2020•2021

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: nucleaire technologie

Masterthesis

Application of neural networks in the analyses of gamma spectra collected during UAV flights

PROMOTOR :

Prof. dr. Wouter SCHROEYERS

PROMOTOR :

Prof. dr. Johan CAMPS

BEGELEIDER :

ing. Stef GEELEN

Daan Van Dyck

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: nucleaire technologie, afstudeerrichting nucleaire en medisch

Gezamenlijke opleiding UHasselt en KU Leuven

2020•2021

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: nucleaire technologie

Masterthesis

Application of neural networks in the analyses of gamma spectra collected during UAV flights

PROMOTOR :

Prof. dr. Wouter SCHROEYERS

PROMOTOR :

Prof. dr. Johan CAMPS

BEGELEIDER :

ing. Stef GEELEN

Daan Van Dyck

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: nucleaire technologie, afstudeerrichting nucleaire en medisch

UHASSELT KU LEUVEN

Acknowledgements

While writing this Master's thesis, I received a lot of support and assistance. Without them, it would not have been possible to achieve this.

First and foremost, I would like to thank prof. dr. Johan Camps in delivering the right guidance and critical insights while writing this thesis and especially for the help in making the right choices for the development of the neural network and creating the dataset needed for this work. Thank you for being stand by for the countless video call meetings and mails.

Furthermore, I would like to thank prof. dr. Wouter Schroeyers, for the excellent guidance during my education and delivering the right help and comments that were needed when encountering some unrealistic results. Many thanks for being stand-by for meetings and steering the thesis in the right direction.

I would also like to show my gratitude towards ing. Stef Geelen, who helped a lot while writing the code, showing me around at SCK CEN and in scheduling the measurements. Otherwise, the process would not have been as fluid as it was. Adding to that, I would like to thank you for lending me some interesting books and being stand by for even the smallest little question.

Moreover, I would like to thank Dr. ir. Fabricio Fiengo Perez for providing very thorough proofreading of my thesis and providing me with amazing and very critical feedback which improved the thesis significantly.

Then I would also like to thank the entire Crisis Management and Decision Support group of SCK CEN, for allowing me to lurk around in the video calls during the very insightful talks and meetings. In specific, ing. Jos Rutten for spending your time reading and correcting my thesis. You gave me very useful feedback allowing the thesis to be corrected substantially.

Finally, I would like to thank my friends and family. Thank you for your belief and support from start to finish.

Index

List of tables

List of figures

Abstract in English

In the characterization of nuclear contamination, UAVs with radiological equipment are an interesting approach for radiation mapping because of their autonomy and flexibility. Yet, UAVs have limitations and combined with the dynamic character of a flight this will present gamma spectra with poor statistics. To tackle the difficulties in quantification and identification, a new approach with neural networks is proposed, where a model is trained on similar data as collected by the UAV system.

This thesis aimed to investigate if it was possible to implement an ANN that could identify ^{137}Cs in various background cases. To achieve this, a DenseNet architecture was implemented with TensorFlow and the model was trained with 6500 spectra of a weak ^{137}Cs source at different distances. This model was tested on a validation dataset which consisted of 2090 spectra collected with differences in time and distance, these results were then compared with a standard radionuclide identification method (i.e. MultiSpect).

The model reached an accuracy of 85% on the validation set, MultiSpect only achieved 44%. At the base of misclassified spectra by the model was a low amount of counts in the photopeak, the minimum amount needed for good results was 34±3 counts. Reducing the number of channels within the spectrum lead to little improvements in accuracy.

In conclusion, with the use of a strong variety of background situations, this method is an effective means for automated radionuclide identification and could be expanded with more radionuclides.

Abstract in Dutch

Door hun flexibiliteit en autonomie zijn UAVs met radiologische meetapparatuur een interessante manier voor de karakterisatie van nucleair gecontamineerde gebieden. Deze UAVs hebben beperkingen (payload, autonomie) en dat gekoppeld met het dynamisch karakter van een vlucht zorgt ervoor dat de telstatistiek beperkt is. Om de moeilijkheden in kwantificatie en identificatie op te lossen, is er een nieuwe aanpak met behulp van neurale netwerken voorgesteld. Hierbij is een model getraind met gelijkaardige data zoals die gezien wordt met het UAV systeem.

Het doel van deze thesis was om te onderzoeken of het mogelijk was een ANN op te stellen die 137Cs kan detecteren in diverse natuurlijke achtergronden. Hiervoor werd een DenseNet architectuur gebruikt met behulp van TensorFlow en een dataset met 6500 spectra met verschillende afstanden. Daarnaast werd het model gevalideerd met een dataset van 2090 spectra met verschillen in afstand en tijdsduur en vergeleken met een standaardmethode (i.e. MultiSpect).

Het ANN behaalde een efficiëntie van 85%, terwijl MultiSpect slechts 44% bereikte. Aan de bron van de misclassificaties lag een laag aantal counts onder de fotopeak, de minimale hoeveelheid voor correcte classificatie was 34±3 counts. Het reduceren van het aantal kanalen in het spectrum leidde tot kleine winsten in efficiëntie.

Tot slot kan er geconcludeerd worden dat deze methode een effectieve manier is om automatisch radionucliden te classificeren. Dit kan uitgebreid worden, mits gebruik van verschillende natuurlijke achtergronden.

1 Introduction

Large scale nuclear contamination, as seen in the events of the Fukushima-Daichi and Chernobyl nuclear power plant accidents, can cause entire regions to be evacuated because of the possible harm that can be done due to the radiological effects [1], [2]. In the case of those contaminations, the main goal is to get these areas decontaminated as effectively as possible. To achieve this, while maintaining the As Low As Reasonably Achievable (ALARA) principle, there is a need to accurately map these areas, to efficiently alleviate nuclear hotspots that can show up due to the dispersion of fission products. In other cases, like the Goiânia accident, there is radioactive contamination due to a forgotten or lost source [3]. Such sources are a risk for public health and have to be localized for effective removal and/or decontamination.

One way to achieve this is by attaching radiation detection devices to Unmanned Aerial Vehicles (UAVs). These UAVs then scan over the contaminated area, to detect the presence of certain radioactive elements. By mapping the activities, hotspots can be localized and analysed by interpreting the spectrum of the source. Currently, a gamma-spectrum analysing software called MultiSpect, with a built-in database is used to find a resemblance between the spectrum and its database. A new proposition is to use Artificial Neural Networks (ANNs) - to automate the identification and analysis in the long term of these contaminated areas. The use of ANNs is rapidly increasing. They are being implemented in numerous applications due to their ability to mimic the human learning process and their capacity for self-correction when exposed to new information. The usual applications ANNs are for classification and regression, yet the possibilities of usage of these techniques extend beyond these fields. This is why ANNs are interesting for a task such as radionuclide identification.

1.1 Problem statement

In this thesis, the UAV of choice is a drone. The main advantage of the use of drones is their versatility This advantage is possible thanks to the various sizes and manufacturers available. The choice of a smaller size for the drone makes it possible to manoeuvre it in urban areas, dangerous or inaccessible locations, without the need for human presence in-situ. This certainly limits the exposure to workers and thus meets the requirements for the ALARA principle. Nevertheless, the choice of small-sized drones has two main drawbacks. Firstly, the limited size is accompanied by a limited battery capacity, this condition reduces the hours of flight and distance covered between recharges. Secondly, the size also limits the weight that a drone can carry. Small-sized drones usually cannot generate enough lift to carry heavy objects. In contrast, larger drones are more capable of carrying loads but at the cost of their manoeuvrability.

These two limitations restrict the capabilities of the detection system to be implemented on the drone. Weight restrictions prompt the use of smaller, more lightweight detectors, which have lower resolution compared to their larger counterparts. The smaller battery capacity subsequently leads to limited measuring time. As a general rule of thumb, longer collection times of spectra lead to better results, that later on facilitates the identification and analysis of the spectral data. Under all these restrictions it difficult for MultiSpect to accurately estimate the radionuclide in the contaminated area. Another drawback of using MultiSpect for spectra analyses is that it can not be performed on the fly but during post-processing. Often, it also requires the addition of multiple spectra to detect the presence of certain radionuclides. Certainly in real-time or in the on-site analysis is desirable where just one spectrum is needed.

Therefore, another technique is required to overcome this limitation in order to achieve more accurate results while using the same data.

One of the most important challenges during the spectra data collection is the determination of the fraction of the background radiation. This is probably the most varying factor since background radiation depends on the location and height of airborne radionuclides, the presence of NORM materials, the cosmic radiation and the probabilistic nature of radiation counting.

Another important problem is that during drone flights, the most varying factor is the fraction of background radiation. Since background radiation is depending on the location and height of airborne radionuclides, the presence of NORM materials, cosmic radiation, along with the probabilistic nature of radiation counting. This all combined makes it difficult to make correct assumptions on spectral data for low counting times.

1.2 Objectives

The main objective is to make an Artificial Neural Network that is able to identify ^{137}Cs in spectra that have been collected in a period of three to five seconds. The developed ANN should perform this identification close to real-time in such a way that the user can be notified as soon as possible in case of the ANN marks the presence of a ^{137}Cs source. In other words, it must complement the expertise of the user to avoid spots passing unnoticed. If the chance of a source being there would be higher than a certain value, the user could receive a prompt. As result, the measurements can be carried out for a longer time in that area, or the height of the measurement in the specified area can be adjusted to quantify better the hotspot. The second objective is to research the capabilities and limitations of the ANN models. This means researching the reason why a model does not predict the expected outcome and attempting to find which part of the gamma spectrum contains the information which is useful for the ANN to achieve the desired output. Since an ANN is still a 'black box', this has to be done by inspecting specific cases where the model fails to predict correctly. The last objective is to compare the ANN-model to a standard method, in specific MultiSpect, with a validation set that will be based on border cases. These border cases would consist out of measurements where there is a limited amount of collection time used for certain measurements and the activity of the source will have large variability, preferably around the point where the peak of ^{137}Cs is almost invisible, challenging both the ANN model and the standard method.

The further course of this thesis will be the following: chapter 2 handles the principles of gamma radiation and the interaction gamma radiation has with materials and chapter 3 shows the principles of scintillation detection these will be shown to understand how gamma spectra are created. This is followed by the different radiological accidents in chapter 4, where there will be a look at how these affected the public and how the remediation was done. Chapter 5 will go deeper into radiation mapping and how UAVs have an impact on this. Subsequently, in chapter 6 there will be an introduction to machine learning and deep learning, rounding off the theoretical background of this thesis. In Chapter 7 the methods and materials for this thesis will be discussed in detail followed by the results and discussion of these results in chapter 8. In chapter 9 a conclusion will be formed and an outlook will be given.

2 Gamma radiation

Radioactive decay occurs when an atom has an excess of energy within the atom which it needs to lose. This causes the atom to rearrange itself internally to achieve this lowest energy state [4]. On certain occasions, this internal rearrangement is not sufficient to reach this lowest energy state, in that case, the nucleus achieves this by emitting some of its elemental particles. By mean of these emissions, the nucleus transforms into another nucleus, reaching higher stability. Such transformations may create perfectly stable nuclei. Frequently, these nuclei are still in an excited state. In this state, the nucleus will rearrange itself again, while losing some of this excitation energy in the form of photons as can be seen in figure 1, where the decays scheme of ^{137}Cs is shown. These photons, which are a quantification of electromagnetic radiation, have no electric charge. The photons which are released by nuclei are often described by gamma radiation.

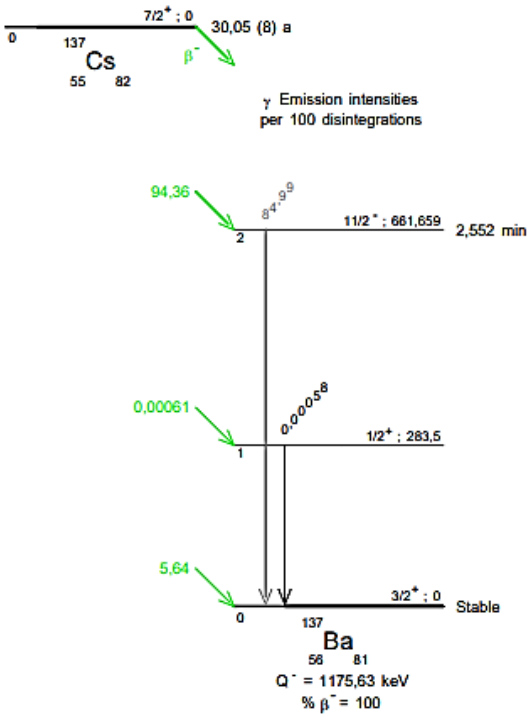


Figure 1 Decay scheme of ^{137}Cs [5]

This gamma radiation is specific for the decay of one certain excited nucleus and thus it contains useful information that can be used for understanding more about that nucleus. As can be seen in the decay scheme of ^{137}Cs, as seen in figure 1, gamma rays have a specific energy and intensity. Although this information is interesting in its own right, its detection still represents an important task to be able to use for practical purposes. In that sense, the question is how these photons interact with matter. In general, detection methods for charged particles rely on the coulomb forces and the excitation and/or ionization of the medium due to these forces. Since photons have no charge, they are not subjected to any nuclear or Coulomb forces; however, they do interact with matter. They primarily interact by photoelectric absorption, Compton scattering and pair production. Other interactions are Rayleigh scattering and Thompson scattering.

With the photoelectric effect, a photon is absorbed by an atom and one atomic electron is released. This released electron is often called the photoelectron, where the kinetic energy of this electron is

equal to the difference between the binding energy of the electron and the energy of the photon. A representation can be seen in figure 2.

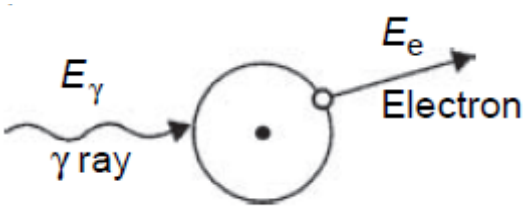


Figure 2 Schematic representation of the photoelectric effect [6]

Where the energy of the atomic electron is equal to:

$$E_e = E_\gamma - BE \tag{1}$$

Where E_e represents the energy of the emitted electron, E_γ is the energy of the incident photon and BE is the binding energy of the released electron.

For this process to happen, there is an important remark that has to be made. For the conservation of the momentum and the energy, a heavy atom is necessary to absorb the amount of momentum to have a little loss in energy. The second interaction process, the Compton scattering, is slightly different as shown in figure 3. There a photon scatters with a free atomic electron, causing it to inherit some of the kinetic energy of the incident photon. The amount of remaining energy of the scattered photon (and thus the amount of energy that scattered electron receives) is characterized by the Compton-scattering formula.

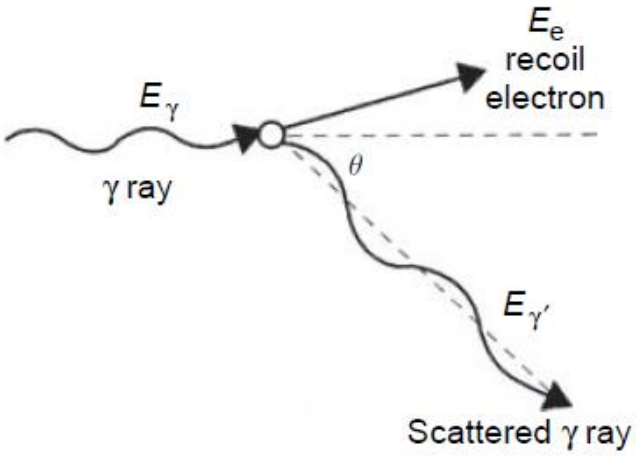


Figure 3 Schematic representation of Compton scattering [6]

The energy of the scattered photon can be calculated with:

$$E'_\gamma = \frac{E_\gamma}{1 + \left(\frac{E_\gamma}{mc^2}\right)(1 - \cos\ \theta)} \tag{2}$$

Where:

- E'_γ is the energy of the scattered photon
- E_γ is the energy of the initial photon
- m is the mass of the photon
- c is the speed of light
- θ is the angle at which the photon has been scattered

The third interaction process is the pair production process. This is characterized by the disappearance of the photon through the creation of an electron-positron pair within the Coulomb field of the nucleus as illustrated in figure 4. It needs to be emphasized that this is only feasible when the energy of the photon is greater than 1,022 MeV.

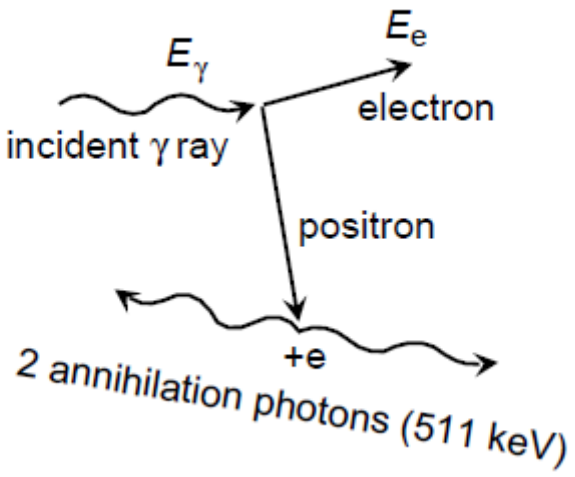


Figure 4 Schematic representation of pair production [6]

As mentioned before, for the conservation of energy, the atom should be massive in comparison to the photon to preserve momentum. However, this can also happen in lighter particles, for example, electrons. When that happens the process is called triplet production. The chance that these interactions occur will depend on the energy of the photon and the atomic number Z of the absorber atoms. In Figure 5 the probability of each of these processes is presented. There, the left line represents the energy where Compton scattering and the photoelectric effect have equal probabilities for a certain atomic number. Analogue to that, the right line describes the equal probability of Compton scattering and pair production.

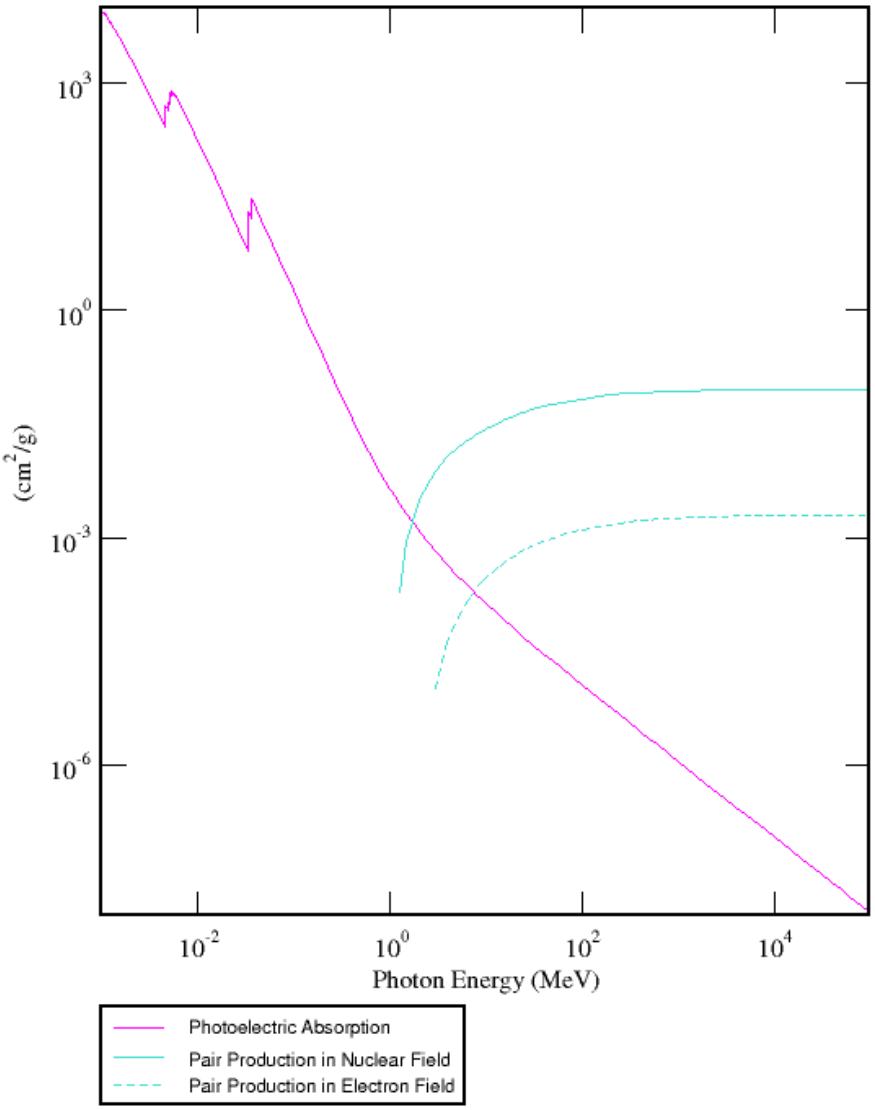


Figure 5 Dominance of the three primary interactions of photons for a CsI detector[7]

For a CsI(Tl) detector, this means that the

In all these processes, the main interaction leads to the production of an electron, with certain energy ranging from zero to the maximum energy of the gamma-ray. All these electrons create charges within the material of a detector, these charges will create electron-hole pairs, also referred to as secondary electrons. It is those secondary electrons that will create an electrical signal and will contribute to a spectrum.

Since every type of interaction depends on (1) the amount of energy of the photon, (2) the types of interactions within detecting materials and (3) the size of the detector, there will be a different amount of reactions happening within the detecting material.

If a large detector can be considered infinitely large, whereas the surface of that detector can be neglected, every gamma-ray will have the opportunity to interact with either one of the possible interactions within the detector's material. All interactions will result in a full energy peak. This is opposed to a small detector, where the gamma rays cannot fully interact in certain interaction mechanisms, due to limitations of size [6]. For these small detectors, the full energy peak will only consist out of photoelectric interactions. The rest of the spectrum will consist out of a single recoil electron, which carries a portion of the Compton scattered gamma-ray. The scattered gamma-ray will leave the detectors material, which loses energy along its path. The loss of energy results in a Compton continuum, reaching from 0 keV to the Compton Edge. The events of pair production will be limited to the so-called double escape peak of the annihilation events because both of these photons will escape from the detector. As can be expected, the real detector lies within these two hypothetical cases. Expected here is that all interaction mechanisms might contribute to the full energy peak There are still other features that present identifiable features within a gamma spectrum. For example, multiple Compton events in which multiple scattering events occur and introduce energies that extend past the Compton edge and before the full energy peak. Another feature could be the single escape peak, in which the only one of the annihilation photons is absorbed, while the other photon escapes from the detector. The energy in which this happens is 511 keV less than the energy of the full energy peak. Figure 6 sums up how all these events would translate to a gamma spectrum.

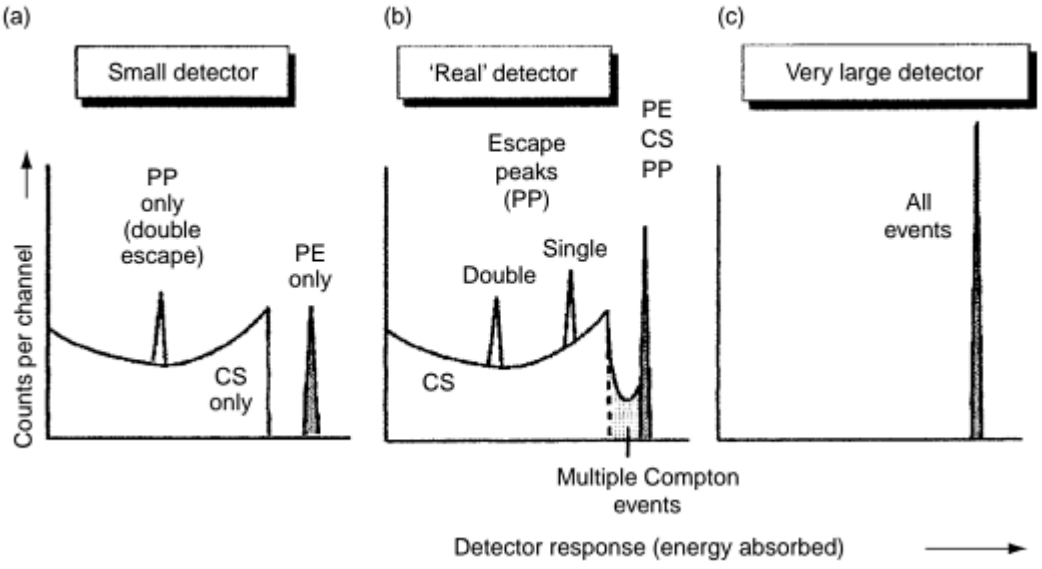


Figure 6 Summary of possible interaction processes [6]

A practical example of a ^{137}Cs spectrum with a subtracted background, measured with a CsI(Tl)-detector is presented in figure 7, where the Escape peaks and Compton edge can be seen:

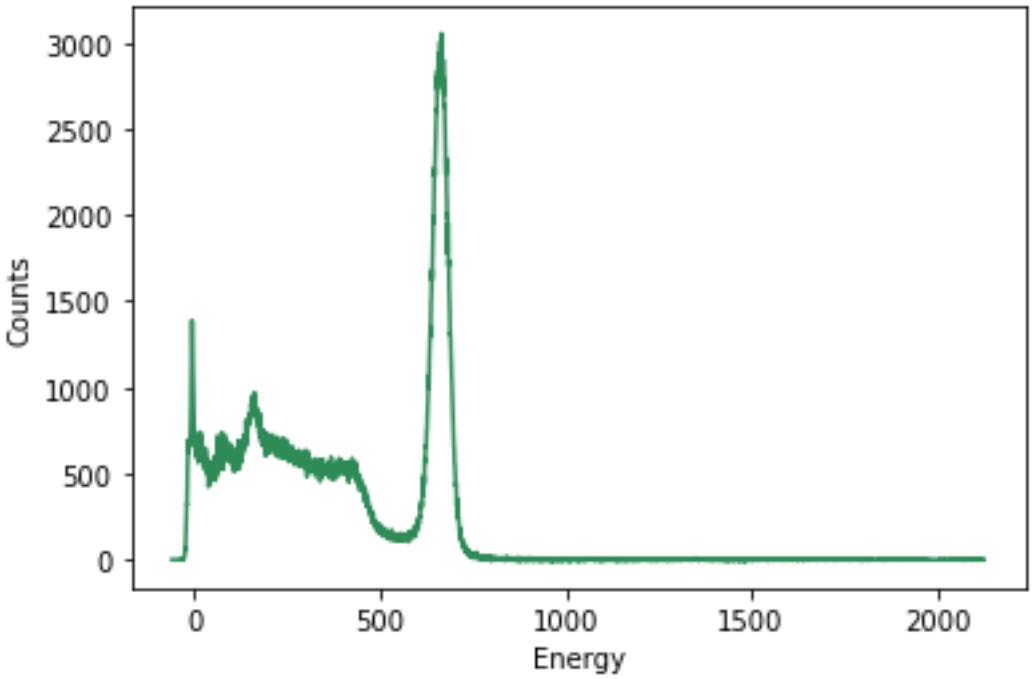


Figure 7 ^{137}Cs spectrum with a Kromek SIGMA50 CsI(Tl)-detector.

3 Gamma-ray spectrometry

As seen in the previous chapter, gamma radiation has three different interactions that can happen within a certain medium. Fortunately, these interactions provide a way to quantify them. Specifically by taking advantage of these interactions to interchange energy with electrons within a medium. In this way the atoms of a medium will be ionized and later by this ionization, these liberated electrons then get collected, either indirectly or directly, and used to quantify the incident photon. The collection of these liberated electrons varies according to the medium of the detector.

3.1 Inorganic scintillation

Scintillation is the process that happens when ionizing radiation produces light within a certain material. This is one of the oldest, but useful techniques for the detection and spectroscopy of different nuclear decays [8]. The mechanism within inorganic scintillators depends on the energy states that are determined by the crystal lattice of the material. Within a material there are certain discrete energy levels in which electrons are present, these energy levels are frequently called bands and are represented in figure 8. The lower band, also known as the valence band, consists out of electrons that are bound at lattice sites, unlike the conduction band in which electrons with sufficient energy can freely travel throughout the crystal. Since both bands consist out of discrete energy levels, in the region in between (the called the forbidden region) electrons are never found within the crystal. In case of absorption of a fraction of energy from radiation, an electron from the valence band will be promoted to the conduction band. This promotion creates a vacancy in the valence band while the electron will stay in the conduction band for a short period of time before returning to the valence band. As results of this process, a photon is created, although it is not in the visible light spectrum. The visibility of these photons is enhanced by adding impurities to the crystal material, named activators. These activators create vacancies within the forbidden region, producing the energy needed for recombination of the lower band. This results in a visible photon, which is at the base of the scintillation process.

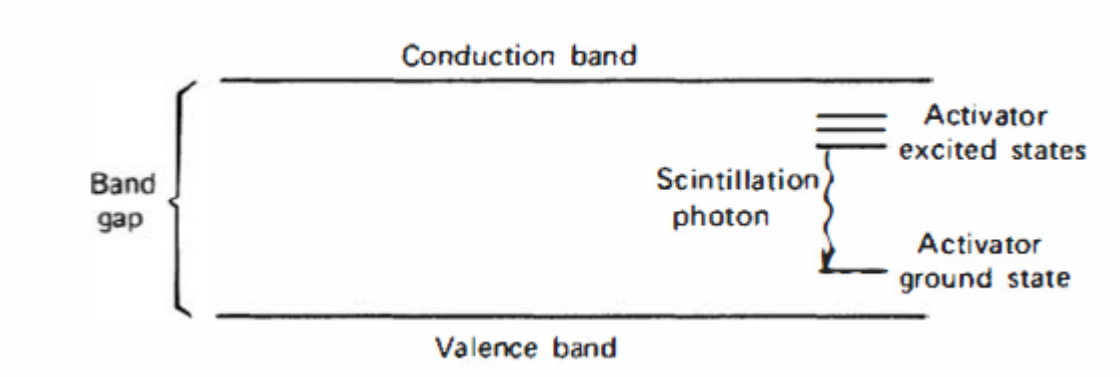


Figure 8 Energy band structure of crystalline detector [8]

When a charged particle interacts within the detector material, it will create electron-hole pairs, also known as primary electrons, causing the holes to drift to an activation site that results to ionize them. Simultaneously, the electron will be able to migrate freely through the crystal, until it encounters an ionized activator. By this recombination, a photon is produced. Moreover, if the activator was the right one, the produced photon will be in the visible light range. Other processes that can occur within this type of material are for example (1) phosphorescence, which is the process where heat causes thermal excitation that results in a significant amount of background lights in the scintillators, (2) the quenching, which happens when an electron is captured at an activator site, resulting in radiationless transitions.

3.2 Photomultiplier tubes

The light that has been created by the photon will be used to quantify the energy of the initially charged particle [6]. The generated light will be used in a photomultiplier tube, which converts light from the incident photon into electrons. This process of photoemission happens in three distinct steps, (1) the conversion of a photon into an electron, followed by (2) the migration of the electron to the surface of the material and (3) the escape of said electron. The energy that can be transferred from a photon to an electron depends on the quantum energy of the photon. The process within the photomultiplier tube (PMT) can be seen in figure 9. The photon that has been created by a scintillator interacts with a light-sensitive layer. Due to this interaction, a photoelectron is emitted. Subsequently, this photoelectron is focused onto dynodes, which are a series of electron multiplier stages that emit more electrons than they receive. This causes an amplification of the signal. Within a PMT there are multiple of these dynodes, creating an amplification down the chain unto an anode. This anode will pass the signal to a measurement circuit.

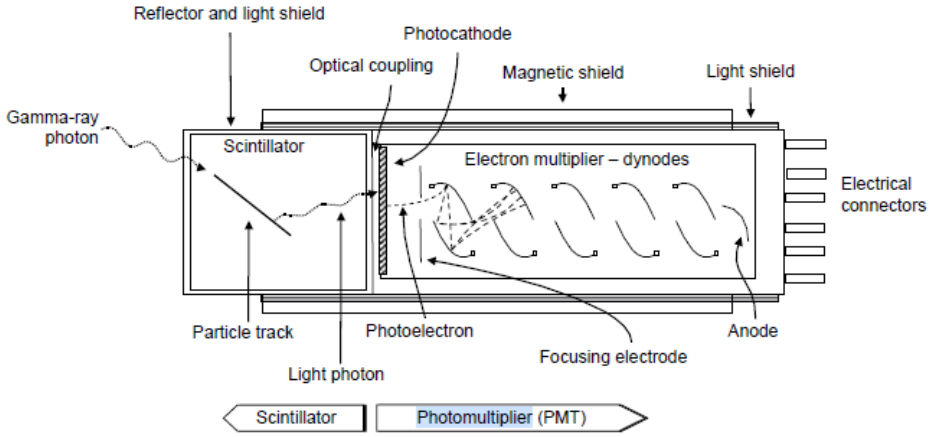


Figure 9 Schematic diagram of scintillator and PMT combination [6]

The output of the PMT presents a certain amount of electrons in the form of a charge. This charge is proportional to the number of electrons that have been created within the PMT. These pulsed charges are collected and sorted by a multichannel analyzer (MCA) by pulse height and counts the number of pulses. This all creates the gamma spectrum which can be used to analyze the radionuclide present in the vicinity of the detector.

3.3 Detectors for airborne environmental radiation monitoring

Since the accidents of the Fukushima-Daiichi and Chernobyl nuclear power plants, mapping the radiation has become integral in finding the location of radiation hotspots [9]. The principle is to quantify the radiation intensity at a certain place within a predefined region of interest within the environment by using either ground-based or airborne measurements. These ground-based measurements (on foot or with a vehicle) implement close scanning patterns and a portable measurement system delivering good results, but due to the point-based measurements, some hotspots may be overlooked. Moreover, the time needed to scan a large area is greater than for airborne measurements. Ground-based measurements will be used to calibrate and validate other methods of measurements or for situations where spatial accuracy is needed rather than as a standalone method due to the slower scanning process. Airborne measurements allow for much more consistent scanning patterns, which include a higher spatial resolution due to the higher field of view the detector has. Although this is true, the height at which a crewed aircraft is allowed to fly is too high for good spatial resolution, which would result in partially overlapping scanning laps. Subsequently,

the overlap would require serious deconvolution to localize the areas of interest. UAVs present a good opportunity to fly closer to the surface which results in an improved spatial resolution, with the added benefit of having a lower overall financial cost.

Historically, the industry standard for airborne measurements were large volume detectors, which means that they require a large aircraft to be used for large scale measurements, these detectors were usually CsI(Tl) scintillators. These larger detectors are good for several reasons due to the increased photon stopping potential. The problem with these large volume detectors is the fact that these cannot be used on a UAV, due to restrictions in the payload. Adding payload to a UAV system reduces the operating time considerably, which means that this drone would lose some essential autonomy in the process. So to account for this, some smaller volume detectors must be considered for these problems. These smaller volume detectors should have a good energy resolution and detection efficiency, added with a high number of optical photons per MeV of energy gathered by the crystal volume. The ability to stop photons should also be sufficient, thus heavy Z materials pose good candidates for this. Lowdon et al. [9] have researched the possible detectors that are good candidates for this application. This research has shown that $LaBr_3$ shows the most promise in these tests, because of the higher detection efficiency, increased count rates and excellent spectral resolution with $CeBr_3$ coming in second. These results came from a GEANT4 simulation, yet the research also suggests that the hygroscopic behaviour could affect these results. CsI(Tl) is relatively invariant for this hygroscopic behaviour and has relatively similar qualities compared to the $LaBr_3$ detector system [10]. The main differences lie in the lower resolution and the higher decay time than $LaBr_3$ as can be seen in table 1.

Table 1 Considered detectors for UAV research [9]

Material	Peak Emission (nm)	Light Yield (ph/MeV)	Density (g/cm³)	Attenuation @ 1.5 MeV	Energy Resolution (@ 661.7 keV)	Decay Time (ns)
NaI(Tl)	415	38,000–55,000	3.67	114.7	7%	250
CsI(Na)	420	38,000–44,000	4.51	140.9	5.8%	630
CsI(Tl)	540–550	52,000–65,000	4.51	140.9	6.9%	1000
BGO	480	8,000–10,000	7.13	222.8	9.7–16%	300
GAGG(Ce)	520	22,000–60,000	6.63	207.2	5.1%	87
LYSO(Ce)	420	30,000–33,000	7.1–7.2	225.0	8-20%	45
$LaBr_3$	380	63,000	5.08–5.22	158.8	2.6–3.5%	16
$CaF_2(Eu)$	435	19,000–30,000	3.19	99.7	5.4%	950
$CeBr_3$	380–390	57,000–66,000	5.1–5.2	159.4	3.8–4%	18–20
$SrI_2(Eu)$	435	80,000–115,000	4.55	142.2	2.8–4%	1200

4 Accidents

Throughout history, there have been several accidents that have been listed on the INES scale. The INES scale is used for communicating the safety significance of events concerning the exposure of the public to radiation sources. If such an event would occur, it would get rated from zero to seven, with zero having no safety significance and 7 being a major accident. A visual aid to this scale can be seen in figure 10. In this chapter, some of these accidents will be highlighted because of their significant repercussions to the public due to ionizing radiation. Every accident will be handled in the same way, discussing the accident, the effects on the public and the remediation of the accident. Discussing these events side by side will give extra insights into the way decontamination is done on such a scale and the different measures that could be taken to apply the ALARA principle in new accidents. The events in specific are the Goîania accident (INES 5), and both the Chernobyl (INES 7) and Fukushima-Daiichi (INES 7) nuclear power plant accidents.

Figure 10 INES Scale [11]

4.1 Chernobyl 1986 (INES 7)

During a test procedure of the RBMK-1000 reactor of the Chernobyl nuclear power plant, the emergency core cooling system was blocked in combination with the plant being operated at half power (1600 MW(th)) for 11 hours [12]. The desired operating power was between 700 and 1000 MW(th), this is why the power went below the minimal allowable operating power, which was 700 MW(th) resulting in dire conditions. Due to operational errors, the power fell to 30 MW(th), which caused the operators to attempt to increase the power again to maintain the level which was originally

planned for the test. Subsequently, their efforts were not successful due to xenon poisoning, reduced coolant void and graphite cooldown. To compensate for this, the choice was made to withdraw some of the control rods which brought the power back to 200 MW(th). Although the power level was stabilized, the Operating Reactivity Margin was violated. With the power at 200 MW(th), the reactor was in an unstable condition when the test was initiated. During the test, the main coolant pumps ran down, leading to a reduced flow rate of the coolant and an increase of nominal power of 15% and within a few seconds a reactor excursion occurred. The reactor excursion led to the rapid increase in the fuel's energy release and the increased heat was transferred to the coolant. The coolant evaporated, increasing the pressure in the reactor vessel and this led to the explosion of the reactor. Figure 11 shows the power sequence of the reactor.

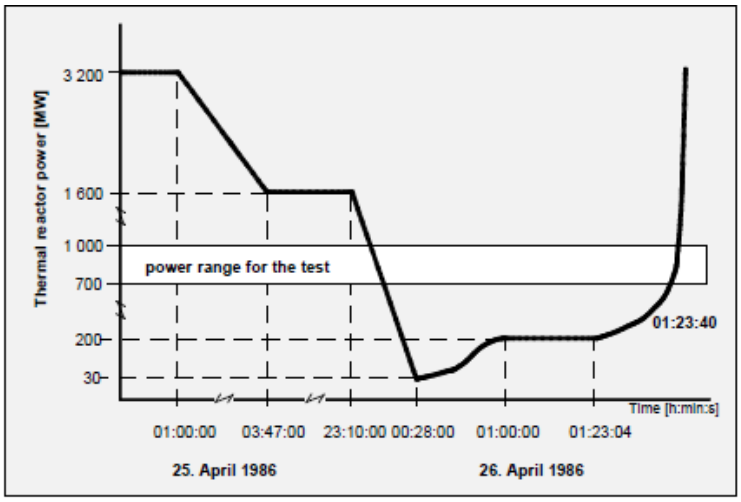


Figure 11 Sequence of the reactor power [13]

According to UNSCEAR, there were 600 workers present on the site right after the events of the accident, 134 of them received a high dose (0.8-16 Gy) and suffered from radiation sickness. Of these workers were also first responders who had no idea of what happened. Among these heavily affected people, 28 died within the first three months and 19 died from causes not directly associated with radiation. Along with the workers, the public was also heavily affected, the group was around 115 000 persons who needed to be evacuated outside the contaminated area. Subsequently, this group received an effective dose of 30 mSv. People who stayed within this area received 9 mSv the following two decades. To remediate the entire region, there were 530 000 registered workers which received a dose between 0.02-0.5 Gy. Due to these events, the average effective dose received by members of the public in distant European countries is estimated to be 1 mSv over the course of a lifetime. This shows how immense the impact of this accident has been on the entire European continent, but particular to Belarus, Ukraine and the Russian Federation. These three countries have seen large aftermath due to the late effects of the accident. In 2005, there were more than 6000 thyroid cancer cases diagnosed, particularly among children, likely due to the consumption of contaminated milk.

The explosion of the reactor resulted in one of the greatest releases of radionuclides to the atmosphere since the nuclear bomb tests. Since there was no external shelter around the reactor building, the core was exposed to the environment yielding enormous releases that would otherwise be contained. The estimated principal released radionuclides can be seen in Table 2. The public got exposed in three possible ways, external dose from cloud passage and radionuclides deposited on the soil or internally by inhalation of the radionuclide cloud or ingestion of contaminated crops and contaminated milk. To have a general idea of the spread of the radionuclides, the choice was made to use ^{137}Cs to model this, since it is of great significance and easy to measure with spectroscopy. Since heavy particles such as

strontium or plutonium bond to larger particles and were deposited within a 100 km range of the power plant. The spread of the airborne ^{137}Cs over the European continent was immense, luckily this was thoroughly researched and documented in the UNSCEAR report of the Chernobyl accident. . Remediation was conducted in the vicinity of the Chernobyl power plant and at distances from 0.5-15 km from the nuclear power plant. To manage the waste, several temporary waste repositories were needed. One being the shelter, built to contain the materials present at the nuclear power plant and the rest for the contaminated surroundings.

Table 2 Estimated releases during the course of the Chernobyl accident [1]

	Half-life	Activity released (PBq)
Inert gases		
Krypton-85	10.72 a	33
Xenon-133	5.25 d	6500
Volatile elements		
Tellurium-129m	33.6 d	240
Tellurium-132	3.26 d	~1150
Iodine-131	8.04 d	~1760
Iodine-133	20.8 h	910
Caesium-134	2.06 a	~47[b]
Caesium-136	13.1 d	36
Caesium-137	30.0 a	~85
Elements with intermediate volatility		
Strontium-89	50.5 d	~115
Strontium-90	29.12 a	~10
Ruthenium-103	39.3 d	>168
Ruthenium-106	368 d	>73
Barium-140	12.7 d	240
Refractory elements (including fuel particles)[c]		
Zirconium-95	64.0 d	84
Molybdenum-99	2.75 d	>72
Cerium-141	32.5 d	84
Cerium-144	284 d	~50
Neptunium-239	2.35 d	400
Plutonium-238	87.74 a	0.015
Plutonium-239	24 065 a	0.013
Plutonium-240	6 537 a	0.018
Plutonium-241	14.4 a	~2.6
Plutonium-242	376 000 a	0.00004
Curium-242	18.1 a	~0.4

Recently, due to technical advancements, the fallout in the Red Forest was mapped using a UAV system by first using a fixed-wing system at 65km/h in a grid pattern and then hotspots got followed up with a rotary drone for a higher resolution [14]. These data points of the radiation maps then got superimposed on a regular map of the area and this shows that there are still some radioactive hotspots present which can still cause harm for tourists with a morbid curiosity. For a full dose map,

all points between the measurement points will be interpolated to achieve a gradient in the map. The map they made can be seen in figure 12 Figure 12A shows the map made by the rotary drone and figure 12B shows the map made by the fixed-wing drone.

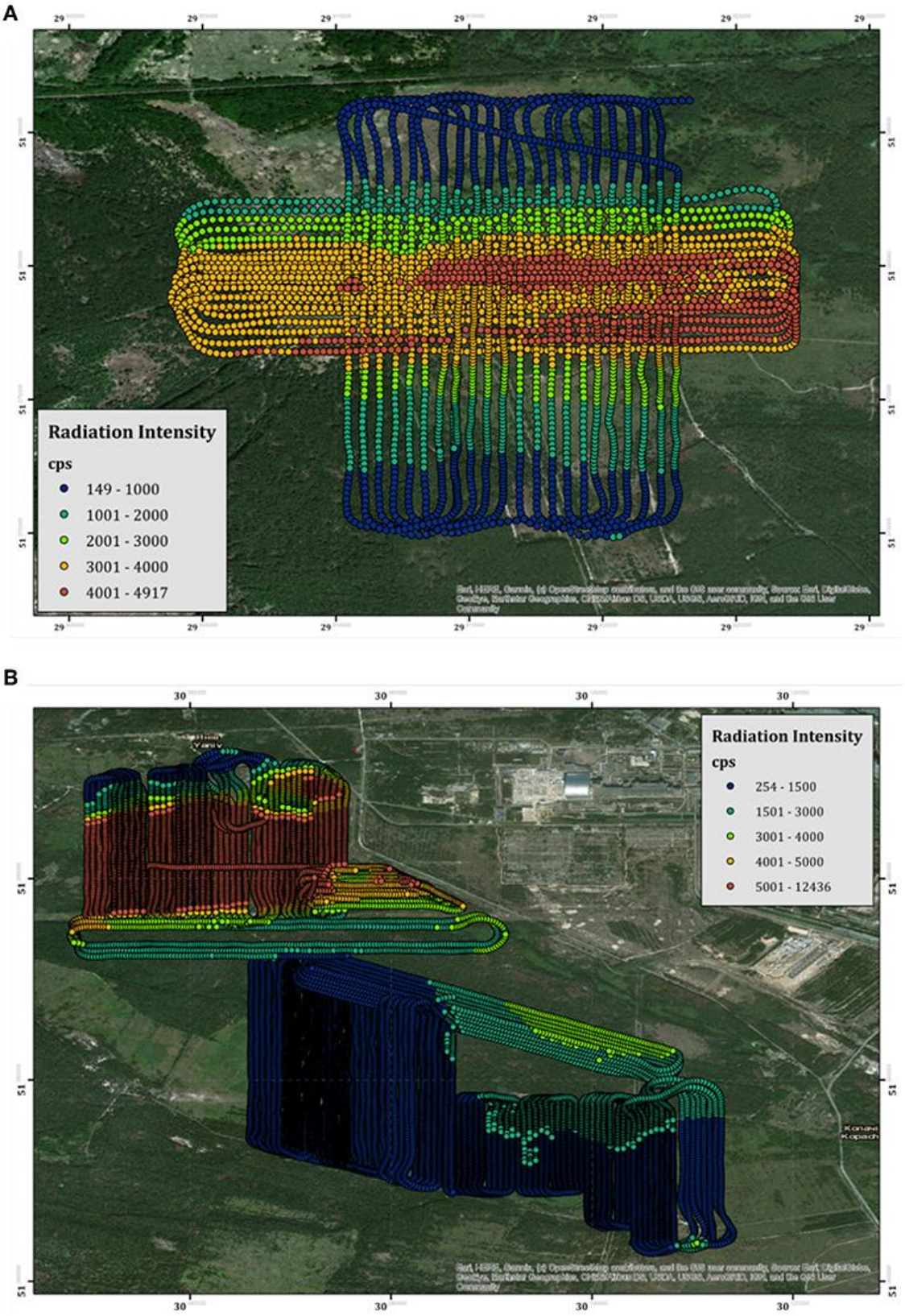


Figure 12 Measurements collected by the UAV in the Red Forest, near the Chernobyl Nuclear Power Plant [14]

4.2 Goîania 1987 (INES 5)

In 1985, a teletherapy unit of the Instituto Goiano de Radiotherpia in Goîania, Brazil, was left behind during the relocation to a new premise [3]. The teletherapy unit in specific was a ^{137}Cs unit that was demolished together with the former premises of the radiotherapy institution. This resulted in the teletherapy unit becoming insecure. After these events, two members of the public entered the former premises and found the source assembly, took it home and subsequently decomposed it as they thought it might have some scrap value. During the dismantling, the source capsule was damaged, contained in the capsule was a caesium chloride salt, which is highly soluble and can be easily dispersed. Subsequently, the material was then sold to a junkyard holder, who mentioned that the material glowed blue in the dark, sparking a sense of fascination that resulted in a dispersion of the material to several families. The size of the distributed fragments was around the size of rice grains.

After five days, the number of people who were exposed to the material was showing gastrointestinal symptoms, which were not initially linked to radiation. This relation was unsure until one of the victims took the remnants of the material to the public health department, which started a chain of events resulting in the discovery of the accident. From this accident, 20 persons were identified as a victim, needing medical care, from which four casualties were noted due to acute radiation syndrome. The estimated dose of these victims ranged from 4.5 to over 6 Gy. In total, the events of the Goîania accident resulted in the monitoring of 112 000 patients, of whom 249 were either internally or externally exposed.

The environment was severely contaminated, resulting in the evacuation and monitoring of the area. The decontamination of the accident was divided into two distinct phases. The first phase was an attempt to bring control over the possible sources of contamination, the second was the remedial phase, aiming to restore normal living conditions to the area. The first phase was to identify the sources and the initial surveys were conducted on foot. In this campaign, seven of the main contaminated areas were identified. Additionally, an aerial survey was conducted with a helicopter, which showed that no areas were overlooked. Over 67 square kilometres of urban areas were overlooked during the first two days and over 85 houses were found to have significant contamination. The remediation phase lasted about three months (Christmas holidays 1987 – March 1988). An investigation of soil, ground water, sediment and river water, drinking water, air and food supplies was conducted. This investigation had shown that the levels of contamination in drinking water were low and that there were only countermeasures needed for soil and fruit within 50 meters of the main hotspots as seen in figure 13. The waste was packaged in 3800 metal drums, 1400 metal boxes, 10 shipping containers and 6 sets of concrete packaging. All in all, about 3500 m³ of contaminated material was stored.

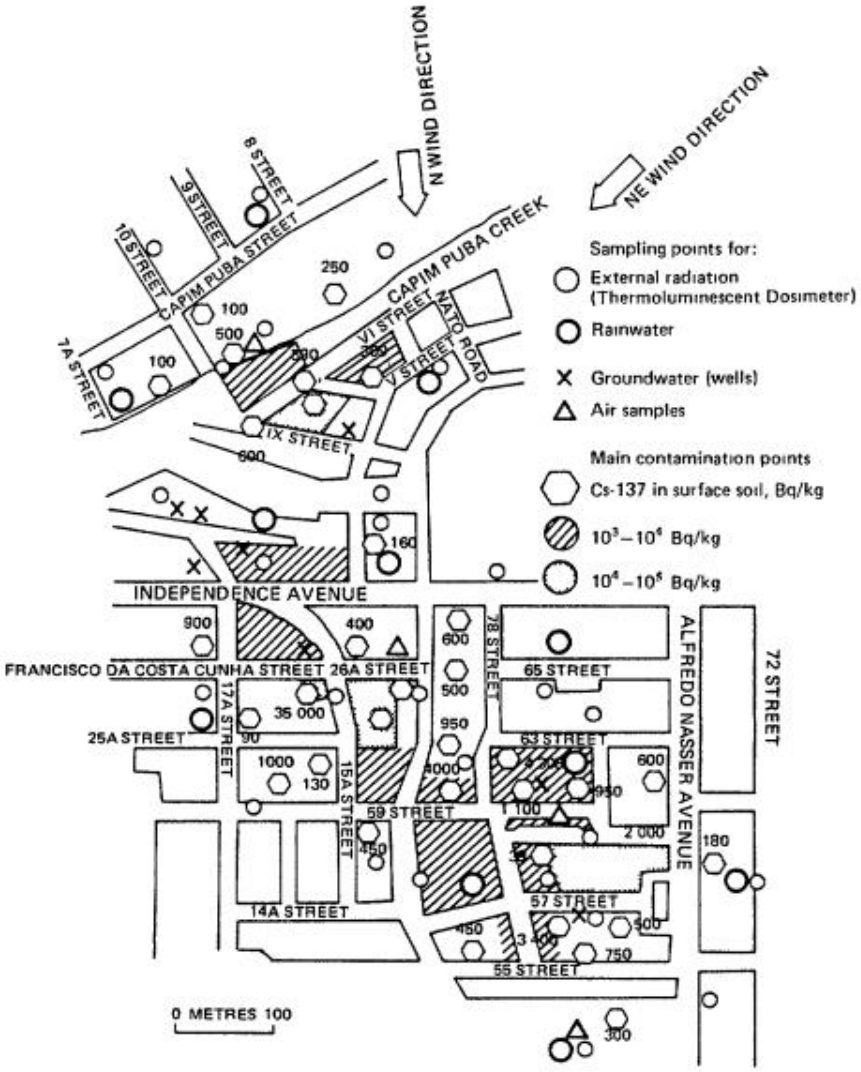


Figure 13 Plan of the principal contamination hotspots of the Goiânia accident[3]

4.3 Fukushima Daiichi 2011 (INES 7)

On 11 March 2011, the Great East Japan Earthquake with a magnitude of 9.0 caused a tsunami which led to great devastation in Japan [15]. Over 15 000 people died, 6000 got injured and 2500 people are still missing. At the time of the earthquake, three of the six reactors at the Fukushima Daiichi Nuclear Powerplant (FDNPP) were operating, the remaining three were shut down for refuelling. The operating reactors were shut down by the safety systems which detected ground motion. Since the fuel was still hot, the cooling system had to be running, but the magnitude of the earthquake inevitably caused problems on the power grid. As a result, the backup diesel generators had to be used for driving the pumps and removing the decay heat to prevent other problems. At this point, the earthquake had already done a lot of harm, but most of the damage was done by the tsunami which followed afterwards. The plant was equipped with seawalls, which were designed to protect it against a maximum height of 5.5 m, yet this was not enough because the wave of the tsunami was approximately 14-15 m. From this moment, the diesel generators got flooded and a station blackout occurred. All units on the site had DC-backup batteries in case of a station blackout, yet due to the flooding, units 1,2 and 4 had no DC power and therefore they could not monitor the essential plant parameters. Units 3,5 and 6 were able to monitor their plant parameters and in unit 3 the pressure rose, which lead to the automatic opening of safety release valves and the operators restarted the reactor core isolation cooling system and shut off non-essential equipment to stretch the lifespan of the DC batteries. Unit 5 was not generating steam and the reactor vessel continued to heat up and pressurize. Unit 6 did not suffer a station blackout, the reactor was not at atmospheric pressure and

the only damage was done by seawater. Conditions in units 1 and 2 were dire, the operators declared a nuclear emergency based on the 'inability of water injection of the emergency cooling systems'. This declaration of nuclear emergency resulted in an evacuation of an area of 3 km around the nuclear power plant. This was later enlarged to 20 km.

In unit 1, the containment vessel pressure had exceeded the maximum design pressure and this had to be vented, meanwhile, the water levels surrounding the fuel elements was dropping. This resulted in the fuel being exposed to air and subsequently, the fuel temperature rose and the fuel rod started to melt. At the same moment, the zirconium cladding in combination with the steam of the remaining coolant was producing hydrogen. This hydrogen escaped through the vents and started to build up within the containment building. Within the containment building, there was a mixture of steam, aerosols, noble gases and hydrogen gas, which caused an explosion. In units 2 and 3 the fuel also melted, but to a lesser extent than unit 1. For unit 2, the operators had chosen to cool the core with seawater to remove the decay heat. Since this happened after the explosion of unit 1, the operators opened the blowout panel to avoid the same happening. The efforts were not enough to prevent the core from melting due to the rapid drop of the water level in the reactor vessel. It has appeared that most of the radioactive releases came from unit 2 due to a leak in the primary containment. In unit 3, the main backup water injection failed, causing water levels to drop rapidly, again the same countermeasures of cooling with seawater were undertaken, the unit was also regularly vented. Due to this venting, there was a very large explosion which blew the roof of. In unit 4, there was also an explosion which resulted in further damage to the structure of unit 3.

The events that happened within the FDNPP were multiple plumes that contained ^{137}Cs and ^{131}I and other volatile fission products. The spread was not only over the entire country of Japan but also had effects on the ecosystems present in the seawater around the Japanese coastline. The spread of these nuclides heavily contributed to a significant increase in exposures on the Japanese mainland. Members of the public got exposed in four different pathways, either internal or external. Internal exposures may result from either ingestion of affected crops or contaminated beverages or inhalation of airborne radioactive particles. External exposure may be caused by submersion in a radioactive cloud or from the radiation of deposited radionuclides. The effective dose that has been received in the first year after the accident was estimated at less than 4mSv, with lifetime effective doses being about 15 mSv. From the accident, there have been no deterministic health effects or deaths observed until 2020.
For the contaminated area to be remediated as fast as possible, there was a need for accurate dose maps, this was done with either grid of individual measurement points made with handheld detectors or continuous measurements with car-borne detectors, a backpack and a remote-controlled helicopter. [16] The actions that have been undertaken in this region were washing of surfaces, abrasion of surfaces or extraction of surface layers, removal or replacement of surfaces and vegetation removal. Overall this has been a very costly endeavour, resulting in the displacement of several millions of cubic meters of soil in the area surrounding the accident.

Lately, new endeavours to map the radiation with UAVs have been conducted in the Fukushima prefecture by the IAEA Nuclear Science and Instrumentation Laboratory [17]. This was done by equipping radiation detectors, GPS devices and cameras which are used to send information to the pilot. The radiation detector in this case was a GM tube, which only presents cps as a measure to map the radiation. Not having any information concerning the material which is present. To make the map in figure 14, the researchers of NSIL flew at an altitude of 10 m, the results of these measurements got superimposed on a LIDAR map, allowing for a very detailed dose map.

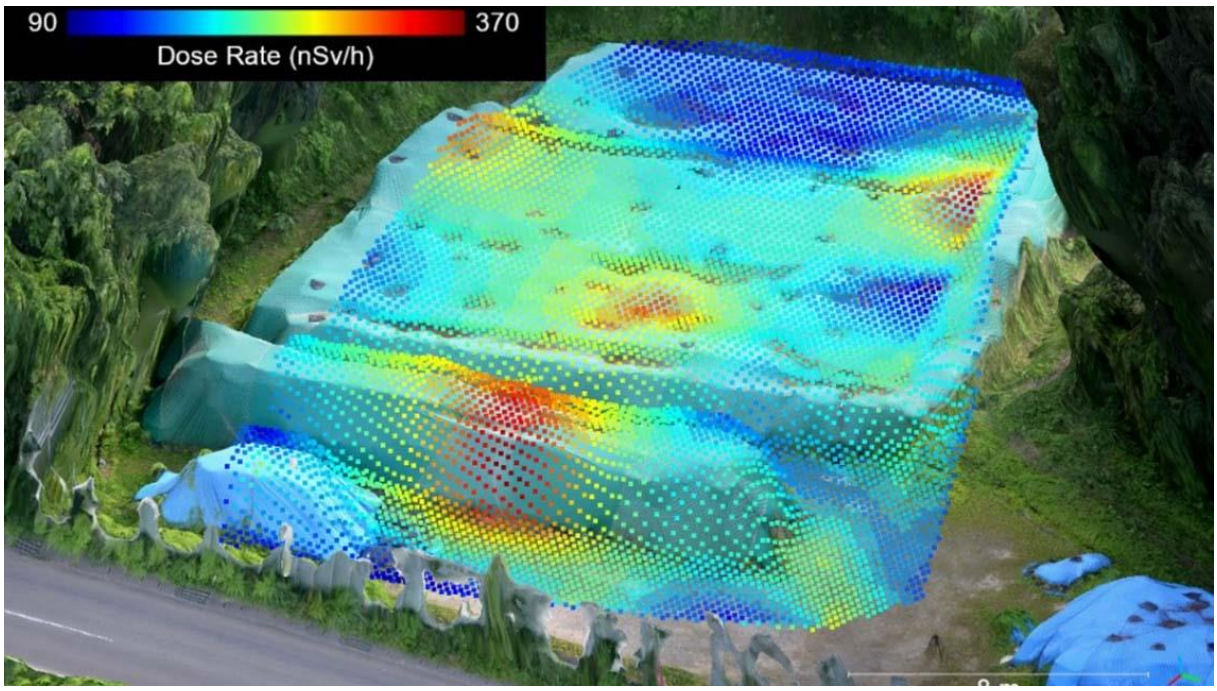


Figure 14 3D map showing the superimposed radiation on a LIDAR map of the area [17]

5 Radiation mapping

As can be concluded from the decontamination done in the aforementioned nuclear accidents, with any nuclear contamination, public health is at stake. If fission products are released into the atmosphere, the spread of these can severely affect the environment, crops and water supplies. Decontaminating these areas should happen as efficiently and effectively as possible to maintain the ALARA principle, thus ensuring the safety of workers doing the decontaminating. Similarly, in the events of an orphan source or the events of a terrorist attack, first responders need to be harnessed against all possible effects of ionizing radiation. The general idea behind the remediation of contaminated areas is similar across all the accidents, the first step is measuring the spread of the released radionuclides and locating hotspots. The second step is using these maps to be as effective as possible in the decontamination of the affected areas/hotspots.

A way to make these decontamination problems easier is by using radiation maps to point out radiological hotspots, but also create a scale for the accident. In the events of a major release on the European continent, the Joint Research Centre has the Radioactivity Environmental Monitoring (REM)-group who has a network of detection systems, which is used to monitor the events of large-scale accidents. An example of measurements of a similar system has been used to map the spread of ^{137}Cs in the European continent during the events of the Chernobyl Nuclear Powerplant accident and is well documented in the UNSCEAR report [1]. Such a system is interesting for monitoring, for decontamination of specific areas this network does not suffice, so for high detail maps the normal way of operation is using foot-based measurements. While these generate improved details within the radiation maps, working this way is costly and requires a lot of time. This coupled with a significant amount of exposure to ionizing radiation, means that this technique is something that has to be improved. Another drawback is that the human body attenuates as much as 30% of the incident gamma rays that otherwise would be collected by the detector [18]. Other possibilities to map the radiation within an area is using either military or public aerial vehicles, carrying heavy and expensive equipment to make radiation maps at altitudes of 150-700 meters [19]. In the case of the Fukushima Daiichi incident, the Japanese aviation law prevented higher spatial resolutions since manned helicopters are prohibited to fly at lower altitudes than 150 meters. these measurements were mainly done to check the effectiveness of the remediation process.

Hence the research of Martin et al. and MacFarlane et al., which have proposed a solution to this problem by implementing Unmanned Aerial Vehicles (UAV) to map contaminated sites of a former uranium mine [18], [20]. MacFarlane et al. have suggested a low-cost and lightweight UAV with a microcontroller and integrated Kromek GR1 CZT gamma spectrometer, GPS and LIDAR, which was able to detect radiation anomalies with a spatial resolution lower than 1 meter. The drone shown in this research had a possible flight time of 12 minutes and to identify the used source, a spectrum was collected for 800 seconds. The results from this study have shown that flying at low altitudes has improved results for the spatial resolution of the radiation map, and the lower a drone is able to fly, the lighter and smaller the detector can be due to the increased sensitivity. This all is due to the applicability of the inverse square law, which states that the intensity of a radiation source is inverse proportional to the square of the distance, or in this case height of the measurement. This is only true for heights of measurements from 1-10 m, otherwise, there has to be a correction with the exponential fall-off of the radiation intensity.

Martin et al [21]. have used the research of MacFarlane et al. to apply this in the field. To make a high-resolution map, the setup presented by MacFarlane et al. was implemented to fly over the contaminated site with a pre-determined route. The route resulted in an encompassing survey grid

and due to the differences in height needed (trees obstructed the flight lines) which led to heights varying from 5-15 m. Due to these contrasting differences in heights, Martin et al. were not able to make a regular survey grid. Furthermore, the speed of the drone used to fly over the grid was 1.5 m s^{-1}. In order to make a specific dose map, the gamma spectrum was converted to cps value, this converted value then got normalized for height above the surface using the inverse square law. In this research, all spectral data was added up for four specific areas to do isotopic fingerprinting. They concluded that this technique was able to provide high-resolution radiation maps for real-world environments, with a much greater rate of collection as compared to traditional systems. This is then accompanied by the ability to determine the nature of the contamination with isotopic fingerprinting. Following this research, Martin et al. have implemented this method in the highly contaminated Kawamata region, surrounding the Fukushima Daiichi accident, to see if the results presented in the study in 2015 also apply in highly contaminated areas [21]. In general, the same process was used as in the previous research, with some slight differences The measurements were carried out at a speed of approximately 1 m s^{-1} and the flights were done at a height of 1-5 meters above the surface and a grid separation of 1.5 m between every flight line. The results as shown in the previous study remained the same. Yet the fingerprinting showed interesting results, containing primarily the specific peaks of several caesium isotopes.

A problem related to these drone measurements is the fact that there are often limits to the number of counts that can be detected. In some regions, the amount of radiation only depends on the background radiation, hence there being a certain Minimal Detectable Amount, or MDA[6]. This MDA is also used in the measurement of environmental samples, it is often important to justify whether or not a sample is radioactive. Therefore, the MDA is used as the upper limit measured in a sample that is not justifiable and in some sense acts as the uncertainty of a certain measurement. In this sense, the MDA of a certain measurement can be calculated with [6]:

$$\mathrm{MDA} = \frac{L_D}{\epsilon * P_\gamma * t_c} \tag{3}$$

Where L_D is the limit of detection in counts, ϵ is the efficiency of the detector at the specific energy of the gamma-ray, P_γ is the probability of the gamma-ray and t_c is the live time count. L_D depends on the degree of confidence of the measurement. This is calculated with the following formula for a confidence of 95% for detection of a peak:

$$L_D = 2.71 + 3.29\sigma_0 \tag{4}$$

Where the uncertainty of the background is described with σ_0 and this then is described by the square root of 2 times the amount of background counts within a certain ROI. If the total background rate is defined by B_R, then the measured background over a ROI with n channels is described by:

$$\sigma_0 = \sqrt{2B} \tag{5}$$

$$B = B_r * t_c * n \tag{6}$$

Where B is the number of background counts. The n channels within the ROI can be described by the FWHM of the peak and can be included within the background as:

$$B = B_r * t_c * \ FWHM \ * \ F/ECAL \tag{7}$$

Here F is a factor introduced for the amount of coverage and ECAL is the energy calibration, to calculate the energy back to channels. When this all is substituted for L_D in the formula, this becomes:

$$\mathrm{MDA} = \frac{2.71 + 3.29\sqrt{2(B_r * t_c * \ FWHM \ * \ F/ECAL)}}{\epsilon * P_\gamma * t_c} \tag{8}$$

The result of this equation has to be as low as possible and will primarily affect the choice of the detector. A lower MDA leads to better results in low count rate systems. Generally, the MDA for two detectors will be compared, to choose the best detector for the application at hand.

6 Basic principles of neural networks and deep learning

6.1 Machine Learning

Neural networks and deep learning is a sub-branch of Machine Learning, this is the field of study where machines are trained to do human-like tasks [22]. The general idea is that rather than explicitly programming code that sets the rules for the program, the machine defines the rules of the program by itself. This is done by feeding data to a model, along with a desired output for the specific application. The machine then must learn the rules from this data to recognize trends and similarities between the data and the output. In general, the most common learning techniques used for ML models are supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning.

Most of these learning techniques differ in the way that data is fed to the ML algorithm. In supervised learning, specific features or labels are supplied to the ML algorithm. This information contains the characteristics that describe a certain point within a dataset that is of interest to the application. Then, these features are correlated by the ML model with the output(s). Next, the ML model tries to predict the desired output. The main advantage of this technique is that there is control over what data the model uses, thus irrelevant data can be excluded if need be. This technique is used for classification problems like, for example, spam detection and image recognition, or regressions (e.g. business or weather forecasting). Unsupervised learning is a learning technique where the ML model is free to decide among the features it describes which is useful or not in order to obtain the desired result. This technique is often used in anomaly detection and for data clustering. In semi-supervised learning, just a subset of the total data is labelled. This often leads to flexible models since they can adjust themselves on the data, extracting important data themselves, rather than models that are supervised which need specifically extracted input variables to work. The last technique is reinforcement learning, this technique differs completely from the previously mentioned techniques. Here, a learning system called an ‘agent’ is used. The ‘agent’ rewards or penalizes based on the outcome, in other words, it rewards the tasks just if they were completed correctly. The program itself is called a policy, this technique is often used for learning AI to walk, learn a system how to balance an object or how to play a game.

These different types of learning have shown that Machine Learning can tackle a lot of repetitive and easy tasks. But all of this largely depends on how an ML model has been trained. Training is how an ML model mathematically approaches the task in question by attempting to fit itself as close as possible to the given data under certain accuracy criteria. In general, this is an iterative process that can be seen in figure 15. The process starts as follows: first, the model is given some data, next the model makes a prediction based on that data. This prediction is then compared to the desired output and based on that; a loss function is made. Subsequently, this loss function is further minimized until a certain accuracy criteria is meet. Often, the minimalization of the loss function leads to over-fitting the dataset, a situation that must be avoided. Over-fitting can be tracked by splitting the data into training, test and validation sets. The loss of both the training and testing set will be tracked during the learning process, and this can be a helpful tool to monitor over-fitting. The prevention of over-fitting is done in specific ways and depends on the type of model that is being used.

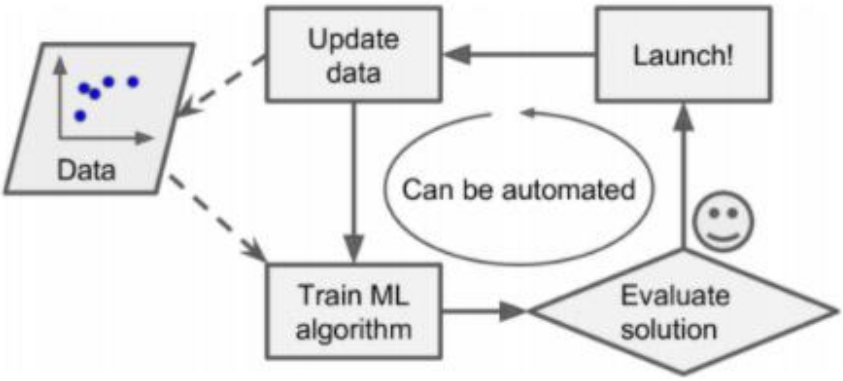


Figure 15 Simple training loop [22]

6.2 Neural Networks

6.2.1 Neurons

To achieve operational and reliable applications, it is important to choose the most adequate model based on the specific application. One of these possible ML models is the Artificial Neural Network (ANN)[23]. An ANN is a versatile model that is easily moldable for various applications and has already been used in multiple scientific fields, from tumour detection in medical imaging to revenue prediction models in economics. An ANN is broadly based upon the human brain, which is made up of neurons. A neuron is a cell that handles information within the brain and can be seen in figure 16. Information enters the cell with dendrites, which sends the information to the nucleus [23]. The nucleus is the central computing unit of the neuron, the nucleus then acts upon the information it has received from the dendrites and passes another signal to the axon and subsequently the axon terminals. These axon terminals are the output of the neuron and they send signals to the synapses which enables the connections between other neurons. In the human brain, there are on average 86 billion neurons which are all connected.

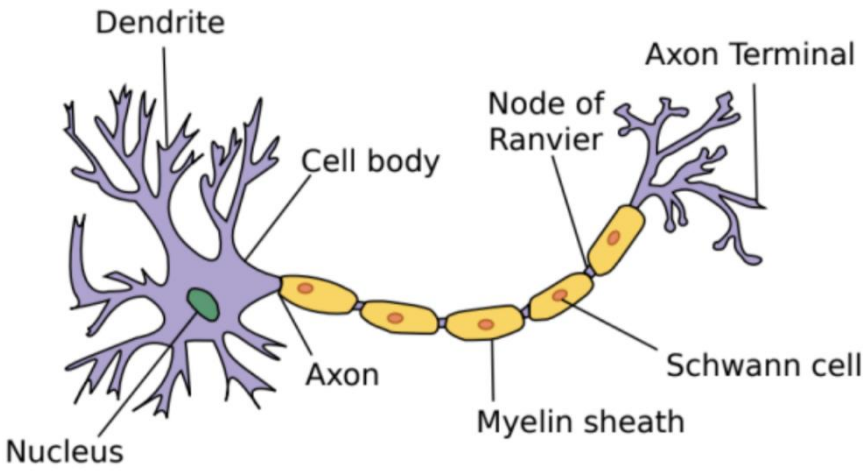


Figure 16 The biological neuron[23]

This is mimicked by the ANN, which also uses neurons to pass information through. Artificial neurons have had a multitude of forms, but the most advanced neuron is the Perceptron. A Perceptron adds up every input it receives, processes it and subsequently applies an activation function to it. These activation functions are used to prevent the inputs to blow up to infinity but also to act as the biological synapses, the activation function decides whether a neuron should produce an output or not. Figure 17 shows an artificial neuron.

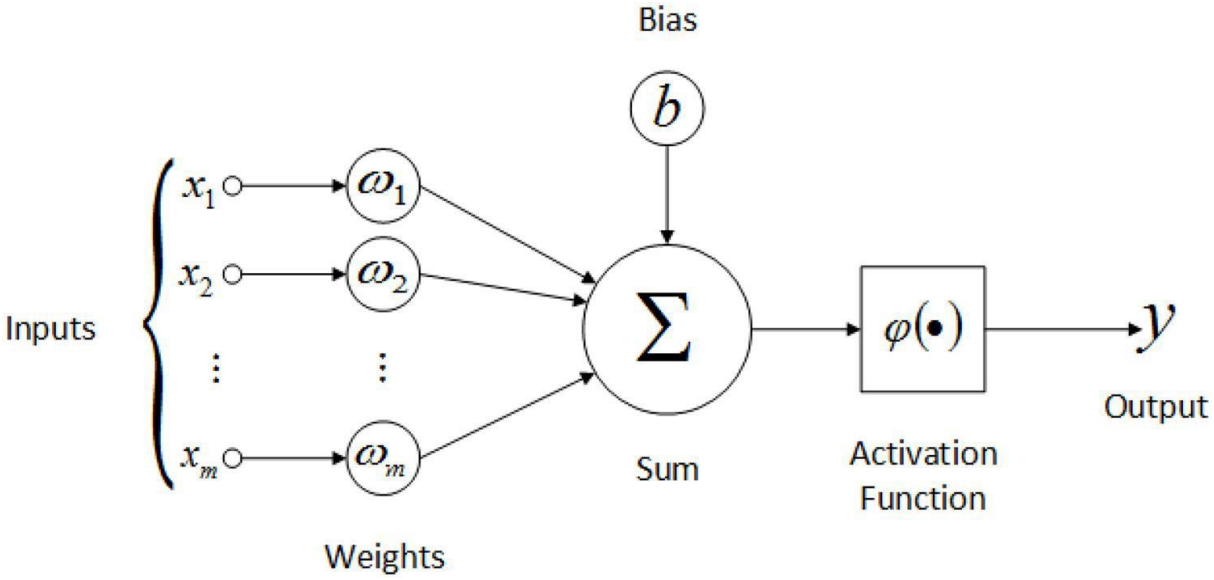


Figure 17 The artificial neuron [24]

The mathematical framework of the ANN relies heavily on linear algebra. Often the data that is being presented and calculated within these applications is given in the form of vectors for multi-input lists or even tensors for multichannel image data. So, to understand what is happening within a neuron, there is a need to understand the mathematics behind it. For a Perceptron with input vector x_i, weight vector w_i which addresses the importance of the input, output y, possible bias b and activation function Φ can be represented by:

$$\boldsymbol{y} = \Phi\left(\sum \boldsymbol{x}_i \boldsymbol{w}_i + b\right) \tag{9}$$

A perceptron on itself is already a tiny neural network and can learn simple linear tasks or patterns. Although this is interesting, most things in life are not linear, thus requiring more abstraction for real-life applications. This is achieved by mimicking biology, wherein the human brain the neurons are heavily interconnected to each other. These interconnections drastically open up more possibilities, by introducing more non-linearities. This is the reason why Multilayer Perceptrons (MLP) are being used. An MLP is an ANN with multiple layers of neurons between input and output, these layers in between the input and output are often referred to as hidden layers since they are in some way a black-box if the ANN is considered as some kind of logic gate. The connections between the different neurons contain weight vectors, these weight vectors introduce links and assign importance to certain input values. A representation of an MLP can be seen in figure 18.

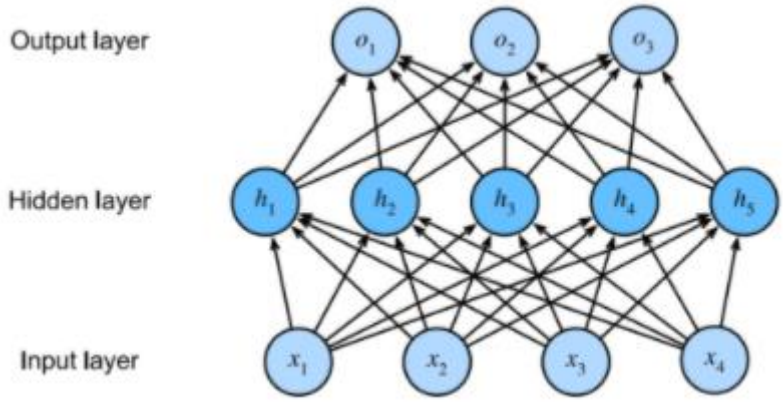


Figure 18 MLP with densely connected neurons consisting of input, output and hidden layer [23]

6.2.2 Activation functions

The most important part of this all is the activation function. Mathematical functions can be decomposed as the sum of several non-linear functions. This is what happens with deep learning, where every neuron has a certain activation function which in itself is a non-linearity being introduced at a certain point within the equation. Moreover, a deep neural network is a chain of neurons. Each hidden layer of the network will mathematically consist out of an input matrix **X**, output matrix **O**, hidden layer **H**, hidden layer weights $\mathbf{W^{(1)}}$, output layer weights $\mathbf{W^{(2)}}$, biases $\mathbf{b^{(1)}}$ and $\mathbf{b^{(2)}}$ and activation function ϕ [23].

$$\boldsymbol{H} = \phi\left(\boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}\right) \qquad (10)$$
$$\boldsymbol{O} = \boldsymbol{H}\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}$$

Additionally, this can be worked out for the output as:

$$\boldsymbol{O} = \left(\phi\left(\boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}\right)\right)\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)} = \phi\boldsymbol{X}\boldsymbol{W}^{(1)}\boldsymbol{W}^{(2)} + \phi\boldsymbol{b}^{(1)}\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)} \qquad (11)$$
$$\boldsymbol{O} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}$$

Where:

$$\boldsymbol{W} = \phi\boldsymbol{W}^{(1)}\boldsymbol{W}^{(2)}$$
$$\boldsymbol{b} = \phi\boldsymbol{b}^{(1)}\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}$$

Adding multiple layers together will result in the multiplication of more activation functions which will generate more intricate models. Activation functions transform inputs to output signals and decide whether or not the specific neuron should be used, with the added value of introducing a non-linearity. As previously mentioned, activation functions are of fundamental importance to deep learning and will have an impact on the efficiency and general outcome of the neural network. Another important feature for these activation functions is the fact that they have to be differentiable in parts. This will be important to train based on the gradient of the loss function. To further understand the concept of an activation function, a closer look will be given to the most popular activation function. The rectified linear unit (ReLu) works simple. When the input is below zero, the activation function will give 0 as an output, while as the input is larger than zero, it remains its input value as can be seen in figure 19. Mathematically this means:

$$ReLu(x) = \begin{cases} y = 0 \mid x < 0 \\ y = x \mid x > 0 \end{cases} \qquad (11)$$

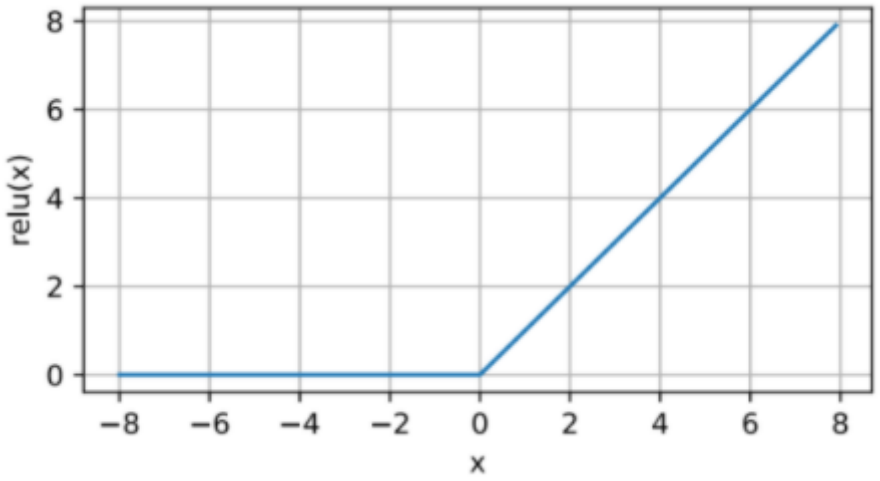


Figure 19 Visual representation of the ReLu activation function [23]

The derivative is either 0 if x is smaller than 0 and 1 for all the rest. This means that the derivatives of these activations will not cause the model to become unstable and will lead to convergence. Added to that is the added value of this ReLu that it is not computationally expensive. Activation functions get calculated after every layer, which means that this will be calculated millions of times within the training loop of a neural network. If this would be an intricate function, this would be a computational bottleneck.

Other popular activation functions are sigmoid, tanh and softmax. The sigmoid activation is analogue to the ReLu activation but this is a continuous function, with a difference that it places the output value between 0 and 1 for every value, while tanh does the same but produces output values between -1 and 1. Softmax is used in particular for classification applications and is a so-called last layer activation in classification problems. It produces an output between 0 and 1 with the value representing the probability of a certain class being classified. In conclusion, activation functions are an important step towards realizing well functioning neural networks, the choice of the right activation function will be an important part of choosing the right hyperparameters (i.e. the parameters used to set up the model).

6.2.3 Learning of neural networks

As previously mentioned, learning of any ML model is an iterative process in which the prediction of a model produces a loss function. This loss function is optimized to achieve the best possible predictions. The weights of the specific layers are the components that are trained and altered. The specific learning process for a neural network can be seen in figure 20:

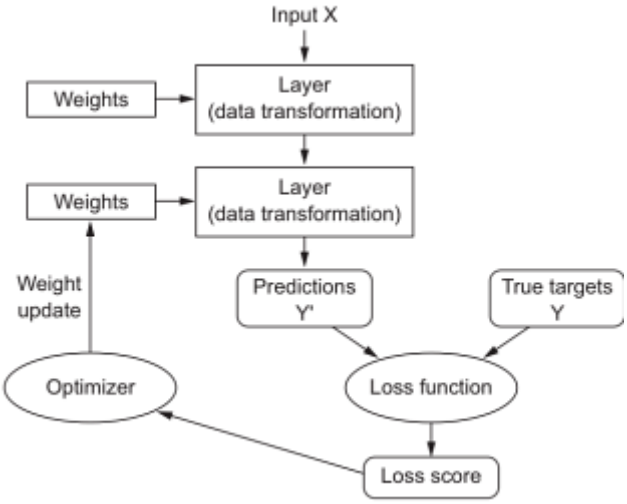


Figure 20 Learning process of a neural network[22]

One addition to the previously mentioned learning algorithm is the optimizer. The loss or cost function is the feedback signal used to achieve the best possible result and it depends on the metric (accuracy or mean average percentile error) that is being used. Loss functions are used by the optimizer to determine the way the layer weights should be altered. The optimizer determines other learning parameters, being the learning rate and also the responsible for the alteration of the weights during the training loop. In order to conceptualize these ideas, the following loss function in figure 21 with gradient descent optimizer can be used:

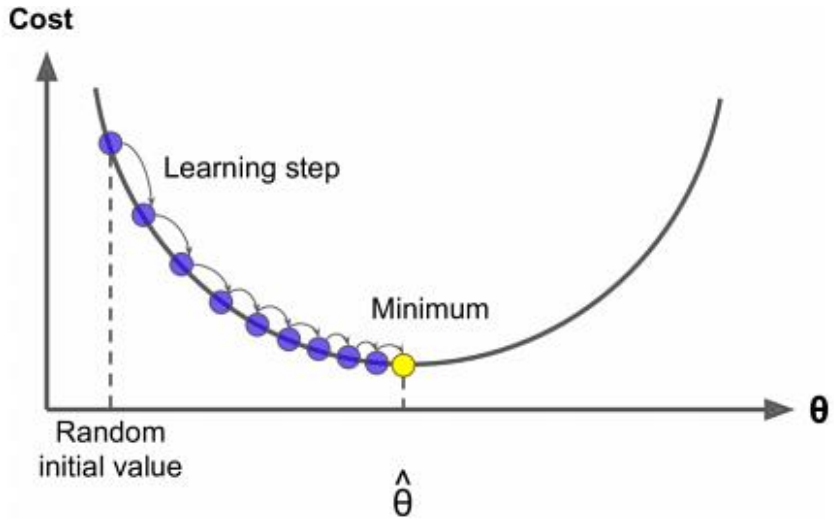


Figure 21 Representation of a loss function [22]

The optimizer aims to find the minimum of the cost function, in this case, this is point $\hat{\theta}$. For the model to find this specific point, the loss function is derived in the current point to achieve the gradient. Based on the gradient that the loss function has at the current point, the optimizer will adapt itself to follow this gradient. The jumps the optimizer makes along the loss function, are representative for the learning rate it uses. As can be seen, the lower the learning rate, the longer it will take to arrive at that point. It is possibility is to choose a very large learning rate, but this results in large variations. Modern optimizers use variable learning rates, like for example the Adam and Adagrad optimizers, which are often used in high level architectures [25], [26]. This means that they change their learning rate based on lower or higher gradients, allowing for more precise control over the progress on the loss function.

7 Method and materials

7.1 Data collection

An ANN is trained by presenting it with a subset of the data that is used that is similar to the data that is used in the application at hand. In that sense, it is important to gather not only a sufficient amount of data but also to cover a broad range of variations of source intensities and background radiation scenarios. Only with these circumstances, it is possible to get a robust well-trained ANN. The general data collection task starts with a Kromek SIGMA CsI(Tl) detector and a weak ^{137}Cs-source level. However, the detector needs to be calibrated before its use with a mixed source of ^{137}Cs/ ^{60}Co. To have a representative data set, spectra will be collected of 5 seconds real-time, with the first 500 spectra of the background, followed by 500 spectra of the ^{137}Cs source. This setup allows to collect data at multiple locations and therefore, get a dataset that is representative of background radiation levels that covers the full low to high range found in the area of interest. A schematic overview of the measurement setup can be seen in figure 22.

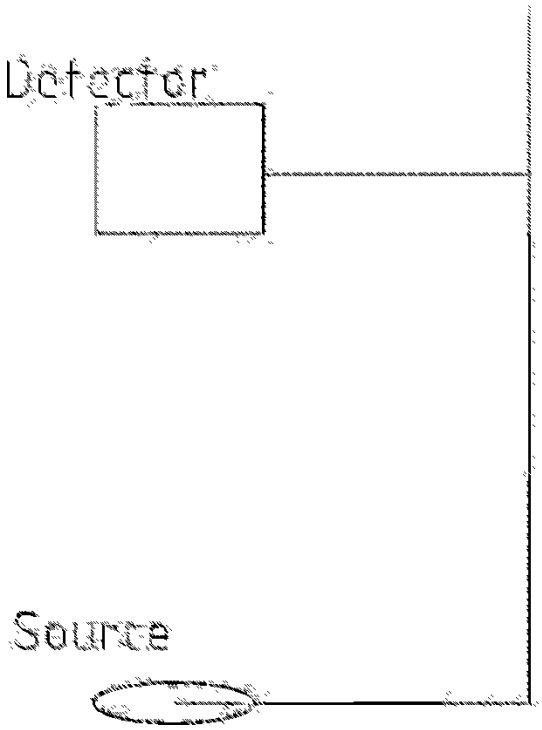


Figure 22 Schematic overview of the measurement setup

One of the greatest challenges for this application is differentiating between different types of background radiation and correctly interpreting what is in a certain spectrum [27]. Naturally Occurring Radioactive Materials (NORM) will present themselves in a multitude of forms (e.g. primordial elements in building materials, fertilizers, bananas), combined with cosmic radiation will trigger false alarms because of the variability they cause in gamma spectrometry data. This means that if a model were to be trained, it should contain data with enough variation in background radiation, to minimize the classifications of background radiation as an actual source. This is why the model will be trained with three different background variations (e.g. low, medium and high background radiation).

For the data to have as little background radiation as possible, shielding is necessary. This means a certain amount of a high Z-material is needed to attenuate the background radiation up to a desired level at which it is virtually non-existing. At SCK CEN there are facilities that provide this condition.

There, a bunker made of old steel from a shipwreck, which is normally used for whole-body counting can be used for this purpose. The levels of background radiation inside the bunker are very low, this circumstance allows to measure spectra that contain only the radiation of the source. The low background will present spectra that contain purely ^{137}Cs and could present more interesting features for the ANN in the region where the background is usually located. Moreover, this could aid the ANN in correctly classifying a spectrum if the activity of the source is low. Furthermore, high background radiation often occurs within buildings that are constructed with building materials such as clay and gypsum. These materials contain traces of primordial radionuclides (i.e. ^{238}U and ^{232}Th) [28]. These eventually decay to ^{226}Ra, ^{222}Rn and ^{220}Rn which can eventually build up within the building over time. In that case, it was interesting to find a building with these specific properties to have some higher levels of background radiation. Other general measurements are done in a desk/lab environment with normal levels of background radiation.

The next step is changing the source-detector distance to account for less active sources. Increasing the distance between a radioactive source and a detector will follow the inverse square law and thus present spectra which appear to have a less active source [29]. Increasing the source-detector distance will reduce the intensity of peaks and minimize the number of counts collected within the photopeak [30]. With less detailed spectra, the model will need to search for other features than the photopeak and making it more versatile. Subsequently, the result will be a model that is less influenced by high activity sources and capable to detect lower activities of ^{137}Cs. In practice, this is done by elevating the detector above the source, to minimize the amount of attenuation of tables and other elements within the lab environment, the only attenuation will be from the air. Two elevation measurements will be used to guarantee the presence of a noticeable difference in the conditions in the training dataset. These distances are 13 cm and 30 cm and are arbitrarily chosen. Both 1000 and 500 spectra respectively will be collected at every source-detector distance.

Furthermore, it is important to benchmark the ANN against a second subset of collected data which contains border cases with even less pronounced peaks and a minimal number of counts in the spectrum and specifically in the photopeak. These border cases are set in order to challenge the model in situations where normal methods struggle, due to the statistics related to these spectra. With this dataset, the ANN will be compared to a standard method (i.e. MultiSpect). Added to these spectra are some more labels, which are used for benchmarking and comparing both methods to each other. The first interesting type of benchmark is the identification accuracy on different measurement times, since radiation occurs with a set time interval (disintegrations/second), reducing the measurement time will result in a reduction of the total number of counts in the spectrum, creating spectra which have fewer data in them. To achieve this, 50 spectra were collected with a source and 50 without, with measurement times varying from one to five seconds. These spectra were collected in either high or average background situations. Subsequently, the same will be done for the source-detector distance, where the spectra will be measured at distances from 10-60 cm at both high and average background radiation. When the prediction of the model is inaccurate, the spectrum itself should be analysed in terms of the total number of counts and the number of counts in the photopeak to quantify the absolute boundary of when a spectrum cannot be classified.

Further, a test will be done where the number of spectra needed to accurately train a model will be quantified by changing the train-test split from 0.001 to 0.9. This means that the model will be trained in 65 spectra to 5850 spectra. Since the data on the drone is in another format due to data transmission restraints, the spectral data which is sent to a user consists out of 256 channels, which means that the spectral data needs to be reduced from 4096 to 256 channels. Moreover, the model will need to be adjusted and retrained for that specific data. The goal of this is to research if the model gains accuracy when concatenating the channels.

7.2 Data pre-processing

An important part of any Deep Learning project is the way the data is pre-processed. This is done to ensure that every file within the project has the structure and is fitted to pass into the specific model. In this case, the data pre-processing was done with Python. Python is a versatile and extensive programming language, interesting for data scientists and engineers. The spectral data is delivered in standardized .spe-files, which can be read in Python so this process can be automated. The important data which should be extracted from the .spe-files are the channels and their corresponding counts per channel, the life- and real-time of the measurements and the energy calibration. In specific, the .spe-files contain 4096 channels with counts and a calibration curve is set. This file system can be read out as a simple .txt file which is useful for automation purposes. Since the data itself can not have any specific metadata containing the setting in which the measurement was done, this information must be embedded in the file name. For this specific dataset, the information that has been embedded was the radioactive source (e.g. ^{137}Cs or background radiation), the source to detector distance, the source-detector angle and the measurement time. By generalizing this, this could be automated and done in another python script of which the code is added in Appendix A.

The spectral data itself can be opened in a simple .txt file, which contains the general information of the spectrum, the energy calibration, but also the measurement time (Real Time and Live Time), the name of the file, the detector type, resolution calibration and the LLD(Lowest Limit of Detection). To train the model, the most important information within this is the energy calibration and the spectral date e.g. the counts in every channel as well as the type of source(background or ^{137}Cs). For structuring the benchmarking dataset, the measurement time and name are mostly used for creating some extra metadata for the model. As one might expect, the entire dataset needs to be split into training and testing fractions which need to be randomized by using a pseudo-randomizer with a specific seed in Python. As a consequence of this pseudo-randomization, the outcome of the trained model is generally speaking the same, the biggest difference will lie within the different model weights determined by the model during the training cycle. The input data has utilised the entire shape of the spectrum, which is an array of 2 by 4096 channels with one array of 4096 being the counts and the other an energies from every channel, calculated with an energy calibration.

7.3 Model development

The development of the model has been done in Jupyter Notebook and pre-processing of the data is done in SPYDER, two IDE's which use the Python 3.0 language, the code for this model development and testing is seen in Appendix B. Python itself can be expanded by using third-part libraries to write stronger code. One of those libraries is TensorFlow, an open-source Machine Learning library developed by Google and used for scientific and consumer applications. Tensorflow offers easy to implement building blocks to build a model suitable to fit the needs of the problem. Other used libraries within the development of the model are Keras, which inherits from Tensorflow and SciKitLearn for splitting the data in randomized train-test splits.

As shown in previous research in this field, the possibility to use Artificial Neural Networks for radionuclide detection was shown by Yoshida et al. [31] in 2002 and 2019 by Kim et al.[32]. Yoshida et al. used the energy of the gamma-ray and the derivatives within the spectra to use as input data for the neural network since 4096 channels at the time were too big for neural networks to handle. Moreover, the neural network architecture was a normal Densely connected neural network. Research by Kim et al. has shown that a multi-isotope identification is possible by extracting a series of channels from a simulated plastic scintillator gamma spectrum in which the model present 9 predictions and a voting decision gets made to classify the model's output and accuracy. Their dataset

consisted out of 43 500 spectra, by extraction of these channels within the spectrum. Although this approach is interesting, it would be difficult to apply this in practical situations where a user gets real-time information back while doing measurements due to the fractioning that has to be done before the model can classify the spectrum. This pre-processing will present itself as a bottleneck in the application of this architecture in real-time. Another paper by Kamuda et al., suggests the use of neural networks as a means of an automated isotope identification method [33]. The total number of counts in each spectrum for training the neural network was between 10^3 and 10^5. These techniques both were trained on simulated data, created with Monte Carlo simulations.

In order to have a "plug and play" application for the proposed problem, the model has been constructed with the objective to extract features from a full 4096 channels of a ^{137}Cs spectrum by itself rather than provide additional information to help the model. An example of such additional information is a library containing the energies of the spectrum, this makes the architecture usable for including new radionuclides with only the spectral data. Another reason behind this is that the spectra that are used have generally less information in them as compared to a spectrum that has been collected for 10 minutes. The spectra of 5 seconds contain mostly the full energy peak of the ^{137}Cs source and some background, which means that classifying features to it might obstruct the model in achieving convergence. Adding to this is the fact that the model is trained on real-world data with different energy calibrations, which means that the model needs to be very flexible in searching the area of the spectrum where specific features or certain peaks are which indicate the presence of a source. Although the features need to be extracted by the model, the labels are pre-defined in the dataset and ready for the model to use. The labels consist of the source from which this spectrum is made (i.e. background or Cs-137) in a one-hot encoded manner. This means that the data is split into a one by two array and will be [1,0] for background radiation and the opposite for ^{137}Cs. Otherwise, the model would not be able to predict these labels correctly. The other label that is presented to the model is an array of 4096 channels containing 200 channels where the peak is located if there is a source present, so that it can use this to predict where a peak is located within the gamma spectrum[34].

Useful for this application is the use of convolutional layers. These have proven to be strong in extracting information out of data, in specific from images [35]. Convolutional layers work within the neural network to extract these interesting features from the data and create feature vectors, which are used to classify the input data. These convolutional layers work by applying convolutions to their input values with filters, these values will then be passed on to the next layer. The result of every convolution represents a feature map, these feature maps are used to make classifications or other predictions like in this instance a peak detection prediction. This method is visually represented in figure 23:

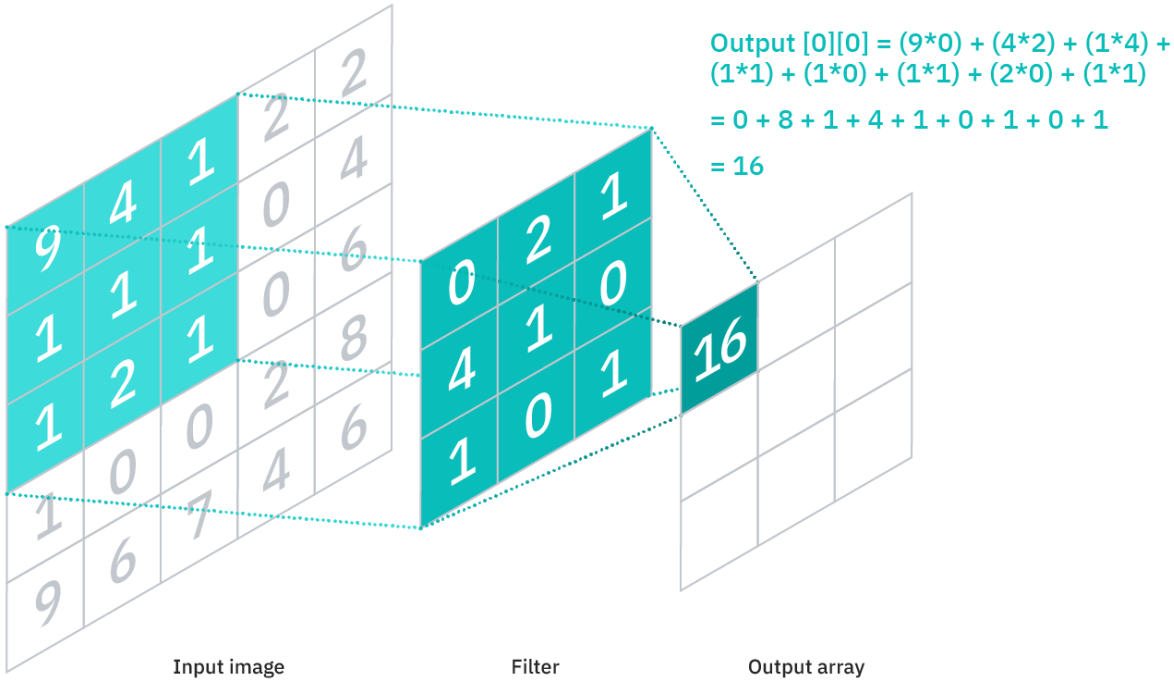


Figure 23 Convolution example of a 2D-array [36]

Although this representation is 2D, the model will need to do this on spectral data, and thus on a 1D array containing the normalized counts and the energy at which these counts are present. The input array is the entire spectrum that can be used to extract data from it.

Despite the brief introduction mentioned in the theoretical background, the selection of the neural network architecture is one of the most important choices to be done. For this project, a model architecture that consists of convolutional layers called DenseNet architecture [37] was selected due to the good performance reported in the presented paper. This architecture overcomes a problem suffered by previous network architectures, where due to the fewer parameters, the vanishing or exploding gradients are not present, this is often a problem in the backpropagation loop of traditional neural networks. This loss of gradient and input data causes the network to perform in a sub-optimal way, the manner in which this is fixed within the DenseNet architecture is by creating shorter paths from later existing layers to earlier layers. A general graph of the DenseNet architecture is shown in figure 24.

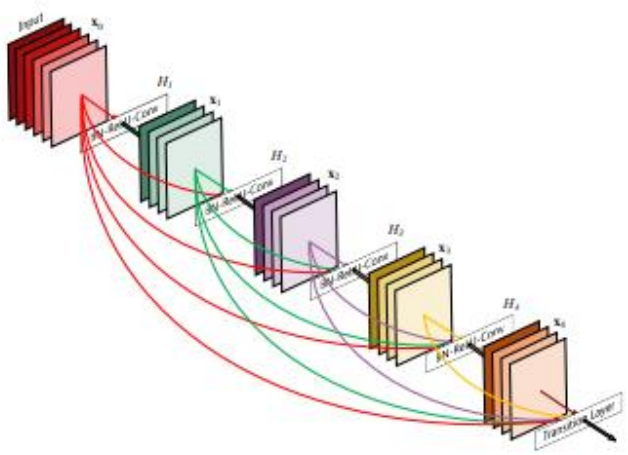


Figure 24 Representation of the DenseNet architecture [37]

Another great advantage of this architecture is that it can extract more features, the deeper the network is, while still taking into account the data and the weights from the previous layers and blocks. Each block contains the feature map of the previous layer and subsequently is able to extract higher-level features from the data. The way this connection to earlier layers is made is by concatenating the feature map of the previous blocks and bundle these as the input of the next layer. After every block, the network can grow at a certain rate. For example, when there are two layers of each 64 feature maps, then the second layer would consist out of 128 feature maps, since the feature maps of the first layer will already be defined and used by the model, the following 64 feature maps of the second layer will not search for features that the first layer has done. This way, the ANN can be made considerably deeper and narrower than the existing state of the art architectures, without using more parameters to tune. This leads to another advantage of this architecture, due to the fewer parameters this architecture is less prone to overfitting and it is more efficient in utilizing its parameters.

In deep learning, there is often made use of 'blocks' instead of layers. One block contains a sequence of several layer combinations. In the case of a DenseNet, the blocks consist out of a batch normalization layer, followed by 3 subsequent convolutional layers and a pooling layer. The batch normalization layer is used to stabilize the neural network by re-focussing and rescaling the layer's inputs by itself. The batch normalization is only used in later layers since the model will otherwise normalize the input information, leading to worse predictions. The pooling layer is used to extract relevant data from the feature maps and even offering to downsample the data if needed, resulting in better performance of the model. After this feature extraction, the model has enough features extracted to perform peak detection and classify the presence of a source. This is done with a simple dense network. The last layer is a dense layer with two neurons and a 'softmax' activation. This activation function will place predictions between 0 and 1, with the sum of each output neuron being 1. In other words, the model presents a probability of a certain label being present.

Addressing the efficiency of the model is done by comparing both labels with each other. This all is contained in the choice of a certain metric, a simple example of this is accuracy. This checks whether or not a prediction is equal to the label. If this is true for all cases the accuracy is 1, yet if it is false the end-user has no knowledge of the way his model is trained. To help this, the F1-score is often used to tackle this problem. It takes into account the number of false-positive and false-negative predicted data points that can be predicted by the model. This is also known as the balanced F-score and can be calculated with formula 12.

$$F1 - score = \frac{\text{True positive}}{\text{True positive} + \frac{1}{2}(\text{False positive} + \text{False negative})} \tag{12}$$

The optimizer used is Adam, which is derived from adaptive moment estimation [26]. What it does is iteratively change the learning rate depending on the gradient of the loss function. Adaptive learning rates usually start with big learning rates, resulting in big jumps in losses, but when the training cycle is longer, the learning rate will become smaller. To further extend this, loss functions, as they were presented in figure 21 are rarely the case, the more accurate representation is a 3D- field with various local minima and maxima. The goal of the Adam optimizer is to find the absolute minimum and thus ensuring that the highest efficiency is met for the prediction. The difference with Adam compared to Stochastic Gradient Descent is that Adam also uses momentum to converge faster.

Finally, the model should be optimized in order to achieve the highest accuracy on the data it is trained on, preferably without overfitting. Overfitting is prevented by using the Dropout layer in Keras, which like the name preludes, drops out irrelevant input vectors. In this model, this is done before the dense classification network and before the final layer activation in the model. To achieve the best epoch and the best hyperparameters within a neural network, hyperparameter optimization needs to be conducted. Hyperparameters can be seen as the bones of the neural network, which represent the number of neurons within each densely connected layer and the number of filters used for convolutional layers. These are not tunable for training the neural network and will be fixed throughout the training cycle. Optimizing these hyperparameters can be done in multiple ways, by either randomly choosing these values or by using Bayesian Optimization which uses the result from the previously predicted values to achieve the highest results on a black-box function. This hyperparameter optimization can be conducted within the Keras library.

8 Results and discussion

8.1 Dataset and model hyperparameters

In order to have a good dataset that can be used to train the model, the collected spectra should resemble the data of these drone measurements as closely as possible. To achieve a strong model, a dataset consisting out of 6500 spectra was created containing three background scenarios. To guarantee a complete range of spectra this data was collected in three different buildings, Bunker, HOB and ENE, which represent low, middle and high background scenarios respectively. The dataset also consists of 61.6% of spectra where a ^{137}Cs source was present and for 38.4% where no source was present. These two fractions within the dataset will be used to have two classifiable labels to quantify the model's accuracy. The training data is represented in figure 25:

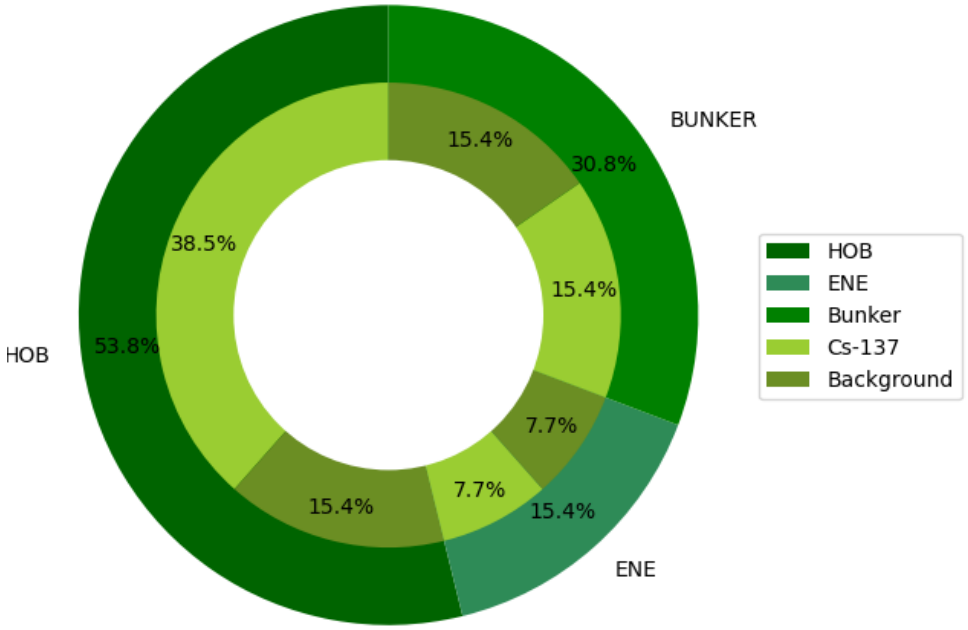


Figure 25 Pie-chart representing the training dataset

Although there is a unbalance in the amount of ^{137}Cs spectra with respect to background spectra, the idea behind this is that for radiological safety the model should be more capable of correctly quantifying ^{137}Cs and preventing the first responders or users to be unnecessarily exposed. The data within both spectra are not drastically different, but little nuances exist that increase the model versatility. The largest differences are appreciated in the different spectra of the background radiation in figure 26. These great differences are not what will be tested by the model, it is the spectra where there is little to no difference between background and ^{137}Cs spectra. To create a more challenging set of data, a less active source is needed. The inverse square law comes in handy, by increasing the distance, the activity will decrease by the square of the distance resulting in spectra that appear to have less pronounced peaks, appearing as if the source was less active. Within the dataset there are 1500 of these borderline spectra where there is an increased distance between the detector and the source (13 and 30 cm) moreover, these spectra are collected at HOB or medium background radiation level which also explains the imbalance.

Figure 26 a and b show spectra that are collected in the Bunker, which means that there should be virtually no external radiation present. This can be seen in figure 24a, where there only 5 peaks within the entire spectrum and this means that the spectrum from the ^{137}Cs source itself can be seen in figure 26b. Figure 26c and 26d represent the cases in which there is moderate background radiation present, this affects the spectra in having some more and higher peaks within the 0-500 keV region in the spectrum, but also in the region after the photopeak of 662 keV. Subsequently, figure 24e and 24f show the spectra from high radiation background where the region between 0 and 500 keV contains much more peaks compared to the spectrum collected in figure 24a. These spectra are well-defined instances of both scenarios where the photopeak is visible and will not present any problem for even the trained human eye. However, the training set does include instances where the photopeak is nearly invisible or appears to be part of the background radiation.

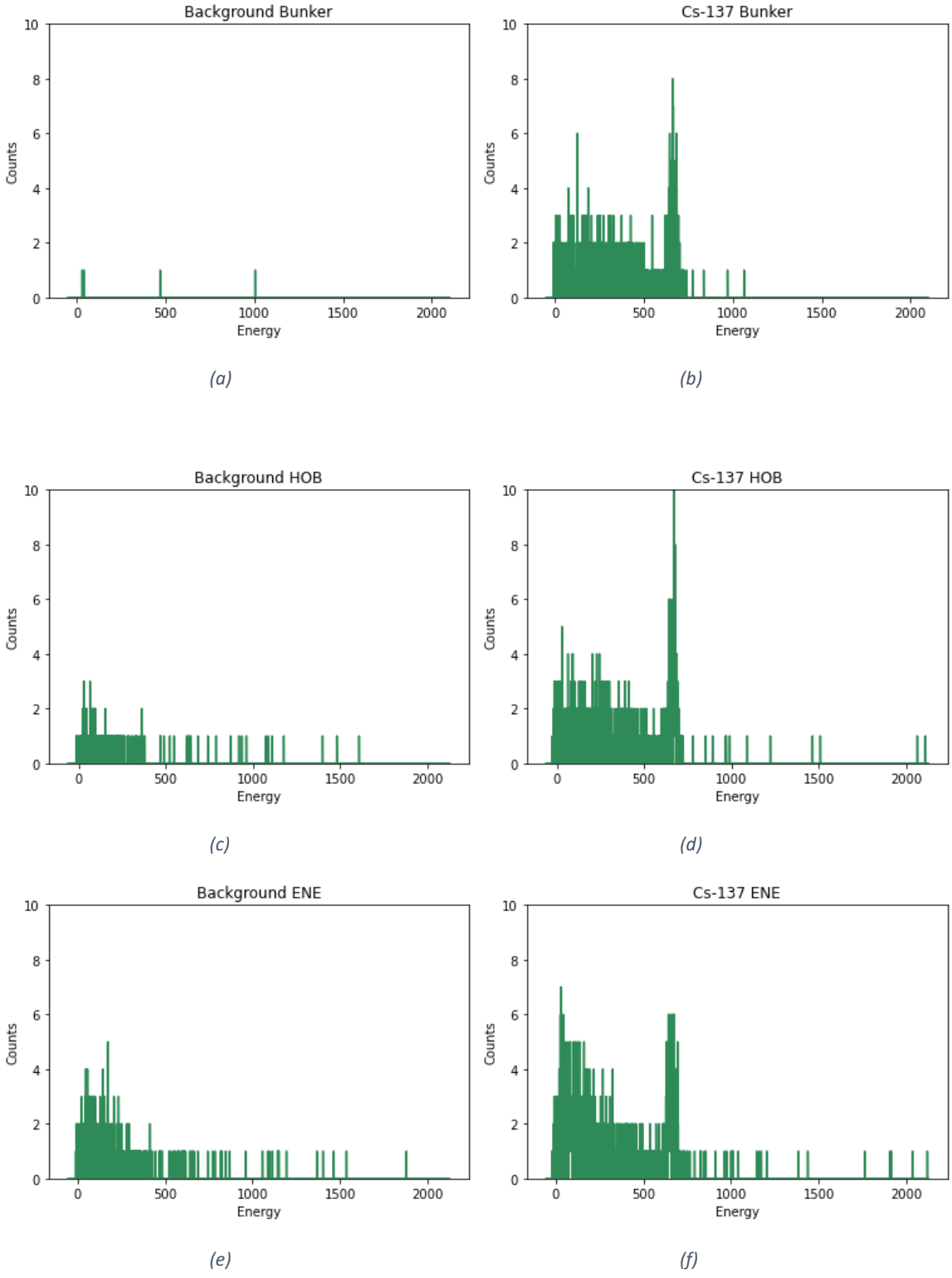


Figure 26 Examples of spectra within the training dataset

The training dataset will be split for 90% in training the model and a fraction of 10% for testing the model while training. Validating the model will be done using a validation dataset. Since the differences between the background and ^{137}Cs spectra are clear, the model should not have problems classifying these correctly. The goal here is to test the model under extreme conditions complete to identify its

limitations. Hence the reason why this dataset contains only HOB and ENE spectra. Figure 25 represents the fractions of validation data used in this dataset.

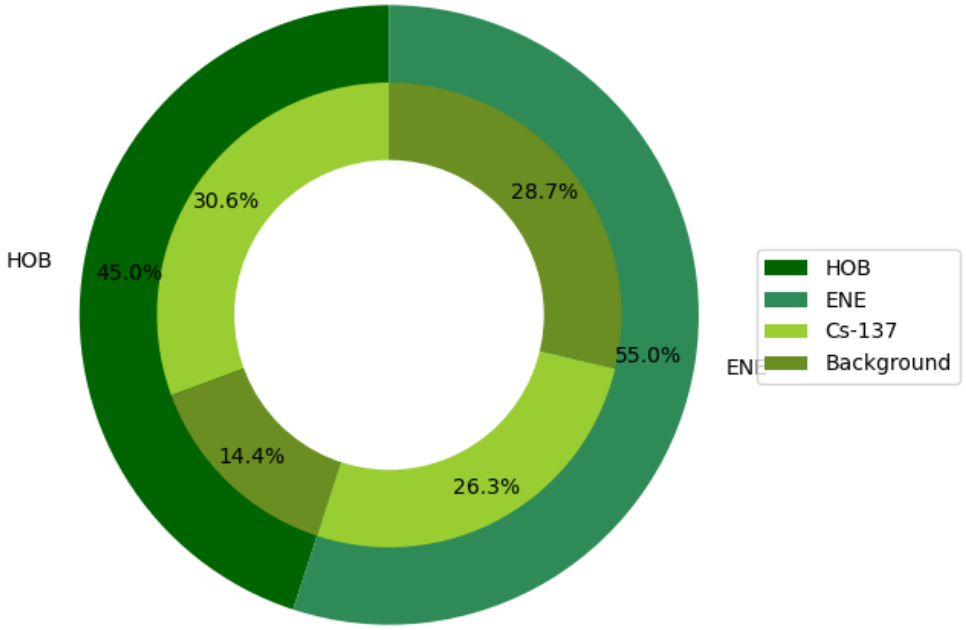


Figure 27 Pie-chart representation of the validation data

The hyperparameters of the model are the fractions of the data that have been obtained from the Bayesian Optimization algorithm within Keras-Tuner. These hyperparameters are optimized to have the greatest accuracy in the shortest amount of time on the testing dataset. Thus, the model consists out of 9 convolutional blocks which are built of four sequential layers of an input layer, which could be the spectrum for the first layer but could also be the weights of the previous layer(s) or even a batch normalization for normalizing the input of the previous layers. As can be seen in figure 26, the amount of filters in every convolutional layer is 35 and is optimized with Bayesian Optimization.

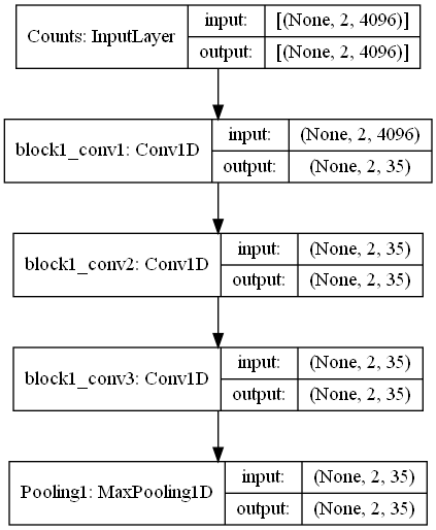


Figure 28 One convolutional block

Behind the feature extraction is the concatenating layer which collects all data from the convolutional blocks and merges this into one big input. This input gets pooled one last time and flattened to make a 1D feature vector. This flattened feature vector is used for the first output with 4096 neurons, which

gives an estimation of where the full energy peak of the ^{137}Cs spectrum is. These layers follow a densely connected neural network which is used for classifying both cases of either background radiation or ^{137}Cs. The hyperparameters of this classification model are subsequently 194, 190, 70, 192, 224, 164 and 122 neurons. The last layer has to have the same amount of layers as the number of labels. Overall, this part of the model can be seen in figure 27:

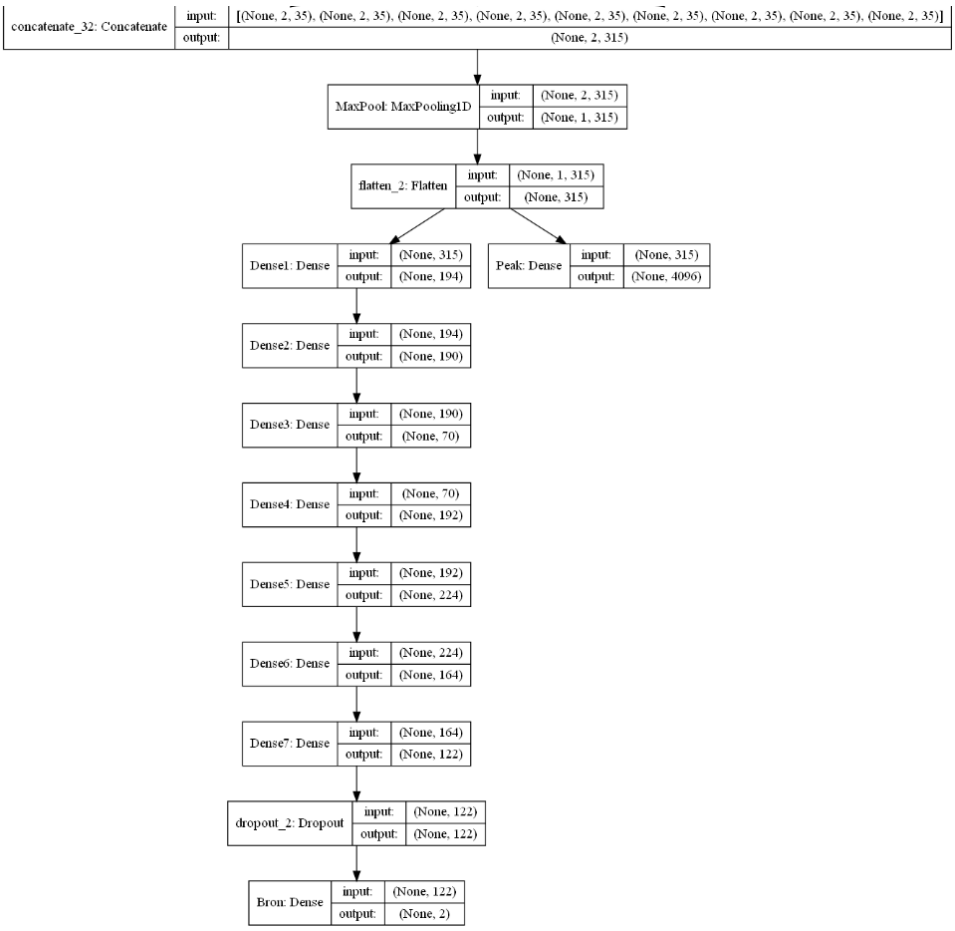


Figure 29 Outputs of the peak detection model and the classification model

8.2 Model training

While training the model, it will extract relevant features from the model using the convolutional layers. This results in features vectors that the model uses to classify the spectral data. The model consists out of 3 013 647 parameters within the model (weights and filters) and training the model takes about 16 epochs to train with a train-test split of 90% training data and 10% testing data, which means that 90% of the data is used to train the model and calculate the model's weights and filters, while the fraction of 10% will be used to test the trained model, to ensure that the model does not overfit and to check if the result is generalized. Due to the low amount of filters in the first convolutional layers, the overall calculation time is kept relatively low. Moreover, due to optimization, convergence is achieved after 2 epochs of training, as can be seen in figure 28, thus requiring a less overall number of epochs. This makes it possible to keep the time needed to fully train the model around 4 minutes.

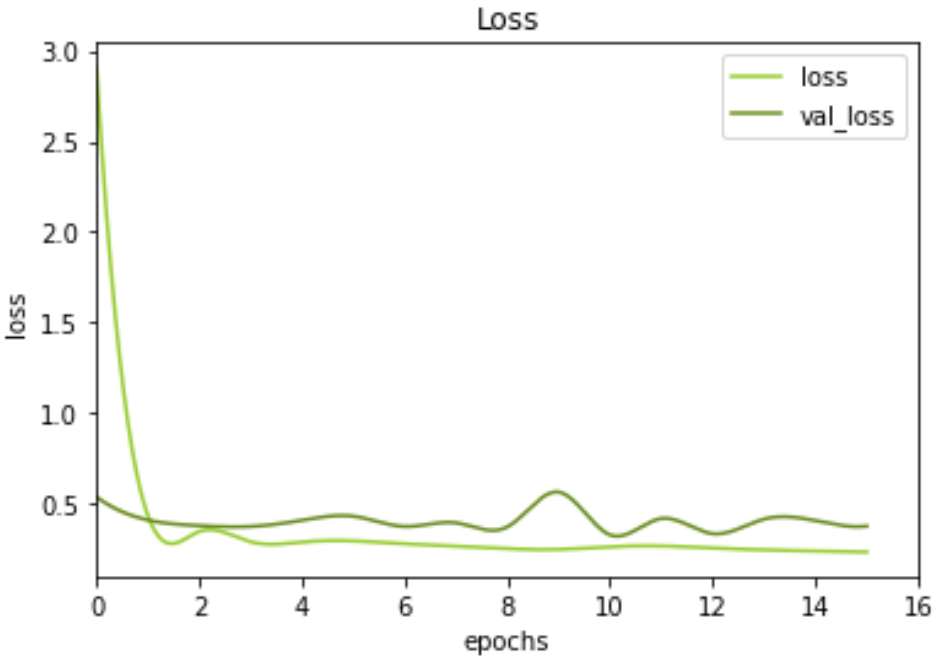
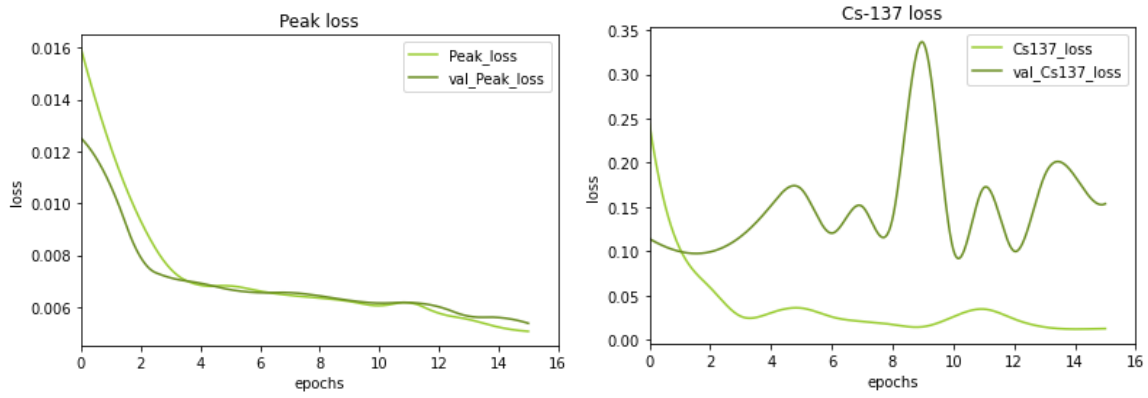


Figure 30 Overall model loss

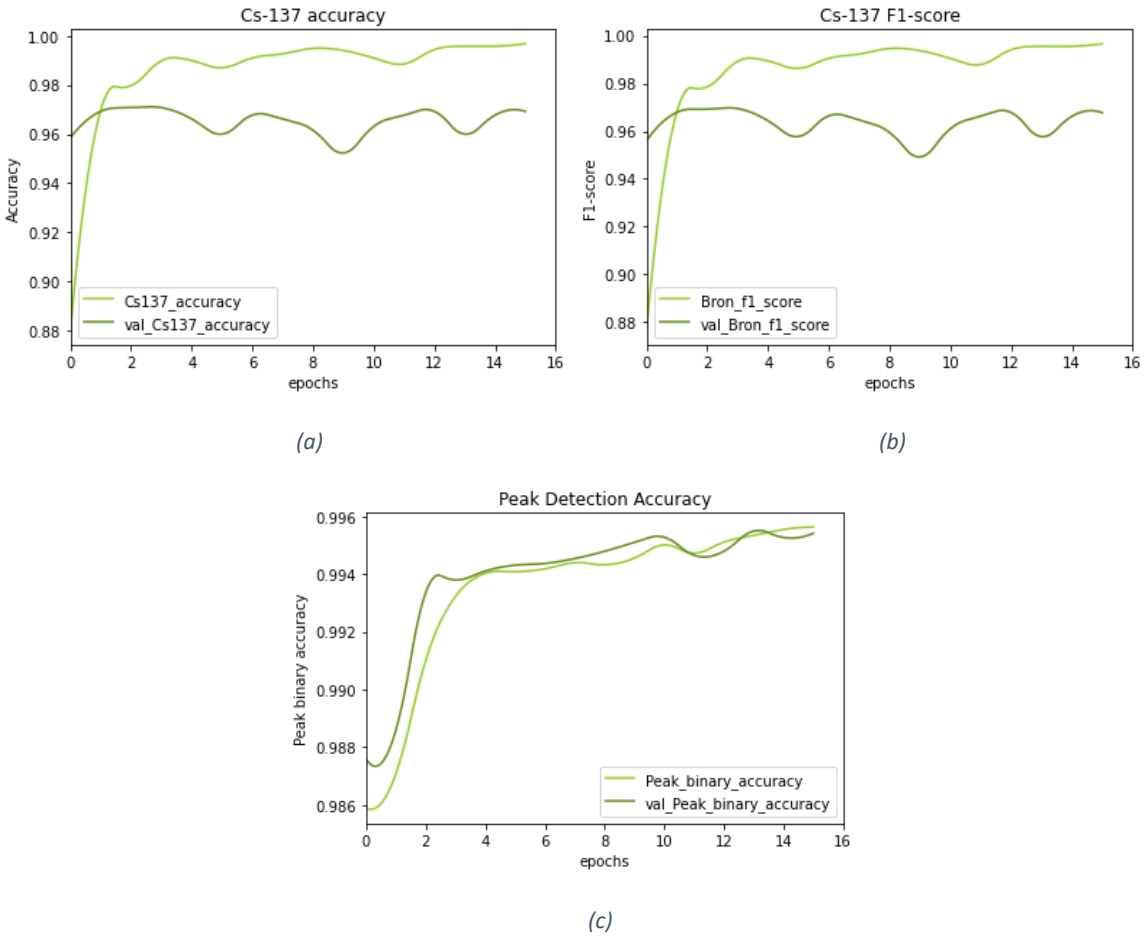
The overall model loss shows that the model reaches convergence at around 1 epoch, which is quick but it does not indicate real convergence. The model reaches appears to achieve a constant loss of 0.5 after one epoch with almost no differences, however, what should be taken into account is the fact that this seems low because it is compared to an initial loss of 3, shown at the beginning of the training cycle. Since the model initializes its parameters at the start with a random value, the initial losses will be very high. Knowing this, it is important to look at the losses of the individual outputs to examine the loss of the prediction. These loss functions can be seen in figure 30. The loss function of the Peak detection algorithm in figure 30a shows that the difference between the training and test set, “val Peak loss” in this figure, is the loss of the peak detection for the test-set, which converges to 0.006 with a continuous descending gradient, but shows that there is no overfitting because there is no difference in the overall trend of the validation peak loss. Moreover, it even shows that the model is capable of achieving even smaller losses. To further illustrate this, the ideal loss of a function is still zero, so the value presented by the model with 0.006 is very small in comparison with that. In the case of image 30b, the behaviour of the model is different compared to the peak detection loss. There is a significant difference between test and training loss, the loss values are much more erratic compared to the ‘clean’ loss function for the peak detection loss. Due to the increased amount of neurons in the classification network, slight differences in weights will have a great impact on the model’s ability to predict correctly. The training loss of the classifier shows no further loss, and the trend in the validation loss even shows a slight increase, which indicates that the model is overfitting.



(a) *(b)*

Figure 31 Loss functions of each output class

Figure 31 shows the differences in how the accuracy and F1 score differs as the training goes on. Unlike what the loss function might seem to suggest, the effects are not as great in the validation accuracy. To interpret these accuracies as good as possible, it is important to note that this is the mean of the total predictions. In terms of the training set, this means that 99% of all the 5850 spectra are classified correctly. Subsequently, the test accuracy shows that the model performs well on non-training data and stabilizes to around 96%. The accuracy and F1 are almost identical, which is a good indication that the model predictions correct. Adding to that, the validation accuracy shows no signs of major overfitting like the previous loss function has led to believe. According to the loss functions of the peak detection, the model was well trained with almost negligible losses, which also can be seen in the accuracy reaching 99,4%. The peak detection algorithm gives a prediction of 1 where the peak is found within the spectrum and 0 where there is no peak present within the spectrum. In addition, the loss is calculated by comparing the labelled data's ones and zeros and calculates then the loss accordingly. This accuracy of 99.4% translates to 24 out of the 4096 channels being misclassified.



(a) *(b)*

(c)

Figure 32 Classification accuracy and F1-score and Peak Detection Accuracy plot

When the model has trained all over again from scratch, all figures will be present consistently a similar behaviour. Also, the efficiencies of the model will not be significantly different. Hence it is important to highlight that in every training cycle, the model weights will be different and the efficiency of the model will also be different. This means that in order to accurately quantify the model's ability to predict on a dataset needs to be either trained and validated multiple times, or this feature should be baked into the model itself to statistically quantify the abilities of the model.

The model's predictions themselves are good, with 96% on the validation set. Even though this is a good result, it is more important to understand the model and find the reasons behind a success or a failure. Moreover, for which type of spectra does the model present bad predictions. As can be seen in figure 32 below, the model struggles in the boundary cases. In both the false-positive spectra, the model has predicted based on a higher amount of counts in the region around 661 keV, as these are not so pronounced in the false-negative predictions. These border cases are also cases in which a human is not able to classify these properly.

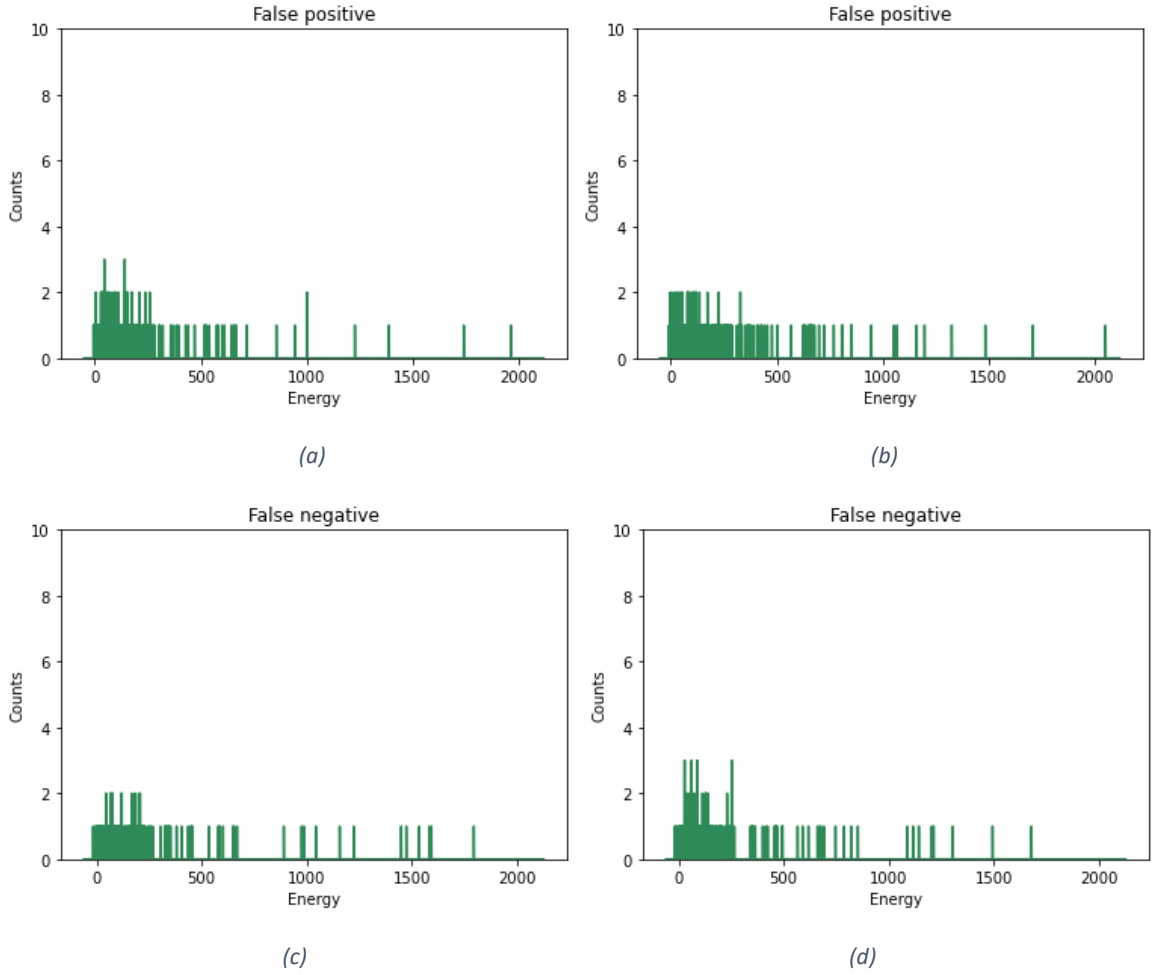


Figure 33 Examples of misclassified spectra in the testing dataset

Figure 33 presents some examples of the peak detection predictions, in specific the predictions of a well-defined peak(a), background in the bunker(b), a spectrum that contains normal background radiation (c) and a spectrum where there is a peak present but it is not as pronounced as in 34a (d). With this peak detection, the model has been trained to indicate where the photopeak of the input data is. What the model does good is setting the boundaries of the photopeak correctly, as seen in figure 34a, the peak prediction output of the model is perfectly able to point out where the photopeak is. Figure 34b shows a background spectrum taken within the bunker at SCK CEN with no source, and again, the model predicts this correctly by predicting zero for every channel of the spectrum.

This result is achieved by correlating the channels where the peak with a binary value, which shows the model that the value one is there for a peak and zeroes for all other channels. However, there is a problem in this peak detection when the peak is not as pronounced. This is visualized in figure 34c and 34d, where the model presents a value of 0.1 and 0.5 respectively for the peak predictions. What this means is that the model quantifies these parts of the spectrum as having a peak, but it also shows that it is not able to predict it with 100% certainty due to the low amount of counts within that region of interest.

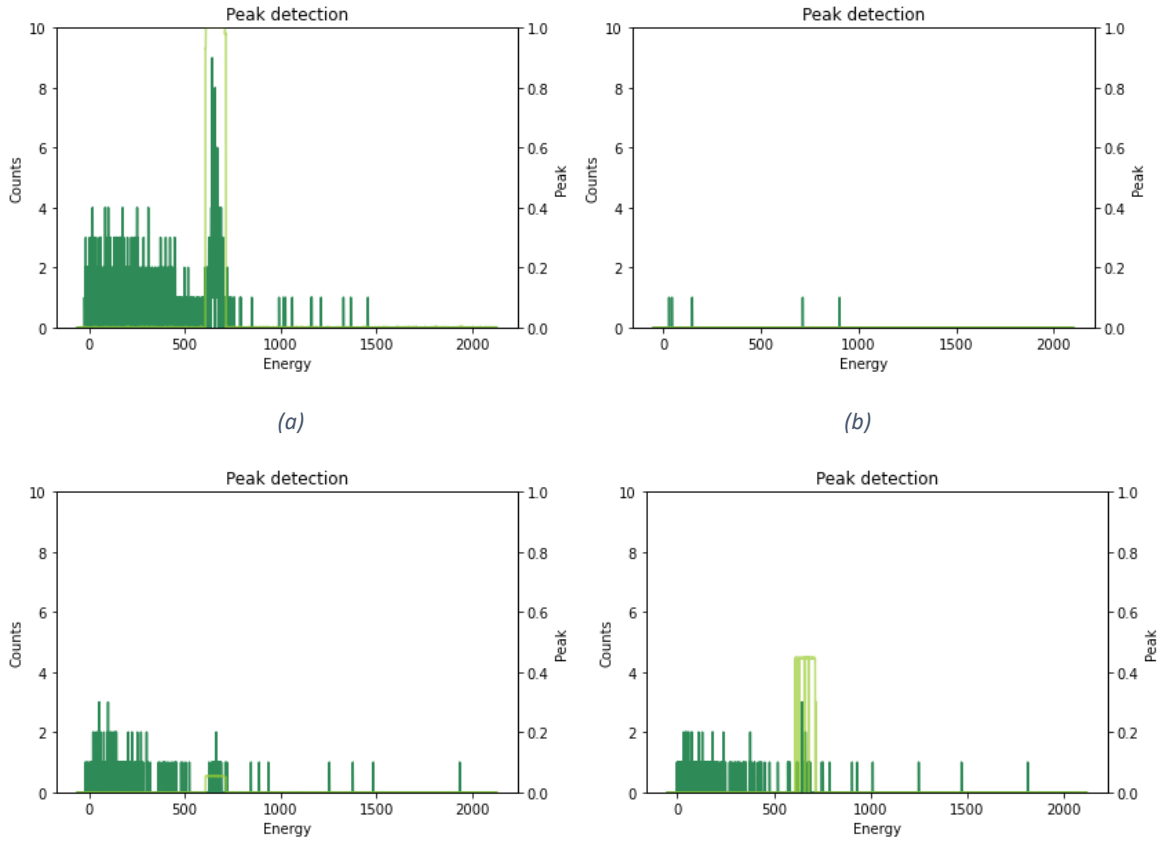
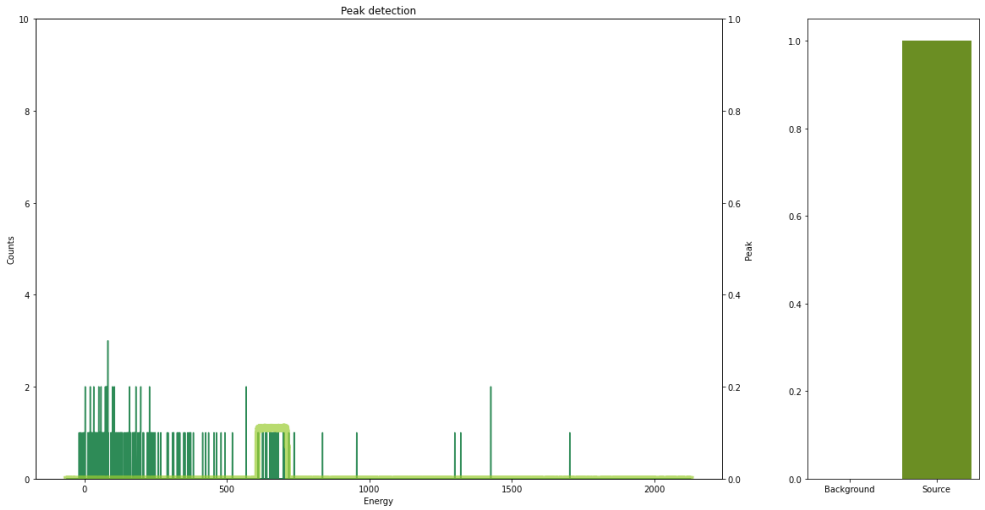
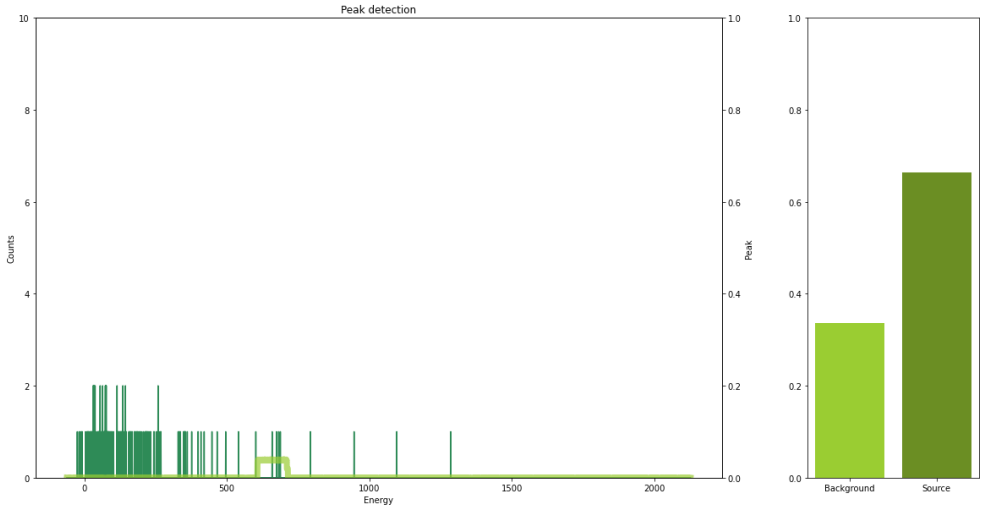


Figure 34 Peak detection examples

This peak detection prediction is done by using the features that the convolutional layers have extracted from the data. However, when relating these peak predictions with the predictions done by the classification network, there can be seen that this peak detection does present some differences. Figure 35a a shows two spectra that are similar in terms of features to the spectra which are shown in figures 34c and 34d, but added with these is the output of the classification network. This shows that the model predicts this as 100% being a source, rather than predicting a value between 100% and 50%, indicating some uncertainty surrounding this measurement. Furthermore, this indicates that when the peak prediction can make a prediction resulting in the detection of a peak, that the classification part of the model will indicate this as there is a presence of a ^{137}Cs source. Meanwhile, when the peak detection is between 0 and 0.1, the classification is more prudent in its prediction. As can be seen in figure 34b, the model predicts with approximately 0.7 that this spectrum contains ^{137}Cs.



(a)



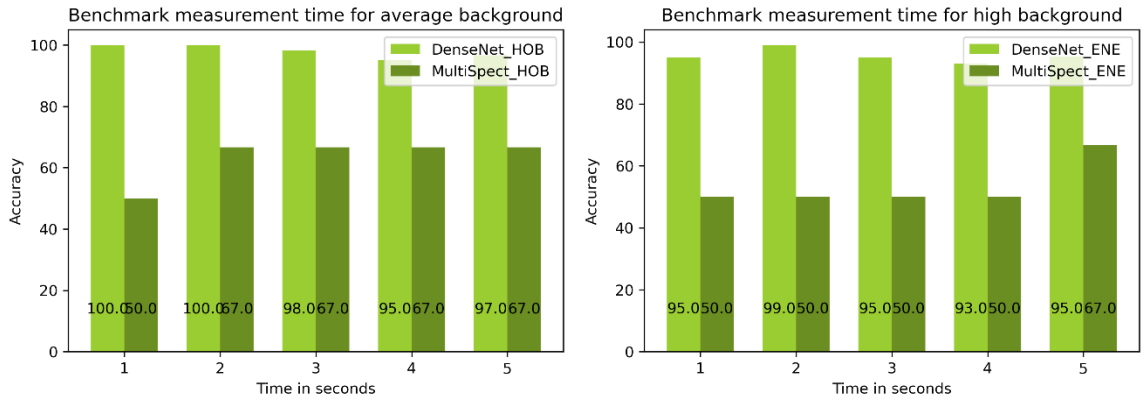
(b)

Figure 35 Closer look at peak predictions and the prediction of the classification output

8.3 Comparing the model to MultiSpect

The following results were found using the standard settings within MultiSpect. MultiSpect can be configured with a selection of radionuclides that the user wants to identify, in this case, the selected radionuclide was ^{137}Cs. Subsequently, Multispect presents colour coded results in the identification tests for users, the colours that are used are red, green and blue which correspond to a failed, succeeded or statistically invalid test. In the case of the benchmarking set, MultiSpect presented a blue colour for every spectrum which required classification. This means that MultiSpect quantified the spectrum to fail the critical limit test for the region in which it was set, i.e. the peak containing ^{137}Cs at approximately 662 keV in every spectrum. The reason behind this failed critical limit test is the limited counting statistics due to the small number of counts within the spectrum. For the following results, the assumption is made that a blue colour represents a failed classification and thus the spectrum contains background radiation. Added to this lower accuracy is the fact that MultiSpect did not have a way to automate the process of loading spectra from memory and this had to be done by hand.

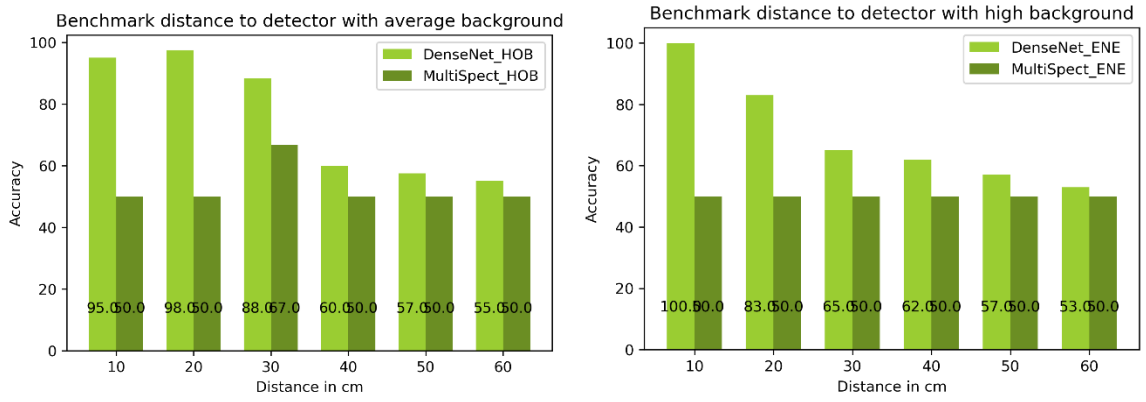
As can be seen in figure 36, the accuracies of the model are being plotted in relation to the measurement time. For both the average (36a) and high (36b) background scenarios the model outperforms MultiSpect in a very significant manner, in some cases close to double the accuracy achieved by MultiSpect. Comparing the results produced by the model show that the model does have some more problems producing accurate predictions for the higher background scenarios due to the presence of some higher peaks at certain channels. Overall these results are all above 93% which is a good indication of the model's capabilities.



(a) *(b)*

Figure 36 Comparison of model and MultiSpect on benchmarking set with differences in measurement times

Figure 37 shows the models prediction for the fraction of the dataset containing the source-detector differences for average (37a) and high (37b) background levels. MultiSpect again fails to make predictions, resulting in an accuracy of 50%. Generally, the model can predict better than 50% for all distances, which is better than guessing. An interesting trend that can be seen in the high background (35b) part is that the accuracy almost resembles the curve described by the inverse square law, whereas this is almost the case for the average background (37a). Furthermore, what can be seen is that the ANN model is not able to make accurate predictions past



(a) *(b)*

Figure 37 Comparison of model and MultiSpect on benchmarking set with differences in height measurements

Another test done was checking if changing the angle between the source and the detector influences the accuracy of the model. This can be seen in figure 38, where both the accuracies at every angle is plotted for the model and MultiSpect. MultiSpect was not able to predict any spectrum correctly, since

the dataset only consists out of ^{137}Cs spectra. So, when assuming that a blue colour result represents a misclassification by MultiSpect, the total accuracy on this dataset is 44%.

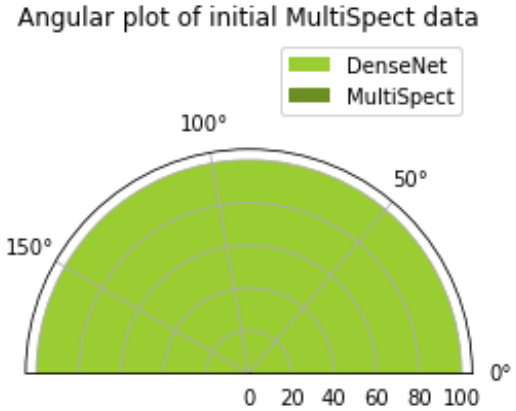
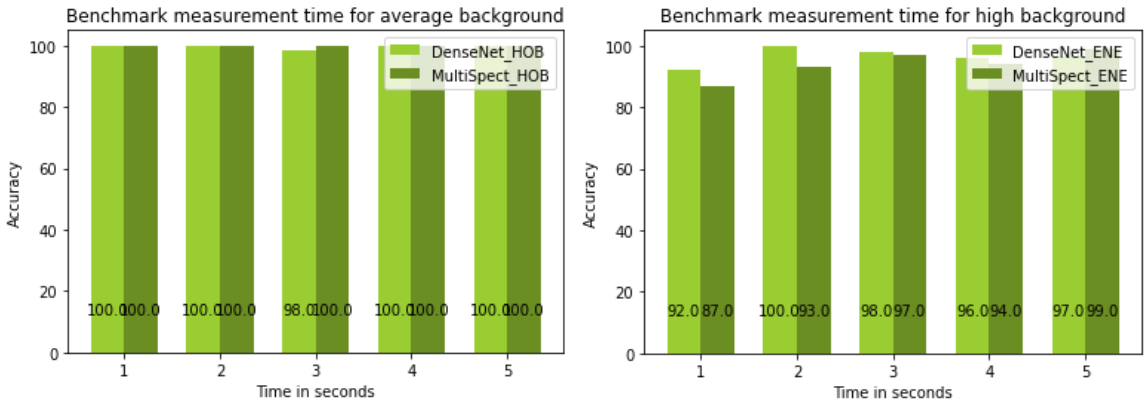


Figure 38 Angular plot for MultiSpect and the ANN model

Although these predictions show that the model is better than the standard settings of MultiSpect, MultiSpect's settings can be altered to achieve better results with the cost of having lower confidence. The standard settings for the confidence limit of MultiSpect is 95%, whereas for these tests it was limited to 68.27%. Moreover, all other settings were set to their respective minimum, for MultiSpect to have better results in general. This lower confidence will also predict the presence of other radionuclides rather than only predicting the outcome for ^{137}Cs. The settings that were in effect in these identification tests were:

- Lower background level = 10.10 keV
- Upper background level = 10.10 keV
- Peak ROI = 95.45%
- Confidence limit = 68.27%
- Min/max half-life = 0%
- Critical limit = 80.5%
- Minimum intensity = 5%
- Energy = 30 – 3000 keV
- Second portion of max = 0.10

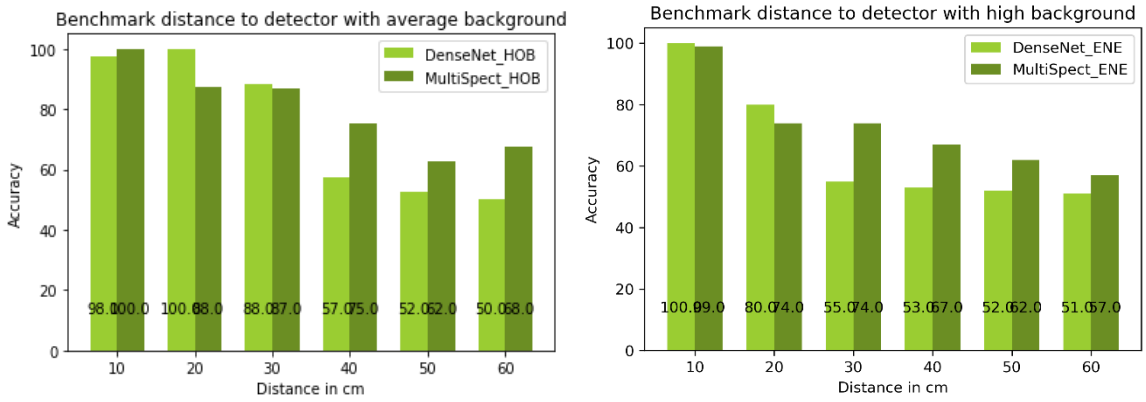
Compared to the previous results, MultiSpect holds itself a bit better compared to the standard settings. The distance between the detector and the source was 0 and 10 cm to create this part of the dataset. Overall, the model holds up quite well in comparison to MultiSpect as can be seen in figure 38, even outperforming it in the high background scenarios. This leads to belief that if the activity of the source is high enough, the model will have no problem making correct predictions with it.



(a) *(b)*

Figure 39 Benchmark results for different time predictions

In the distance predictions in figure 39, the model is not as accurate as the time predictions and is mostly outperformed, especially in distances greater than 30 cm and in the higher background radiation setting. The model is for average background radiations on par with MultiSpect until 30 cm, but from then it drops to guessing and only barely being better than a coin toss or a guess.



(a) *(b)*

Figure 40 Benchmark results for different distance predictions

Overall the accuracy and the F1-score of the model for this benchmarking dataset was 85.98%, while MultiSpect was able to achieve a higher overall score for these settings of 88.50 %. One of the possible reasons why MultiSpect outperformed the neural network could point to training data. Where the model struggles the hardest is past 30 cm of height, which is also the last height used within the training dataset. This means that either by the inverse square law the model or problems with the limited amount of data present within these spectra. Adding to this, the model did not have the same distance data present in the high background setting, resulting in worse predictions. The peak detection has an efficiency of 97%.

Since the model is trained and will try and reach convergence with different weights every time, this needs to be trained multiple times to make good conclusions. After training the model 50 times, the outcome can be seen in figure 40. The histogram shows the accuracies that the model has when training. Two models did not perform as good as the rest, but overall there can be said with a confidence of 95% that the accuracy of the model is [85.318 ± 0.116]%. Since the F1-score can give some results on how every different class performs, these are also shown. The model shows less than average predictions on the background of the dataset as compared to the accuracy, with a result of [84.852 ± 0.201]. Subsequently, the confidence interval of these predictions is wider. This contrasts

with the slightly better source scores, these being [85,711 ± 0.081] which have a narrower confidence interval. This difference between the background and source predictions can be taken back to the training data, where there is a slight imbalance between the amount of source and background spectra. This also shows that the model at this point is better at classifying ^{137}Cs spectra as compared to classifying the background.

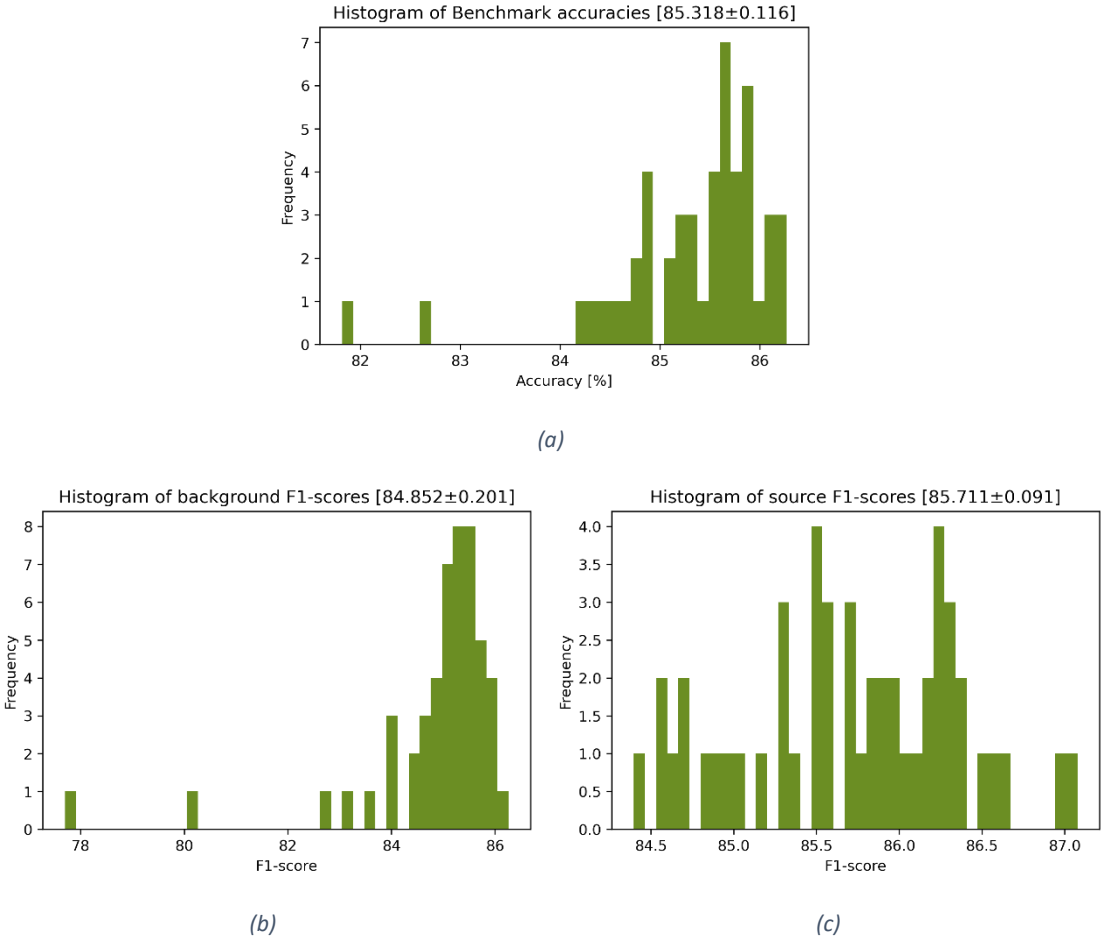


Figure 41 Histograms of 50 trained models

8.4 Tests to further understand the model

8.4.1 The minimal amount of data needed

With the idea of increasing the number of nuclides at which the model can make predictions accurately, the minimal amount of data that is needed for the model to have decent results needs to be quantified. This can be tested by altering the train-test spilt at which the model is trained and subsequently testing it with the benchmark set will indicate which moment the model will have enough data. Figure 41 shows a plot in which the model F1-score per prediction class is plotted in relation to the number of spectra that are being used. What is interesting to see is that there is a difference between the F1-score of the background classification and the ^{137}Cs classification. The model is more able to predict correctly when there is a ^{137}Cssource within a spectrum as compared to the background, in the case of the first point, the model has been trained with 6 spectra. This can be either by chance since there is an approximate 60/40 difference in the different spectra. As can be seen, is the model able to make accurate predictions from 1000 spectra, this comes to around 84% prediction efficiency from then on with only marginal gain going forward. From this, the conclusion can be made that the

model is needed at least a mixed batch of 1000 spectra or approximately 500 spectra of each labelled source, with enough variation in background data and source activities.

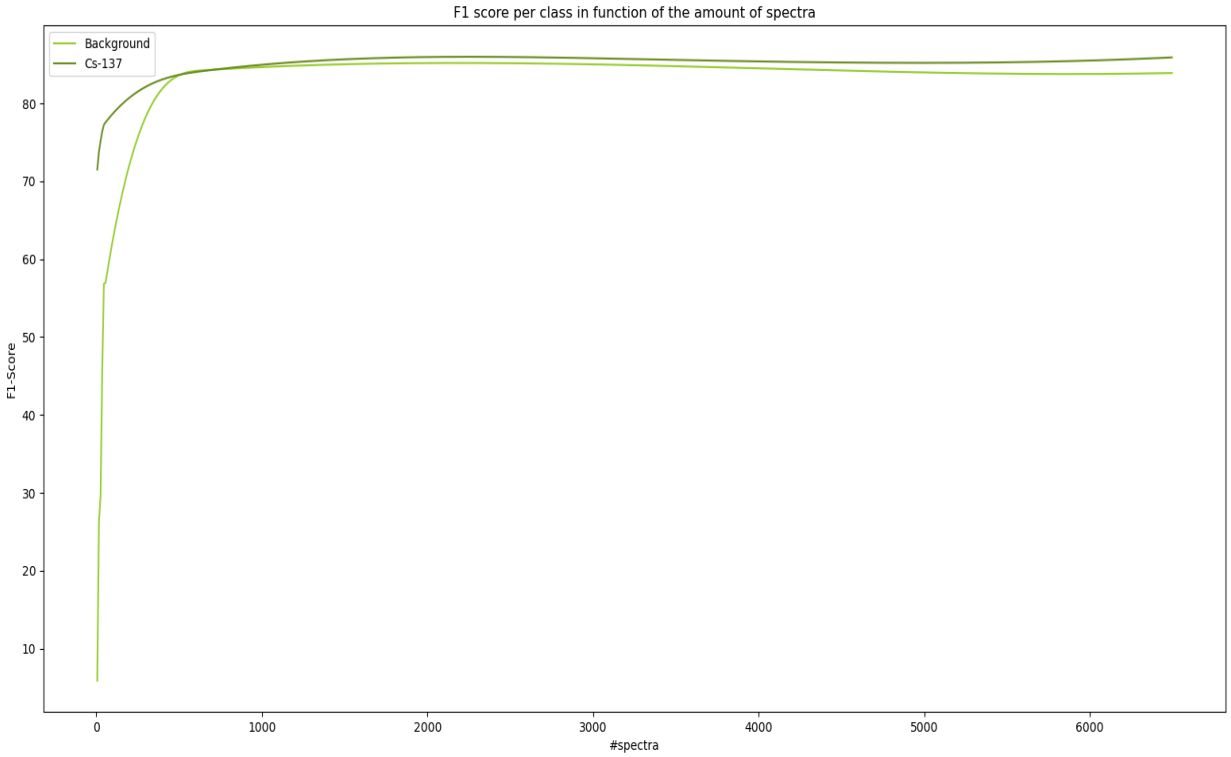
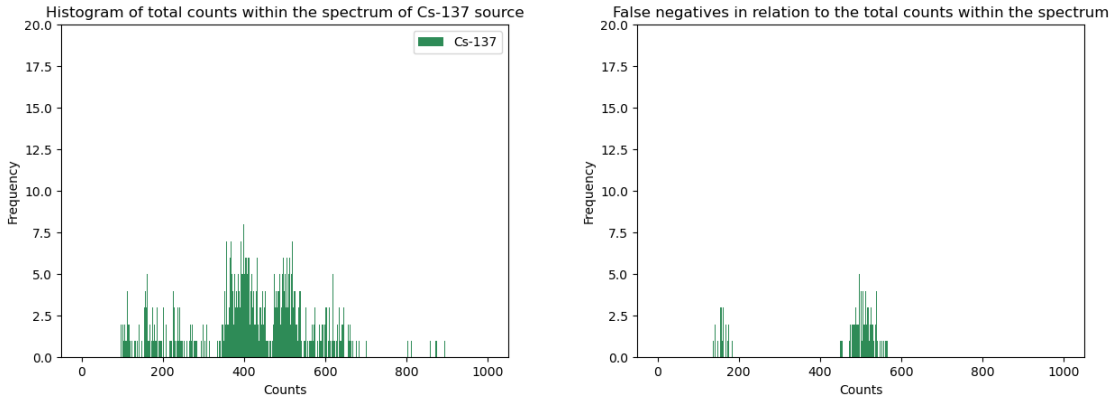


Figure 42 Amount of data needed to train the model fully

8.4.2 Relation total counts and prediction efficiency

In the MultiSpect test, it became clear that the model is not able to make accurate predictions if the distance becomes greater than 30 cm. So in that sense, the model will have some problems when there is only a certain amount of counts present in a gamma spectrum where there is a certainty that there is a source present. By visualizing this with the amount of false negatives results in the classification of the spectra at those counts, there can be seen whether or not there is indeed a minimal amount of activity needed so that the model can identify radionuclides correctly. Since the dataset only consists out of ^{137}Csspectra, the false positives will not deliver any relevant information since every positive test is a ^{137}Csclassification.

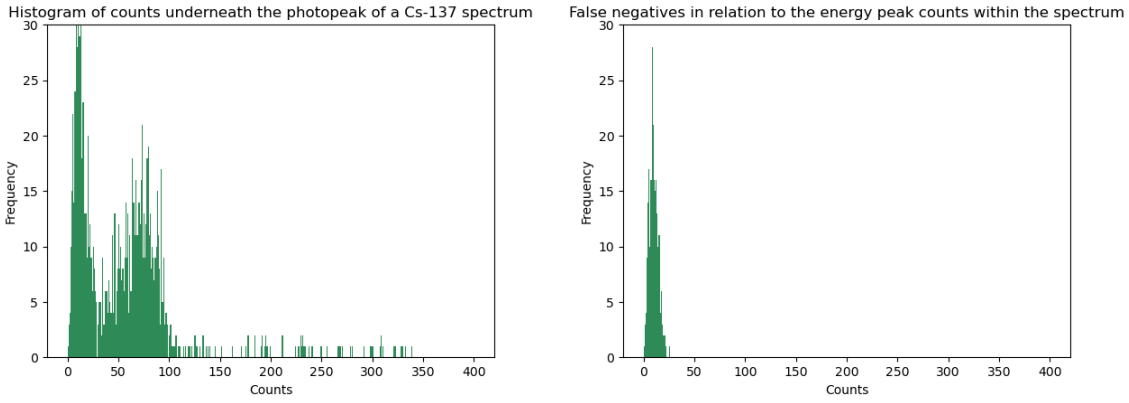
As can be seen in figure 42, the total amount of the counts exists out of two very distinct peaks around 500 counts and that there is also a smaller peak of around 180 counts. If there was a limited or minimal amount of counts needed for correct classification, the expectation would be that the model will perform less good on the spectra with a lower amount of counts and will perform better on those with more counts present. The histogram containing the false negatives shows that there are even more spectra misclassified around 500 counts than around 200 counts.



(a) *(b)*

Figure 43 Histograms showing the number of counts in every ^{137}Csspectrum of the dataset

From the previous figure, the conclusion would be that there is no relation between the number of counts within the spectrum and the misclassification by the model. And thus there is no thus the quantification of a minimal amount of counts would be unachievable. Yet, in order to classify a spectrum correctly, the model requires a certain full energy peak within the spectrum. This can be seen in figure 43 where the counts underneath the photopeak are represented, along with the number of counts that are classified as a false negative by the model. What can be seen here is that there is a certain minimal amount of counts that must be present within the photopeak. From the moment that there are more than 25 counts within the full energy peak, the model can classify it correctly.



(a) *(b)*

Figure 44 Histograms of photopeak counts

This is a representation of one trained model, which shows that this particular model can make accurate predictions if there is a source present and when there are more than 25 counts in the photopeak. But as previously mentioned, the model weights differ from one training cycle to another. By taking the maximum value at which the model predicts a false negative for 50 trained models, the actual mean of this value can be derived. As can be seen in Figure 43, the mean of this is [34 ± 3] counts for a confidence interval of 95%.

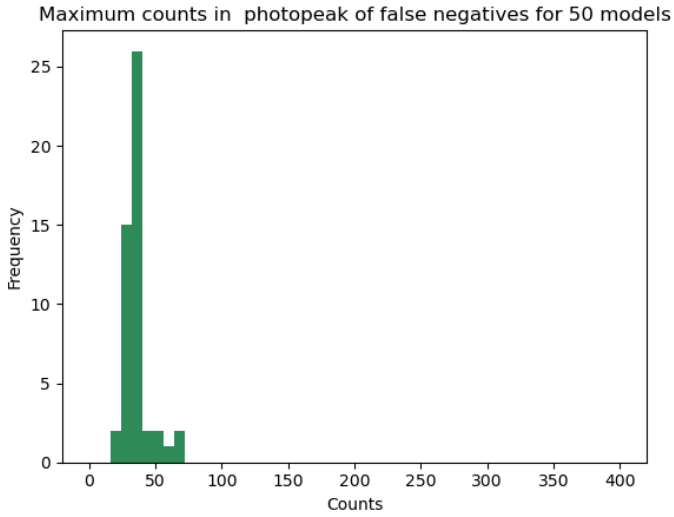
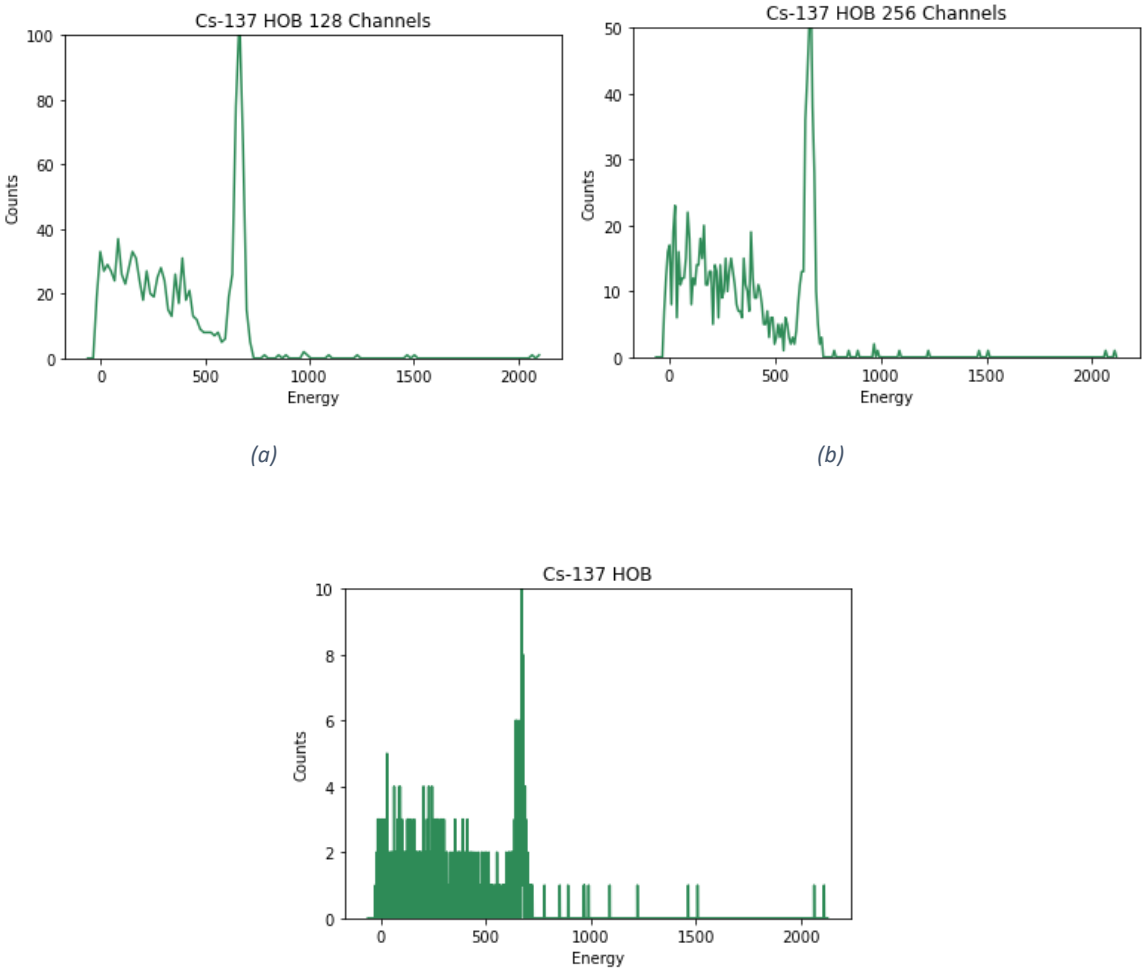


Figure 45 Maximum counts in photopeak where the model presents false negative predictions for 50 different models

8.4.3 Effect of reducing channels

In the events of contamination in an urban environment, such as in the events seen in the Goiânia accident, the first responders or in this case the drone pilots will need direct feedback of what data the drone sees. To keep these data transfers as fast as possible, the amount of channels is reduced from 4096 to 256 for storage on the drone and 128 for sending from the drone to the base station. This means that the current model will not be able to make predictions for these spectra, thus the model should be modified and the spectra must be altered. Luckily, this is a problem within the Tensorflow interface.

The spectral information is reduced significantly, as can be seen In figure 44, where the same spectrum is plotted with either 128,256 and 4096 channels. For one, the peak around 661 keV and the background radiation is 10 times larger in the 128 Channel data, compared to the 4096 channels. This is only 5 times for the spectrum with 256 channels. Visually, the spectra with fewer channels seem to have a more pronounced peak and a higher background, which only could be a benefit for the model to classify this. Moreover, the reduced amount of channels might make it possible to train the model even quicker and make predictions instantly.



(a) *(b)*

(c)

Figure 46 Spectral data with reduced channels

As can be seen in Table 3, reducing the number of channels within the spectra does indeed increase the accuracy of the model, thus having even better results when the implementation is used on a drone. The time per epoch is also greatly reduced, meaning that the speed at which the model does its predictions is also greatly improved.

Table 3 Results of reducing the number of channels

Channels	4096	256	128
Time per epoch	15 seconds	5 seconds	8 seconds
Accuracy	85.32%	86.60%	88,90%
F1-score	85.32%	86.60%	88.89%
Peak detection	97%	96.41%	97.38%

Comparing neural network model objectively can only be done by utilizing the same data in either train, test and validation steps. Moreover, the models presented in the literature utilized a dataset that consisted out of multiple radionuclides which were all simulated with MCNP6 code for Germanium and plastic detectors, rather than two actual measured cases with CsI(Tl) detector [31], [32], [33], [38]. Moreover, these models were constructed with different hard-and software, thus comparing these is difficult. However, comparing the model to the literature shows that this model performs less on this

specific dataset as compared to these other models. This is due to the intended border cases used to quantify the model's abilities and research where its limits are. However, in training the model there are similarities to be drawn. First, in terms of the number of epochs needed to train the model, this closely resembles the results achieved by Sahiner et al. [38], where the model achieved convergence in 15 epochs and outperforms the model presented by Kamuda et al. which required more than 500 iterations to achieve convergence, yet an important note that has to be made is the fact that this model could classify mixtures of 25 different radionuclides with multiple photopeaks, hence the reason why the model needed a higher number of iterations. The model of Kim et al. required 1600 epochs to achieve this. In terms of time measurements, the model presented by Kim et al. dropped from 100% to 93% in accuracy when the spectral acquisition time went below 4 seconds, while this model only suffered these losses when confronted with higher background radiation. Furthermore, the minimum number of counts needed for identification was more than 500 counts per radioisotope where the FP and FN were less than 5%, while in this thesis the minimum number of counts needed was [34 ± 3] counts. This again can be drawn to the type of data where the model is trained with, in specific the lower counting times and overall counts in every spectrum.

9 Conclusion and outlook

The objective set for this research was to investigate if a neural network could be trained to be implemented in the measurements of spectra that are collected during UAV flights. The characteristics of these spectra are that they have a counting time of maximally 5 seconds and contain mainly variations in background radiation. To achieve this the implemented model's architecture was a DenseNet. Because this could extract features itself by the use of convolutions. The training dataset consisted out of 6500 spectra of which 90% was used in a train-test split. The validation or benchmarking set consisted out of two main sets i.e. height and time measurements, which were used to compare the model to a standard method, the MultiSpect method developed by Kromek. The model reached an efficiency of 85.32±0.12% on the validation set, which was almost double the accuracy MultiSpect was able to achieve on the same dataset with standard settings. When altering MultiSpect's settings to allow for lower uncertainties, MultiSpect outperformed the ANN-model with a 3% difference. The greatest differences between the two methods were observed during the distance test, where the efficiency of the neural network started to decline as the distance increased beyond 30 cm with high background radiation conditions, this was since the distance was excluded from the training dataset. However, having to include more data within the training set taken at further distances, the model would not only recognize these spectra but the number of false negatives would be lower. Thereafter, the number of spectra that are needed to train the model accurately are 500 spectra for each label, with a wide variety of backgrounds and activities. Subsequently, the minimal number of counts that must be present within a photopeak to have a correct prediction must be greater than [34 ± 3] counts for a spectrum to be classified correctly. Finally, reducing the number of channels within the spectrum to be able to use the model in the field, improving the accuracy of the model by 1.28% for 4096/256 conversion and 1.71 % for 4096/128 conversion. Moreover, it sped up the training cycle from 15 to 5 seconds per epoch.

In conclusion, neural networks are a feasible and reliable alternative to the standard method, if the model is trained with enough variation in either activity and background radiation. The dataset used in this thesis has shown to lack some essential features which caused it to perform well on the training dataset and the time measurements but made it fail on the distance test of the high background scenario.

Future work should start with an extension of the current dataset to include more radionuclides and more background scenarios to have a better and more robust model so it can be implemented fully on the drone. Furthermore, the architecture could be extended to have more features with for example activity estimation, adding uncertainty for each prediction by implementing a Probabilistic Neural Network and providing visual feedback for first responders. Extending the existing architecture with more radionuclides will present more efforts in calculating the possible predictions that are made by the ANN. The model is currently optimized for the dataset of ^{137}Cs and background radiation. Extending this would mean that the model should present different labels and thus some hyperparameters may be optimized further for optimal use. Subsequently, the influence of temperature, weather conditions and other external factors have not been inspected, this could be researched in specific with relation to this specific neural network since a UAV must operate in multiple different weather conditions.

Reference list

[1] IAEA, “Environmental Consequences Of The Chernobyl Accident And Their Remediation: Twenty Years Of Experience,” *Radiol. Assess. REPORTS Ser.*, p. 180, 2006.

[2] IAEA, “The Fukushima Daiichi Accident: Radiological Consequences,” *Fukushima Daiichi Accid. Vol. 4/5*, vol. 4, p. 237, 2015.

[3] IAEA, “The Radiological Accident in Goiânia,” *Radiol. Accid. Goiânia*, p. 157, 1988.

[4] K. S. KRANE, “Krane - Introductory Nuclear Physics.pdf.” p. 831, 1987.

[5] L. N. H. Becquerel, “Cs-137 tables,” *Atomic and Nuclear Data*, 2007. http://www.lnhb.fr/nuclides/Cs-137_tables.pdf.

[6] G. R. Gilmore, *Practical Gamma-Ray Spectrometry*. 2008.

[7] and K. O. M.J. Berger, J.H. Hubbell, S.M. Seltzer, J. Chang, J.S. Coursey, R. Sukumar, D.S. Zucker, “XCOM: Photon Cross Sections Database,” *NIST Standard Reference Database 8 (XGAM)*. 2010, doi: https://dx.doi.org/10.18434/T48G6X.

[8] G. F. Knoll, *Radiation Detection and Measurement*, 4th ed. Wiley, 2010.

[9] M. Lowdon *et al.*, “Evaluation of scintillator detection materials for application within airborne environmental radiation monitoring,” *Sensors (Switzerland)*, vol. 19, no. 18, 2019, doi: 10.3390/s19183828.

[10] P. Yang, C. D. Harmon, F. P. Doty, and J. A. Ohlhausen, “Effect of humidity on scintillation performance in Na and Tl activated CsI crystals,” *IEEE Trans. Nucl. Sci.*, vol. 61, no. 2, pp. 1024–1031, 2014, doi: 10.1109/TNS.2014.2300471.

[11] IAEA, *INES THE INTERNATIONAL NUCLEAR AND RADIOLOGICAL EVENT SCALE USER’S MANUAL*. 2013.

[12] IAEA, *The Chernobyl Accident: Updating of INSAG-1*, vol. 19, no. 5. 1993.

[13] GRS, “The Accident and the Safety of RBMK-Reactors February 1996 GRS in Eastern Europe,” no. February, 1996.

[14] D. T. Connor *et al.*, “Radiological Mapping of Post-Disaster Nuclear Environments Using Fixed-Wing Unmanned Aerial Systems: A Study From Chornobyl,” *Front. Robot. AI*, vol. 6, no. January, pp. 1–14, 2020, doi: 10.3389/frobt.2019.00149.

[15] World Nuclear Association, “Fukushima Daiichi Accident - World Nuclear Association,” 2020. https://www.world-nuclear.org/information-library/safety-and-security/safety-of-plants/fukushima-daiichi-accident.aspx.

[16] S. M. L. Hardie and I. G. McKinley, “Fukushima remediation: Status and overview of future plans,” *J. Environ. Radioact.*, vol. 133, no. December 2017, pp. 75–85, 2014, doi: 10.1016/j.jenvrad.2013.08.002.

[17] A. (IAEA D. of N. S. and A. Peeva, “Now Available: New Drone Technology for Radiological Monitoring in Emergency Situations,” 2021. https://www.iaea.org/newscenter/news/now-available-new-drone-technology-for-radiological-monitoring-in-emergency-situations (accessed Jun. 05, 2021).

[18] P. G. Martin, O. D. Payton, J. S. Fardoulis, D. A. Richards, and T. B. Scott, “The use of unmanned aerial systems for the mapping of legacy uranium mines,” *J. Environ. Radioact.*, vol. 143, pp.

135–140, 2015, doi: 10.1016/j.jenvrad.2015.02.004.

[19] P. G. Martin, O. D. Payton, J. S. Fardoulis, D. A. Richards, Y. Yamashiki, and T. B. Scott, "Low altitude unmanned aerial vehicle for characterising remediation effectiveness following the FDNPP accident," *J. Environ. Radioact.*, vol. 151, pp. 58–63, 2016, doi: 10.1016/j.jenvrad.2015.09.007.

[20] J. W. MacFarlane *et al.*, "Lightweight aerial vehicles for monitoring, assessment and mapping of radiation anomalies," *J. Environ. Radioact.*, vol. 136, pp. 127–130, 2014, doi: 10.1016/j.jenvrad.2014.05.008.

[21] P. G. Martin *et al.*, "3D unmanned aerial vehicle radiation mapping for assessing contaminant distribution and mobility," *Int. J. Appl. Earth Obs. Geoinf.*, vol. 52, pp. 12–19, 2016, doi: 10.1016/j.jag.2016.05.007.

[22] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. 2017.

[23] A. J. S. Aston Zhang, Zachary C. Lipton, Mu Li, *Dive Into Deep Learning*, vol. 15, no. 1. 2020.

[24] J. B. Ahire, "The Artificial Neural Networks handbook: Part 1," *Medium*, 2018. https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4 (accessed Jun. 07, 2021).

[25] J. C. Duchi, P. L. Bartlett, and M. J. Wainwright, "Randomized smoothing for (parallel) stochastic optimization," *Proc. IEEE Conf. Decis. Control*, vol. 12, pp. 5442–5444, 2012, doi: 10.1109/CDC.2012.6426698.

[26] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.

[27] L. Marques, A. Vale, and P. Vaz, "State-of-the-art mobile radiation detection systems for different scenarios," *Sensors (Switzerland)*, vol. 21, no. 4, pp. 1–67, 2021, doi: 10.3390/s21041051.

[28] C. Sabbarese, F. Ambrosino, A. D'Onofrio, and V. Roca, "Radiological characterization of natural building materials from the Campania region (Southern Italy)," *Constr. Build. Mater.*, no. xxxx, p. 121087, 2020, doi: 10.1016/j.conbuildmat.2020.121087.

[29] W. Khan, C. He, Y. Cao, R. Khan, and W. Yang, "A detector system for searching lost γ-ray source," *Nucl. Eng. Technol.*, vol. 52, no. 7, pp. 1524–1531, 2020, doi: 10.1016/j.net.2019.12.021.

[30] Junios, F. Haryanto, Z. Su'ud, and Novitrian, "The effect of radiation source distance to the scintillation detector based on Monte Carlo simulation," *J. Phys. Conf. Ser.*, vol. 1493, no. 1, 2020, doi: 10.1088/1742-6596/1493/1/012012.

[31] E. Yoshida, K. Shizuma, S. Endo, and T. Oka, "Application of neural networks for the analysis of gamma-ray spectra measured with a Ge spectrometer," *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.*, vol. 484, no. 1–3, pp. 557–563, 2002, doi: 10.1016/S0168-9002(01)01962-3.

[32] J. Kim, K. Park, and G. Cho, "Multi-radioisotope identification algorithm using an artificial neural network for plastic gamma spectra," *Appl. Radiat. Isot.*, vol. 147, no. February, pp. 83–90, 2019, doi: 10.1016/j.apradiso.2019.01.005.

[33] M. Kamuda, J. Stinnett, and C. J. Sullivan, "Automated Isotope Identification Algorithm Using

Artificial Neural Networks," *IEEE Trans. Nucl. Sci.*, vol. 64, no. 7, pp. 1858–1864, 2017, doi: 10.1109/TNS.2017.2693152.

[34] D. Oh *et al.*, "CNN-Peaks: ChIP-Seq peak detection pipeline using convolutional neural networks that imitate human visual inspection," *Sci. Rep.*, vol. 10, no. 1, pp. 1–12, 2020, doi: 10.1038/s41598-020-64655-4.

[35] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.

[36] IBM, "Convolutional Neural Networks," 2020. https://www.ibm.com/cloud/learn/convolutional-neural-networks.

[37] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 2261–2269, 2017, doi: 10.1109/CVPR.2017.243.

[38] H. Sahiner and X. Liu, "Gamma spectral analysis by artificial neural network coupled with Monte Carlo simulations," *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.*, vol. 953, no. October 2019, p. 163062, 2020, doi: 10.1016/j.nima.2019.163062.

Appendix A: Python code for making the dataset

```
# -*- coding: utf-8 -*-

"""
Created on Mon Sep  7 16:15:18 2020

@author: daanv


"""
import pandas as pd
import os
from os import listdir
from os.path import isfile,join
import numpy as np
import sys
import pickle
import matplotlib.pyplot as plt




#progressbar om te zien hoe lang het nog zal duren
def progress(count, total):
    bar_len = 60
    filled_len = int(round(bar_len * count / float(total)))
    percents = (100.0 * count / float(total))
    percents = round(percents,2)
    bar = '#' * filled_len + ' ' * (bar_len - filled_len)
    sys.stdout.write('\r')
    sys.stdout.write('[%s] %s%s' % (bar, percents, '%'))
    sys.stdout.write('\r')
    sys.stdout.flush()

#os.remove('data.h5')

def before(value, a):
    # Find first part and return slice before it.
    pos_a = value.find(a)
    if pos_a == -1: return ""
    return value[0:pos_a]
#bron:https://www.dotnetperls.com/between-before-after-python

def after(value, a):
    # Find and validate first part.
    pos_a = value.rfind(a)
    if pos_a == -1: return ""
    # Returns chars after the found string.
    adjusted_pos_a = pos_a + len(a)
    if adjusted_pos_a >= len(value): return ""
    return value[adjusted_pos_a:]

#bron:https://www.dotnetperls.com/between-before-after-python
```

```
def between(value, a, b):
    # Find and validate before-part.
    pos_a = value.find(a)
    if pos_a == -1: return ""
    # Find and validate after part.
    pos_b = value.rfind(b)
    if pos_b == -1: return ""
    # Return middle part.
    adjusted_pos_a = pos_a + len(a)
    if adjusted_pos_a >= pos_b: return ""
    return value[adjusted_pos_a:pos_b]



def Write():
    j=0
    #huidige directory toevoegen
    path=os.getcwd()
    #path zetten voor datafolder
    list_name = []


    HDF = pd.DataFrame({"RealTime":[],
                            "LifeTime":[],
                            "Energie":[],
                            "Counts":[],
                            "Afstand":[],
                            "Activiteit":[],
                            "Hoek":[],
                            "Bron":[]    })


    for root, directories, files in os.walk(path, topdown=False):
        for name in directories:
            list_name.append(os.path.join(root,name))



        for i in range(0, len(list_name)):
            data_path = list_name[i]
            # bestandsnamen zonder extensies binnenhalen
            file_names=[".".join(f.split(".")[:-1]) for f in listdir(data_path) if isfile
(join(data_path,f))]
            # bestandsnamen met extensies binnenhalen
            full_file_names=[f for f in listdir(data_path) if isfile (join(data_path,f))]


            Data = pd.DataFrame({"RealTime":[],
                            "LifeTime":[],
                            "Energie":[],
                            "Counts":[],
                            "Peak":[],
                            "Afstand":[],
                            "Activiteit":[],
                            "Hoek":[],
                            "Bron":[]    })

            for s in full_file_names:
                j=j+1
```

```
txt = pd.read_csv(data_path+'\\'+s)
#alles uit .spe file halen
#live time uit de meting
lifetime = float(before(txt.iat[7,0]," "))
#real time uit de meting
realtime = float(after(txt.iat[7,0]," "))
# gebruiken van de gemeten waardes
counts = []
#Energiekalibratie van het meettoestel
EKal = []
c0 = float(before(txt.iat[4107,0]," "))
c1 = float(after(txt.iat[4107,0]," "))


for k in range(0,4096):
   EKal.append(c0+c1*k)
   c = int(txt.iat[k+10,0])
   counts.append(c)
counts = np.array(counts)
Ekal = np.array(counts)
Channel = round((661-c0)/c1)
peak = np.zeros(4096)
bron = str(before(str(s),"_"))



#Bepaling welke bron er aanwezig is
if(bron == str('background')):
    bron = "Background"
    activiteit = 0
if(str('Cs137') ==bron):
    L = np.log(2)/30.01
    activiteit = 9.25*10**3*np.exp(-L*21.666666)
    for i in range (Channel - 100, Channel + 100):
        peak[i] = 1
#print(bron)
afstand = 0
#bepaling of bron onder een gemeten hoek lag
if ('°' in str(s)):
    hoek = float(between('s','°','_'))
else:
    hoek = float(0)
if('cm' in str(s)):
    afstand= int(between(str(s),"_", "cm"))
else:
    afstand = 0

if (j < 6500):
    sys.stdout.write('\r')
    progress(j, 6500)

if( i%650 ==0):
    plt.figure(i)
    plt.plot(EKal, counts)
    plt.plot(EKal, peak)
```

```
                Data = Data.append({"RealTime":realtime,
                            "LifeTime":lifetime,
                            "Energie":EKal,
                            "Counts":counts,
                            "Peak":peak,
                            "Afstand":afstand,
                            "Activiteit":activiteit,
                            "Hoek":hoek,
                            "Bron":bron   }, ignore_index = True)
            # data toevoegen aan hdf5 file om verder te verwerken in ML-algoritme
            HDF = HDF.append(Data,ignore_index = True)

    return(HDF)
HDF = Write()
print(HDF)
picklefile = open('data.pkl','wb')

pickle.dump(HDF,picklefile)
picklefile.close()
```

```
import tensorflow as tf
import pandas as pd
from datetime import datetime
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import regularizers
from sklearn.utils import shuffle
import tensorflow_addons as tfa

from sklearn.model_selection import train_test_split
import IPython
import random
import matplotlib.pyplot as plt
import copy
import h5py
import pickle
```

In [57]:

```
data = open("data.pkl",'rb')
data = pickle.load(data,encoding='bytes')

lijst = copy.copy(data)
voc_List = lijst.pop("Bron")


for naam in voc_List:
  vocabulair = voc_List.unique()



Y= data.pop("Bron")
print(Y)

vertaling = pd.DataFrame({
    "0" :[],
    "1" :[]
})



for i in range(0,len(Y)):
  a = np.where(vocabulair == Y[i])
  a = int(a[0])
  if(a == 1):
    Cs = 1
    Bg = 0
  else:
    Cs = 0
    Bg = 1
  vertaling = vertaling.append({
      "0": Bg,
      "1": Cs
```

```
  }, ignore_index = True)


vertaling = np.array(vertaling)
Peak_Y = data.pop('Peak')
Peak_Y = np.array(Peak_Y)
Peak_Y = Peak_Y.tolist()



Z = data.pop("Counts")
X= data.pop("Energie")

Y2 = data.pop("Activiteit")
Y2 = Y2.tolist()
for i in range (0, len(Y2)):
  a = Y2[i]
  a = a/1000
  Y2[i]= a


Y2 = np.array(Y2)

spectrum = pd.DataFrame()
energie = []
counts = []
n =random.randint(0,len(X))
spect = [500,1100,2650,3365,4000,5500 ]
labels_spect = ['Background Bunker','Background HOB', 'Background ENE', 'Cs-137 ENE', 'Cs-137
Bunker', 'Cs-137 HOB' ]
for i in range(0, len(X)):
    a = X[i]
    b = Z[i]
    a = np.array(a)
    b = np.array(b)
    energie.append(a)
    counts.append(b)

spectrum['Counts'] = counts
spectrum['Energie']= energie
spectrum = spectrum.to_numpy()
spectrum = spectrum.tolist()

counts = np.array(counts)
energie = np.array(energie)
x_train, x_test, y_train, y_test = train_test_split(spectrum,vertaling, test_size=0.1,
random_state = 42)



x_test_spect = x_test.copy()


x_train = tf.convert_to_tensor(x_train)
y_train = tf.convert_to_tensor(y_train)
x_test = tf.convert_to_tensor(x_test)
y_test = tf.convert_to_tensor(y_test)
```

```
x2_train,x2_test,y2_train,y2_test = train_test_split(data,Peak_Y, test_size = 0.1,
random_state=42)

y2_train = tf.convert_to_tensor(y2_train)
y2_test = tf.convert_to_tensor(y2_test)

#extra code om van de data nog een validatiesplit te maken, niet echt nodig, maar kwam om
gelijkaardige resultaten uit als met test-data gewoon

"""
data_val = validatie
Y_val= data_val.pop("Bron")
Y2_val = data_val.pop("Activiteit")
Y2_val = Y2_val/1000




vertaal = pd.DataFrame({
    "0" :[],
    "1" :[]
})



for i in range(val_percent,val_percent + len(Y_val)):
  a = np.where(vocabulair == Y_val[i])
  a = int(a[0])
  if(a == 1):
    Cs = 1
    Bg = 0
  else:
    Cs = 0
    Bg = 1
  vertaal = vertaal.append({
      "0": Bg,
      "1": Cs
  }, ignore_index = True)


y_val = tf.convert_to_tensor(vertaal)


Z_val = data_val.pop("Counts")
X_val= data_val.pop("Energie")


spectrum = pd.DataFrame()
energie = []
counts = []
for i in range(val_percent,val_percent + len(X_val)):
  a = X_val[i]
  b = Z_val[i]
  a = np.array(a)
```

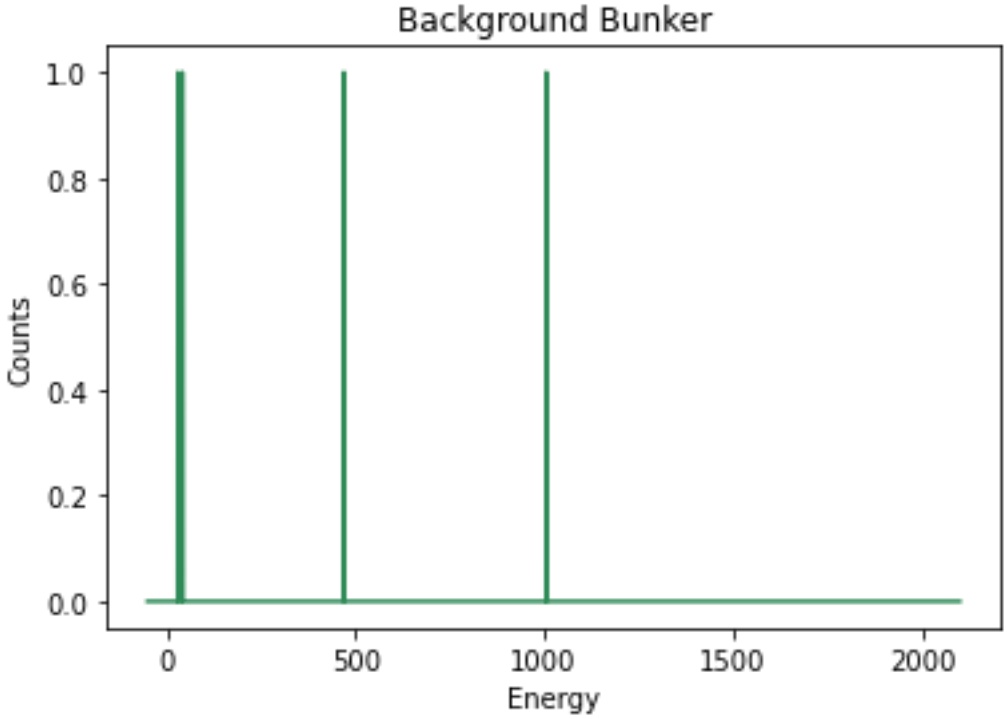
```
  b = np.array(b)

  energie.append(a)
  counts.append(b)

n = 50


spectrum['Counts'] = counts
spectrum['Energie']= energie
spectrum = spectrum.to_numpy()
spectrum = spectrum.tolist()
x_val = tf.convert_to_tensor(spectrum)
x2_val = data_val


print(model1.predict(spectrum))
print(Y_val)

"""
0       Background
1       Background
2       Background
3       Background
4       Background
           ...
6495      Cs137
6496      Cs137
6497      Cs137
6498      Cs137
6499      Cs137
Name: Bron, Length: 6500, dtype: object
```

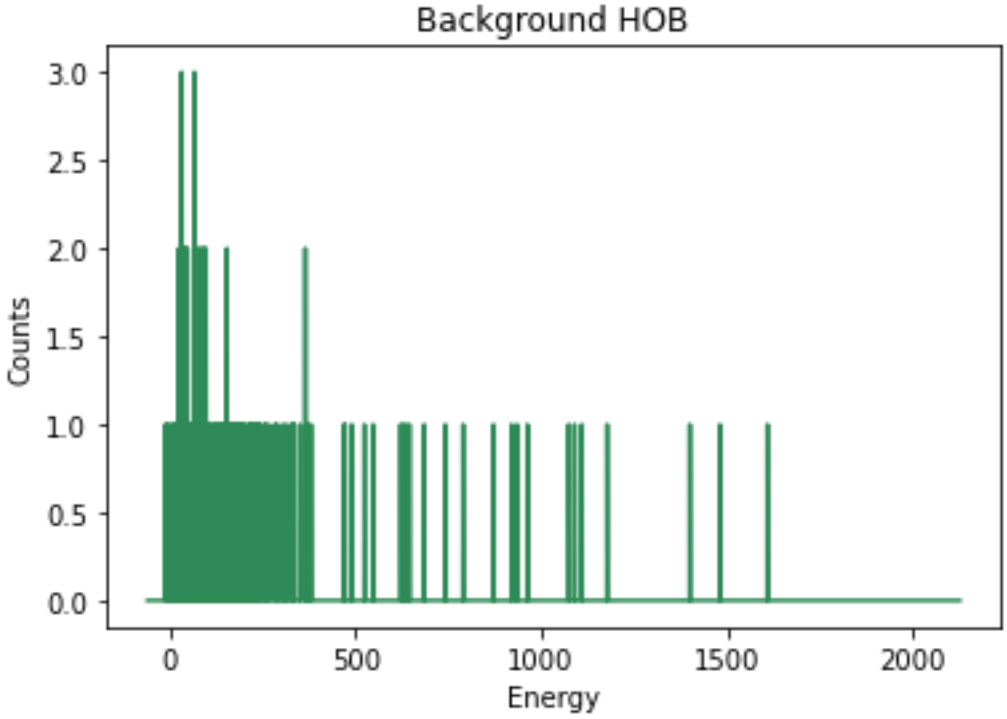
Out[57]:

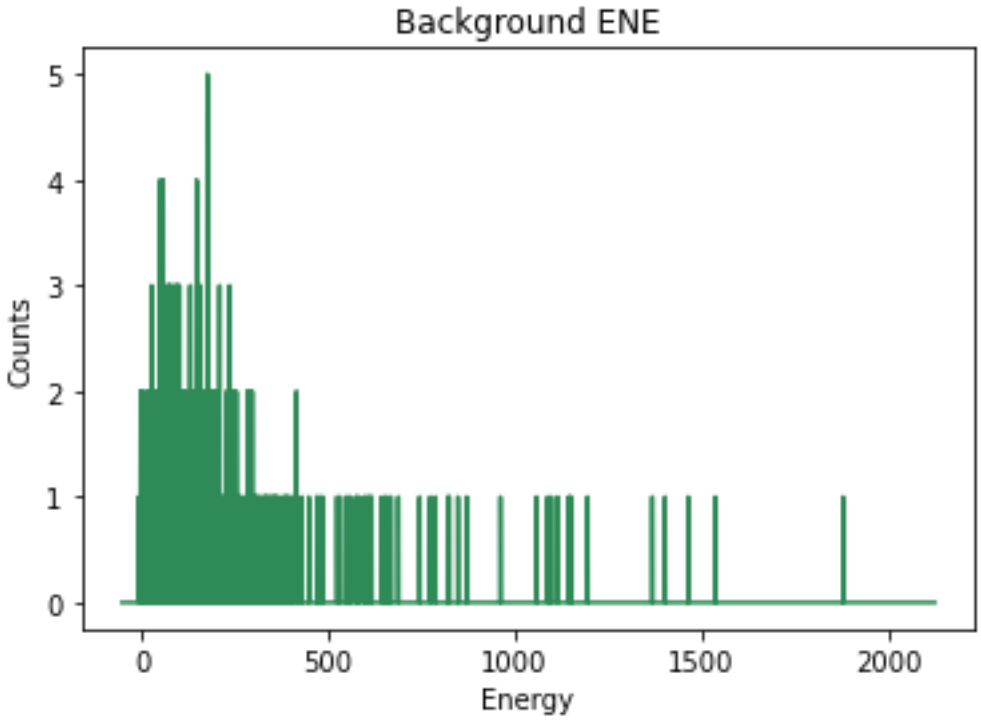
```
'\ndata_val = validatie\nY_val= data_val.pop("Bron")\nY2_val = data_val.pop("Activiteit")\nY2_
val = Y2_val/1000\n\n\n\n\n\nvertaal = pd.DataFrame({\n    "0" :[],\n    "1" :[]\n})\n\n\nfo
r i in range(val_percent,val_percent + len(Y_val)):\n  a = np.where(vocabulair == Y_val[i])\n
a = int(a[0])\n  if(a == 1):\n    Cs = 1\n    Bg = 0\n  else:\n    Cs = 0\n    Bg = 1\n  verta
al = vertaal.append({\n      "0": Bg,\n      "1": Cs\n  }, ignore_index = True)\n\n\ny_val = t
f.convert_to_tensor(vertaal)\n\n\nZ_val = data_val.pop("Counts")\nX_val= data_val.pop("Energie
")\n\n\nspectrum = pd.DataFrame()\nenergie = []\ncounts = []\nfor i in range(val_percent,val_p
ercent + len(X_val)):\n  a = X_val[i]\n  b = Z_val[i]\n  a = np.array(a)\n\n  b = np.array(b)\
n\n  energie.append(a)\n  counts.append(b)\n\nn = 50\n\n  \nspectrum[\'Counts\'] = counts\nspe
ctrum[\'Energie\']= energie\nspectrum = spectrum.to_numpy()\nspectrum = spectrum.tolist()\nx_v
al = tf.convert_to_tensor(spectrum)\nx2_val = data_val\n\n\nprint(model1.predict(spectrum))\np
rint(Y_val)\n\n'
```

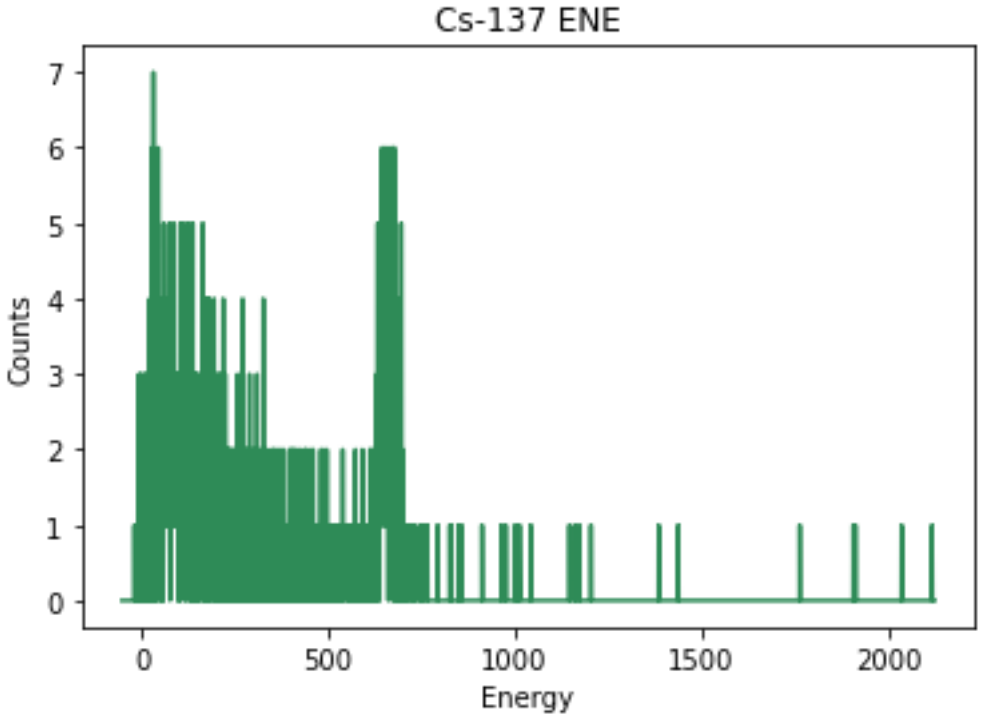
In [58]:

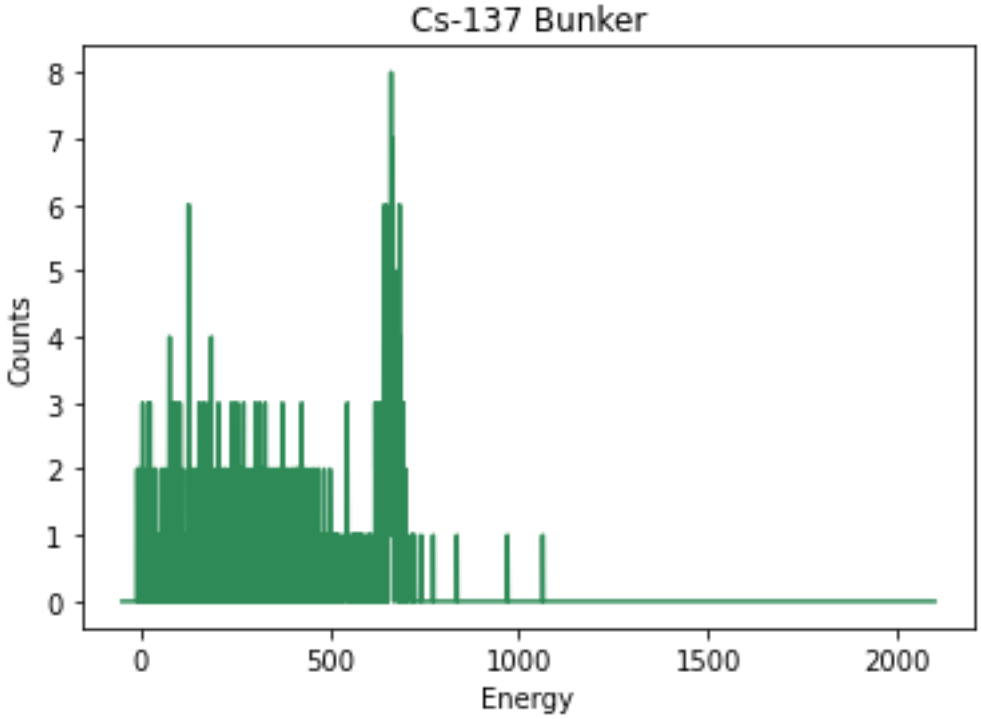
```
for spe in spect:

        rand = spe
        labeler = spect.index(rand)
        plt.plot(X[rand],Z[rand], color = 'seagreen')
        plt.ylabel('Counts')
        plt.xlabel('Energy')
        plt.title(labels_spect[labeler])
        plt.show()
```

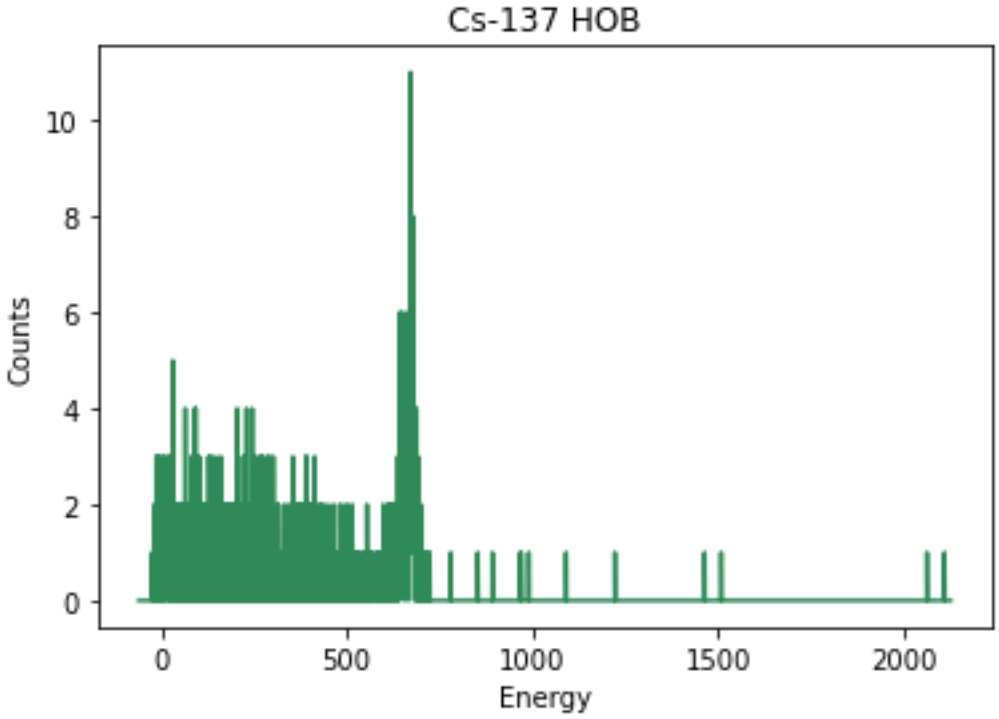

Background Bunker
Counts
Energy
0.0
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
2000


Background HOB
Counts
Energy
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
500
1000
1500
2000


Background ENE
Counts
Energy
0
1
2
3
4
5
0
500
1000
1500
2000


Cs-137 ENE
Counts
Energy
0
1
2
3
4
5
6
7
0
500
1000
1500
2000





In [62]:

```
# Clear any logs from previous runs
!rm -rf ./logs/
#import tensorflow_probability as tfds


def build_model():
```

```
    hp1 = 35
    hp10 = 194
    hp11 = 190
    hp12 = 70
    hp13 = 192
    hp14 = 224
    hp15 = 164
    hp16 = 122
    ps = 1

    input1 = keras.Input(shape =(2,4096), name = "Counts")
    x1 = tf.keras.layers.Convolution1D(hp1,1, strides = 1,padding = 'same', activation =
'relu', kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001),name =
'block1_conv1')(input1)
    x1 = tf.keras.layers.Convolution1D(hp1,1, strides = 1,padding = 'same', activation =
'relu', kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001),name =
'block1_conv2')(x1)
    x1 = tf.keras.layers.Convolution1D(hp1,1, strides = 1,padding = 'same', activation =
'relu', kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block1_conv3')(x1)
    x1 = tf.keras.layers.MaxPooling1D(pool_size = ps, strides = 1, padding = 'same', name =
'Pooling1')(x1)

    x2 = tf.keras.layers.concatenate([x1,input1])
    x2 = tf.keras.layers.BatchNormalization()(x2)
    x2 = tf.keras.layers.Convolution1D(hp1,1 , padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block2_conv1')(x2)
    x2 = tf.keras.layers.Convolution1D(hp1,1 , padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block2_conv2')(x2)
    x2 = tf.keras.layers.Convolution1D(hp1,1 , padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block2_conv3')(x2)
    x2 = tf.keras.layers.MaxPooling1D(pool_size = ps, strides = 1, padding = 'same', name =
'Pooling4')(x2)

    x3 = tf.keras.layers.concatenate([input1,x1,x2])
    x3 = tf.keras.layers.BatchNormalization()(x3)
    x3 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block3_conv1')(x3)
    x3 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block3_conv2')(x3)
    x3 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block3_conv3')(x3)
    x3 = tf.keras.layers.MaxPooling1D(pool_size = ps, strides = 1, padding = 'same', name =
'Pooling5')(x3)

    x4 = tf.keras.layers.concatenate([input1,x1,x2,x3])
    x4 = tf.keras.layers.BatchNormalization()(x4)
    x4 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same',activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block4_conv1')(x4)
    x4 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block4_conv2')(x4)
```

```
    x4 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block4_conv3')(x4)
    x4 = tf.keras.layers.MaxPooling1D(pool_size = ps, strides = 1, padding = 'same', name =
'Pooling6')(x4)

    x5 = tf.keras.layers.concatenate([input1,x1,x2,x3,x4])
    x5 = tf.keras.layers.BatchNormalization()(x5)
    x5 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block5_conv1')(x5)
    x5 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block5_conv2')(x5)
    x5 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block5_conv3')(x5)
    x5 = tf.keras.layers.MaxPooling1D(pool_size = ps, strides = 1, padding = 'same', name =
'Pooling7')(x5)

    x6 = tf.keras.layers.concatenate([input1,x1,x2,x3,x4,x5])
    x6 = tf.keras.layers.BatchNormalization()(x6)
    x6 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block6_conv1')(x6)
    x6 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block6_conv2')(x6)
    x6 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block6_conv3')(x6)
    x6 = tf.keras.layers.MaxPooling1D(pool_size = ps, strides = 1, padding = 'same', name =
'Pooling8')(x6)

    x7= tf.keras.layers.concatenate([input1,x1,x2,x3,x4,x5,x6])
    x7 = tf.keras.layers.BatchNormalization()(x7)
    x7 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block7_conv1')(x7)
    x7 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block7_conv2')(x7)
    x7 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block7_conv3')(x7)
    x7 = tf.keras.layers.MaxPooling1D(pool_size = ps, strides = 1, padding = 'same', name =
'Pooling9')(x7)

    x8= tf.keras.layers.concatenate([input1,x1,x2,x3,x4,x5,x6,x7])
    x8 = tf.keras.layers.BatchNormalization()(x8)
    x8 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same',activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block8_conv1')(x8)
    x8 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block8_conv2')(x8)
```

```
    x8 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block8_conv3')(x8)
    x8 = tf.keras.layers.MaxPooling1D(pool_size = ps, strides = 1, padding = 'same', name =
'Pooling10')(x8)

    x9= tf.keras.layers.concatenate([input1,x1,x2,x3,x4,x5,x6,x7,x8])
    x9 = tf.keras.layers.BatchNormalization()(x9)
    x9 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block9_conv1')(x9)
    x9 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block9_conv2')(x9)
    x9 = tf.keras.layers.Convolution1D(hp1,1,padding = 'same', activation =
'relu',kernel_regularizer = tf.keras.regularizers.L1L2(l1=0.001, l2=0.001), name =
'block9_conv3')(x9)
    x9 = tf.keras.layers.MaxPooling1D(pool_size = ps, strides = 1, padding = 'same', name =
'Pooling11')(x9)

    conc = tf.keras.layers.concatenate([x1,x2,x3,x4,x5,x6,x7,x8,x9])
    conc = tf.keras.layers.MaxPooling1D(pool_size = 1, strides = 4, name = 'MaxPool')(conc)
    x = tf.keras.layers.Flatten()(conc)
    output2 = tf.keras.layers.Dense(4096, activation = 'swish', name = 'Peak')(x)


    x = tf.keras.layers.Dense(hp10, activation = 'relu', name = 'Dense1')(x)
    x = tf.keras.layers.Dense(hp11, activation = 'relu', name = 'Dense2')(x)
    x = tf.keras.layers.Dense(hp12, activation = 'relu', name = 'Dense3')(x)
    x = tf.keras.layers.Dense(hp13, activation = 'relu', name = 'Dense4')(x)
    x = tf.keras.layers.Dense(hp14, activation = 'relu', name = 'Dense5')(x)
    x = tf.keras.layers.Dense(hp15, activation = 'relu', name = 'Dense6')(x)
    x = tf.keras.layers.Dense(hp16, activation = 'relu', name = 'Dense7')(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    output1 = tf.keras.layers.Dense(2, activation = 'softmax', name = 'Bron')(x)


    model = keras.Model(inputs = (input1), outputs=(output1, output2))

    losses = {
        'Bron' : tf.keras.losses.CategoricalCrossentropy(from_logits = True),
        'Peak' : tf.keras.losses.MeanAbsoluteError()
    }

    Metrics = {
        'Bron' : ['accuracy',tfa.metrics.F1Score(num_classes = 2, threshold = 0.5)],
        'Peak': [tf.keras.metrics.BinaryAccuracy()]

    }

    model.compile(loss=losses,
                  optimizer=keras.optimizers.Adam(),
                  metrics= Metrics )


    return model

# Data opslaan om te visualiseren met tensorboard
```

```
logdir="logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq = 1,
write_images = True)

model1 = build_model()

model1.summary()
'rm' is not recognized as an internal or external command,
operable program or batch file.
Model: "functional_37"
_________________________________________________________________________________________
____
Layer (type)                 Output Shape              Param #     Connected to
=========================================================================================
====
Counts (InputLayer)          [(None, 2, 4096)]         0
_________________________________________________________________________________________
____
block1_conv1 (Conv1D)        (None, 2, 35)             143395      Counts[0][0]
_________________________________________________________________________________________
____
block1_conv2 (Conv1D)        (None, 2, 35)             1260        block1_conv1[0][0]
_________________________________________________________________________________________
____
block1_conv3 (Conv1D)        (None, 2, 35)             1260        block1_conv2[0][0]
_________________________________________________________________________________________
____
Pooling1 (MaxPooling1D)      (None, 2, 35)             0           block1_conv3[0][0]
_________________________________________________________________________________________
____
concatenate_173 (Concatenate)  (None, 2, 4131)         0           Pooling1[0][0]
                                                                   Counts[0][0]
_________________________________________________________________________________________
____
batch_normalization_152 (BatchN (None, 2, 4131)        16524       concatenate_173[0][0]
_________________________________________________________________________________________
____
block2_conv1 (Conv1D)        (None, 2, 35)             144620      batch_normalization_152[0][0]
_________________________________________________________________________________________
____
block2_conv2 (Conv1D)        (None, 2, 35)             1260        block2_conv1[0][0]
_________________________________________________________________________________________
____
block2_conv3 (Conv1D)        (None, 2, 35)             1260        block2_conv2[0][0]
_________________________________________________________________________________________
____
Pooling4 (MaxPooling1D)      (None, 2, 35)             0           block2_conv3[0][0]
_________________________________________________________________________________________
____
concatenate_174 (Concatenate)  (None, 2, 4166)         0           Counts[0][0]
                                                                   Pooling1[0][0]
                                                                   Pooling4[0][0]
_________________________________________________________________________________________
____
batch_normalization_153 (BatchN (None, 2, 4166)        16664       concatenate_174[0][0]
_________________________________________________________________________________________
____
block3_conv1 (Conv1D)        (None, 2, 35)             145845      batch_normalization_153[0][0]
_________________________________________________________________________________________
____
block3_conv2 (Conv1D)        (None, 2, 35)             1260        block3_conv1[0][0]
_________________________________________________________________________________________
____
block3_conv3 (Conv1D)        (None, 2, 35)             1260        block3_conv2[0][0]
_________________________________________________________________________________________
____
Pooling5 (MaxPooling1D)      (None, 2, 35)             0           block3_conv3[0][0]
_________________________________________________________________________________________
____
concatenate_175 (Concatenate)  (None, 2, 4201)         0           Counts[0][0]
                                                                   Pooling1[0][0]
                                                                   Pooling4[0][0]
                                                                   Pooling5[0][0]
```

batch_normalization_154 (BatchN	(None, 2, 4201)	16804	concatenate_175[0][0]
block4_conv1 (Conv1D)	(None, 2, 35)	147070	batch_normalization_154[0][0]
block4_conv2 (Conv1D)	(None, 2, 35)	1260	block4_conv1[0][0]
block4_conv3 (Conv1D)	(None, 2, 35)	1260	block4_conv2[0][0]
Pooling6 (MaxPooling1D)	(None, 2, 35)	0	block4_conv3[0][0]
concatenate_176 (Concatenate)	(None, 2, 4236)	0	Counts[0][0] Pooling1[0][0] Pooling4[0][0] Pooling5[0][0] Pooling6[0][0]
batch_normalization_155 (BatchN	(None, 2, 4236)	16944	concatenate_176[0][0]
block5_conv1 (Conv1D)	(None, 2, 35)	148295	batch_normalization_155[0][0]
block5_conv2 (Conv1D)	(None, 2, 35)	1260	block5_conv1[0][0]
block5_conv3 (Conv1D)	(None, 2, 35)	1260	block5_conv2[0][0]
Pooling7 (MaxPooling1D)	(None, 2, 35)	0	block5_conv3[0][0]
concatenate_177 (Concatenate)	(None, 2, 4271)	0	Counts[0][0] Pooling1[0][0] Pooling4[0][0] Pooling5[0][0] Pooling6[0][0] Pooling7[0][0]
batch_normalization_156 (BatchN	(None, 2, 4271)	17084	concatenate_177[0][0]
block6_conv1 (Conv1D)	(None, 2, 35)	149520	batch_normalization_156[0][0]
block6_conv2 (Conv1D)	(None, 2, 35)	1260	block6_conv1[0][0]
block6_conv3 (Conv1D)	(None, 2, 35)	1260	block6_conv2[0][0]
Pooling8 (MaxPooling1D)	(None, 2, 35)	0	block6_conv3[0][0]
concatenate_178 (Concatenate)	(None, 2, 4306)	0	Counts[0][0] Pooling1[0][0] Pooling4[0][0] Pooling5[0][0] Pooling6[0][0] Pooling7[0][0] Pooling8[0][0]
batch_normalization_157 (BatchN	(None, 2, 4306)	17224	concatenate_178[0][0]
block7_conv1 (Conv1D)	(None, 2, 35)	150745	batch_normalization_157[0][0]
block7_conv2 (Conv1D)	(None, 2, 35)	1260	block7_conv1[0][0]

block7_conv3 (Conv1D)	(None, 2, 35)	1260	block7_conv2[0][0]
Pooling9 (MaxPooling1D)	(None, 2, 35)	0	block7_conv3[0][0]
concatenate_179 (Concatenate)	(None, 2, 4341)	0	Counts[0][0] Pooling1[0][0] Pooling4[0][0] Pooling5[0][0] Pooling6[0][0] Pooling7[0][0] Pooling8[0][0] Pooling9[0][0]
batch_normalization_158 (BatchN	(None, 2, 4341)	17364	concatenate_179[0][0]
block8_conv1 (Conv1D)	(None, 2, 35)	151970	batch_normalization_158[0][0]
block8_conv2 (Conv1D)	(None, 2, 35)	1260	block8_conv1[0][0]
block8_conv3 (Conv1D)	(None, 2, 35)	1260	block8_conv2[0][0]
Pooling10 (MaxPooling1D)	(None, 2, 35)	0	block8_conv3[0][0]
concatenate_180 (Concatenate)	(None, 2, 4376)	0	Counts[0][0] Pooling1[0][0] Pooling4[0][0] Pooling5[0][0] Pooling6[0][0] Pooling7[0][0] Pooling8[0][0] Pooling9[0][0] Pooling10[0][0]
batch_normalization_159 (BatchN	(None, 2, 4376)	17504	concatenate_180[0][0]
block9_conv1 (Conv1D)	(None, 2, 35)	153195	batch_normalization_159[0][0]
block9_conv2 (Conv1D)	(None, 2, 35)	1260	block9_conv1[0][0]
block9_conv3 (Conv1D)	(None, 2, 35)	1260	block9_conv2[0][0]
Pooling11 (MaxPooling1D)	(None, 2, 35)	0	block9_conv3[0][0]
concatenate_181 (Concatenate)	(None, 2, 315)	0	Pooling1[0][0] Pooling4[0][0] Pooling5[0][0] Pooling6[0][0] Pooling7[0][0] Pooling8[0][0] Pooling9[0][0] Pooling10[0][0] Pooling11[0][0]
MaxPool (MaxPooling1D)	(None, 1, 315)	0	concatenate_181[0][0]
flatten_18 (Flatten)	(None, 315)	0	MaxPool[0][0]

```
__________________________________________________________________________________________
____
Dense1 (Dense)                  (None, 194)          61304       flatten_18[0][0]
__________________________________________________________________________________________
____
Dense2 (Dense)                  (None, 190)          37050       Dense1[0][0]
__________________________________________________________________________________________
____
Dense3 (Dense)                  (None, 70)           13370       Dense2[0][0]
__________________________________________________________________________________________
____
Dense4 (Dense)                  (None, 192)          13632       Dense3[0][0]
__________________________________________________________________________________________
____
Dense5 (Dense)                  (None, 224)          43232       Dense4[0][0]
__________________________________________________________________________________________
____
Dense6 (Dense)                  (None, 164)          36900       Dense5[0][0]
__________________________________________________________________________________________
____
Dense7 (Dense)                  (None, 122)          20130       Dense6[0][0]
__________________________________________________________________________________________
____
dropout_23 (Dropout)            (None, 122)          0           Dense7[0][0]
__________________________________________________________________________________________
_____
Bron (Dense)                    (None, 2)            246         dropout_23[0][0]
__________________________________________________________________________________________
____
Peak (Dense)                    (None, 4096)         1294336     flatten_18[0][0]
==========================================================================================
====
Total params: 3,013,647
Trainable params: 2,945,591
Non-trainable params: 68,056
__________________________________________________________________________________________
_____
```

In [60]:

```
#model1 = tf.keras.models.load_model('identification_model.h5')
```

In [63]:

```
NUM_EPOCHS = 16
BATCH_SIZE = 128
history1 = model1.fit(x_train, (y_train,y2_train), batch_size = BATCH_SIZE, epochs
=NUM_EPOCHS, verbose = 1, validation_data = (x_test,(y_test,y2_test)))


#model1.save('identification_model.h5')
```

```
Epoch 1/16
46/46 [==============================] - 10s 212ms/step - loss: 12.4032 - Bron_loss: 0.5554 -
Peak_loss: 0.0460 - Bron_accuracy: 0.7304 - Bron_f1_score: 0.7205 - Peak_binary_accuracy: 0.98
01 - val_loss: 4.4547 - val_Bron_loss: 0.4561 - val_Peak_loss: 0.0381 - val_Bron_accuracy: 0.8
954 - val_Bron_f1_score: 0.8852 - val_Peak_binary_accuracy: 0.9888
Epoch 2/16
46/46 [==============================] - 9s 194ms/step - loss: 3.6166 - Bron_loss: 0.4062 - Pe
ak_loss: 0.0455 - Bron_accuracy: 0.9062 - Bron_f1_score: 0.9024 - Peak_binary_accuracy: 0.9881
- val_loss: 2.8114 - val_Bron_loss: 0.3433 - val_Peak_loss: 0.0303 - val_Bron_accuracy: 0.9708
- val_Bron_f1_score: 0.9694 - val_Peak_binary_accuracy: 0.9888
Epoch 3/16
46/46 [==============================] - 8s 185ms/step - loss: 2.4291 - Bron_loss: 0.3479 - Pe
ak_loss: 0.0228 - Bron_accuracy: 0.9653 - Bron_f1_score: 0.9634 - Peak_binary_accuracy: 0.9881
- val_loss: 2.1165 - val_Bron_loss: 0.3531 - val_Peak_loss: 0.0175 - val_Bron_accuracy: 0.9585
- val_Bron_f1_score: 0.9566 - val_Peak_binary_accuracy: 0.9888
Epoch 4/16
46/46 [==============================] - 9s 185ms/step - loss: 1.8553 - Bron_loss: 0.3410 - Pe
ak_loss: 0.0164 - Bron_accuracy: 0.9701 - Bron_f1_score: 0.9684 - Peak_binary_accuracy: 0.9881
- val_loss: 1.6380 - val_Bron_loss: 0.3540 - val_Peak_loss: 0.0154 - val_Bron_accuracy: 0.9585
- val_Bron_f1_score: 0.9569 - val_Peak_binary_accuracy: 0.9888
Epoch 5/16
46/46 [==============================] - 9s 185ms/step - loss: 1.4383 - Bron_loss: 0.3359 - Pe
ak_loss: 0.0160 - Bron_accuracy: 0.9762 - Bron_f1_score: 0.9750 - Peak_binary_accuracy: 0.9881
- val_loss: 1.2751 - val_Bron_loss: 0.3506 - val_Peak_loss: 0.0143 - val_Bron_accuracy: 0.9662
- val_Bron_f1_score: 0.9644 - val_Peak_binary_accuracy: 0.9888
Epoch 6/16
46/46 [==============================] - 8s 184ms/step - loss: 1.1236 - Bron_loss: 0.3307 - Pe
ak_loss: 0.0147 - Bron_accuracy: 0.9829 - Bron_f1_score: 0.9820 - Peak_binary_accuracy: 0.9881
```

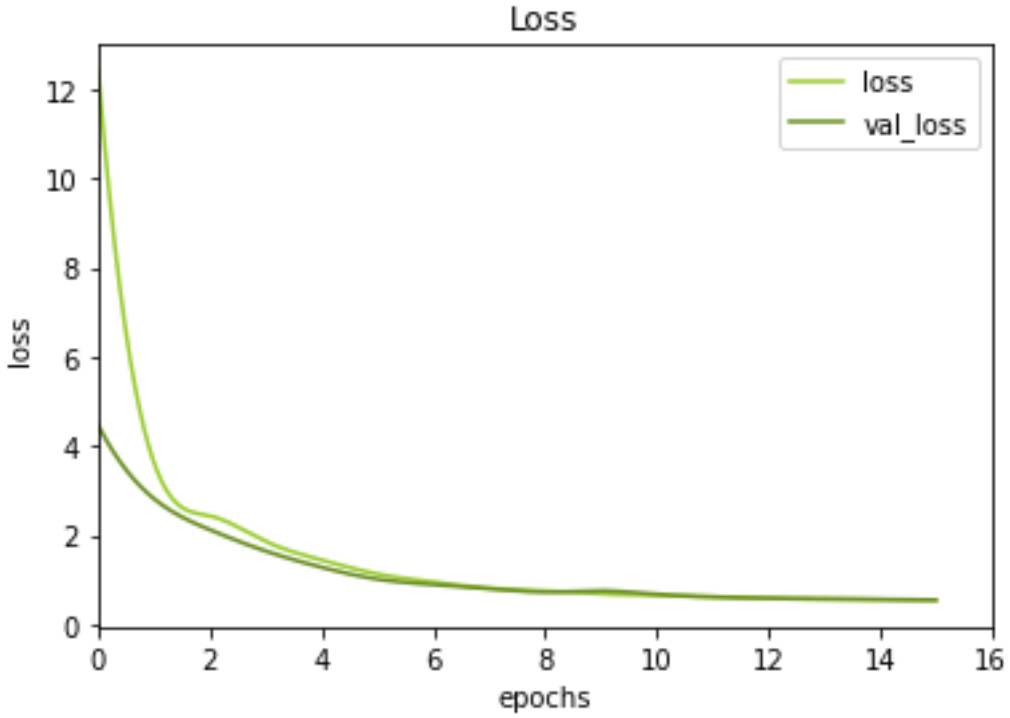
```
- val_loss: 1.0082 - val_Bron_loss: 0.3440 - val_Peak_loss: 0.0133 - val_Bron_accuracy: 0.9692
- val_Bron_f1_score: 0.9677 - val_Peak_binary_accuracy: 0.9888
Epoch 7/16
46/46 [==============================] - 9s 192ms/step - loss: 0.9465 - Bron_loss: 0.3383 - Pe
ak_loss: 0.0154 - Bron_accuracy: 0.9735 - Bron_f1_score: 0.9721 - Peak_binary_accuracy: 0.9881
- val_loss: 0.8875 - val_Bron_loss: 0.3494 - val_Peak_loss: 0.0141 - val_Bron_accuracy: 0.9615
- val_Bron_f1_score: 0.9595 - val_Peak_binary_accuracy: 0.9888
Epoch 8/16
46/46 [==============================] - 9s 195ms/step - loss: 0.8054 - Bron_loss: 0.3276 - Pe
ak_loss: 0.0140 - Bron_accuracy: 0.9853 - Bron_f1_score: 0.9845 - Peak_binary_accuracy: 0.9881
- val_loss: 0.7940 - val_Bron_loss: 0.3659 - val_Peak_loss: 0.0132 - val_Bron_accuracy: 0.9462
- val_Bron_f1_score: 0.9423 - val_Peak_binary_accuracy: 0.9888
Epoch 9/16
46/46 [==============================] - 9s 189ms/step - loss: 0.7465 - Bron_loss: 0.3405 - Pe
ak_loss: 0.0143 - Bron_accuracy: 0.9725 - Bron_f1_score: 0.9711 - Peak_binary_accuracy: 0.9881
- val_loss: 0.7196 - val_Bron_loss: 0.3450 - val_Peak_loss: 0.0131 - val_Bron_accuracy: 0.9692
- val_Bron_f1_score: 0.9676 - val_Peak_binary_accuracy: 0.9888
Epoch 10/16
46/46 [==============================] - 8s 184ms/step - loss: 0.6791 - Bron_loss: 0.3326 - Pe
ak_loss: 0.0138 - Bron_accuracy: 0.9802 - Bron_f1_score: 0.9791 - Peak_binary_accuracy: 0.9881
- val_loss: 0.7509 - val_Bron_loss: 0.4199 - val_Peak_loss: 0.0147 - val_Bron_accuracy: 0.8831
- val_Bron_f1_score: 0.8695 - val_Peak_binary_accuracy: 0.9888
Epoch 11/16
46/46 [==============================] - 9s 188ms/step - loss: 0.6462 - Bron_loss: 0.3306 - Pe
ak_loss: 0.0140 - Bron_accuracy: 0.9819 - Bron_f1_score: 0.9808 - Peak_binary_accuracy: 0.9881
- val_loss: 0.6704 - val_Bron_loss: 0.3746 - val_Peak_loss: 0.0127 - val_Bron_accuracy: 0.9354
- val_Bron_f1_score: 0.9303 - val_Peak_binary_accuracy: 0.9888
Epoch 12/16
46/46 [==============================] - 9s 186ms/step - loss: 0.6085 - Bron_loss: 0.3252 - Pe
ak_loss: 0.0135 - Bron_accuracy: 0.9879 - Bron_f1_score: 0.9872 - Peak_binary_accuracy: 0.9881
- val_loss: 0.6103 - val_Bron_loss: 0.3430 - val_Peak_loss: 0.0127 - val_Bron_accuracy: 0.9692
- val_Bron_f1_score: 0.9675 - val_Peak_binary_accuracy: 0.9888
Epoch 13/16
46/46 [==============================] - 9s 185ms/step - loss: 0.5773 - Bron_loss: 0.3214 - Pe
ak_loss: 0.0133 - Bron_accuracy: 0.9918 - Bron_f1_score: 0.9913 - Peak_binary_accuracy: 0.9881
- val_loss: 0.5805 - val_Bron_loss: 0.3369 - val_Peak_loss: 0.0127 - val_Bron_accuracy: 0.9769
- val_Bron_f1_score: 0.9759 - val_Peak_binary_accuracy: 0.9888
Epoch 14/16
46/46 [==============================] - 9s 187ms/step - loss: 0.5585 - Bron_loss: 0.3212 - Pe
ak_loss: 0.0134 - Bron_accuracy: 0.9920 - Bron_f1_score: 0.9915 - Peak_binary_accuracy: 0.9881
- val_loss: 0.5673 - val_Bron_loss: 0.3386 - val_Peak_loss: 0.0125 - val_Bron_accuracy: 0.9738
- val_Bron_f1_score: 0.9726 - val_Peak_binary_accuracy: 0.9888
Epoch 15/16
46/46 [==============================] - 9s 197ms/step - loss: 0.5429 - Bron_loss: 0.3203 - Pe
ak_loss: 0.0132 - Bron_accuracy: 0.9928 - Bron_f1_score: 0.9924 - Peak_binary_accuracy: 0.9881
- val_loss: 0.5586 - val_Bron_loss: 0.3411 - val_Peak_loss: 0.0127 - val_Bron_accuracy: 0.9708
- val_Bron_f1_score: 0.9692 - val_Peak_binary_accuracy: 0.9888
Epoch 16/16
46/46 [==============================] - 9s 187ms/step - loss: 0.5344 - Bron_loss: 0.3207 - Pe
ak_loss: 0.0132 - Bron_accuracy: 0.9923 - Bron_f1_score: 0.9919 - Peak_binary_accuracy: 0.9881
- val_loss: 0.5415 - val_Bron_loss: 0.3331 - val_Peak_loss: 0.0124 - val_Bron_accuracy: 0.9800
- val_Bron_f1_score: 0.9790 - val_Peak_binary_accuracy: 0.9888
```

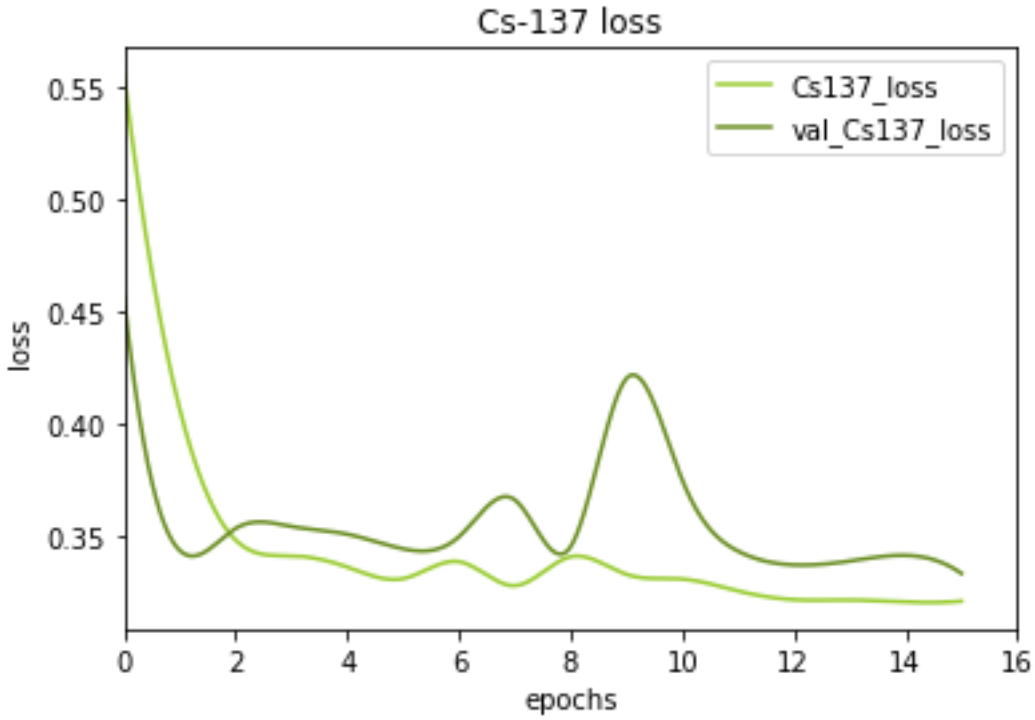
In [64]:

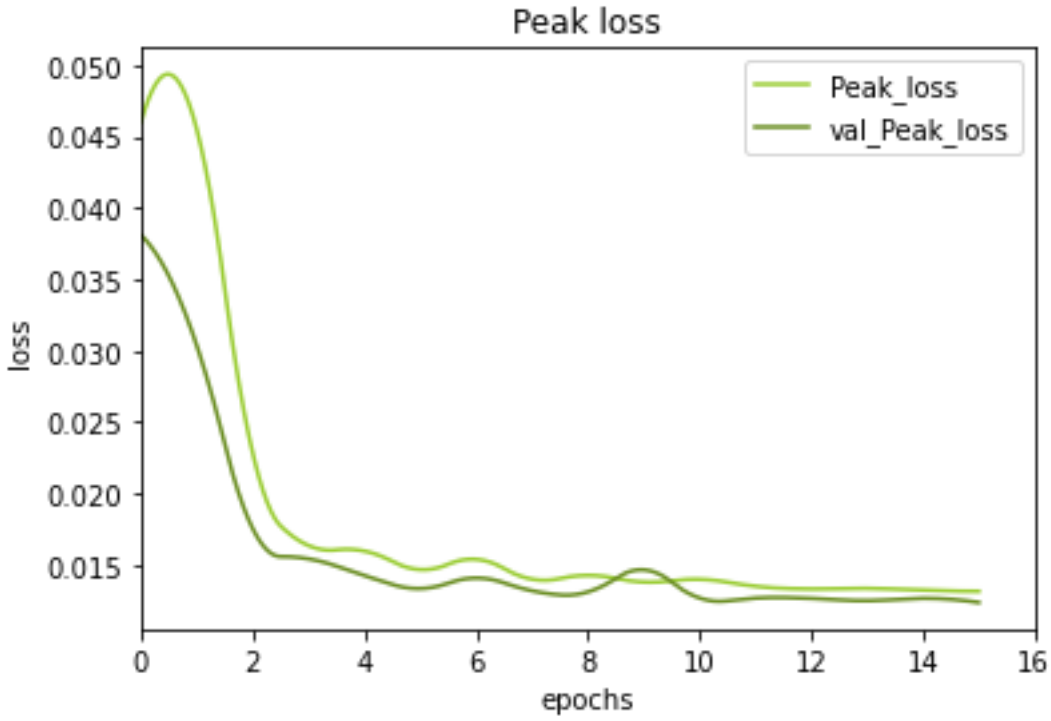
```
from scipy.interpolate import  make_interp_spline

val_bron_f1 = history1.history['val_Bron_f1_score']
bron_f1 = history1.history['Bron_f1_score']
for i in range(0, len(bron_f1)):
    values_val = val_bron_f1[i]
    values = bron_f1[i]
    value_val = np.average(values_val)
    value = np.average(values)
    val_bron_f1[i] = value_val
    bron_f1[i] = value


x = np.arange(0,len(bron_f1), 1)
xnew = np.arange(0,len(bron_f1)-1,0.0001)
length = NUM_EPOCHS
```

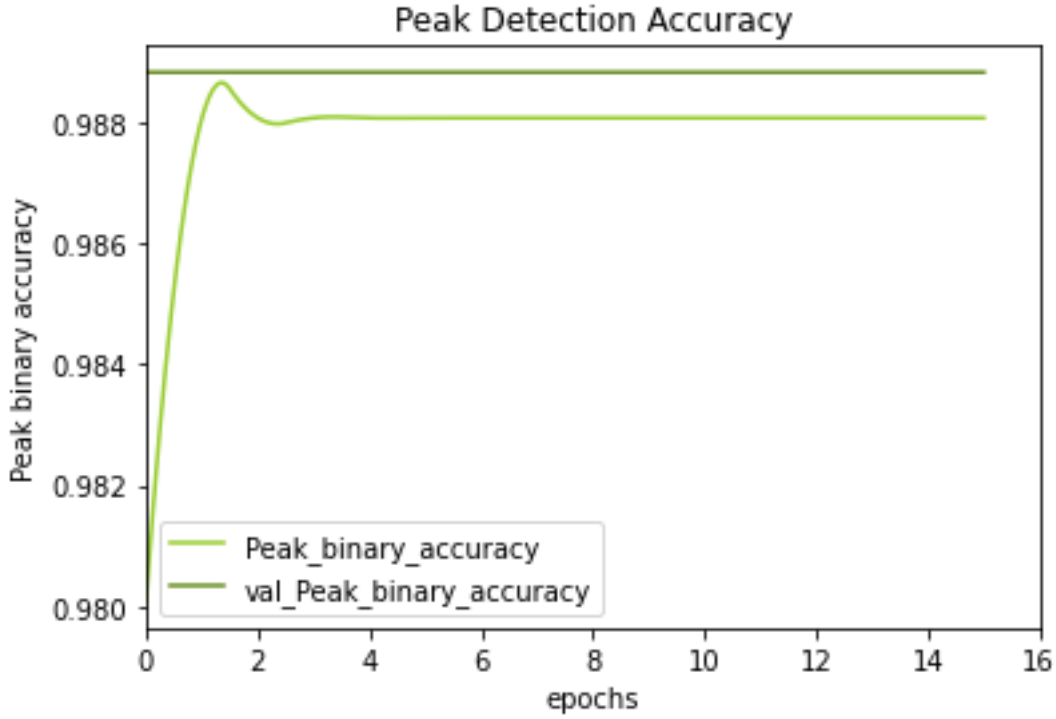
```
loss = make_interp_spline(x,history1.history['loss'],k=3)
Loss = loss(xnew)
val_loss = make_interp_spline(x,history1.history['val_loss'], k=3)
Val_loss = val_loss(xnew)
plt.title('Loss')
plt.plot(xnew, Loss, color = 'yellowgreen')
plt.plot(xnew,Val_loss, color = 'olivedrab')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(('loss', 'val_loss'))
plt.xlim([0,length])
plt.show()

bron_loss = make_interp_spline(x,history1.history['Bron_loss'],k=3)
Bron_loss = bron_loss(xnew)
bron_val_loss = make_interp_spline(x,history1.history['val_Bron_loss'], k=3)
Bron_val_loss = bron_val_loss(xnew)
plt.title('Cs-137 loss')
plt.plot(xnew, Bron_loss, color = 'yellowgreen', label = 'Cs137_loss')
plt.plot(xnew, Bron_val_loss, color = 'olivedrab', label = 'val_Cs137_loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend()
plt.xlim([0,length])
plt.show()

p_l = make_interp_spline(x,history1.history['Peak_loss'] , k = 2)
P_l = p_l(xnew)
v_p_l= make_interp_spline(x,history1.history['val_Peak_loss'] , k = 2)
V_p_l = v_p_l(xnew)

plt.plot(xnew,P_l, color = 'yellowgreen', label = 'Peak_loss')
plt.plot(xnew, V_p_l, color = 'olivedrab', label = 'val_Peak_loss')
plt.title('Peak loss')
plt.xlim([0,length])
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend()
plt.show()
"""
br_f1 = make_interp_spline(x, bron_f1, k = 2)
bron_f1 = br_f1(xnew)
val_br_f1 = make_interp_spline(x, val_bron_f1, k = 2)
val_bron_f1 = val_br_f1(xnew)
plt.plot(xnew,bron_f1, color = 'yellowgreen', label ='Bron_f1_score')
plt.plot(xnew,val_bron_f1, color = 'olivedrab', label = 'val_Bron_f1_score')
plt.xlim([0,length])
plt.title('Cs-137 F1-score')
plt.xlabel('epochs')
plt.ylabel('F1-score')
plt.legend()
plt.show()
"""
peak_acc = make_interp_spline(x,history1.history['Peak_binary_accuracy'], k=2 )
Peak_acc = peak_acc(xnew)
```

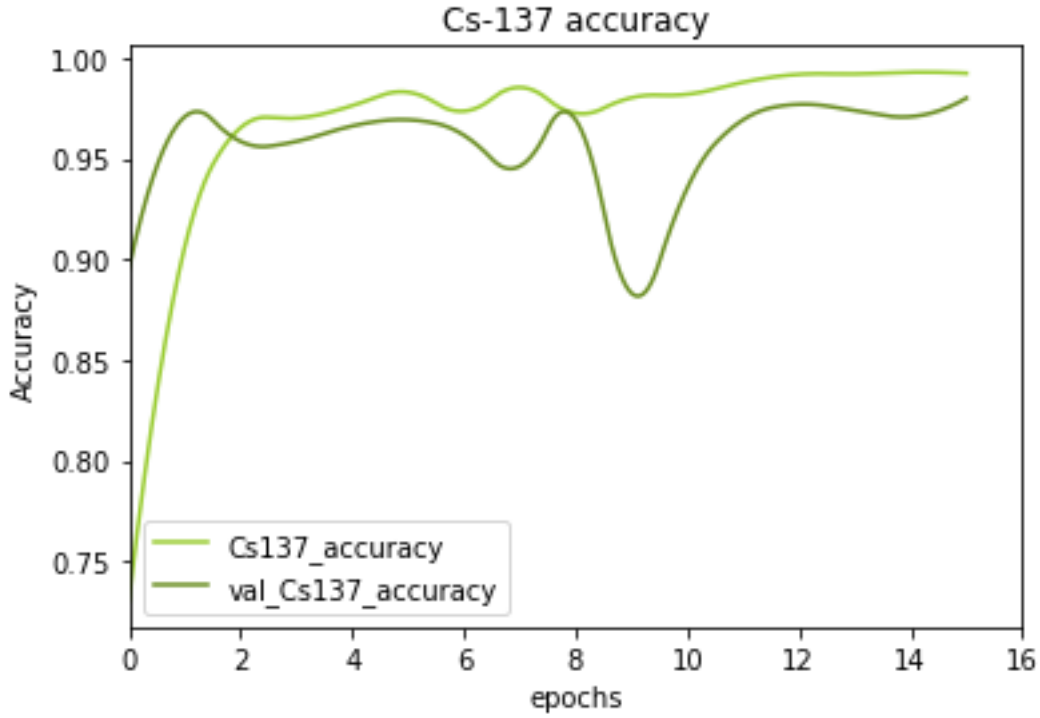
```
val_peak_acc = make_interp_spline(x,history1.history['val_Peak_binary_accuracy'], k=2 )
Val_peak_acc = val_peak_acc(xnew)
plt.plot(xnew, Peak_acc, color = 'yellowgreen', label ='Peak_binary_accuracy' )
plt.plot(xnew, Val_peak_acc, color = 'olivedrab', label = 'val_Peak_binary_accuracy')
plt.title('Peak Detection Accuracy')
plt.ylabel('Peak binary accuracy')
plt.xlabel('epochs')
plt.legend()
plt.xlim([0,length])

plt.show()
b_a = make_interp_spline(x,history1.history['Bron_accuracy'] , k = 2)
B_a = b_a(xnew)

v_b_a = make_interp_spline(x,history1.history['val_Bron_accuracy'] , k = 2)
V_b_a = v_b_a(xnew)
plt.plot(xnew, B_a, color = 'yellowgreen', label = 'Cs137_accuracy')
plt.plot(xnew,V_b_a, color= 'olivedrab', label = 'val_Cs137_accuracy')
plt.title('Cs-137 accuracy')
plt.ylabel('Accuracy')
plt.xlabel('epochs')
plt.xlim([0,length])
plt.legend()


plt.show()
```




Cs-137 loss
Cs137_loss
val_Cs137_loss
loss
epochs
0.55
0.50
0.45
0.40
0.35
0
2
4
6
8
10
12
14
16


Peak loss
Peak_loss
val_Peak_loss
loss
epochs
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0
2
4
6
8
10
12
14
16





In [66]:

```
(model1.evaluate((x_test),(y_test, y2_test), batch_size = 1))
```

```
650/650 [==============================] - 4s 6ms/step - loss: 0.5415 - Bron_loss: 0.3331 - Pe
ak_loss: 0.0124 - Bron_accuracy: 0.9800 - Bron_f1_score: 0.9790 - Peak_binary_accuracy: 0.9888
```

Out[66]:

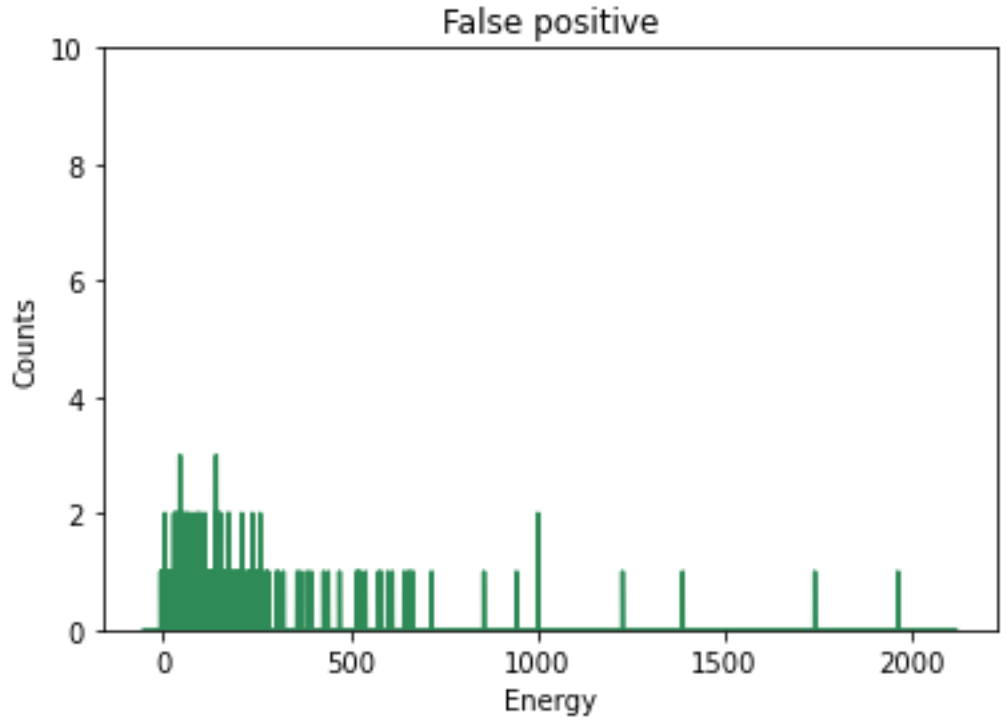
```
[0.5414845943450928,
 0.33305418491363525,
 0.012370091862976551,
 0.9800000190734863,
 array([0.97445977, 0.98356515], dtype=float32),
 0.9888408780097961]
```

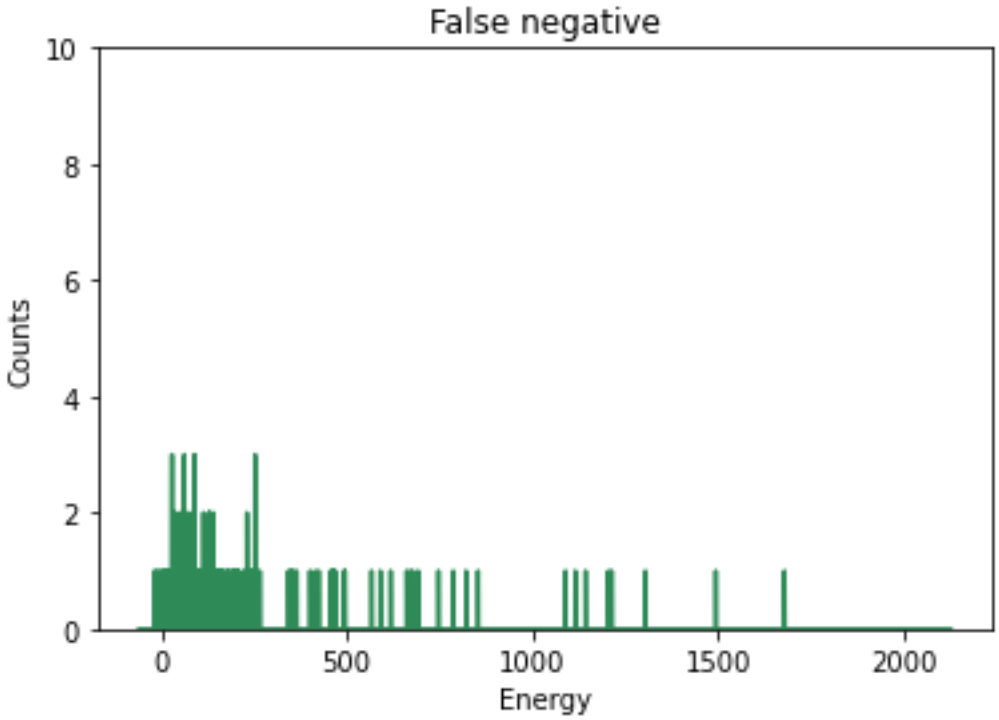
In [67]:

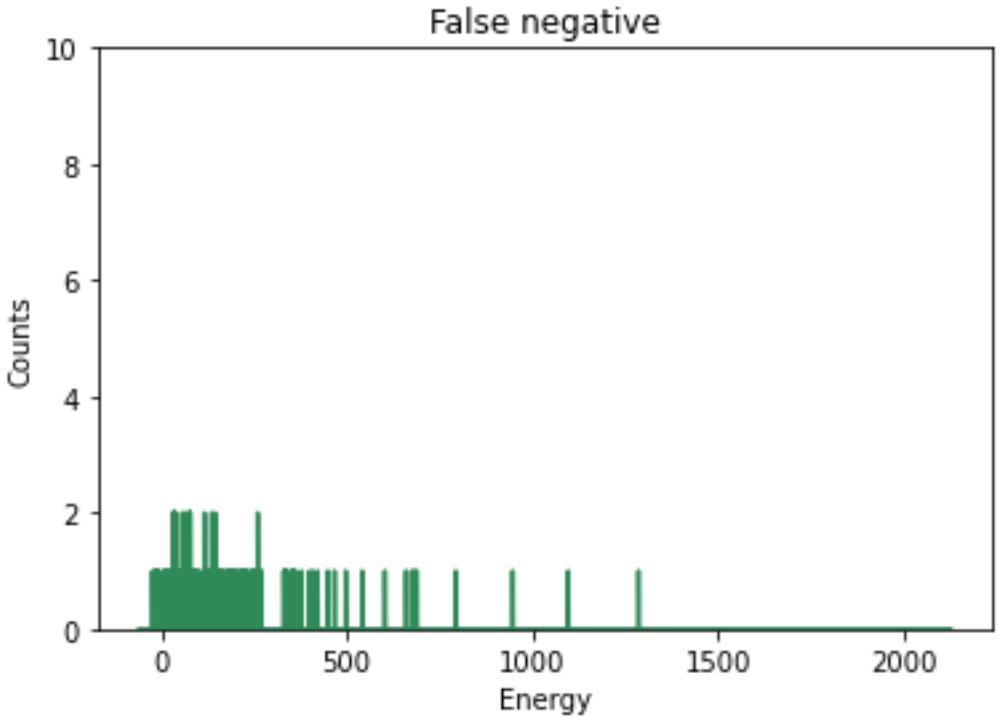
```
prediction, peakLocations = model1.predict(x_test)
peakLocations = (pd.DataFrame(peakLocations))
prediction = np.round(prediction)
a = pd.DataFrame(prediction)
print(a)
spect = (pd.DataFrame(x_test_spect))

c = spect[0]
e = spect[1]

b = y_test.numpy()

b = pd.DataFrame(b)
print(b)
b = pd.DataFrame(y_test)

FOUTenergie = []
FOUTcounts = []
FOUT = []

for i in range(0, len(x_test)):

    if 400< i <410:
        fig,ax= plt.subplots()
        rand = random.randint(0,len(x_test))
        ax.plot(e[rand],c[rand], color = 'seagreen')
        ax.set_ylim([0,10])
        ax2 = ax.twinx()
        ax2.plot(e[rand],peakLocations.iloc[rand,:], alpha = 0.7, color = 'yellowgreen')
        ax2.set_ylim([0,1])
        ax2.set_ylabel('Peak')
        ax.set_xlabel('Energy')
        ax.set_ylabel('Counts')

        ax.set_title('Peak detection')


    if(a.iloc[i,0]!=b.iloc[i,0]):
        FOUTenergie.append(e[i])
        FOUTcounts.append(c[i])
        FOUT.append(b.iloc[i,1])
        plt.figure(i+1)
        plt.plot(e[i],c[i], color = 'seagreen')
        plt.xlabel('Energy')
        plt.ylabel('Counts')
        plt.ylim(0,10)
        if(b.iloc[i,0] == 1):
            plt.title('False negative')
        else:
            plt.title('False positive')
```

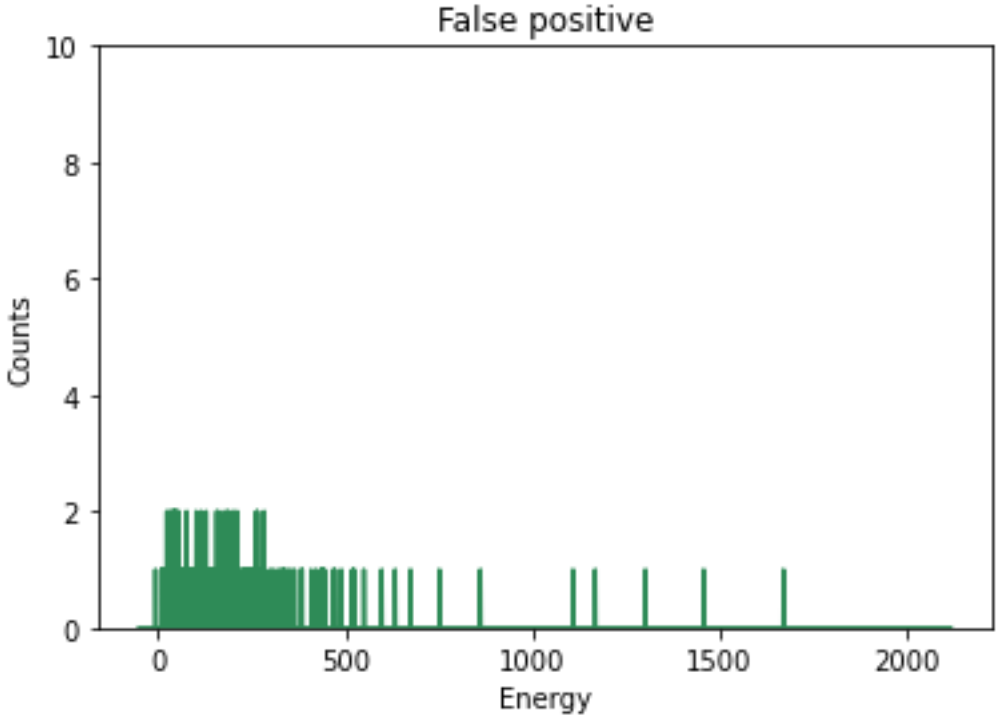
```
       0    1
0    0.0  1.0
1    0.0  1.0
2    1.0  0.0
3    0.0  1.0
4    0.0  1.0
..   ...  ...
645  0.0  1.0
646  1.0  0.0
647  0.0  1.0
```

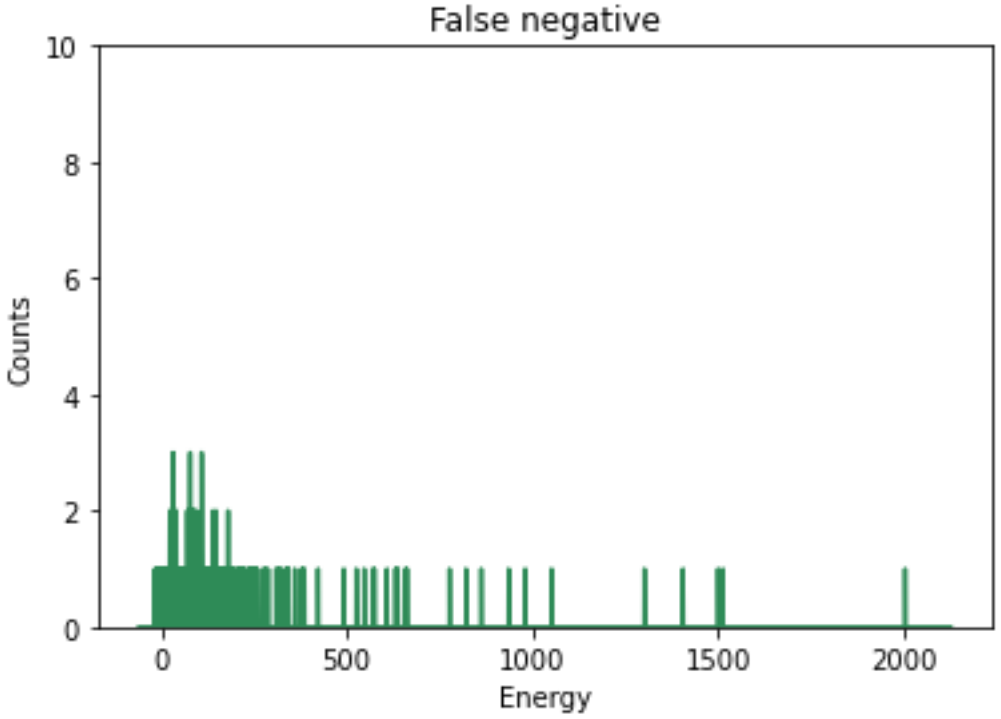
```
648  0.0  1.0
649  1.0  0.0

[650 rows x 2 columns]
       0    1
0    0.0  1.0
1    0.0  1.0
2    1.0  0.0
3    0.0  1.0
4    0.0  1.0
..   ...  ...
645  0.0  1.0
646  1.0  0.0
647  0.0  1.0
648  0.0  1.0
649  1.0  0.0

[650 rows x 2 columns]
<ipython-input-67-af4579672c1c>:42: RuntimeWarning: More than 20 figures have been opened. Fig
ures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until expl
icitly closed and may consume too much memory. (To control this warning, see the rcParam `figu
re.max_open_warning`).
  plt.figure(i+1)
```

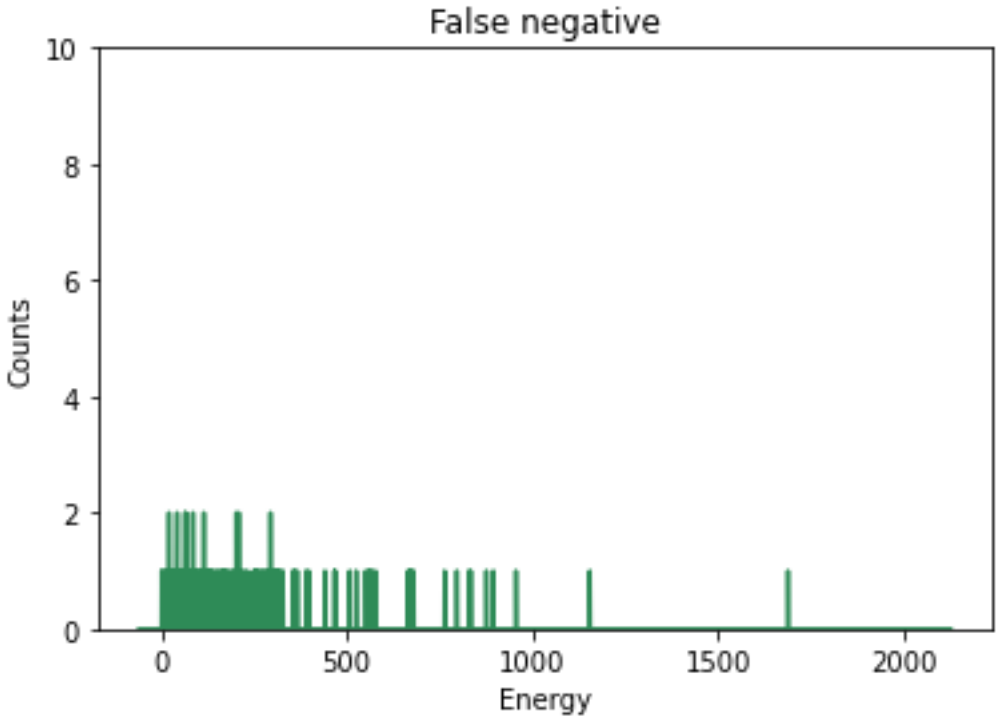


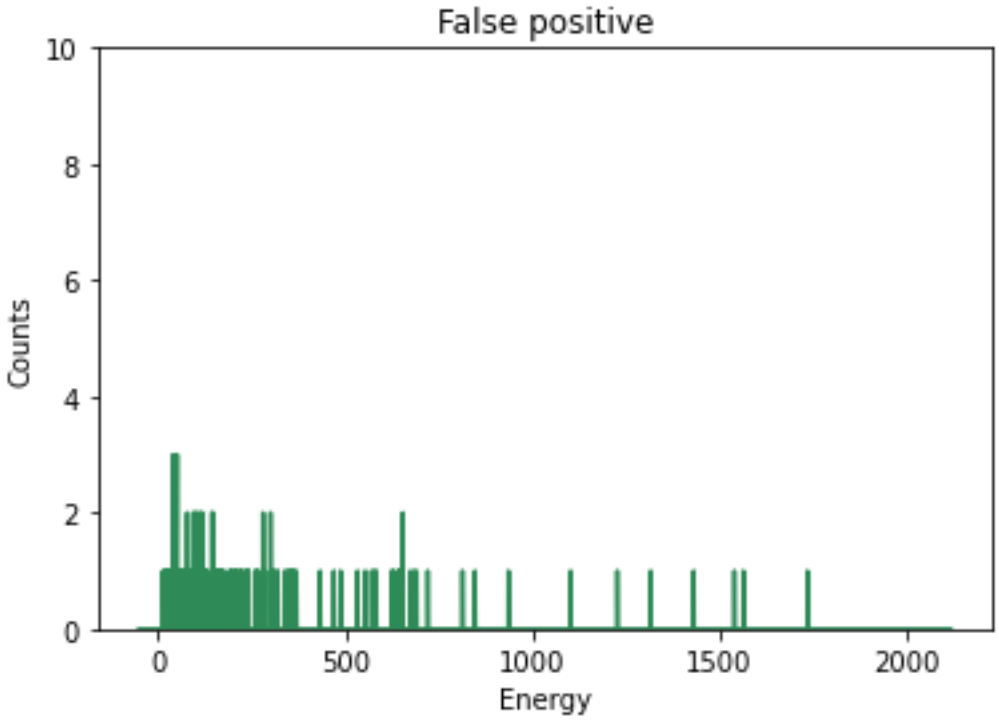

False negative
Counts
Energy
0
2
4
6
8
10
0
500
1000
1500
2000

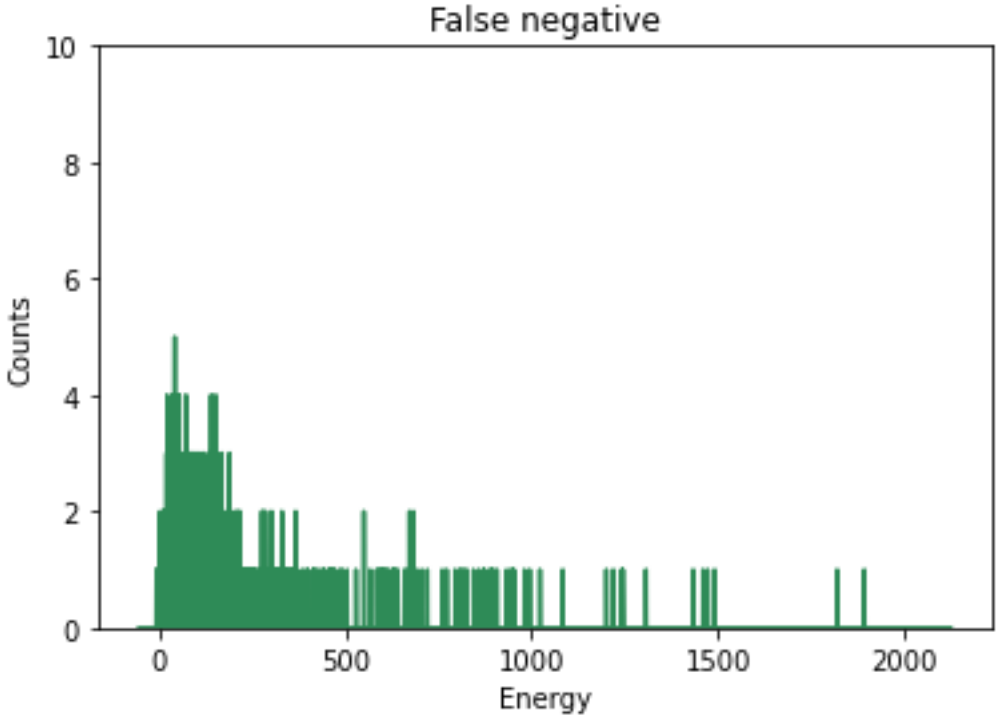

False negative
Counts
Energy
0
2
4
6
8
10
0
500
1000
1500
2000

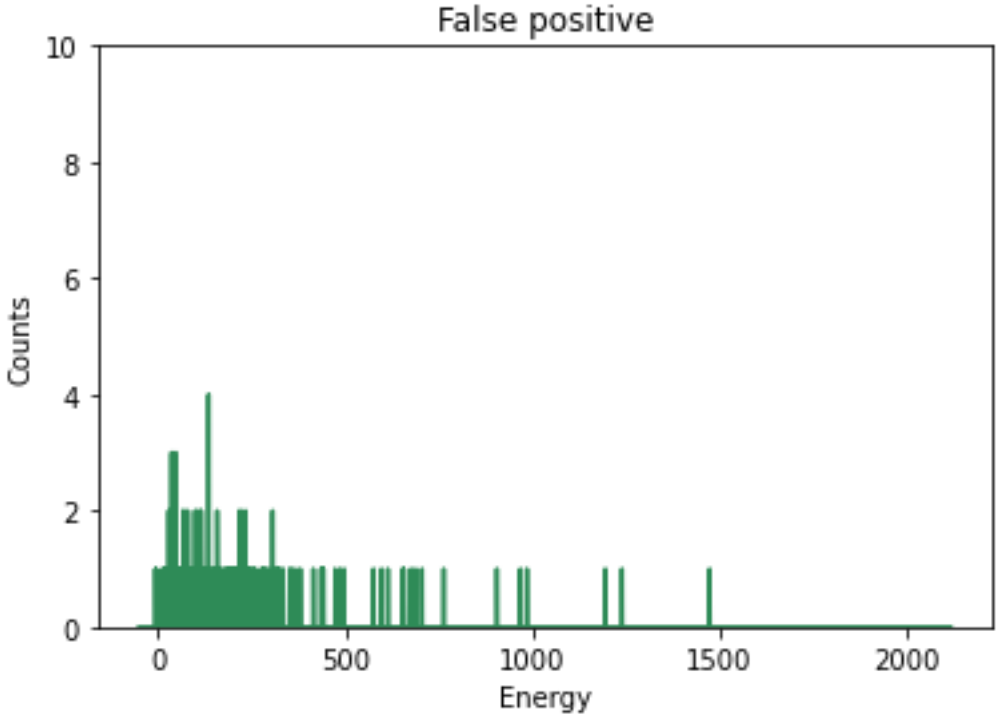

False positive
Counts
Energy
0
500
1000
1500
2000
0
2
4
6
8
10

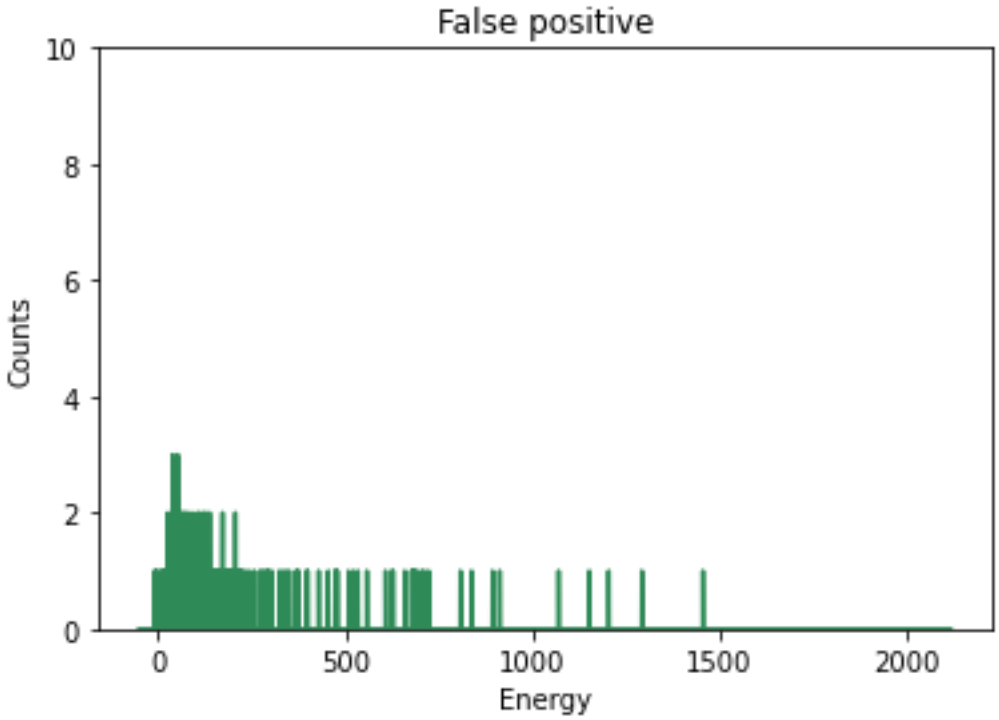

False negative
Counts
Energy
0
500
1000
1500
2000
0
2
4
6
8
10

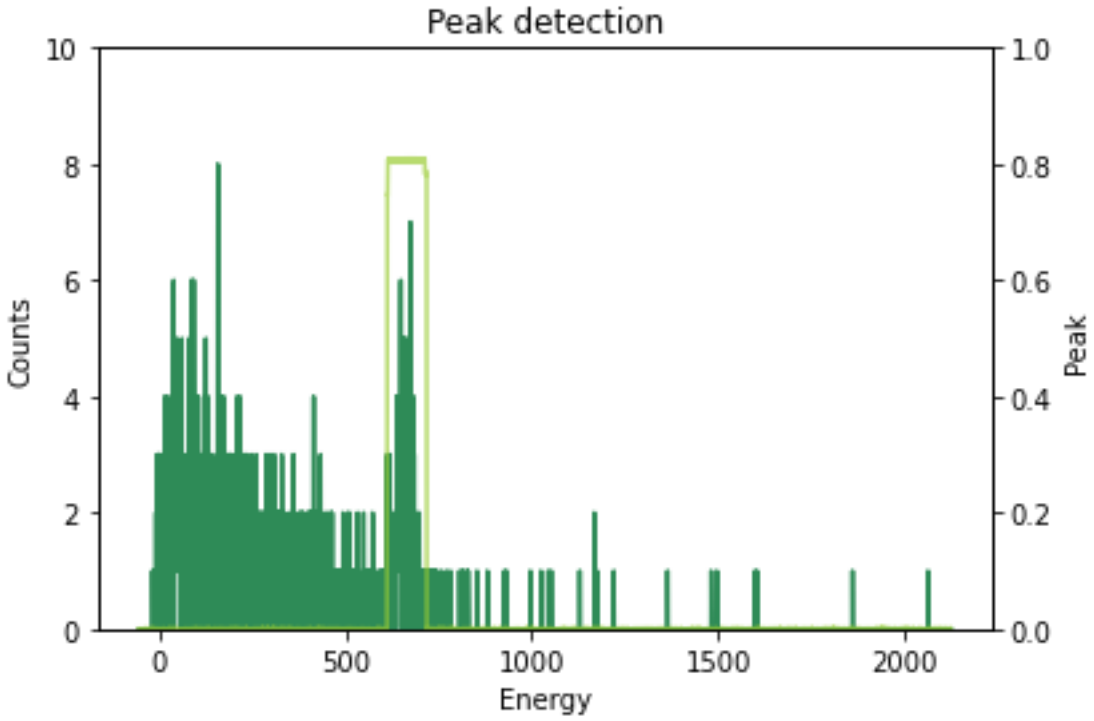

False negative
Counts
Energy
0
500
1000
1500
2000
0
2
4
6
8
10

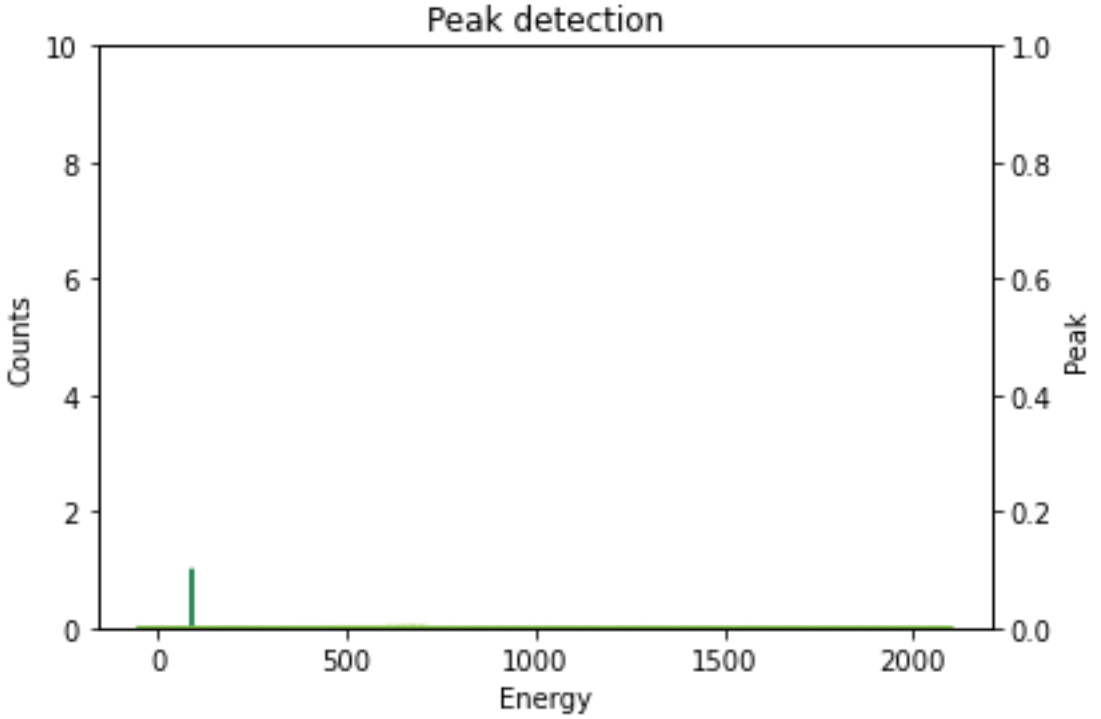

False positive
Counts
Energy
0
500
1000
1500
2000
0
2
4
6
8
10

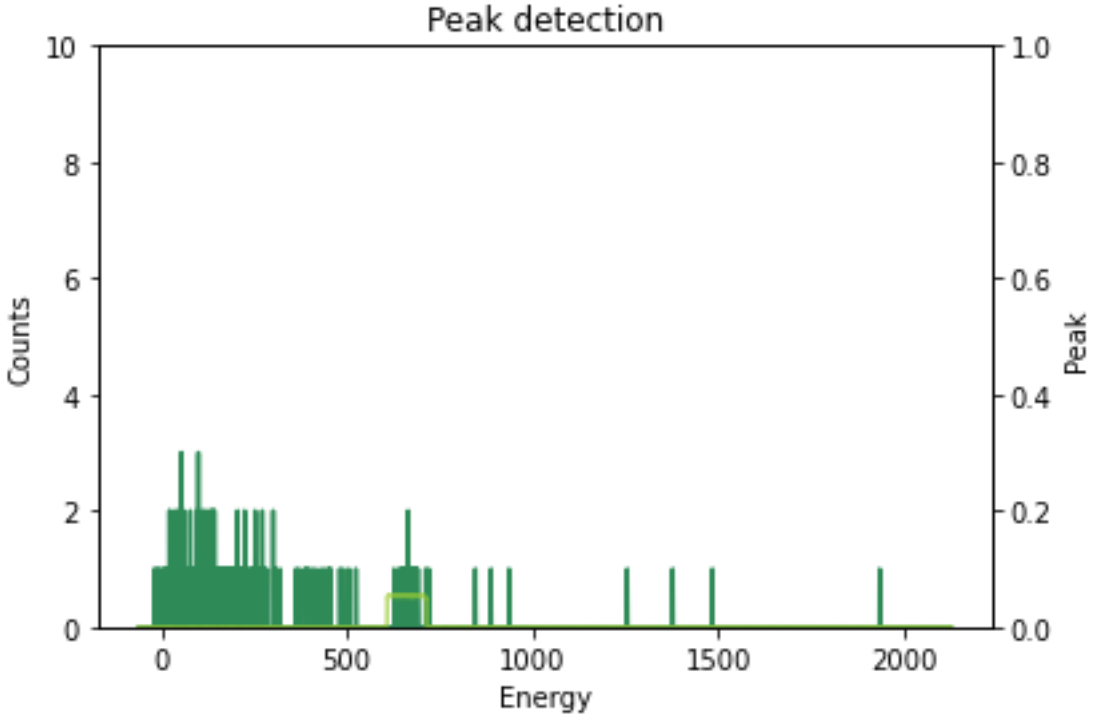

False negative
Counts
Energy
0
2
4
6
8
10
0
500
1000
1500
2000

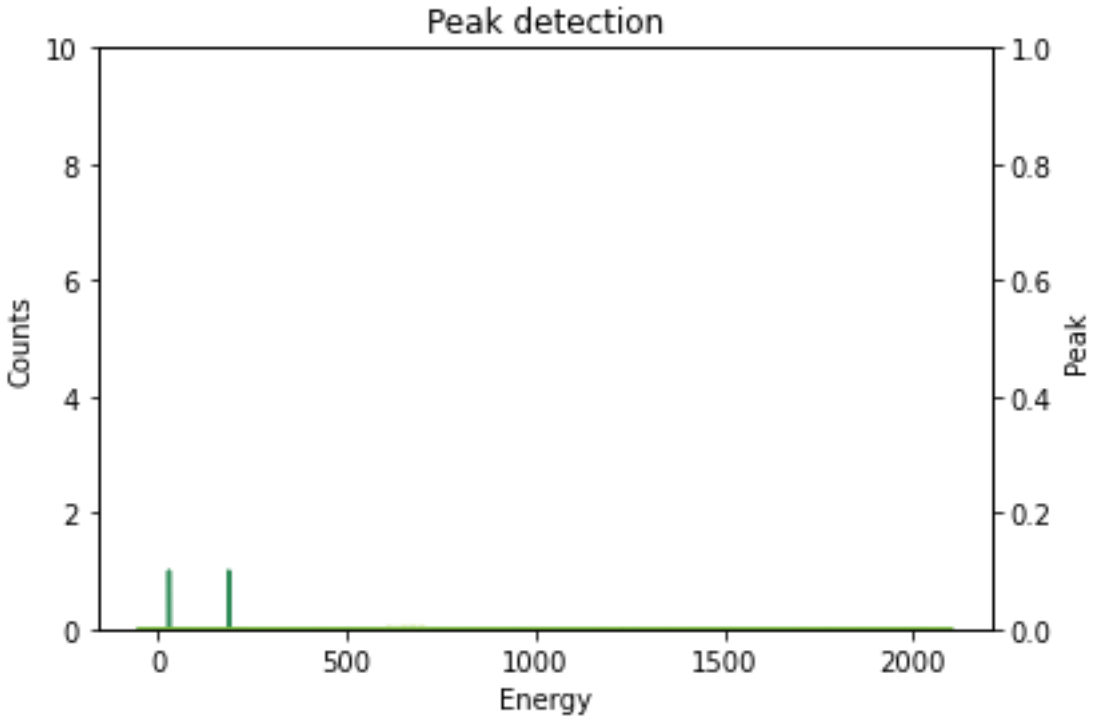

False positive
Counts
Energy
0
2
4
6
8
10
0
500
1000
1500
2000

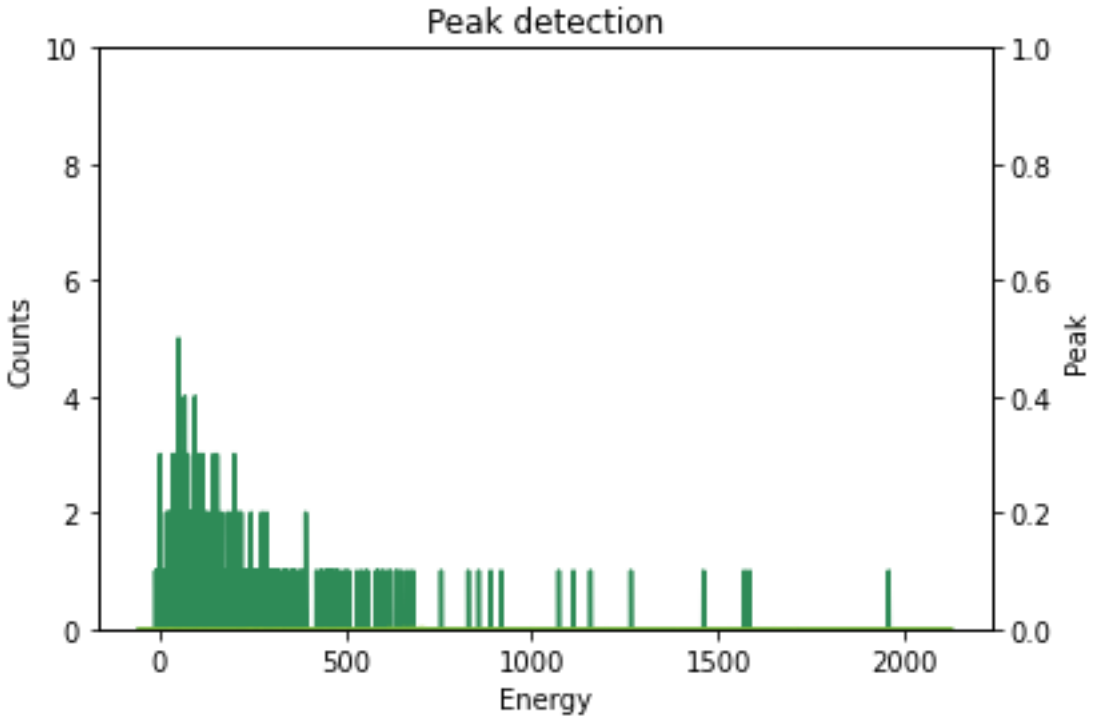

False positive
Counts
Energy
0
2
4
6
8
10
0
500
1000
1500
2000

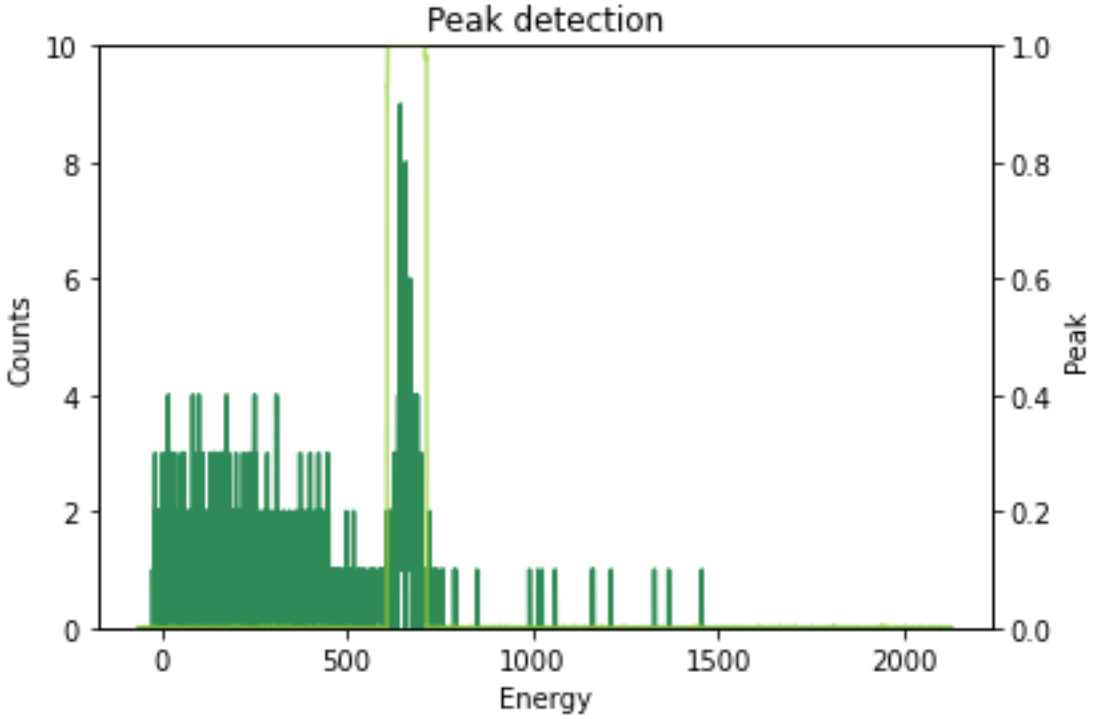

Peak detection
Counts
Peak
Energy
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
2000

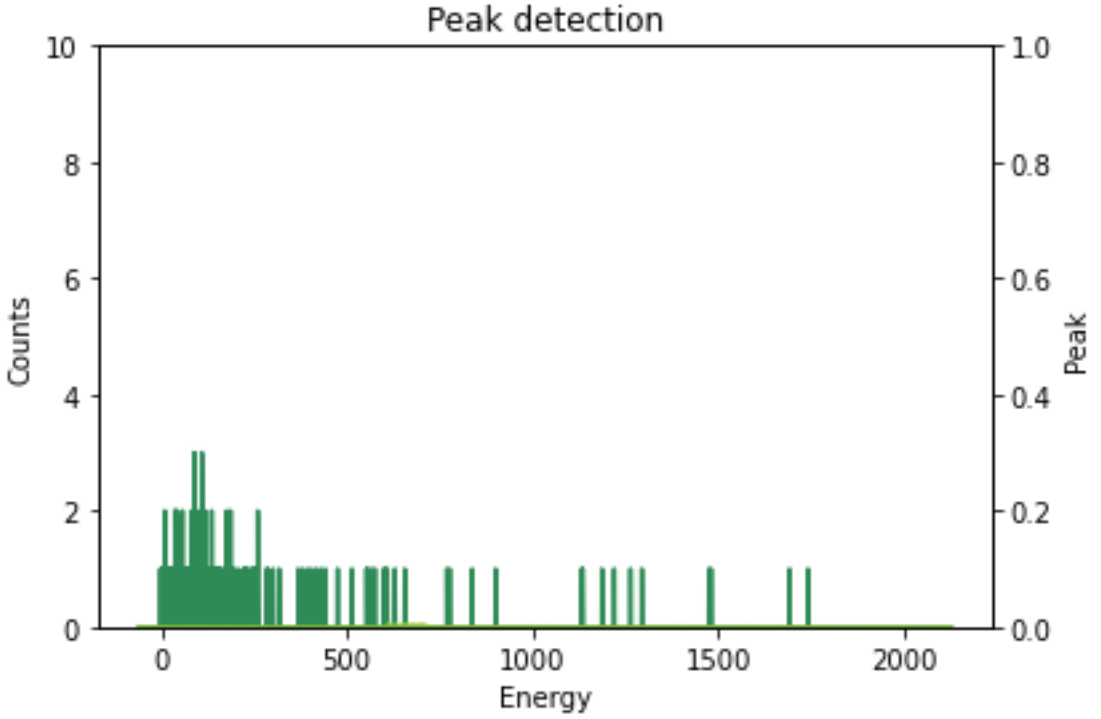

Peak detection
Counts
Peak
Energy
0
500
1000
1500
2000
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0

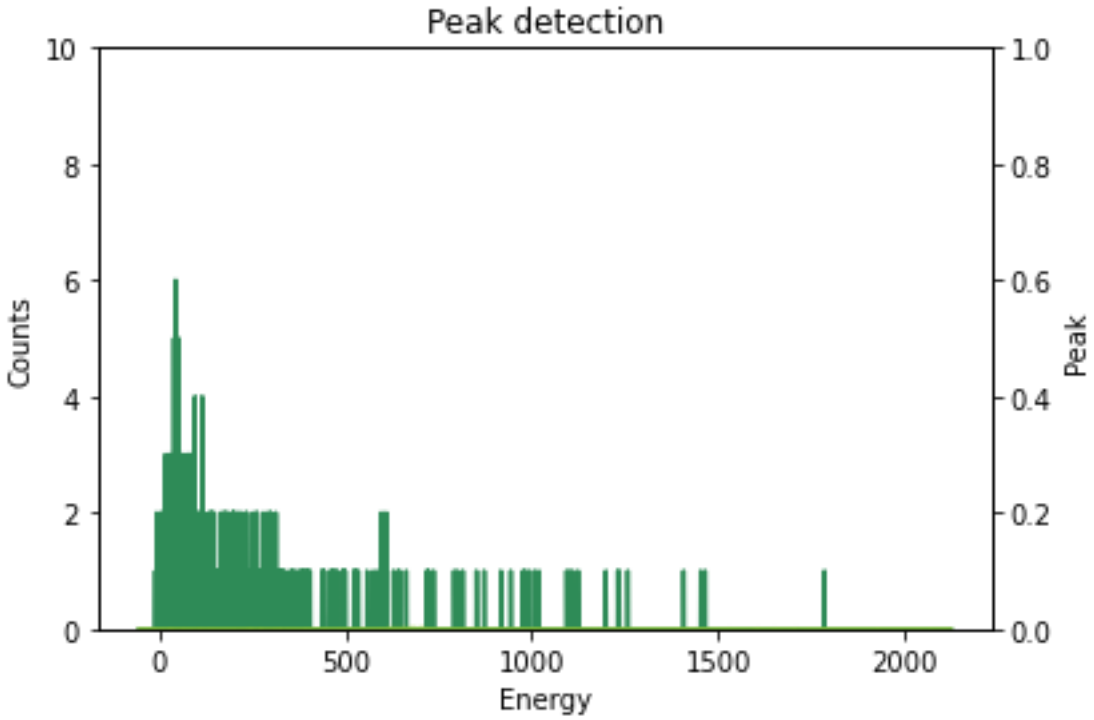

Peak detection
Counts
Peak
Energy
0
500
1000
1500
2000
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0

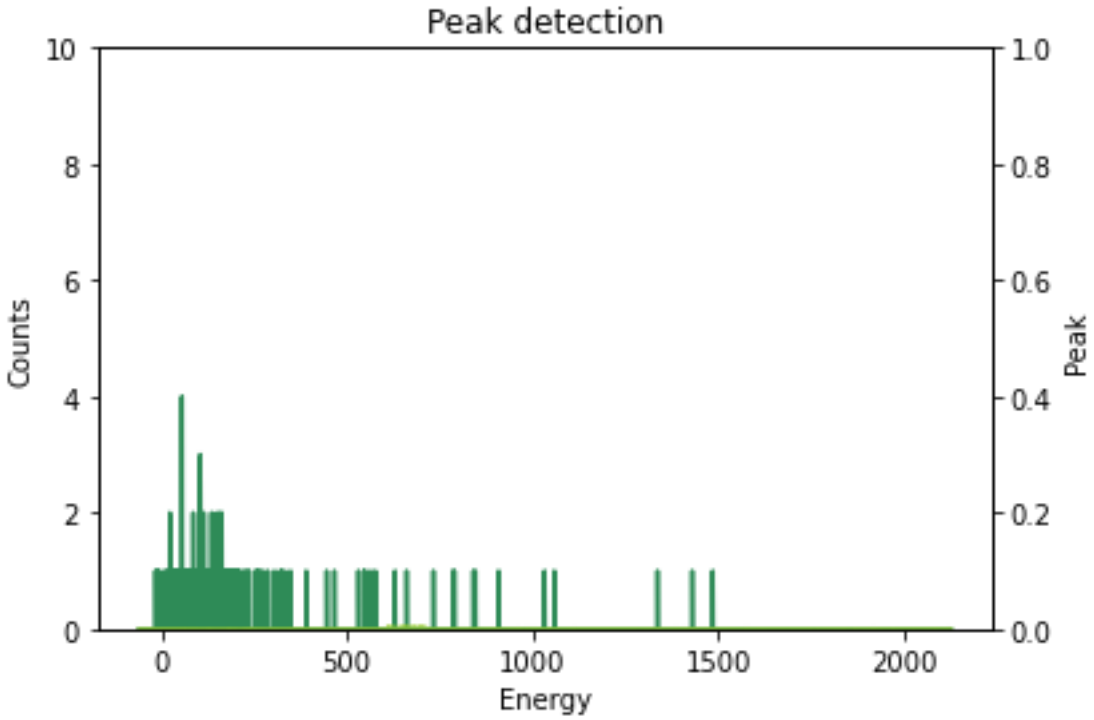

Peak detection
Counts
Peak
Energy
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
2000

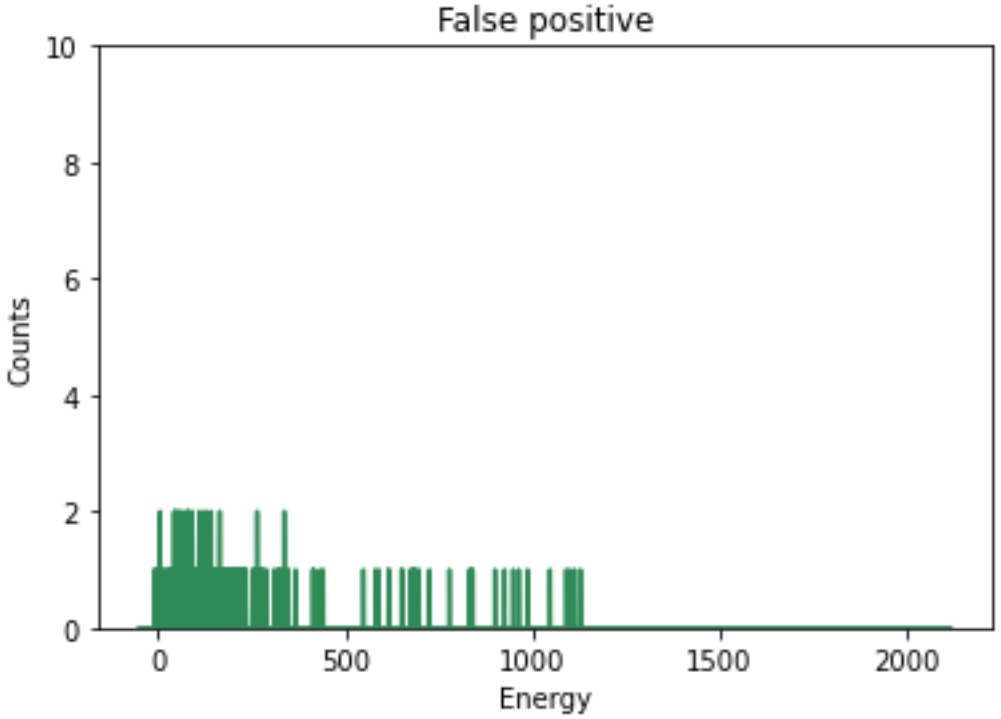

Peak detection
Counts
Peak
Energy
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
2000

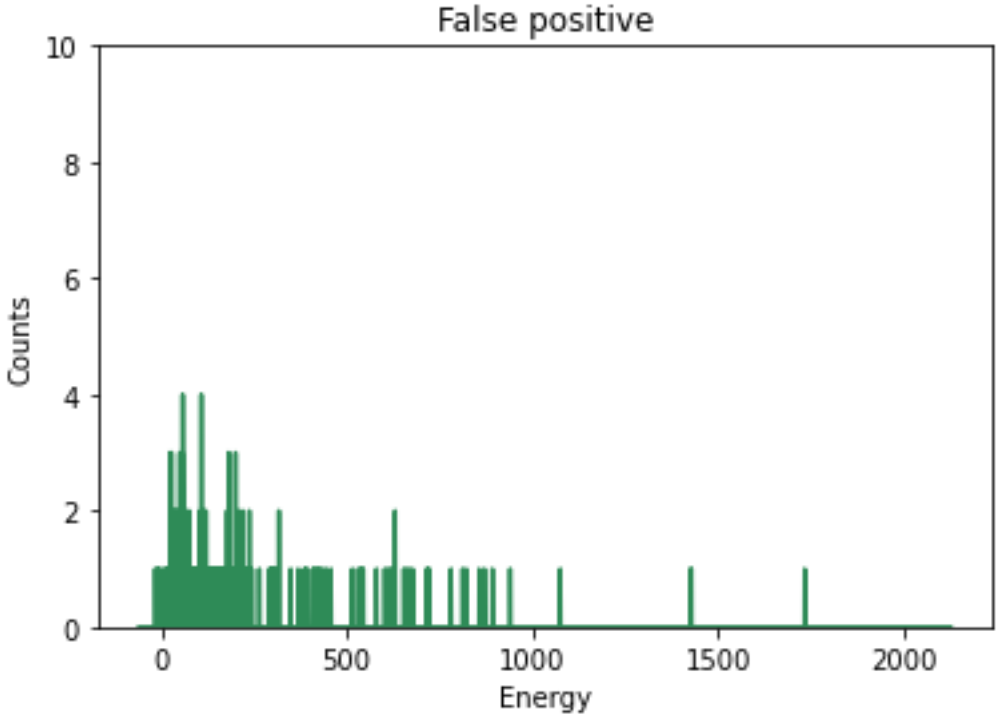

Peak detection
Counts
Peak
Energy
0
500
1000
1500
2000
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0

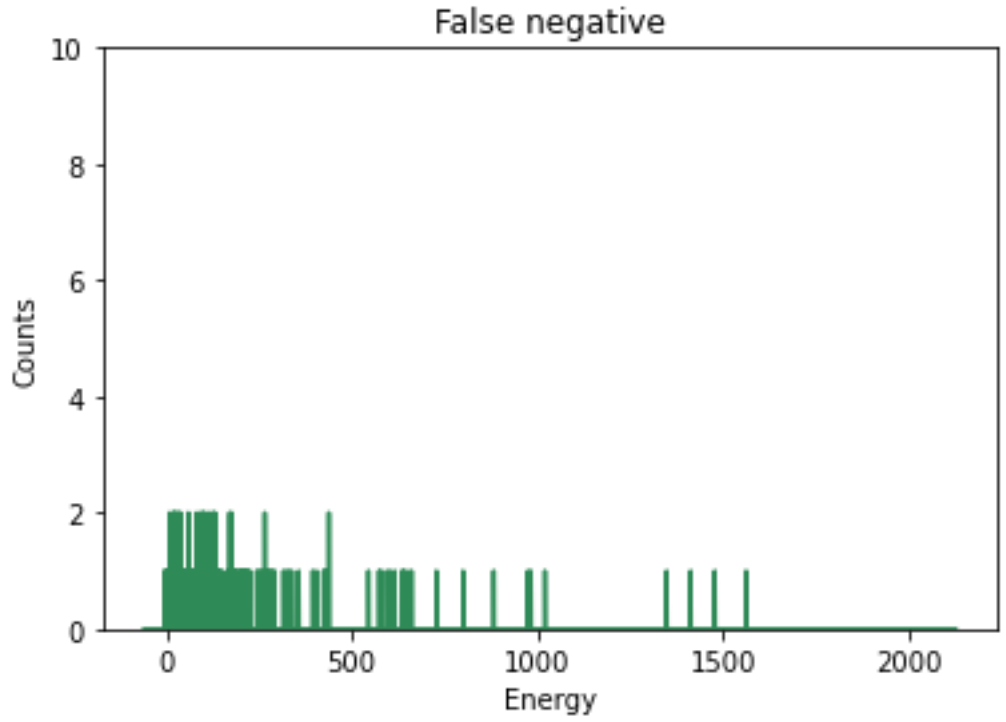

Peak detection
Counts
Peak
Energy
0
500
1000
1500
2000
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0


Peak detection
Counts
Peak
Energy
0
500
1000
1500
2000
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0


Peak detection
Counts
Peak
Energy
0
500
1000
1500
2000
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0


False positive
Counts
Energy
0
2
4
6
8
10
0
500
1000
1500
2000


False positive
Counts
Energy
0
2
4
6
8
10
0
500
1000
1500
2000



In [68]:

```
bench = open("Validation.pkl",'rb')
bench = pickle.load(bench,encoding='bytes')
multispec = open('MultiData_standard.pkl','rb')
multispec = pickle.load(multispec, encoding = 'bytes')


HOB_bench = pd.DataFrame()
ENE_bench = pd.DataFrame()
benchmarks = []

Multi_ENE = pd.DataFrame({'Tijd':[], 'Afstand':[], 'Hoek':[], 'Prediction':[]})
Multi_HOB = pd.DataFrame({'Tijd':[],'Afstand':[], 'Hoek':[], 'Prediction':[]})


for i in range(0,len(bench)):
    lijn = bench.iloc[i]
    multi_lijn = multispec.iloc[i]
    if(lijn['Locatie']=="HOB"):
        HOB_bench = HOB_bench.append(lijn,ignore_index = True)
        Multi_HOB = Multi_HOB.append({'Tijd': lijn['RealTime'], 'Afstand': lijn['Afstand'],
'Hoek' : lijn['Hoek'],  'Prediction': multi_lijn['Prediction']}, ignore_index = True)


    elif(lijn['Locatie']=='ENE'):
        ENE_bench = ENE_bench.append(lijn,ignore_index = True)
        Multi_ENE = Multi_ENE.append({'Tijd': lijn['RealTime'], 'Afstand': lijn['Afstand'],
'Hoek' : lijn['Hoek'],  'Prediction': multi_lijn['Prediction']}, ignore_index = True)


benchmarks.append(HOB_bench)
benchmarks.append(ENE_bench)

tijd_bench = pd.DataFrame()
hoogte_bench = pd.DataFrame()
hoek_bench = pd.DataFrame()
```

```
tijd2_bench = pd.DataFrame()
hoogte2_bench = pd.DataFrame()
hoek2_bench = pd.DataFrame()

tijd3_bench = pd.DataFrame()
hoogte3_bench = pd.DataFrame()
hoek3_bench = pd.DataFrame()

tijd4_bench = pd.DataFrame()
hoogte4_bench = pd.DataFrame()
hoek4_bench = pd.DataFrame()



val = bench.copy()
def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        height = height.round()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, 10),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom',
                   fontsize =10)



for i in range (0, len(HOB_bench)):
    lijn = HOB_bench.iloc[i]
    multi_lijn = Multi_HOB.iloc[i]
    if(lijn['Afstand'] == 9):
        hoek_bench = hoek_bench.append(lijn, ignore_index = True)
        hoek3_bench = hoek3_bench.append(multi_lijn, ignore_index = True)

    elif(lijn['Hoek'] == 0 and lijn['Afstand']==0):

        a = lijn['RealTime'].copy()
        lijn.loc['RealTime'] = (np.round(a))
        tijd_bench = tijd_bench.append(lijn, ignore_index = True)
        tijd3_bench = tijd3_bench.append(multi_lijn, ignore_index = True)

    elif(lijn['Afstand']!=0 and lijn['Afstand']!=9):
        hoogte_bench = hoogte_bench.append(lijn, ignore_index = True)
        hoogte3_bench = hoogte3_bench.append(multi_lijn, ignore_index = True)

for i in range (0, len(ENE_bench)):
    lijn = ENE_bench.iloc[i]
    multi_lijn = Multi_ENE.iloc[i]
    if(lijn['Afstand'] == 9):
        hoek2_bench = hoek2_bench.append(lijn, ignore_index = True)
        hoek4_bench = hoek4_bench.append(multi_lijn, ignore_index = True)

    elif(lijn['Hoek'] == 0 and lijn['Afstand']==0):
```

```
        a = lijn['RealTime'].copy()
        lijn.loc['RealTime'] = (np.round(a))
        tijd2_bench = tijd2_bench.append(lijn, ignore_index = True)
        tijd4_bench = tijd4_bench.append(multi_lijn, ignore_index = True)


    elif(lijn['Afstand']!=0 and lijn['Afstand']!=9):
        hoogte2_bench = hoogte2_bench.append(lijn, ignore_index = True)
        hoogte4_bench = hoogte4_bench.append(multi_lijn, ignore_index = True)


#Tijdbenchmark

## Bewerken van de inputdata om voorspellingen te kunnen doen
## Deze data is voor de metingen van HOB

tijd_bench_spectrum = pd.DataFrame()
tijd_bench_energie = []
tijd_bench_counts = []


tijd_bench_Y= tijd_bench.pop("Bron")
tijd_vertaling = pd.DataFrame({"0":[],
                               "1":[]})
for i in range(0,len(tijd_bench_Y)):
    Voc = tijd_bench_Y[i]
    if(Voc == 'background'):
        Voc = 'Background'

    a = np.where(vocabulair == Voc)
    a = int(a[0])
    if(a == 1):
        Cs = 1
        Bg = 0
    else:
        Cs = 0
        Bg = 1
    tijd_vertaling = tijd_vertaling.append({
          "0": Bg,
          "1": Cs}, ignore_index = True)


tijd_vertaling = np.array(tijd_vertaling)
tijd_bench_X = tijd_bench.pop('Counts')
tijd_bench_Z=tijd_bench.pop('Energie')
for i in range(0, len(tijd_bench)):



  a = np.array(tijd_bench_Z[i])
  b = np.array(tijd_bench_X[i])


  tijd_bench_energie.append(a)
  tijd_bench_counts.append(b)

tijd_bench_spectrum['Counts'] = tijd_bench_counts
tijd_bench_spectrum['Energie']= tijd_bench_energie
```

```
tijd_bench_spectrum = tf.convert_to_tensor((tijd_bench_spectrum.to_numpy()).tolist())
predictionT, peakT = model1.predict(tijd_bench_spectrum)
tijd_voorspellingen1 = np.round(predictionT)
```

```
## Deze data is voor de metingen van ENE

tijd2_bench_spectrum = pd.DataFrame()
tijd2_bench_energie = []
tijd2_bench_counts = []


tijd2_bench_Y= tijd2_bench.pop("Bron")
tijd2_vertaling = pd.DataFrame({"0":[],
                                "1":[]})
for i in range(0,len(tijd2_bench_Y)):
    Voc = tijd2_bench_Y[i]
    if(Voc == 'background'):
        Voc = 'Background'

    a = np.where(vocabulair == Voc)
    a = int(a[0])
    if(a == 1):
        Cs = 1
        Bg = 0
    else:
        Cs = 0
        Bg = 1
    tijd2_vertaling = tijd2_vertaling.append({
          "0": Bg,
          "1": Cs}, ignore_index = True)


tijd2_vertaling = np.array(tijd2_vertaling)
tijd2_bench_X = tijd2_bench.pop('Counts')
tijd2_bench_Z=tijd2_bench.pop('Energie')
for i in range(0, len(tijd2_bench)):



  a = np.array(tijd2_bench_Z[i])
  b = np.array(tijd2_bench_X[i])


  tijd2_bench_energie.append(a)
  tijd2_bench_counts.append(b)

tijd2_bench_spectrum['Counts'] = tijd2_bench_counts
```

```
tijd2_bench_spectrum['Energie']= tijd2_bench_energie



tijd2_bench_spectrum = tf.convert_to_tensor((tijd2_bench_spectrum.to_numpy()).tolist())
predictionT2, peakT2 = model1.predict(tijd2_bench_spectrum)
tijd_voorspellingen2 = np.round(predictionT2)


#Vanaf hier werken om de twee te quantificeren
tijd_correct1 = []
tijd_fout1 = []
tijd_correct2 = []
tijd_fout2 = []

tijd_correct3 = []
tijd_fout3 = []
tijd_correct4 = []
tijd_fout4 = []


FP1 = []
FP2 = []
FN1 = []
FN2 = []

print(tijd3_bench)

for i in range(0, len(tijd_bench)):
    if(tijd_voorspellingen1[i,0] ==tijd_vertaling[i,0]):
        tijd_correct1.append(tijd_bench['RealTime'].iloc[i])
    else:
        tijd_fout1.append(tijd_bench['RealTime'].iloc[i])
        if(tijd_voorspellingen1[i,0] == 0):
            FP1.append(tijd_bench['RealTime'].iloc[i])
        else:
            FN1.append(tijd_bench['RealTime'].iloc[i])


    if(int(tijd3_bench['Prediction'].iloc[i]) == tijd_vertaling[i,1]):
        tijd_correct3.append(np.round(tijd3_bench['Tijd'].iloc[i]))

    else:
        tijd_fout3.append(np.round(tijd3_bench['Tijd'].iloc[i]))




for i in range(0,len(tijd2_bench)):

    if(tijd_voorspellingen2[i,0] ==tijd2_vertaling[i,0]):
        tijd_correct2.append(tijd2_bench['RealTime'].iloc[i])
    else:
        tijd_fout2.append(tijd2_bench['RealTime'].iloc[i])
        if(tijd_voorspellingen2[i,0] == 0):
            FP2.append(tijd2_bench['RealTime'].iloc[i])
        else:
            FN2.append(tijd2_bench['RealTime'].iloc[i])
```

```
    if(int(tijd4_bench['Prediction'].iloc[i]) == tijd2_vertaling[i,1]):
        tijd_correct4.append(np.round(tijd4_bench['Tijd'].iloc[i]))
    else:
        tijd_fout4.append(np.round(tijd4_bench['Tijd'].iloc[i]))

unique_tijd = tijd_bench['RealTime'].unique()
tijd_resultaat = pd.DataFrame({'tijd' : [],
                               'resultaat1' : [],
                               'resultaat2' :[],
                               'resultaat3':[],
                               'resultaat4':[],
                               'FP1':[],
                               'FN1':[],
                               'FP2':[],
                               'FN2':[]})


for tijd in unique_tijd:
    correct1 = 0
    fout1 = 0
    FalseP1 = 0
    FalseN1 = 0



    correct1 = tijd_correct1.count(tijd)
    fout1 = tijd_fout1.count(tijd)

    if not FP1:
        FalseP1 = 0
    else:
        FalseP1 = FP1.count(tijd)

    if not FN1:
        FalseN1 = 0
    else:
        FalseN1 = FN1.count(tijd)

    correct2 = 0
    fout2 = 0

    correct2 = tijd_correct2.count(tijd)
    fout2 = tijd_fout2.count(tijd)



    correct3 = 0
    fout3 = 0

    correct3 = tijd_correct3.count(tijd)
    fout3 = tijd_fout3.count(tijd)

    correct4 = 0
    fout4 = 0
```

```
    correct4 = tijd_correct4.count(tijd)
    fout4 = tijd_fout4.count(tijd)

    if not FP2:
        FalseP2 = 0
    else:
        FalseP2 = FP2.count(tijd)

    if not FN2:
        FalseN2 = 0
    else:
        FalseN2 = FN2.count(tijd)

    percentage1 = (correct1)/(correct1 + fout1) * 100
    percentage2 = (correct2)/(correct2 + fout2) * 100
    percentage3 = (correct3)/(correct3 + fout3) * 100
    percentage4 = (correct4)/(correct4 + fout4) * 100

    False_Negative1 = (FalseN1/(fout1+correct1))*100
    False_Negative2 = (FalseN2/(fout2+correct2)) * 100

    False_Positive1 = (FalseP1/(fout1+ correct1)) * 100
    False_Positive2 = (FalseP2/(fout2+correct2)) * 100


    tijd_resultaat = tijd_resultaat.append({
        'tijd': tijd,
        'resultaat1' : percentage1,
        'resultaat2': percentage2,
        'resultaat3':percentage3,
        'resultaat4':percentage4,
        'FP1': False_Positive1,
        'FN1' : False_Negative1,
        'FP2': False_Positive2,
        'FN2': False_Negative2
        }, ignore_index = True)

tijd_resultaat = tijd_resultaat.sort_values(by=['tijd'])



fig,(ax) = plt.subplots(dpi = 300)
mod1 = ax.bar(tijd_resultaat['tijd']-0.175, tijd_resultaat['resultaat1'], width =0.35, label =
'DenseNet_HOB', color = 'yellowgreen' )
mod3 = ax.bar(tijd_resultaat['tijd']+0.175, tijd_resultaat['resultaat3'], width =0.35, label =
'MultiSpect_HOB', color = 'olivedrab' )
autolabel(mod1)
autolabel(mod3)
ax.set_xlabel('Time in seconds')
ax.set_ylabel('Accuracy')
ax.legend()
ax.set_title('Benchmark measurement time for average background')
plt.show()


fig,(ax) = plt.subplots(dpi = 300)
mod2 = ax.bar(tijd_resultaat['tijd']-0.175, tijd_resultaat['resultaat2'], width =0.35, label =
'DenseNet_ENE', color = 'yellowgreen' )
```

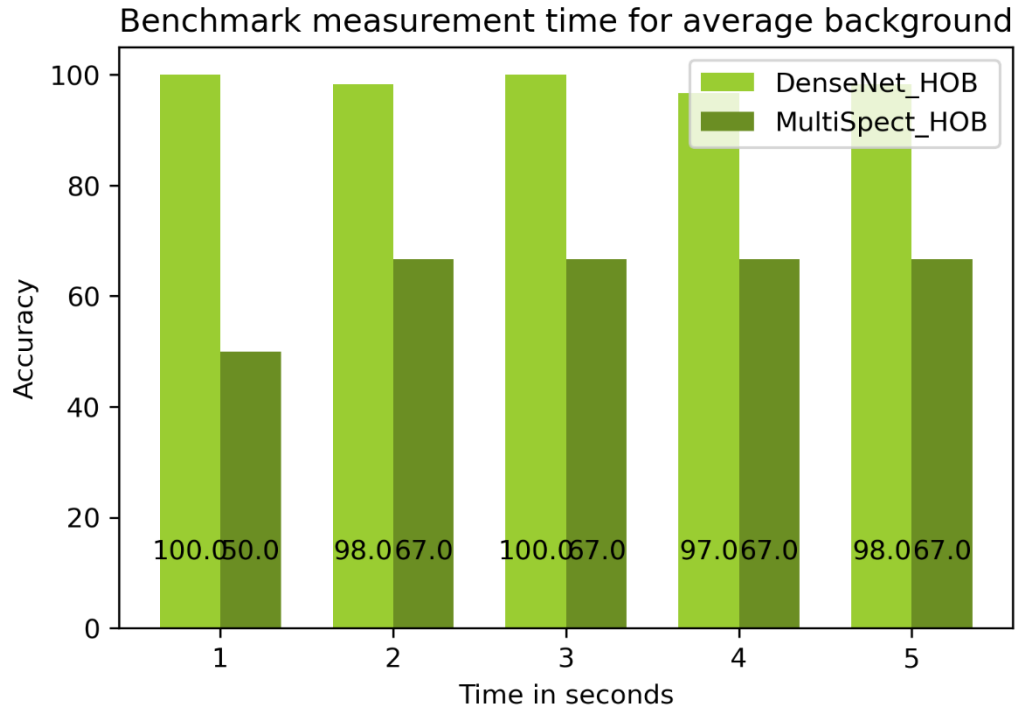
```
mod4 = ax.bar(tijd_resultaat['tijd']+0.175, tijd_resultaat['resultaat4'], width =0.35, label =
'MultiSpect_ENE', color = 'olivedrab' )
autolabel(mod4)
autolabel(mod2)
ax.set_xlabel('Time in seconds')
ax.set_ylabel('Accuracy')
ax.legend()
ax.set_title('Benchmark measurement time for high background')
plt.show()


print(tijd_resultaat)

##############################################
##############################################

########################################
########################################
#hoogtenebenchmark
hoogte_bench_spectrum = pd.DataFrame()
hoogte_bench_energie = []
hoogte_bench_counts = []


hoogte_bench_Y= hoogte_bench.pop("Bron")
hoogte_vertaling = pd.DataFrame({"0":[],
                                 "1":[]})
for i in range(0,len(hoogte_bench_Y)):
    Voc = hoogte_bench_Y[i]
    if(Voc == 'background'):
        Voc = 'Background'

    a = np.where(vocabulair == Voc)
    a = int(a[0])
    if(a == 1):
        Cs = 1
        Bg = 0
    else:
        Cs = 0
        Bg = 1
    hoogte_vertaling = hoogte_vertaling.append({
          "0": Bg,
          "1": Cs}, ignore_index = True)


hoogte_vertaling = np.array(hoogte_vertaling)
hoogte_bench_X = hoogte_bench.pop('Counts')
hoogte_bench_Z=hoogte_bench.pop('Energie')
for i in range(0, len(hoogte_bench)):

  a = np.array(hoogte_bench_Z[i])
  b = np.array(hoogte_bench_X[i])

  hoogte_bench_energie.append(a)
  hoogte_bench_counts.append(b)
```

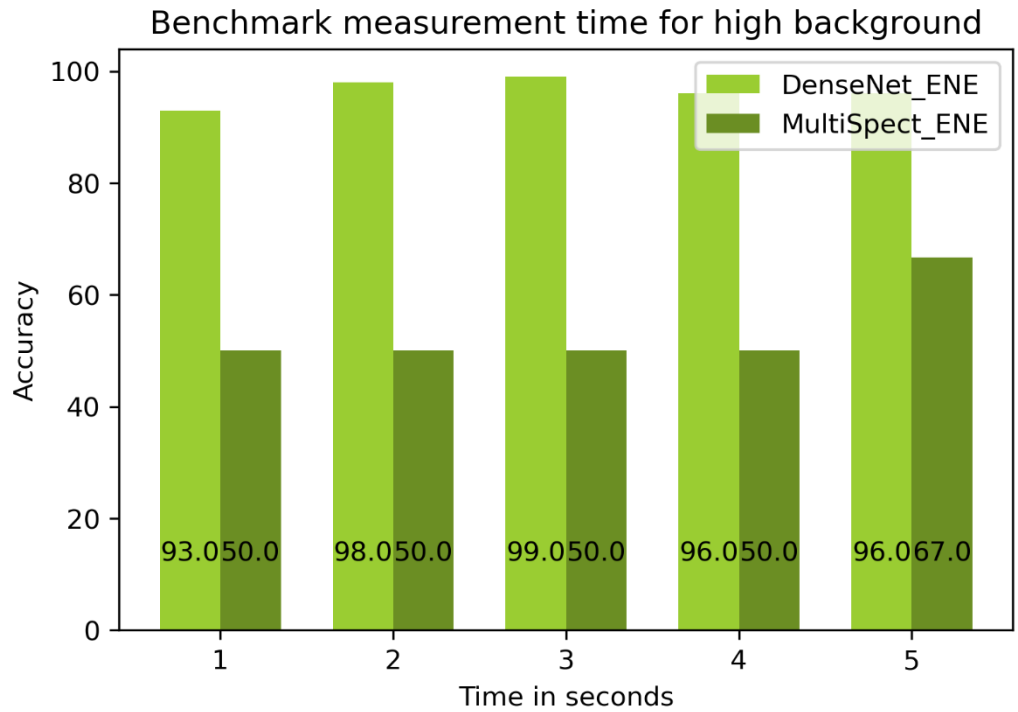
```
hoogte_bench_spectrum['Counts'] = hoogte_bench_counts
hoogte_bench_spectrum['Energie']= hoogte_bench_energie
hoogte_bench_spectrum = tf.convert_to_tensor((hoogte_bench_spectrum.to_numpy()).tolist())

predictionH, peakH = model1.predict(hoogte_bench_spectrum)
hoogte_voorspellingen1 = np.round(predictionH)

#ENE

hoogte_bench_spectrum2 = pd.DataFrame()
hoogte_bench_energie2 = []
hoogte_bench_counts2 = []


hoogte_bench_Y2= hoogte2_bench.pop("Bron")
hoogte_vertaling2 = pd.DataFrame({"0":[],"1":[]})
for i in range(0,len(hoogte_bench_Y2)):
    Voc = hoogte_bench_Y2[i]
    if(Voc == 'background'):
        Voc = 'Background'

    a = np.where(vocabulair == Voc)
    a = int(a[0])
    if(a == 1):
        Cs = 1
        Bg = 0
    else:
        Cs = 0
        Bg = 1
    hoogte_vertaling2 = hoogte_vertaling2.append({ "0": Bg, "1": Cs}, ignore_index = True)


hoogte_vertaling2 = np.array(hoogte_vertaling2)
hoogte_bench_X2 = hoogte2_bench.pop('Counts')
hoogte_bench_Z2=hoogte2_bench.pop('Energie')
for i in range(0, len(hoogte2_bench)):

  a = np.array(hoogte_bench_Z2[i])
  b = np.array(hoogte_bench_X2[i])

  hoogte_bench_energie2.append(a)
  hoogte_bench_counts2.append(b)

hoogte_bench_spectrum2['Counts'] = hoogte_bench_counts2
hoogte_bench_spectrum2['Energie']= hoogte_bench_energie2
hoogte_bench_spectrum2 = tf.convert_to_tensor((hoogte_bench_spectrum2.to_numpy()).tolist())

predictionH2, peakH2 = model1.predict(hoogte_bench_spectrum2)
hoogte_voorspellingen2 = np.round(predictionH2)

hoogte_correct1 = []
hoogte_fout1 = []

hoogte_correct2 = []
hoogte_fout2 = []

hoogte_correct3 = []
hoogte_fout3 = []
```

```
hoogte_correct4 = []
hoogte_fout4 = []



for i in range(0, len(hoogte_bench)):
    if(hoogte_voorspellingen1[i,0] ==hoogte_vertaling[i,0]):
        hoogte_correct1.append(hoogte_bench['Afstand'].iloc[i])
    else:
        hoogte_fout1.append(hoogte_bench['Afstand'].iloc[i])
    if(int(hoogte3_bench['Prediction'].iloc[i]) == hoogte_vertaling[i,1]):
        hoogte_correct3.append(hoogte3_bench['Afstand'].iloc[i])
    else:
        hoogte_fout3.append(hoogte3_bench['Afstand'].iloc[i])

for i in range(0, len(hoogte2_bench)):
    if(hoogte_voorspellingen2[i,0] ==hoogte_vertaling2[i,0]):
        hoogte_correct2.append(hoogte2_bench['Afstand'].iloc[i])
    else:
        hoogte_fout2.append(hoogte2_bench['Afstand'].iloc[i])

    if(int(hoogte4_bench['Prediction'].iloc[i]) == hoogte_vertaling2[i,1]):
        hoogte_correct4.append(hoogte4_bench['Afstand'].iloc[i])
    else:
        hoogte_fout4.append(hoogte4_bench['Afstand'].iloc[i])


unique_hoogte = hoogte_bench['Afstand'].unique()
hoogte_resultaat = pd.DataFrame({'hoogte' : [], 'resultaat1' : [], 'resultaat2':[],
'resultaat3':[], 'resultaat4':[]})

for hoogte in unique_hoogte:
    correct1 = 0
    correct2 =0
    correct3 = 0
    correct4 = 0
    fout1 = 0
    fout2 = 0
    fout3 = 0
    fout4 = 0



    correct1 = hoogte_correct1.count(hoogte)
    fout1 = hoogte_fout1.count(hoogte)
    percentage1 = (correct1)/(correct1 + fout1) * 100

    correct2 = hoogte_correct2.count(hoogte)
    fout2 = hoogte_fout2.count(hoogte)
    percentage2 = (correct2)/(correct2 + fout2) * 100

    correct3 = hoogte_correct3.count(hoogte)
    fout3 = hoogte_fout3.count(hoogte)
    percentage3 = (correct3)/(correct3 + fout3) * 100

    correct4 = hoogte_correct4.count(hoogte)
```

```
    fout4 = hoogte_fout4.count(hoogte)
    percentage4 = (correct4)/(correct4 + fout4) * 100



    hoogte_resultaat = hoogte_resultaat.append({
        'hoogte': hoogte,
        'resultaat1' : percentage1
        ,'resultaat2' : percentage2
        ,'resultaat3' : percentage3
        ,'resultaat4' : percentage4

        }, ignore_index = True)

hoogte_resultaat = hoogte_resultaat.sort_values(by=['hoogte'])


print(hoogte_resultaat)

fig,ax = plt.subplots(dpi = 300)
mod1 = ax.bar(hoogte_resultaat['hoogte']-1.75,hoogte_resultaat['resultaat1'] ,width =
3.5,label = 'DenseNet_HOB', color = 'yellowgreen')
mod3 = ax.bar(hoogte_resultaat['hoogte']+1.75,hoogte_resultaat['resultaat3'] ,width =
3.5,label = 'MultiSpect_HOB', color = 'olivedrab')
autolabel(mod1)
autolabel(mod3)
plt.xlabel('Distance in cm')
plt.ylabel('Accuracy')
ax.legend()
plt.title('Benchmark distance to detector with average background')
plt.show()



fig,ax = plt.subplots(dpi = 300)
mod2 = ax.bar(hoogte_resultaat['hoogte']-1.75,hoogte_resultaat['resultaat2'], width =3.5,label
= 'DenseNet_ENE', color = 'yellowgreen' )
mod4 = ax.bar(hoogte_resultaat['hoogte']+1.75,hoogte_resultaat['resultaat4'], width =3.5,label
= 'MultiSpect_ENE', color = 'olivedrab' )
autolabel(mod2)
autolabel(mod4)
plt.xlabel('Distance in cm')
plt.ylabel('Accuracy')
ax.legend()
plt.title('Benchmark distance to detector with high background')
plt.show()
C:\Users\Tones\anaconda3\lib\site-packages\pandas\core\indexing.py:1637: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
C:\Users\Tones\anaconda3\lib\site-packages\pandas\core\indexing.py:692: SettingWithCopyWarning
:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
  iloc._setitem_with_indexer(indexer, value, self.name)
     Afstand  Hoek  Prediction   Tijd
0        0.0   0.0         0.0  1.919
1        0.0   0.0         0.0  1.892
2        0.0   0.0         0.0  1.940
```

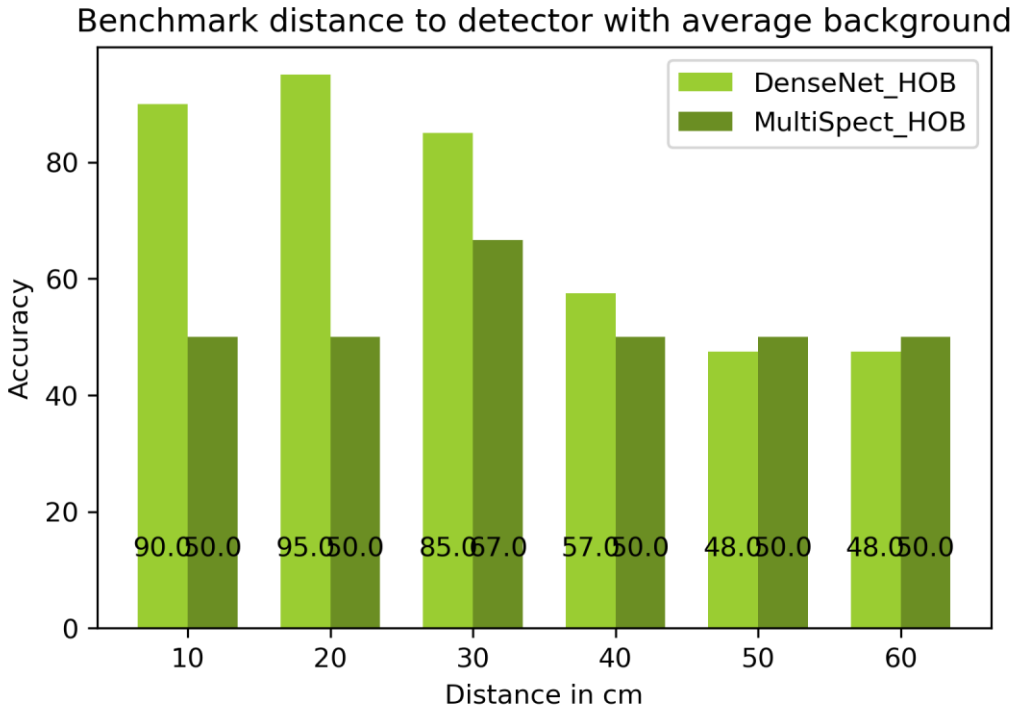
```
3         0.0   0.0          0.0  1.886
4         0.0   0.0          0.0  2.008
..        ...   ...          ...    ...
275       0.0   0.0          0.0  4.928
276       0.0   0.0          0.0  5.008
277       0.0   0.0          0.0  4.919
278       0.0   0.0          0.0  4.997
279       0.0   0.0          0.0  4.893

[280 rows x 4 columns]
```

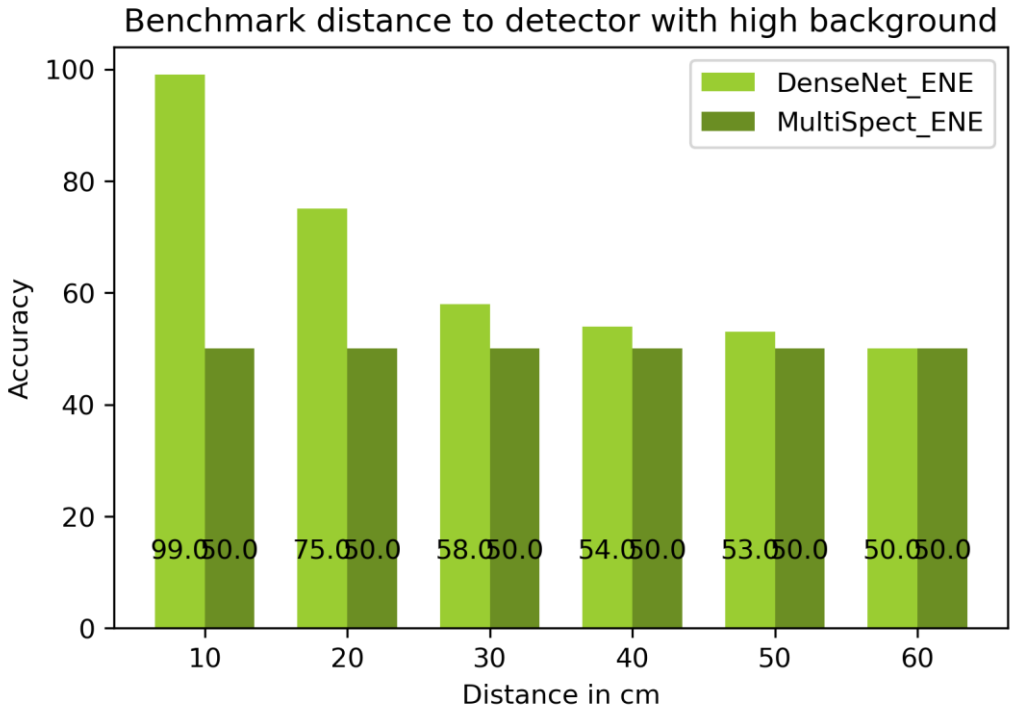





```
   tijd  resultaat1  resultaat2  resultaat3  resultaat4       FP1  FN1  FP2  \
4   1.0  100.000000        93.0   50.000000   50.000000  0.000000  0.0  0.0
0   2.0   98.333333        98.0   66.666667   50.000000  1.666667  0.0  0.0
1   3.0  100.000000        99.0   66.666667   50.000000  0.000000  0.0  1.0
```

```
2   4.0   96.666667        96.0   66.666667   50.000000  3.333333  0.0  4.0
3   5.0   98.333333        96.0   66.666667   66.666667  1.666667  0.0  4.0

   FN2
4  7.0
0  2.0
1  0.0
2  0.0
3  0.0
   hoogte  resultaat1  resultaat2  resultaat3  resultaat4
1    10.0        90.0        99.0   50.000000        50.0
2    20.0        95.0        75.0   50.000000        50.0
0    30.0        85.0        58.0   66.666667        50.0
3    40.0        57.5        54.0   50.000000        50.0
4    50.0        47.5        53.0   50.000000        50.0
5    60.0        47.5        50.0   50.000000        50.0
```





In [ ]:

```
print(hoek_bench)
```

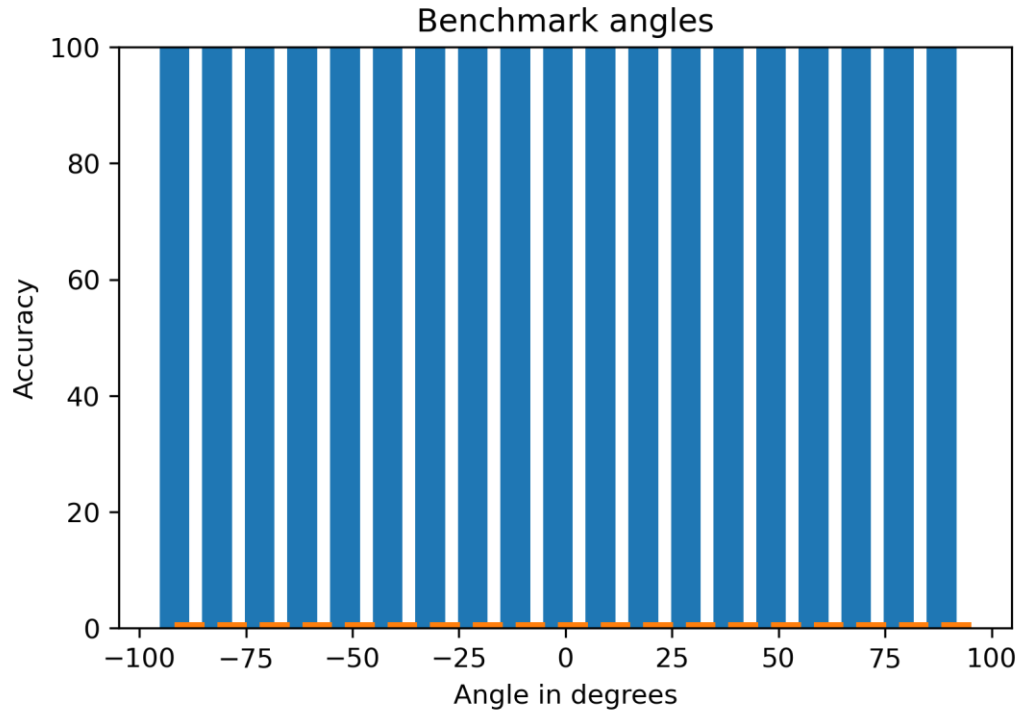
In [69]:

```
#hoekbenchmark
hoek_bench_spectrum = pd.DataFrame()
hoek_bench_energie = []
hoek_bench_counts = []


hoek_bench_Y= hoek_bench.pop("Bron")
hoek_vertaling = pd.DataFrame({"0":[],
                               "1":[]})
for i in range(0,len(hoek_bench_Y)):
    Voc = hoek_bench_Y[i]
    if(Voc == 'Background'):
        Voc = Voc.lower()

    a = np.where(vocabulair == Voc)
    a = int(a[0])
    if(a == 1):
        Cs = 1
        Bg = 0
    else:
        Cs = 0
        Bg = 1
    hoek_vertaling = hoek_vertaling.append({
          "0": Bg,
          "1": Cs}, ignore_index = True)


hoek_vertaling = np.array(hoek_vertaling)
hoek_bench_X = hoek_bench.pop('Counts')
hoek_bench_Z=hoek_bench.pop('Energie')
for i in range(0, len(hoek_bench)):



  a = np.array(hoek_bench_Z[i])
  b = np.array(hoek_bench_X[i])


  hoek_bench_energie.append(a)
  hoek_bench_counts.append(b)

hoek_bench_spectrum['Counts'] = hoek_bench_counts
hoek_bench_spectrum['Energie']= hoek_bench_energie
hoek_bench_spectrum = tf.convert_to_tensor((hoek_bench_spectrum.to_numpy()).tolist())
hoek_voorspellingen, hoek_peak = (model1.predict(hoek_bench_spectrum))
hoek_voorspellingen = np.round(hoek_voorspellingen)

hoek_correct = []
hoek_multi_correct = []
hoek_multi_fout = []
hoek_fout = []
for i in range(0, len(hoek_bench)):
    if(hoek_voorspellingen[i,0] ==hoek_vertaling[i,0]):
        hoek_correct.append(hoek_bench['Hoek'].iloc[i])
```

```
    else:
        hoek_fout.append(hoek_bench['Hoek'].iloc[i])
    if(hoek3_bench['Prediction'].iloc[i]==hoek_vertaling[i,0]):
        hoek_multi_correct.append(hoek_bench['Hoek'].iloc[i])
    else:
        hoek_multi_fout.append(hoek_bench['Hoek'].iloc[i])
unique_hoek = hoek_bench['Hoek'].unique()
hoek_resultaat = pd.DataFrame({'hoek' : [], 'resultaat' : [], 'hoek2':[], 'resultaat2':[]})

for hoek in unique_hoek:
    correct = 0
    fout = 0
    multi_correct = 0
    multi_fout = 0
    correct = hoek_correct.count(hoek)
    multi_correct = hoek_multi_correct.count(hoek)
    multi_fout = hoek_multi_fout.count(hoek)
    fout = hoek_fout.count(hoek)
    percentage = (correct)/(correct + fout) * 100
    m_percentage = multi_correct / (multi_correct + multi_fout)
    hoek_resultaat = hoek_resultaat.append({
        'hoek':hoek,
        'resultaat' : percentage,
        'hoek2': hoek,
        'resultaat2':m_percentage
    }, ignore_index = True)
hoek_resultaat = hoek_resultaat.sort_values(by=['hoek'])
plt.figure(dpi = 300)
plt.bar(hoek_resultaat['hoek'] - 1.75, hoek_resultaat['resultaat'],width = 7)
plt.bar(hoek_resultaat['hoek']+1.75 ,hoek_resultaat['resultaat2'], width = 7 )
plt.ylim(0,100)
plt.xlabel('Angle in degrees')
plt.ylabel('Accuracy')
plt.title('Benchmark angles')
```

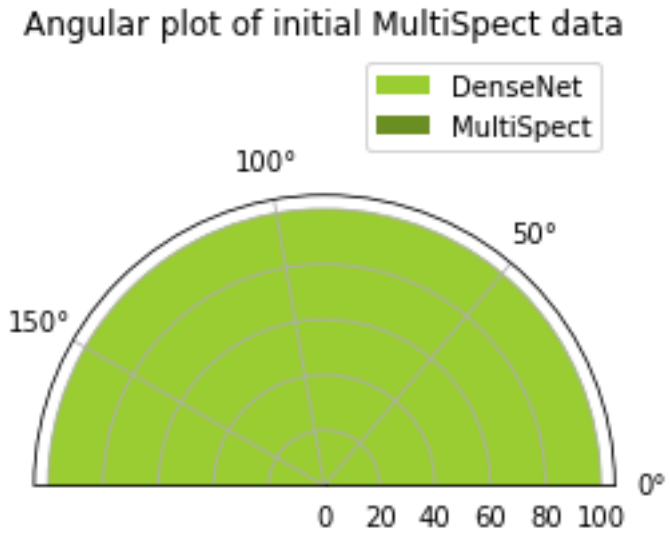
Out[69]:

```
Text(0.5, 1.0, 'Benchmark angles')
```



In [70]:

```
ax = plt.subplot(projection = 'polar')
ax.set_title('Angular plot of initial MultiSpect data')
ax.set_thetamin(0)
ax.set_thetamax(180)
mod1 = ax.bar(hoek_resultaat['hoek']+90, hoek_resultaat['resultaat'], color = 'yellowgreen',
label = 'DenseNet')
mod2 = ax.bar(hoek_resultaat['hoek']+90, hoek_resultaat['resultaat2'], color = 'olivedrab',
label = 'MultiSpect')
ax.legend()
plt.show()
```



In [71]:

```
validatie = open("validation.pkl", 'rb')
validatie = pickle.load(validatie, encoding = 'bytes')

valY = validatie.pop('Bron')

vertaling_val = pd.DataFrame({
    "0" :[],
    "1" :[]
})


for i in range(0,len(valY)):
  bron = valY[i]
  if bron == 'background':
        bron = 'Background'
  a = np.where(vocabulair == bron)
  a = int(a[0])
  if(a == 1):
    Cs = 1
    Bg = 0
  else:
    Cs = 0
    Bg = 1
```

```
  vertaling_val = vertaling_val.append({
      "0": Bg,
      "1": Cs
  }, ignore_index = True)


valY = vertaling_val


Zval = validatie.pop("Counts")
Xval= validatie.pop("Energie")


val_spectrum = pd.DataFrame()
energie = []
counts = []
n =random.randint(0,len(Xval))

for i in range(0, len(Xval)):
  a = Xval[i]
  b = Zval[i]
  a = np.array(a)

  b = np.array(b)

  energie.append(a)
  counts.append(b)



val_spectrum['Counts'] = counts
val_spectrum['Energie']= energie
val_spectrum = val_spectrum.to_numpy()
val_spectrum = val_spectrum.tolist()

val_Peak_Y = validatie.pop('Peak')
val_Peak_Y = np.array(val_Peak_Y)
val_Peak_Y = val_Peak_Y.tolist()




x_val = val_spectrum
y_val1 = valY
y_val2 = val_Peak_Y

x_val = tf.convert_to_tensor(x_val)
y_val1 = tf.convert_to_tensor(y_val1)
y_val2 = tf.convert_to_tensor(y_val2)

resultaat = model1.evaluate((x_val),(y_val1, y_val2), batch_size = 1)
```

```
2090/2090 [==============================] - 13s 6ms/step - loss: 0.6794 - Bron_loss: 0.4588 -
Peak_loss: 0.0246 - Bron_accuracy: 0.8522 - Bron_f1_score: 0.8522 - Peak_binary_accuracy: 0.97
38
```

Aangezien elke keer een model getrained wordt, er verschillen opduiken, is hier ene stuk code dat 100 modellen trained en daar de standaardafwijking en het gemiddelde uit tracht te halen om een onzekerheid te hebben op de efficientie van het model.

In [72]:

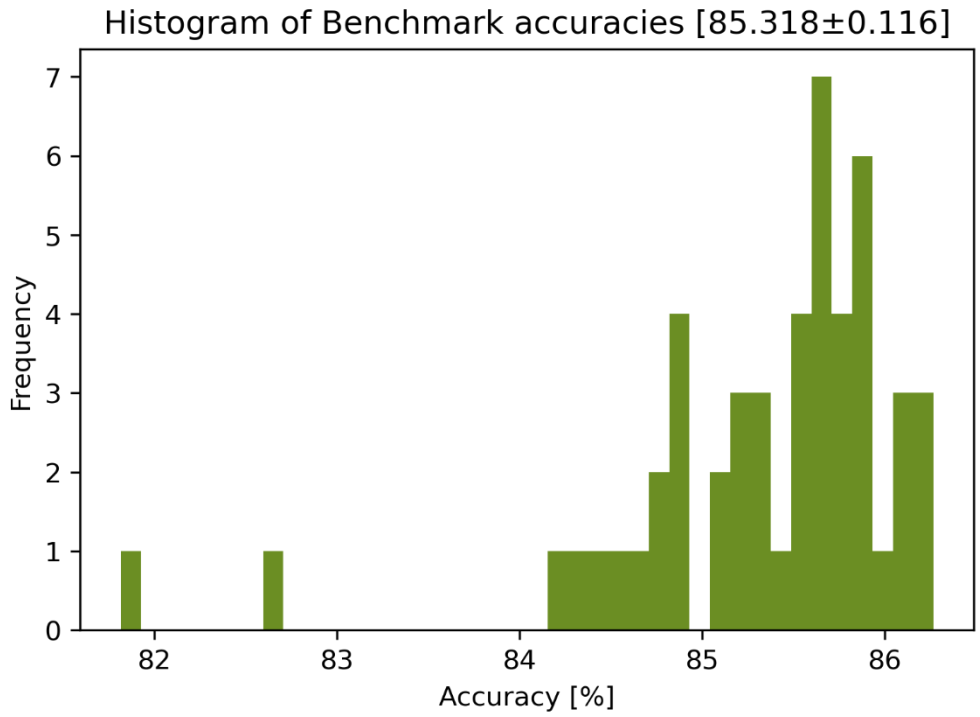
```
print(resultaat
     )
```

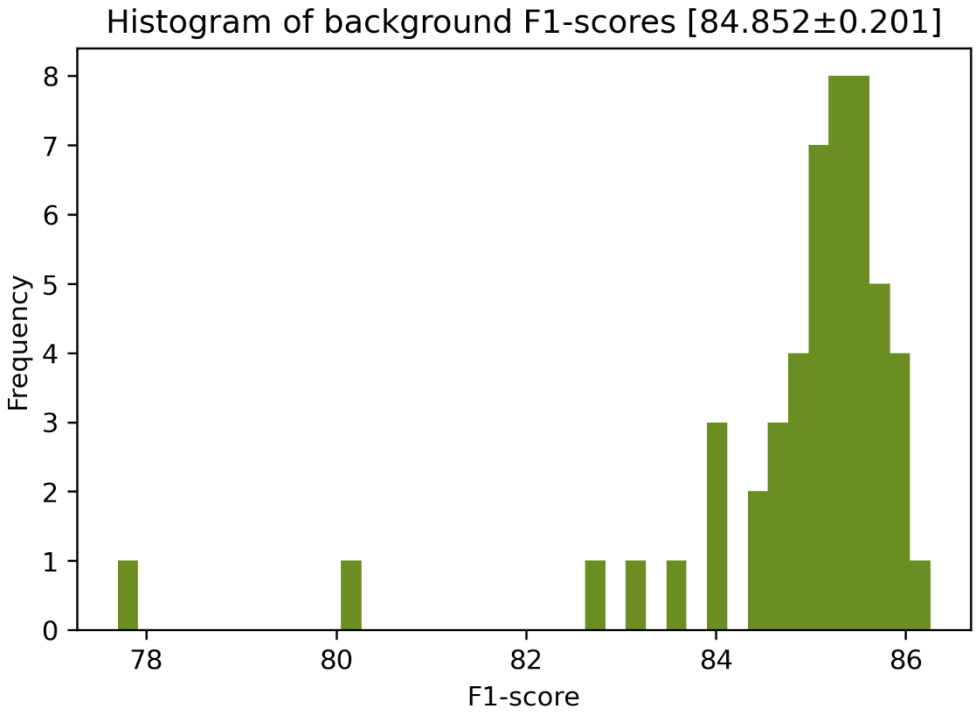
```
[0.6794423460960388, 0.45880118012428284, 0.024574415758252144, 0.8521531224250793, array([0.8
522239, 0.8520823], dtype=float32), 0.9738063812255859]
```

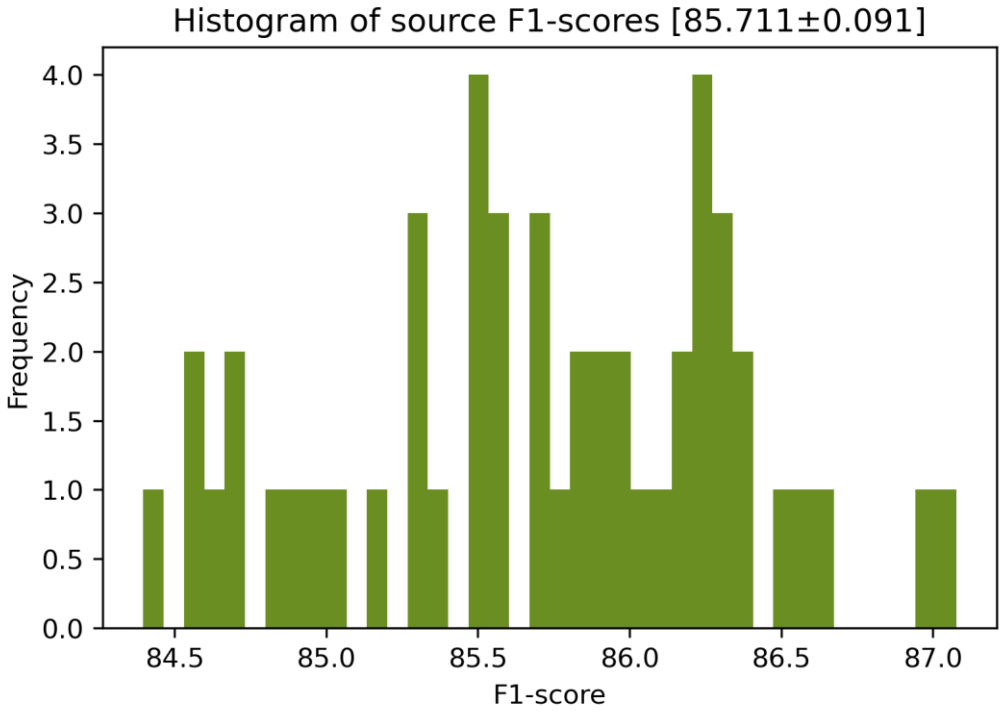
In [17]:

```
models = []
histories = []
results = pd.DataFrame({'Accuracy':[],'F1-score':[]})
for i in range(0,50):
    model = build_model()
    history = model.fit(x_train, (y_train,y2_train), batch_size = 16, epochs =16,verbose = 3
,validation_data = (x_test,(y_test,y2_test)))
    result = model.evaluate((x_val),(y_val1, y_val2), batch_size = 1, verbose = 1)
    models.append(model)
    histories.append(history)
    results = results.append({'Accuracy': result[3],'F1-score': result[4]}, ignore_index =
True )


import pickle

output1 = open('models.pkl','wb')
pickle.dump(models,output1)
output1.close()

output2 = open('results.pkl','wb')
pickle.dump(results,output2 )
output2.close()

output3= open('histories.pkl', 'wb')
pickle.dump(histories, output3)
output2.close()


mean = round(np.average(results['Accuracy']*100),3)
std_dev =  round(np.std(results['Accuracy']*100),3)
z = 1.96
se =round( std_dev / np.sqrt(len(resultaat)),3)
cL = mean - se*z
cU = mean + se*z

plt.figure(dpi = 300)
resultaat = results['Accuracy']*100
resultaat.to_numpy()
plt.hist(resultaat, bins = 40, color = 'olivedrab')
plt.xlabel('Accuracy [%]')
plt.ylabel('Frequency')
title ='['+ str(mean) +'±'+ str(se)+']'
plt.title('Histogram of Benchmark accuracies ' +title)
plt.show()




background = []
source =[]
```

```
for i in range(0, len(results['F1-score'])):
    result = results['F1-score'][i]
    background.append(result[0]*100)
    source.append(result[1]*100)

mean = round(np.average(background),3)
std_dev =  round(np.std(background),3)
z = 1.96
se = np.round(std_dev / np.sqrt(len(background)),3)


cL = mean - se*z
cU = mean + se*z
plt.figure(dpi = 300)
plt.hist(background, bins = 40, color = ['olivedrab'])
plt.xlabel('F1-score')
plt.ylabel('Frequency')
title = '[' +str(mean) +'±'+ str(se)+']'
plt.title('Histogram of background F1-scores ' + title)
plt.show()


mean = round(np.average(source),4)
std_dev =  round(np.std(source),4)
z = 1.96
se = round(std_dev / np.sqrt(len(background)),3)

plt.figure(dpi = 300)
plt.hist(source, bins = 40, color = ['olivedrab'])
plt.xlabel('F1-score')
title = '['+ str(mean) +'±'+ str(se)+']'
plt.ylabel('Frequency')
plt.title('Histogram of source F1-scores ' + title)
plt.show()
```




Histogram of background F1-scores [84.852±0.201]
Frequency
F1-score
0
1
2
3
4
5
6
7
8
78
80
82
84
86


Histogram of source F1-scores [85.711±0.091]
Frequency
F1-score
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
84.5
85.0
85.5
86.0
86.5
87.0