UNIVERSITY OF COPENHAGEN

FOREIGN OBJECT DETECTION IN X-RAY IMAGES USING MACHINE LEARNING

Master's Thesis in Physics
submitted by

JESPER RASK PEDERSEN

2020

Jesper: Foreign object detection in x-ray images using machine learning

This Master's Thesis has been carried out at

NIELS BOHR INSTITUTE, COPENHAGEN

under the supervision of

PROF. BRIAN VINTER

and

ASSISTANT PROF. KENNETH SKOVHEDE

ACKNOWLEDGMENTS

Many people has had a hand in making this thesis, and deserves a large thanks.

Thanks to my supervisors Brian Vinter and Kenneth Skovhede. Brian Vinter was my main supervisor for the first year, and was responsible for establishing the contact to FOSS, and for giving me a good start. Kenneth Skovhede had to take over in the middle of the Summer, and without his support it would probably not have passed the finish line.

I have had the pleasure of working with three daily supervisors. Elisabeth Ulrikkeholm was my intial contact person at FOSS, and also served as the main supervisor from FOSS until December. I am grateful to her for introducing me to the people at FOSS. Then Erik Dreier took over, and he had a massive influence on the project with artificial foreign objects. I also appreciate the time he took to proof read this thesis, and for all the meetings. From NBI's side I have been working closely with Carl-Johannes Johnsen during most of the project. I am grateful for the fact that you always had time to talk, even though you have been supervising N other students. There is also no doubt that the thesis itself improved by your countless readings.

It has been a long project and without the great collaboration at the eScience group it would have been a very bleak time indeed. A great thanks to all the people involved.

I got an amazing reception at FOSS, and everyone has always been willing to help, and answer my emails. That played a large part in me feeling welcome, and kept the motivation up.

Finally I have to thank my family and friends from outside the University for supporting me and keeping me sane. A special thanks to my girlfriend Cilie for having to actually live with me during this, at times, stressing period. Furthermore, she also did large parts of the proof reading, often in the early iterations, and for that I am deeply grateful.

ABSTRACT

The development of better detectors, x-ray sources, as well as faster and cheaper computer processing resources, has made x-ray imaging have numerous applications: From airport security, through medical industry to food processing. One way to get more out of x-ray imaging is to use dual-energy x-rays. Dual-energy increases the possibilities with x-ray to measure the contents of the scanned objects. For many applications large amounts of data are collected since x-ray is introduced to do quality control. This is also the case for the food processing industry, where applications of x-ray imaging can be used in-line to scan the full production. The large amounts of data collected requires automated processing to be effective. This thesis explores Machine Learning in the data processing pipeline of a real world machine: The Meat Master II, which generates dual-energy x-ray images.

Concretely, the challenge is to detect foreign objects in the images. To detect the foreign objects a Convolutional Neural Network was trained. Furthermore, the use of synthetic data was explored. We find that it is possible to train a Convolutional Neural Network to 98.74% accuracy on detecting foreign objects, using a sliding window algorithm to preprocess the data from the Meat Master II. We observed a significant drop in accuracy when the model is evaluated on similar but yet unseen data. This is a significant issue since it is hard to guarantee that the training data represents the full test distribution. It is possible to alleviate this drop in performance, as measured by the Area Under the ROC-curve, by using our synthetic data in the form of artificial foreign objects. The accuracy is currently not good enough for real world use, but with a larger dataset to train on, it should be possible to successfully introduce machine learning to automate the detection of foreign objects in meat using machine learning.

CONTENTS

X-RAY PHYSICS AND SETUP

FOREIGN OBJECT DETECTION WITH DEEP LEARNING

APPENDIX

ACRONYMS

FCNN	Fully Connected Neural Network
CNN	Convolutional Neural Network
RELU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
ADAM	Adaptive Moment Estimation
ROC-CURVE	Receiver Operating Characteristic Curve
AUC	Area Under the ROC-Curve
GAN	Generative Adversarial Nets
VAE	Variational Autoencoder

Part I

X-RAY PHYSICS AND SETUP

1 INTRODUCTION

Wilhelm Röntgen discovered x-rays in 1895 as a penetrating electro-magnetic radiation. The ability to penetrate matter made it possible to take pictures of the insides of solid objects, and this found use almost immediately in the medical industry. This resulted in Wilhelm recieving the first Nobel prize in physics in 1901 [2]. Today, x-rays have numerous applications: From airport security, in the medical industry to food processing. In the food processing industry, the ability to do non-destructive inspection of food is an incredibly cost effective way to make quality control. Contrary to a destructive procedure using statistical sampling to test, with x-rays it is possible to scan objects non-destructively. For this reason, x-ray technologies has been adopted in various food lines in factories.[3]

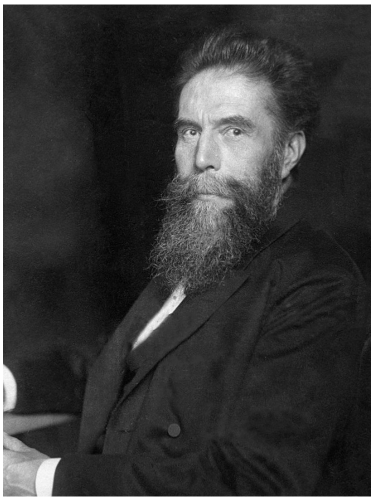


Figure 1.1: Wilhelm Röntgen, from [1]

The ability to inspect all the food in a production line generates enormous amounts of data. To make proper use of the data, the analysis needs to be automatic. The field of computer vision concerns the automated analysis of images. The past years, the field of computer vision has undergone a revolution with the introduction of deep learning [4]. The development of deep learning has been driven by companies like Google and Facebook, who are pushing the boundaries of what is possible in many areas of application. With deep learning, it is possible to take advantage of large datasets to create models of greater size than before in order to get extremely good performance.

A company selling optical instruments to use in food production is FOSS, and this piece of work has been birthed in a collaboration with them. FOSS sells the Meat Master II [6] which is used in-line in meat production to scan meat. The Meat Master II uses x-rays to primarily determine the fat composition of meats. As a secondary task it scans the meat for foreign objects. In meat possible foreign elements could be naturally appearing matter such as bone. More troublesome, is the chance of plastic or metal, which could appear from unfortunate accidents, or perhaps more worrisome, in the form of tampering of the product [7].

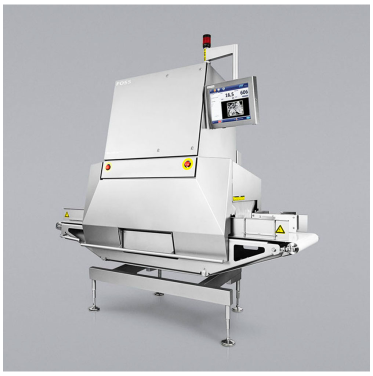


Figure 1.2: Meat Master II, from [5]

The current methods used in the Meat Master II for foreign object detection are threshold-based using computer vision. With their threshold-based algorithm, pieces of metal are the easiest to find, small pieces of bone is almost undetectable, and thin slices of plastic are impossible. The aim of this thesis has been to explore machine learning, and more specifically deep learning, to do foreign object detection. The expectation is that it should be possible to implement an algorithm that can detect foreign objects, of metal and bone. This takes starting point in data from the Meat Master II, but the methods explored here might have more general applications.

This thesis is divided into two parts: X-ray Physics and Setup, followed by Foreign Object Detection with Deep Learning.

X-ray Physics and Setup introduces the physics of x-rays and the data made available by FOSS. This includes the electromagnetic interactions of interest and the source and detector necessary for x-ray imaging (Chapter 2). Then the images that constitute the available data are presented and the labelling of the foreign elements is shown (Chapter 3).

PART I

Foreign Object Detection with Deep Learning follows the four steps of a deep learning algorithm as laid out by the main textbook cited [see 8, sec 5.10]: *"... combine a specification of a dataset, a cost function, an optimization procedure and a model"*. First, the data processing leading to the final dataset is specified. This entailed implementing a sliding window algorithm to preprocess the data into a format better suited for the model. Furthermore, simple data augmentation transformations, along with a more complex one of my own design which introduces synthetic/artificial foreign objects are described (Chapter 4). Then the cost function and optimizers used are presented. The cross-entropy of the negative log-likelihood was used for cost function. For optimizers, Stochastic Gradient Descent (SGD) with momentum along with variations of Adaptive Moment Estimation (Adam) was used (Chapter 5). The design of the deep learning model is presented next (Chapter 6). This concludes the four steps describing the deep learning algorithm, and left is only the evaluation of the approach. The results are presented, both for windows and full pictures. Furthermore, the performance of the model outside its training regime is explored (Chapter 7). Drawing it all to a close, the discussion and conclusion are last (Chapter 8).

PART II

For deep learning I used the PyTorch library. All of the code, for both the thesis and the models, should be available at https://github.com/jrpedersen/mlfoss.

CONTENTS

X-RAY IMAGES 2

In this chapter the physics governing x-ray attenuation contrast imaging is presented. To do so, I will try to shine a light on the interactions between electromagnetic radiation and matter. These interactions are the foundation on which photography is founded, and are key in the making of x-ray images. X-rays are electromagnetic radiation with a typical wavelength in the range of $10pm$ to $10nm$. Central to the usefulness of x-rays is the ability to penetrate solid materials.

The layout of the chapter will be as follows: In the first section, § 2.1, the physics and equations governing the interactions that are relevant for x-rays is showed. These are the photoelectric absorption, Compton scattering, pair production and coherent scattering. The exposition of the material follows Radiation Detection and Measurement [9]. The second section, § 2.2, gathers these interactions in the context of attenuation contrast imaging. The third section, § 2.3, concerns itself with the creation[1] and detection of x-ray photons. Finally, a short section is devoted to introduce dual energy x-rays, § 2.4.

2.1 PHOTONS

Photons or electromagnetic radiation occupy a central place in physics, and is used to investigate anything from the biggest stars, to the smallest of atoms. Now the photon is famously both a particle and a wave at the same time, and when viewing the interactions of importance it is perhaps better to think of the photons as single particles undergoing interactions, which will let it either pass through a given material, or be stopped. For a single photon, the energy, E_p, and momentum, p, is given by:

$$E_p = h\nu,$$
$$p = \frac{E_p}{c},$$

with h being planks constant, and ν the frequency. Furthermore the conversion between wavelength and frequency is given by:

$$c = \lambda\nu,$$

With c the speed of light, λ the wavelength. The interactions of interest are the ones which transfer energy from the particle to the matter. [2] For the following I have used [see 9, chap. 2]

[1] In other words, the *source*

[2] That these interactions are fundamental to physics, can also be seen by the Nobel prizes given for discovering an interaction. Given to Einstein, Compton and Blackett.

2.1.1 Photoelectric absorption

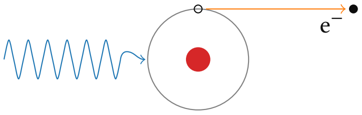


Figure 2.1: The photoelectric absorption. An incident wave (blue) is absorbed by an electron (black) which is in turn ejected from it bound state.

Photoelectric absorption happens when a photon interacts with an atom and the photon excites an electron from the atom to the continuum. A simplistic representation of this is shown in Fig. 2.1. In this process the photon is absorbed. For photoelectric absorption to happen, the energy of the photon has to be greater than the energy with which the electron is bound to the nucleus, E_b. The resulting energy of the excited electron, E_{e^-} is:

$$E_{e^-} = E_p - E_b,$$

The resulting atom is now ionized, and depending on which shell the electron exited belonged to, further reactions may take place, resulting in the emission of new photons.

The full description of this effect, how it alters the probability of absorption, is rather complex. As an approximation in the x-ray regime, it is described by the equation:

$$\tau \propto \frac{Z^N}{E^{3.5}},$$

with τ being the rate of absorption given Z, the atom number, and $N \in [4, 5]$ an experimentally determined constant. Thus we see that heavier nucleus absorbs more photons, which is why these are used to shield x-ray devices with especially lead being common.

2.1.2 Compton Scattering

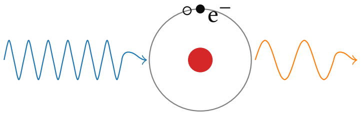


Figure 2.2: Compton scattering. An incident wave (blue) is inelastically scattered by an electron (black) which in turn recoils.

Compton scattering describes the inelastic deflection of an incoming photon when interacting with an electron of the target material. A simplistic representation of this is shown in Fig. 2.2. In this process the photon transfers energy to the electron it interacts with. Using conservation of energy and momentum, for a given scattering angle θ, we have:

$$h\nu' = \frac{h\nu}{1 + \frac{h\nu}{m_0c^2}(1 - \cos\theta)},$$

where ν' is the frequency of the outgoing photon, and m_0c^2 is the rest-mass energy of the electron.

2.1.3 *Pair production*

Pair production is an interaction between the photon and the nucleus at high energies. The result is that the photon energy is converted to an electron-positron pair. A simplistic representation of this is shown in Fig. 2.3. Since the rest-mass energy of a pair of electrons is $1.02MeV$, the photon energies must be larger than this. This effect only becomes predominant at even larger energies well outside the scope of the x-rays used for this thesis.

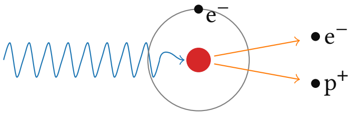


Figure 2.3: Pair production. An incident wave (blue) interacts with the nucleus (red), which transforms its' total energy into an electron-positron pair.

2.1.4 *Coherent Scattering*

Included for completeness is coherent scattering. It is a fully elastic interaction which leaves the energy of the photon unchanged. For the low energies this interaction is roughly a couple of percentages of the total attenuation coefficient. The surroundings can also by coherent scattering introduce noise in the images.

2.2 ATTENUATION CONTRAST

To create attenuation contrast images we use the property that a beam of x-rays will be attenuated when transmitting through a given sample. Areas of high density, ρ, and containing elements with a high atomic number, Z, will have a higher probability to absorb photons from our beam, contrasting with regions lacking these attributes.

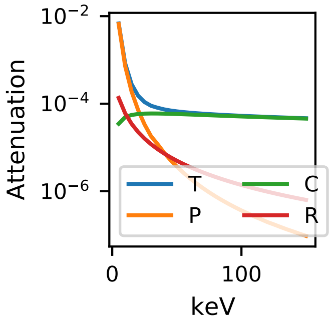


Figure 2.4: Attenuation for all the different processes for Carbon in the energy range 5 – 150 keV. T for total, P for photoelectric, C for Compton Scattering, and R for Rayleigh Scattering (Coherent Scattering). From [10]

To model all of the above interactions as one probability for the photon to be absorbed, we can introduce the linear attenuation coefficient, μ, as:

$$\mu = \mu_{PE} + \mu_{Comp} + \mu_{PP} + \mu_{Coherent}, \tag{2.1}$$

with μ_{PE} the contribution from the photoelectric effect, μ_{Comp} the contribution from Compton scattering, $\mu_{Coherent}$ the contribution from coherent scattering and μ_{PP} the contribution from pair production. A plot of the total attenuation can be seen in Fig. 2.4 where Carbon is chosen as the target. Thus if the intensity of a photon beam is I_0 the intensity after passing through a given material is then given by the Lambert-Beer's law:

LAMBERT-BEER'S LAW

$$I(x) = I_0\, e^{-\mu x}. \tag{2.2}$$

That the density is important can be inferred from the fact that our description of interactions is per atom. Thus higher densities lead to a higher number of atoms, which leads to more interactions that attenuate our beam. To get a density independent description of μ we can use the mass attenuation coefficient μ_ρ.

$$\mu_\rho = \frac{\mu}{\rho}$$

For compound materials, the total mass attenuation coefficient can be calculated from:

$$\mu_\rho(total) = \sum_i w_i \cdot \mu_\rho(i)$$

where the $\sum_i$ is over the i elements and w_i is the fraction of the weight of element i to the total weight. With these equations in hand, it's possible to infer the material properties of an unknown sample, giving rise to a host of applications.

2.3 CREATION AND DETECTION OF X-RAYS

In practice the quality of x-ray imaging is heavily dependent on both the quality of the created x-rays, and the quality of the dector. The source of x-rays used[3] for this thesis is an x-ray tube. The main process being used in conventional x-ray tubes to generate x-rays is called Bremsstrahlung which we will look at first. Then, scintillator based detectors will be quickly reviewed, since these are the ones FOSS used to generate the images for this thesis.

[3] by FOSS.

2.3.1 X-ray tubes & Bremsstrahlung

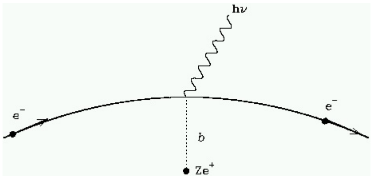


Figure 2.5: Bremsstrahlung. From [11]

Bremsstrahlung is the radiation due to electrons de-accelerating in matter and converting their energy to electro-magnetic radiation. A stylistic example of this can be seen in Fig. 2.5. This results in a broad spectrum of energies, bounded by the maximum energy of the electron above, and dominated by the low energies. In order to get a good conversion of electron energy to radiation, the material used to slow the electrons has to have a high atomic number, and the electron energy has to be high as well. In x-ray tubes, a beam of electrons is shot at a target which is responsible for the de-acceleration. The electrons have a chance to excite the atoms of the target which when the atom decays result in some characteristic x-rays of a given energy, on top of the Bremsstrahlung spectrum. In practice, one is mostly interested in the high end of the spectrum which leads one to apply a filter to remove the low energy x-rays. For our purposes photons in the energy range $10 - 200$ keV is of interest.

2.3.2 *Detector*

FOSS uses a scintillator based detector. The scintillator converts high energy photons to visible light. Then it is possible to measure the amount of visible light with a conventional camera. This requires that the scintillator is transparent to the wavelength of its own emitted light.

The total performance of the scintillator is given by its ability to convert x-rays into its own emitted light. Furthermore, this conversion should be linear, such that the light emitted is proportional to energy input.

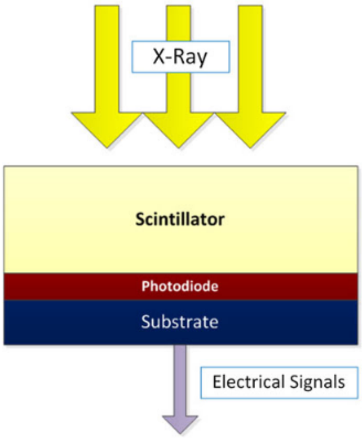


Figure 2.6: Scintilator detector. From [12]

2.4 DUAL ENERGY

This sections draws heavily on [13]. The above admittedly very simple picture of attenuation excluded one very important fact. The interactions between photon and material are very dependent on the energy of the photon, E_p. Thus the linear attenuation coefficients are too. Introducing this dependence formally by writing $\mu(E_p)$ in Eq. (2.1) and Eq. (2.2) results in:

$$I(E_p, x) = I(E_p, 0)\; e^{-\mu(E_p)x}.$$

Here we have also already anticipated the possibility that the incident beam is not monochromatic, meaning I_0 becomes $I(E_p, 0)$. From § 2.3 we know that this is very much the fact. Leaving that aside for now, if we assume that we have two perfectly monochromatic beams on the same target, with intensities I_1, I_2, we can use the relation:

$$\frac{\log \frac{I(E_{p1},x)}{I(E_{p1},0)}}{\log \frac{I(E_{p2},x)}{I(E_{p2},0)}} = \frac{\mu(E_{p1})}{\mu(E_{p2})},$$

to get a length independent estimate of the composition of the target. If we assume that our target, with length L, consists of two materials, we get:

$$\mu L \approx \mu_1 L_1 + \mu_2 L_2,$$

which is a solvable system of two equations. For a non-monochromatic beam one has to integrate over the energy of the beam to get the full transmission. This gives us, under the assumption of dual-energy beams, the equations:

$$\int_{E_1} I_1(E_1, 0)\; e^{-\mu_1(E_1)L_1 - \mu_2(E_1)L_2} dE_1 = I_1,$$
$$\int_{E_2} I_2(E_2, 0)\; e^{-\mu_1(E_2)L_1 - \mu_2(E_2)L_2} dE_2 = I_2.$$

These are the two intensities resulting in the images we have been working with.[4] Our approach has been to use machine learning with these as inputs, instead of using any physical way to solve the system. We will show examples of the actual dual channel images in the next chapter.

2.4.1 *Dual energy scintillator*

FOSS uses a sandwich detector to obtain their dual energy images. In this setup, there is one source, with two detectors stacked on top of each other. The first scintillator absorbs the low energy x-rays and transmits most of the high energy x-rays. The second scintillator is designed to absorb the remaining high energy photons. This result in the two energies being correlated with each other, since the high energy signal also has to pass through the low energy detector. This effect can be reduced by placing a metal filter between the two scintillators to remove all the low energy photons before the second scintillator. An example can be seen in Fig. 2.7.

[4] Assuming perfect detector efficiency.

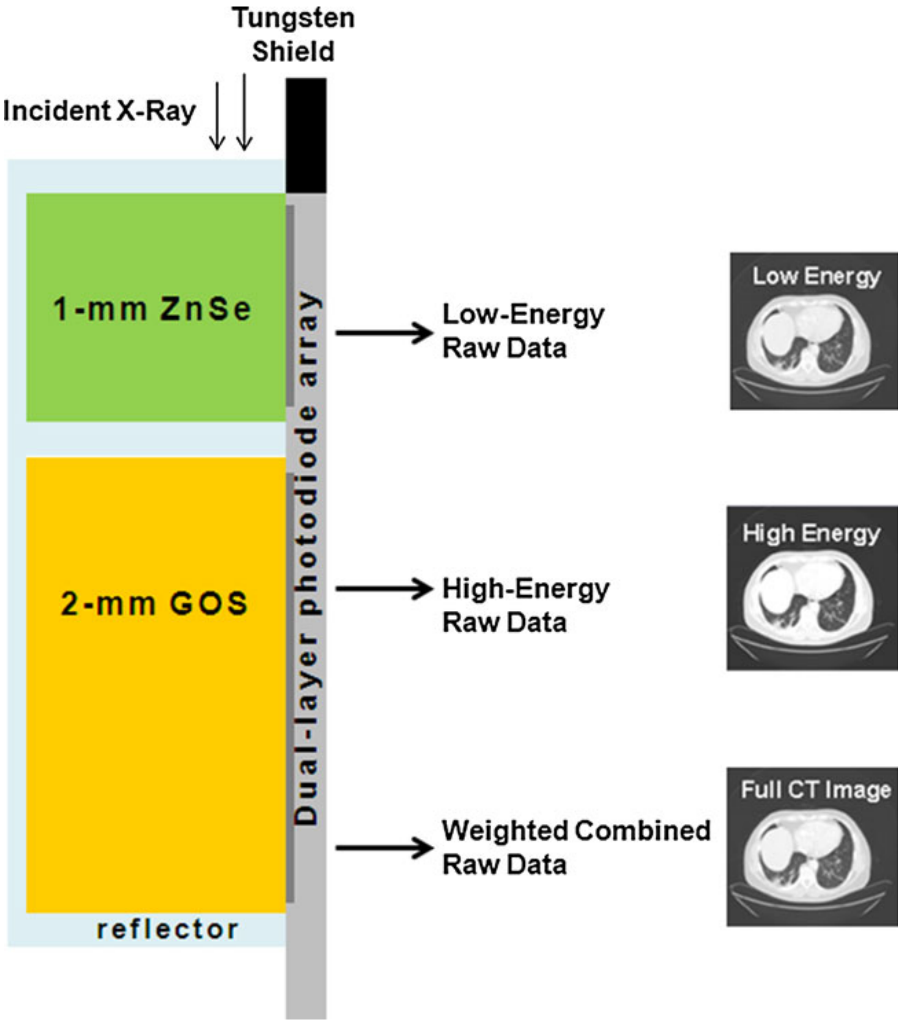


Figure 2.7: One sandwich detector illustrated, not the one FOSS uses. From [12].

CONTENTS

3 DATA PRESENTATION

In this chapter the data are presented. The chapter is split in two sections. The first section, § 3.1, presents the data that was made available from FOSS. The data are dual energy x-ray images obtained using the Meat Master II. I was involved in a quarter of the experiments. The second section, § 3.2, goes in depth with the critical task of labelling the data, which I did for all the images. The labels are used to both train and evaluate the performance. Thus, to a large extend, the quality of the labelling bounds the performance of the algorithm.

This chapter will present the raw data and methods used for annotation. Then, in the next chapter, we will look at the data in the context of machine learning.

3.1 FOSS IMAGES

The full dataset is made up of four parts. In order to keep track of each, we will refer to them as: Circles, Squares, Pens, and Uniform. Examples from each are presented in Fig. 3.1. Each image has two channels, one for the low- and the high energy part. When visualizing images, we visualize the low energy channel, unless otherwise written. The images are captured on a conveyor belt and have the height of 384 pixels. The width is dependent on the objects being scanned, roughly ranging from 380 to 550 pixels. The pixels have a physical size of 1.6 mm. As the first preprocessing step, the width across images belonging to the same dataset was made constant. They would differ by $1 - 10$ pixels, so in effect it meant cutting of small strips at the edges. The concrete composition of the datasets are as follows:

Circles. This part of the data contains 20 images, where half of the images have three phantoms each. Every phantom consists of 13 spheres of metal, giving each image a total of 39 foreign objects. An example of an image belonging to this dataset can be seen in Fig. 3.1.**A**. This image contains phantoms, but they are not necessarily easy to see. This row, shows both the low- and high energy channel, along with their difference. The difference is plotted to show that there is some extra information in having the two channels. It also makes it slightly easier to spot the foreign objects.

Squares. This part of the data contains 116 images. A quarter of the images contain phantoms. These images have two distinct backgrounds. In Fig. 3.1.**B** the first and third images shows the two kinds of backgrounds, and the second and fourth are examples that includes foreign objects. For this dataset, the foreign objects consists of both cubes of metal and the phantoms from before.

Pens. This part consists of 40 images. These images have four distinct background, examples of which can be seen in Fig. 3.1.**C**. The foreign objects, of this dataset, are real world objects and there are two configurations. This means they are more complex than the previous types of objects. Examples are shown in row two and three of Fig. 3.1.**C**.

Uniform. In total there is 47 images with the same, mostly uniform, backgrounds, shown in Fig. 3.1.**D**. This is the dataset I helped to create. The uniform background is made up of a varying number of POM plates, from 2 to 17. Each image contain two phantoms identical to the ones used in

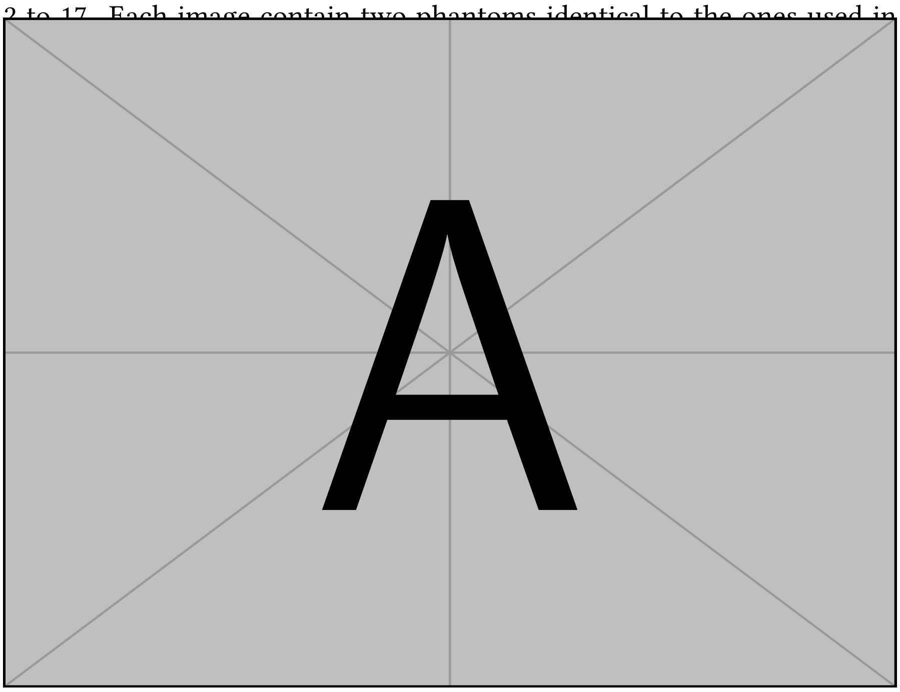


Figure 3.1: **A** Visualization of one image from the dataset called **Cirles**. The image has two channels, corresponding to the low energy (LE) and high energy (HE) channels. Their difference is shown in the third picture. The example image contains phantoms. **B** The low energy channel of four images from **Squares**. These images have either round or square meats as background, and shown is two examples of each. The first two are without phantom, the last two with. **C** The low energy channel of 12 images from **Pens**. The four distinct backgrounds are shown along the columns, with each set foreign objects on each row. **D** The final row is four images of the low energy channel from **Uniform**. The range of backgrounds is from 17 to 2 POM plates. All of these images contain foreign objects.

We will be using the first three sets of images, **Circles**, **Squares** and **Pens**, as the images that are the main challenge of the thesis. The background in these images are real meat, and thus they resemble the real world the best. The fourth set, **Uniform**, will be used to test my final model in a more contrived setting, helping to put some boundaries on the performance.

3.2 LABELLING

Two different pieces of software was used to label the data. The first, labelImg ([14]), allowed one to make rectangular bounding boxes. The second, Labelme ([15]), also has support for polygonal and circular annotations. In both cases, the labelling resulted in a binary mask. Fig. 3.2 shows different annotations of the same image with their difference, using an zoomed in example from **Pens**. The plot of the difference shows that the rectangular boxes will wrongly label a significant amount of pixels in the vicinity of non-square objects. Depending on the eventual algorithm used, this could have an effect on the final result. Furthermore, it is possible to see three objects that are not labelled as foreign objects. These are the hair tie between the pen and the trapeze. Above that a circular shape is also shown. Finally there is a small stone to the center right. These were hard to consistently see and as such they are not included in general. It seems the Labelme would be ideal for this dataset. Unfortunately, I began using it late in the process and only have a version of **Pens** labelled this way.

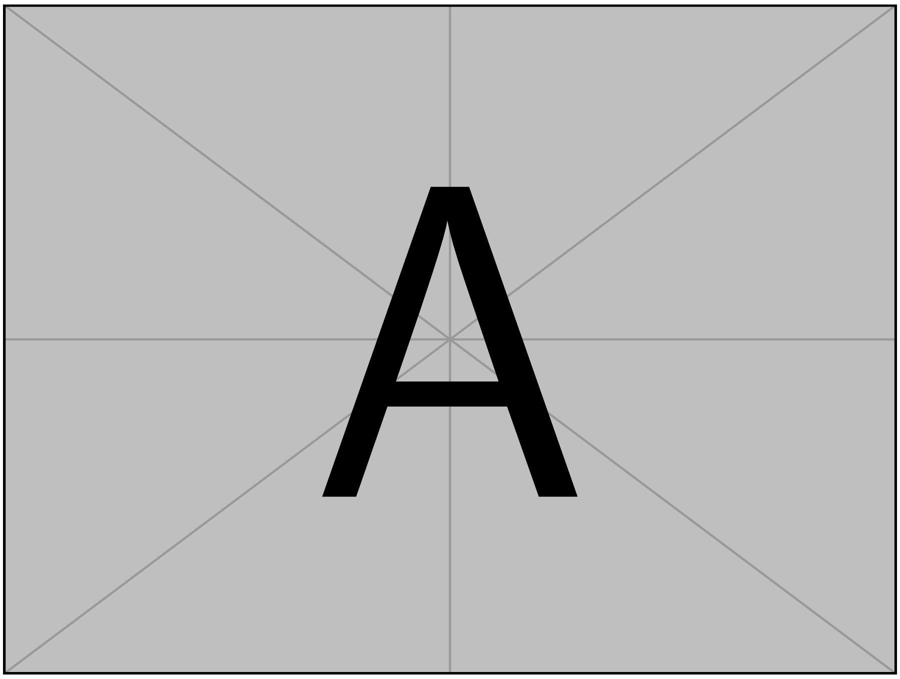


Figure 3.2: The two ways to label the data and their difference. The first image was labelled using LabelImg which only allows rectangles. The second was annotated with Labelme which allows for more complex shapes. Finally, the difference of the masks is plotted, with the yellow representing the extra pixels due to the rectangles. The dark blue corresponds to pixels not marked with the polynomial labelling, but only with the rectangular.

Part II

FOREIGN OBJECT DETECTION WITH DEEP LEARNING

By now it is perhaps evident that I have eschewed the traditional IMRaD struture of a thesis. This is done for two reasons.

The first, was to keep the structure closer to actual real world pipeline, from x-rays to images to model.

Secondly, I view the process of creating a machine learning algorithm as the collection of many building blocks, as Jeff Clune defines the approach to manual AI in [16]. In that sense, by presenting a deep learning algorithm as consisting of 4 steps, and for each step presenting the building blocks I have used, this modular approach has been made as clear as possible.

CONTENTS

4 DATA PROCESSING

After having labelled the available images the question is: How do you go from a labelled dataset to a reliable model? First of all, the quality of the available data limits the performance of our model. The labelling lets us know whether the images contain foreign objects, which is the process we would like to automate. Thus, errors in labelling might result in errors in our model. Secondly, the images we have are only a sub-sample of the larger distribution of data which our model is supposed to work on. Thus any avenue which increase the quality or suitability of our data, compared to our model, is worth pursuing.

The approach were as follows: The images were cut into windows using an sliding window algorithm (§ 4.1). In order to make the model robust to perturbations, we introduced augmentations in the dataset. First, the simple data augmentations of rotation and mirroring (§ 4.2). Secondly, we created synthetic data, in the form of artificial foreign objects (§ 4.3).

4.1 SLIDING WINDOW

A sliding window algorithm was used to generate 32 × 32 pixel windows from the images with 50% overlap. The windows was used as input to the model. The smaller size of the windows, compared to the full image size, allowed for expanding[1] the initial limited dataset of approximately 200 images considerably. Since the task of detecting foreign objects was harder for smaller objects, keeping the input of a size comparable to the expected objects seemed sensible. Furthermore, this greatly increased the number of background only windows to learn from (true negatives), as one tiny metal sphere of a size of 5 × 5 pixels would not reduce a full picture to positive.

4.1.1 *Training and test split*

In order to verify the results of ones model, it is ubiquitous in machine learning to split ones dataset into **Train** and **Test** sets. The **Test** part is a hold out set, which you use for testing your final model in order to evaluate its performance. If these test results are to be believable it is key that the test data are kept strictly separate from the training process. This is due to the fact that the models used in deep learning sometimes have the capacity to fit

[1] Expand perhaps in an artificial way.

to the noise in the data itself. This is called overfitting, and the problem is that it is impossible to know whether you overfit if you do not keep a hold out set. The training set is the data you have to learn with. It is common to split training into training and validation sets. Thus you only train on part of your training set, and keep the validation set to optimize hyper-parameters. Optimizing hyper-parameters directly on the training set can also lead to overfitting. Finally, we have used a k-fold validation split. K-fold validation is when you split at dataset into k parts, and then use $k - 1$ for training, and the last part for validation. This allows you to validate k partitions of your dataset. This makes k-fold validation most viable for smaller datasets, since validation of k models can be too computationally expensive for large models.

4.1.2 Distributional Statistics

All the windows from **Circles**, **Squares** and **Pens** were collected into one dataset called **Mixed**. These three sets of images were the initial data made available from FOSS, and they represent various ways to evaluate the model. **Mixed** was then split into a **Train-**, **Validation-** and **Test Set** using the ratios 0.64, 0.16, 0.20, respectively. This was done by first splitting train and test 80% to 20%. Then the training set was split one-to-five in validation and train. The **Mixed** dataset was labelled with rectangular bounding boxes. The label distribution of the windows can be seen in Fig. 4.1. We see that 9.0% of the windows contain foreign objects. This means that our dataset is unbalanced, which can potentially lead to issues when training. Furthermore, this is also much higher than what we would expect to see in the real world. This is a problem when reporting test results, since the test result are from a clearly biased distribution.

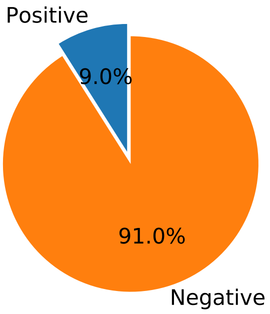


Figure 4.1: Distribution of labels for the **Mixed** dataset.

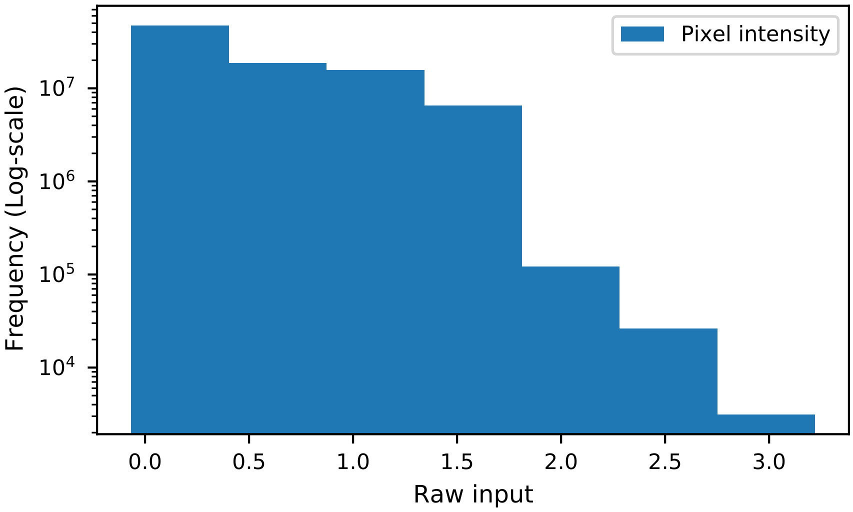


Figure 4.2: Log-histogram of low energy pixel values.

A logarithmic histogram of the distribution of pixel intensities is plotted in Fig. 4.2. The most prevalent values for a pixel is between $0.0 - 0.5$. These pixels are all uncovered conveyor belt, and are not of interest for our model. They can relatively easily be cut away, but we also assume that the model would have no problem figuring out that these are not foreign objects. As we will see in Chapter 7 that is indeed the case. The other end of the scale is where our foreign objects lie. Not all of these pixels corresponds to foreign objects but most do. Preprocessing methods of interest such as histogram equilization and the like were briefly tried, but never fully implemented and tested.

Looking at the windows themselves, the mean value of each pixel, normalized by the range of values to lie between 0 and 1, is plotted in Fig. 4.3. Ideally this would have looked like homogenous white noise. This seems to be the case in the vertical direction, where the variation seems to be on the order of less than 1% of the range of inputs. Varying horizontally, this is not the case. We see that the variations corresponds to 4% of the total range measured. I assume this is a bias due to the sliding window algorithm, and not a bias in the full images themselves. When looking at the two halves of the plot, divided vertically, we see the same pattern repeating just with lower values to the right. This, at least, makes sense since our sliding windows overlap 50%.

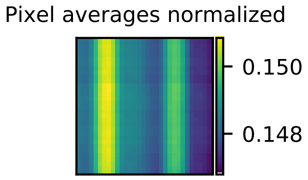


Figure 4.3: Normalized pixel absorption averages. The normalization is over the full range of values the pixel can take as inputs.

4.2 DATA TRANSFORMATIONS

This section describes the various steps in the preprocessing pipeline after loading the windows.

4.2.1 *Standardization*

The windows was standardized as has become standard practice for many deep learning algorithms [17]. This is done channel-wise for all pixels in the window with mean and standard deviate estimated from all the images of the training set. Standardizing is done in order to improve the convergence of the model, and more generally, to make the different data dimensions equal. To be clear the formula is:

$$X_n = \frac{X - \langle X \rangle}{\sigma_X}$$

I use Pytorch's *transforms.Normalize*, but it does in effect standardize [18].

4.2.2 Rotations and mirroring

The images we work with were captured from above, and since the subject is meat, with possibly some non-meat objects we would like identified, the algorithm should be rotationally invariant. Thus, rotations and mirrorings were introduced to augment my data. Since my windows are square, by design, rotations of 90° are especially easy. Combined with mirroring either the vertically or horizontally the number of windows were effectively increased eight-fold. Rotations of other degree than multiples of 90° could have been implemented using linear interpolation, but this would potentially result in a loss of information so we chose not to.

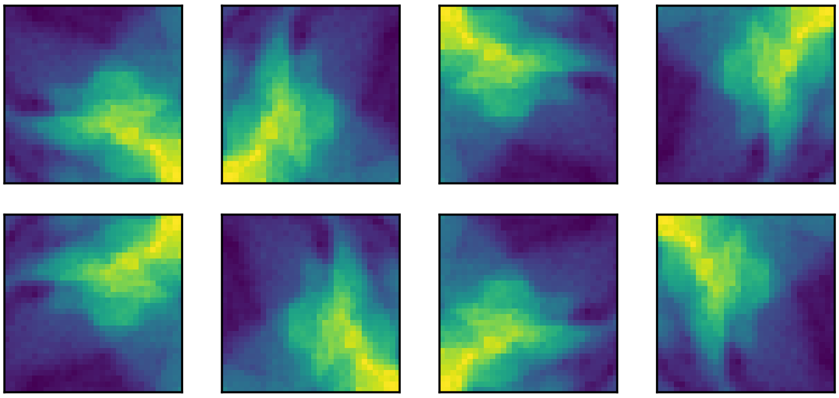


Figure 4.4: Rotations and mirrorings illustrated on a random window.

4.3 ARTIFICIAL FOREIGN OBJECTS

If your available data does not cover the full distribution it is sampled from, it will bias any model trained on it. Collecting more data to fix this is time consuming and expensive, especially if it also has to be labelled manually. This has lead to the emergence of a field of synthetic data. [19] Synthetic data is artificially generated data, it can for example be the results of simulation. In general, simulation is used in many fields of engineering and science and extending it to deep learning has been tried in various formats. Examples of ways to simulate data includes algorithms such as Generative Adversarial Nets (GAN)' [20] and Variational Autoencoder (VAE) [21]. These are typically larger models, and for this project with only 200 images I deemed it infeasible to try creating one of these. For interested readers, a concurrent Masters project employed a GAN in a x-ray potato dataset [22].

Our training set consisted of a limited amount of shapes. This might have induced any model trained on it to overfit to these specific shapes.[2] Thus, in order to remove any possible shape dependence, a training set containing many different kinds of artificial shapes was created. Our algorithm draws

[2] This had in fact happened on a previous pilot project at FOSS.

inspiration from the creation of our images. They were physically created by "adding" foreign objects on top of different meats. We tried creating the obvious pendant by adding foreign objects on top of our meat algorithmically. In the best case scenario, this would let us learn to recognise any shape of objects, even though we had only trained on circles and squares. Further motivation for why this approach might work was two fold:

Firstly, the Lambert-Beers equation (2.2) suggest that by splitting μ in μ_{meat} and μ_{FO}, we get

$$I(x) = I_0\, e^{-\mu_{meat}x_{meat}-\mu_{FO}x_{FO}}.$$

Taking the negative log of this ratio of intensities we get to the format of our images:

$$-\log\frac{I(x)}{I_0} = \mu_{meat}x_{meat} + \mu_{FO}x_{FO}$$

This view is simplistic as it disregards the dual energy. However, it suggest that adding metal is additive in the signal. Furthermore, if we can subtract the background from images with metal, what we have left would be the contribution of the metal. This can then be used to create arbitrary shapes as artificial foreign objects.

Secondly, an alternative way to motivate our approach is in the literature on synthetic data. Specifically, we could take inspiration from the paper [19], which introduces an approach they term: Cut, Paste and Learn. This creates synthetic data as you could imagine: By cutting objects from one setting and pasting them into another. That paper concerns instance detection, which is a sub category of object detection. For instance detection one has to able to differentiate between different instances of the same object.[3] They show that combining synthetic data with real data they can increase the relative performance of their trained model by 21% on benchmarks. While this dataset is significantly different, the idea that one could make useful synthetic data simply by cutting and pasting lends credit to our approach. They work with RBG images, but for x-rays, I believe a reasonable replacement of pasting is addition. Pasting would after all overwrite the local information, where as addition mimics the real world effect as shown above.

The overall idea in our algorithm is to create random shapes, give these plausible intensities generated from our pool of labelled foreign objects, and then add these to otherwise empty windows. For any given dataset the steps required are summarised in Fig. 4.5, and further elaborated below.

[3] A possible example being distinguishing one brand of canned soup from another in a supermarket for example.

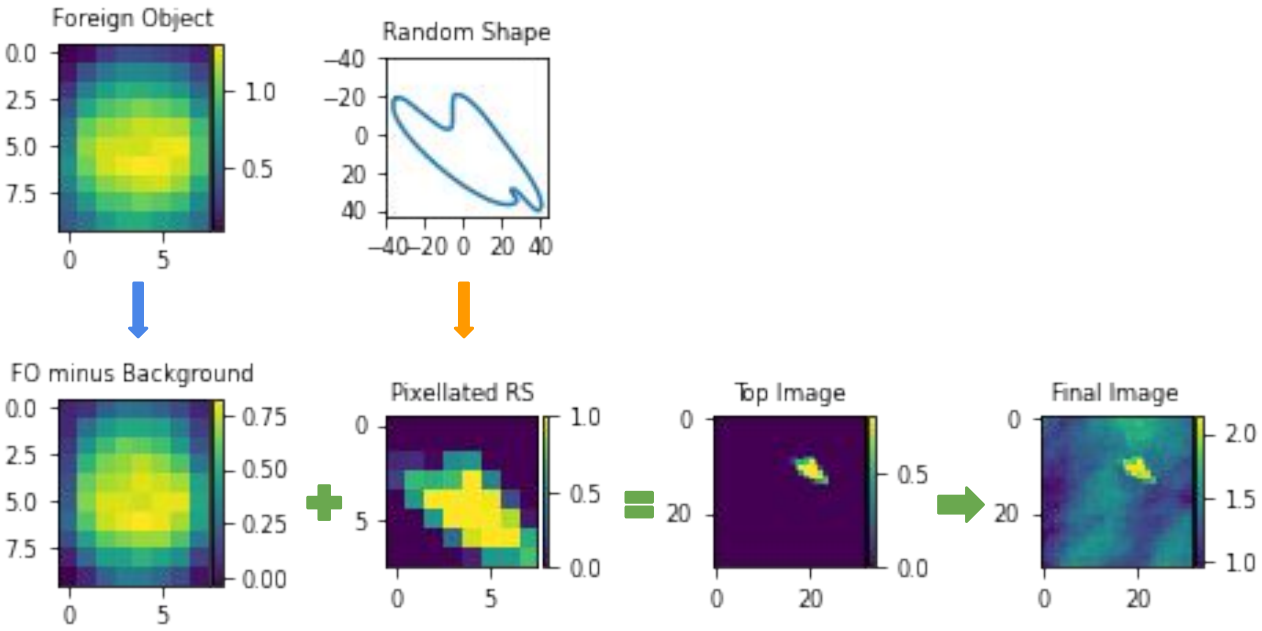


Figure 4.5: Overview of the artificial foreign object pipeline. First we draw a random foreign object from our set of bounding boxes containing foreign objects. Alongside it, we create a pseudo-random shape. Examples of both shown in the upper row. From the foreign object we subtract the background (blue arrow). The random shape is pixelated to the size we are interested in (orange arrow). Using the results of both processes we can create window containing only zeroes and our new foreign object. This can then be added to a window containing only background.

4.3.1 *Gathering the Foreign Objects*

To obtain an approximation of a foreign object's linear attenuation coefficient, we gathered all the bounding boxes containing foreign objects. Before collecting them the background image was subtracted from the images the boxes belonged to. The background was approximated by applying a channel-wise median filter of size 25×25 to the full picture. This way to approximate the background is reminiscent of one of the steps taken in the threshold algorithm used to do foreign object detection by FOSS, and it was suggested to me by Erik Dreier. This subtraction of the background is represented by the blue arrow in the Fig. 4.5, which leads from a bounding box containing a metal ball to one with the background subtracted. The values of the colorbar changes, and the corner pixels change values relative to each other. Further examples of the result of this process, and of different bounding boxes, can be seen in Fig. 4.6

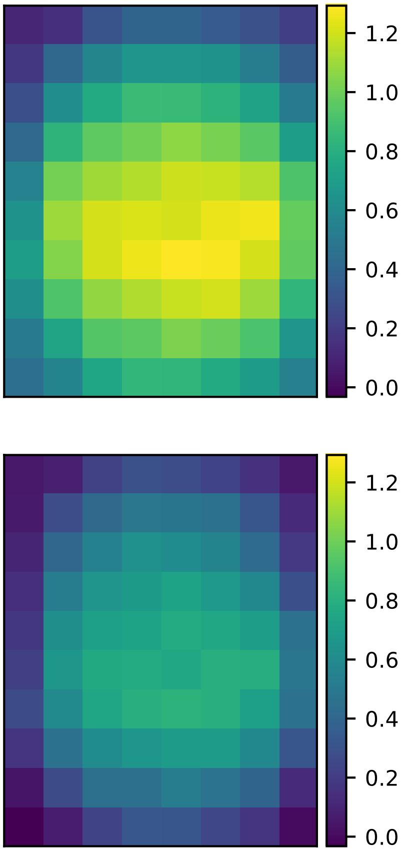


Figure 4.6: In the upper row is shown a sample of four bounding boxes with foreign objects. The lower row is the same objects with the background subtracted.

4.3.2 Creating Random Shapes

The shapes/objects are created in two steps. First, a "high resolution" shape is created and it is then pixellated to fit into a window. The shapes created are 2D. A more complete simulation could include 3D objects and then project them to 2D if specific 3D shapes was of interest. The high resolution random shapes were created using the approach presented by the user ImportanceOfBeingErnest on stackexchange in reply to [23]. In the following, the method for creating random shapes, which is based upon Bézier curves of the third order, is described.

Bézier curves is named after Pierre Bézier [24] who used them for designing cars in the sixties. Today, they are widely used in computer graphics. A Bézier curve of the third order is specified by four points $P_i,\ i \in (0, \ldots 3)$, with an example shown in blue in Fig. 4.7. Here P_0 and P_3 are end points and P_1, P_2 the intermediate points. For the interval $t \in [0, 1]$ the curve is defined as:

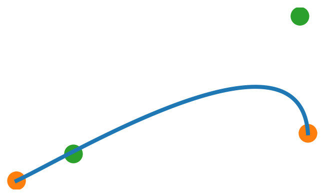


Figure 4.7: In blue a Bézier curve of the third order. The orange points are the end points, with the green the intermediate ones.

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3$$

This is recognized as the expectation of a function, $f(k)$, of a random variable, k, given by a Bernouilli distribution [25]:

$$B(t) = \sum_{k=0}^{n} f(k/n) \binom{n}{k} t^k (1-t)^{n-k}$$

A third order Bézier curve has $n = 3$ and $f(k) = P_k$. In the limits of $t = 0, 1$ we have $B(0), B(1) = P_0, P_3$ which are the end points. As $t \longrightarrow 1$ the intermediate points contribute to the curve.

Now, in order to get a smooth pseudo random shape we need to draw a number of random points[4] which will serve as endpoints for our Bézier curves. Thus, each pair of neighbouring points on the clock is connected through Bézier curves. To get a smooth shape, the intermediate points of each curve must be specified carefully. First of all, the angle of the curves through each point is the average of the angles between it and its two neighbours, unless the parameter edgy is defined. Edgy controls the edginess of the corners of the shapes by being a weight in the averaging of angles. Then the intermediate points between two end points is set using the angle of the curve at the endpoints and continuing in a straight line for some distance. This distance is determined by a radius, which has the corresponding parameter rad in the code. A overview plot of both the parameters effect on random shapes is shown in Fig. 4.8.

[4] Three or more.

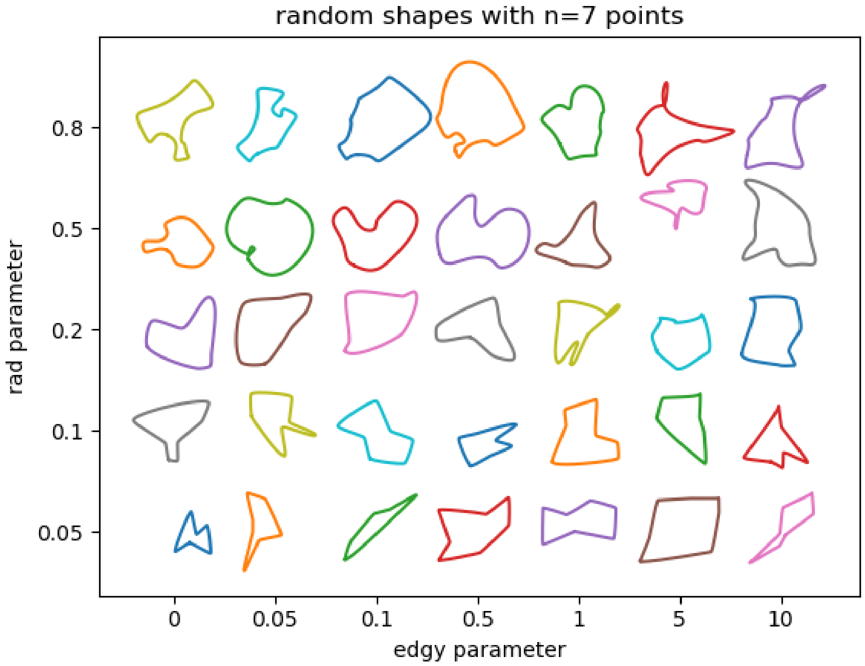


Figure 4.8: Algorithm to create random shapes overview. Figure taken from [23]

The shape was pixelated by initially creating a it on a ten times finer scale than the one desired. This scale was then reduced by averaging whether each

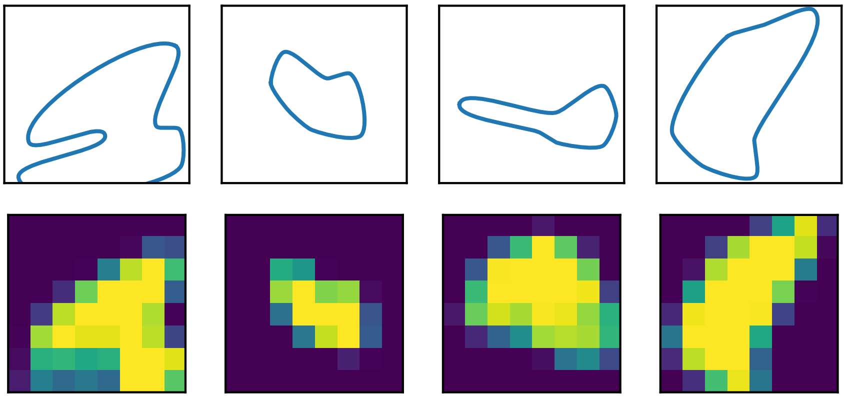


Figure 4.9: Four pseudo-randomly generated shapes is shown on the upper row, with the resulting pixelation shown below.

fine grained cell was inside or outside the shape in question. In this way the grid is reduced by a ratio of 10×10. This leaves a smaller grid, in our case 8×8, with each grid cell giving the percentage to which the cell is inside the shape. In the overview (Fig. 4.5) the averaging process is represented by the orange arrow. Examples of fine-scale curves and their corresponding reduction is given in Fig. 4.9.

4.3.3 New windows

Finally, a random foreign object is drawn from our bounding box set. From the foreign object, we took the channel-wise mean of the four highest values to use as a representation of a foreign object. The use of four values is more or less entirely arbitrary, to the extend that choosing more than one reduced the statistical uncertainty. This mean was multiplied with the pixelated shape resulting in an artificial foreign object. This object was thus approximated to be made of the same material and thickness as the foreign object drawn. The new artificial foreign object was then inserted on a random position in a zeroed array of the same size as the windows. This served as a "top window" which could then add on top of any negative window to create a positive sample. This allowed us to enhance the training set with more positives, and more importantly, positives with "any" shape.

4.3.4 AFO transformation

Finally, the above was implemented as a Pytorch transformation[5]. This transformation is different from the standard data augmentation transformations because it also changes the label of the windows. Furthermore, I decided to only apply the transformation to windows that did not have any FO's to

[5] From the github link: /cnn2/model/ArtificialFO.py

begin with in order to not dilute the original true signal. The transformation was applied to negative windows with a 20% chance when it was used. This probability is a tune-able hyper parameter. Examples of empty backgrounds before and after the addition of an artificial foreign object can be seen in Fig. 4.10.

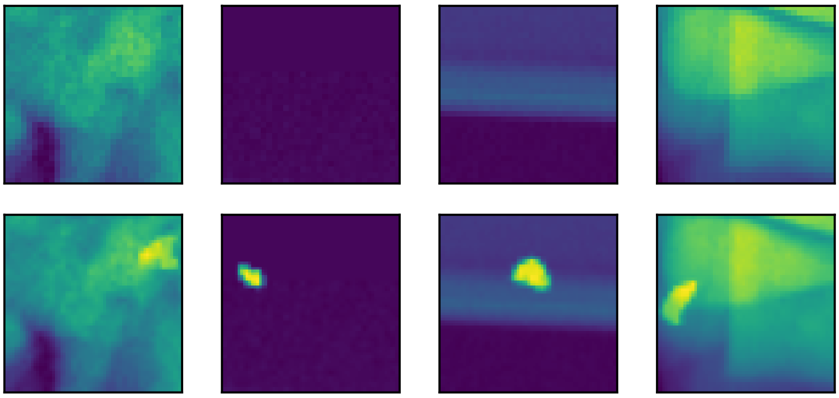


Figure 4.10: In the top row is showed the low energy channel of a sample of windows form the training set. The second row show the same windows with the addition of artificial foreign objects generated using the shapes from Fig. 4.9

CONTENTS

5 LEARNING

Machine learning as a field aims to automate learning. The learning itself can be defined various ways, but keeping to our recipe from [8][1], the learning is a result of choosing a cost function and an optimization procedure. The cost function specifies what you are interested in learning, and the optimization procedure specifies how you will combine your data and model to learn it.

First we will look at a definition of supervised learning (§ 5.1). Then we will look at what we are learning by introducing the cost function[2] (§ 5.2). Finally, different algorithms that use the first order derivatives to update the parameters are described, along with a custom combination of them (§ 5.3). The main work of reference for this chapter is the Deep Learning Book [8].

5.1 SUPERVISED LEARNING

For a thorough exposition of what learning algorithms encompass I recommend [see 8, sec 5.1].

There are two broad categories of learning: supervised and unsupervised learning. For our purposes, it is enough to discuss supervised learning. The task of supervised learning is trying to model:

$$p(y|x),$$

where y is the label, and x the input to the model. If discrete values are used for y to encode different categories we call it classification. Since the data was[3] labelled, we are doing supervised learning. The target was classification approach, since it is fundamentally the binary question we are interested in answering for the real world usage of the Meat Master II.

We used one-hot encoding for my classification. Thus classification formally is the learning a function:

$$f : \mathbb{R}^n \to \mathbb{R}^k,$$

with $n = 32^2$ and $k = 2$. In our case with X representing our data we have.

$$f(X) = z_i, \;\; i \in \{0, 1\}.$$

To get a final prediction from our output, z_i, we have to make some choice. For our model, unless otherwise noted, the choice will be to take argmax of z_i.

[1] "Nearly all deep learning algorithms can be described as particular instances of a fairly simple recipe: combine a specification of a dataset, a cost function, an optimization procedure and a model."

[2] I will use the term loss function and cost function interchangeably.

[3] Painstakingly

5.2 LOSS FUNCTION

The loss function is what we optimize. Thus it has to be constructed in a way that optimizing this results in our model learning. We are not interested in only fitting our data, rather we want to learn the underlying phenomenon to generalize to unseen data. The loss function is interesting in the sense that it serves as an expected loss on unseen data. Where we want to make $p(y|x)_{model} = p(y|x)_{True}$. We use the negative log likelihood as the cross-entropy loss function between the empirical distribution defined by the training set and probability distribution of our model. To derive this we follow the approach of main reference [see 8, sec 3.13].

We start by defining the Shannon entropy as the expected information of a given distribution, $P(X)$, as:

SHANNON ENTROPY

$$\begin{aligned} H(x) &= -\mathbb{E}_{x \in P}[\log P(x)] \\ &= -\sum_i p_i \log p_i \end{aligned}$$

From this we then define the Kullback-Leibler divergence, which is measure of the difference between two distributions, $P(X), Q(X)$, of the same random variable:

KULLBACK-LEIBLER DIVERGENCE

$$\begin{aligned} D_{KL}(P||Q) &= \mathbb{E}_{x \in P}\left[\log \frac{P(x)}{Q(x)}\right] \\ &= \sum_i p_i \log p_i - \sum_i p_i \log q_i \end{aligned}$$

This lets us define the cross-entropy as:

CROSS ENTROPY

$$\begin{aligned} H(P,Q) &= H(P) + D_{KL}(P||Q) \\ &= -\sum_i p_i \log q_i \end{aligned}$$

Thus we see that cross-entropy is the term in $D_{KL}(P||Q)$ dependent of our model $Q(x)$. The way we use the above, is by having q_i being a the probability the model outputs for a given category, and p_i being the ground truth, encoded as an one-hot vector. For the final output of the model, z_i, we use the softmax function to normalize it:

SOFTMAX FUNCTION

$$q_i = \frac{\exp z_i}{\sum_j^k \exp z_j}$$

5.2.1 *Label Smoothing*

One issue[4] is the optimal solution to our optimization problem above. The use of negative log likelihood with the softmax reduces the loss function with assuming that p_i is a one-hot encoding of our category we find that for the label i:

$$-\sum_j p_j \log q_j = -(z_i - \log \sum_j \exp z_j)$$

Thus, the negative log likelihood has maximum when $z_i^* \longrightarrow \inf$. This is good since this mean we can keep training forever. It is bad for the exact same reason: Training forever will potentially lead to overfitting. Furthermore, we are not necessarily interested in encouraging our network extreme confidence in its predictions.[26]

A remedy is to introduce label smoothing as done in [26]. The idea is to replace our labels by:

$$p_i' = (1-\epsilon)\delta_{ij} + (1-\delta_{ij})\epsilon\, u(i)$$

where ϵ is a small number and $u(i)$ is a distribution that is independent of the label of the data points, such as $u(i) = 1/(k-1)$ as in [27]. Now the optimal solution z_i^* becomes:

$$z_j^* = \delta_{ij} \log \frac{(k-1)(1-\epsilon)}{\epsilon} + \alpha$$

here α is a real number which depend on the implementation. Thus the model will converge a finite output. In the two papers cited for this section they found empirical improvements using label smoothing.

5.2.2 *Regularization*

One way to limit the degree to which a model can overfit is to limit the capacity of the model. Without going into details exactly what capacity means explicitly a simple approximation is the number of parameters of the model. One way to limit the capacity is done by adding a term to the loss function that penalizes the weights:

$$J = L + \alpha\Omega(\theta).$$

Here J is the total cost function, with L the loss term and $\Omega(\theta)$ the regularization. α determines the tradeoff between the two terms. In our case, L^2

[4] Or feature depending on your view.

regularization was used. Here, we penalize the model by its norm of the parameters unrolled into a vector.

$$\Omega(\theta) = ||\theta||^2$$

This limits the capacity of the model by adding a term in the gradient to keep the weights small. Thus, the parameter-space of the model is constrained.

5.3 OPTIMIZERS

The field of optimization is vast, and for our purposes we need only a tiny part which nevertheless play an outsized role. That tiny part is the methods that rely on the gradient of the cost function to do gradient descent in the loss landscape. Consigning us to first order gradient based methods, from Stochastic Gradient Descent (SGD) to a series of optimizers using estimates of the second moments to modulate the learning rate, Adaptive Moment Estimations (ADAMS). The optimizers are presented in approximately ascending order of complexity.[5]

[5] Descending order of age.

As a starting point, if we define the total loss, $J(\theta; X)$ corresponding to a loss function, $J(\cdot)$, as a function of the dataset, X, and the parameters of our model, θ, we can define the derivative of this loss with regards to the models parameters as:

$$\nabla_\theta J(\theta; X).$$

This gives us the following update rule for gradient descent:

GRADIENT DESCENT

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta; X).$$

Here we have introduced the *learning rate* η, which controls the step size of the gradient descent. This will be the building block of the methods to follow. Furthermore, we tacitly assume that this gradient is possible to calculate which it is for the models described in this thesis.

First, we will look at SGD with and without momentum. Momentum is the first moment of the gradient, and with it we can introduce estimates of the second moments and use both to create ADAM. ADAM was not the first algorithm to use second order moments, but it has gained a lot of popularity, and spawned many derivates, some of which will be presented below. All proofs of convergence in convex and non-convex settings are beyond the scope of this thesis, and as such we refer the reader to the references.

5.3.1 Stochastic gradient descent

When using gradient descent for very large datasets, it is efficient to batch the data into smaller groups, X_B. If the batches are sampled randomly from the dataset we have SGD [28],[29]. Following [30] we can write SGD as:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta; X_B).$$

SGD

Momentum

Momentum, m_t, [31] is used to increase the rate of convergence of the optimization. The idea is that the best direction in loss space of update is obtained by keeping a running, exponentially decaying, average of the past gradient steps. The intend is that the part of the update that is due to random noise is averaged out.

$$\begin{aligned} m_t &= \gamma m_{t-1} + \eta \nabla_\theta J(\theta; X_B), \\ \theta_{t+1} &= \theta_t - m_t, \end{aligned}$$

with γ determining the exponential decay of past gradients, and a typical value is $\gamma = 0.9$. From the above it is evident that part of the update at time t was already known at $t-1$. Namely, the first part of the momentum γm_{t-1} which is γ times the update at the last timestep. This part can we subtract from the momentum before taking the derivative. This is called Nesterov momentum [32], [33]. We can write:

$$\begin{aligned} m_t &= \gamma m_{t-1} + \eta \nabla_\theta J(\theta - \gamma m_{t-1}; X_B) \\ \theta_{t+1} &= \theta_t - m_t \end{aligned}$$

SGD WITH NESTEROV MOMENTUM

Which, for later convenience can be rewritten as [34]:

$$\begin{aligned} g_t &= \nabla_\theta J(\theta_{t-1}; x^i, y^i) \\ m_t &= \gamma m_{t-1} + \eta g_t \\ \theta_t &= \theta_{t-1} - (\gamma m_t + \eta g_t), \end{aligned}$$

assuming $m_0 = 0$, i.e. the momentum is initialized to zero.

5.3.2 Adam & its derivatives

Expanding the estimation of moments to also include second order moments, we can get different algorithms suchs as AdaGrad, RMSprop and Adam. Since

ADAMS family

1) *Adam [35]*
2) *NAdam [34]*
3) *Amsgrad [36]*
4) *PAdam [37]*
5) *AdamW [38]*
6) *Yogi [39]*
7) *RAdam [40]*
8) *AdaBound [41]*
9) *AdaBelief [42]*

RMSprop is Adam with $\beta_1 = 0$ and AdaGrad is very similar, we focus on the family of ADAMS which are the ones used in this thesis. Some of the variations can be seen in the list to the left. For the following α is used in place of η for the learning rate.

Adam

ADAM was in the words of its authors [35] intended to combine the advantages of AdaGrad, RMSprop, and as such serves as our starting point of optimizers with adaptive learning rate. [6] The idea is to estimate the second order moments and use these to normalize the learning rate individually for each parameter. The second order moments are equal to the uncentred variance. Thus, a large variance in the gradients for some parameters should correspond to a low learning rate. The concrete formula is:

$$
\begin{aligned}
g_t &= \nabla_\theta J(\theta_{t-1}; X_B) \\
m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\
v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \\
\hat{m}_t &= m_t / (1 - \beta_1^t) \\
\hat{v}_t &= v_t / (1 - \beta_2^t) \\
\theta_t &= \theta_{t-1} - \alpha_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}.
\end{aligned}
$$

The exponentially decaying averages, m_t, v_t now have distinct parameters for each estimate β_1, β_2 with out of the box values of 0.9, 0.999. Furthermore, to get an unbiased estimate $\hat{m}$ is needed instead of m directly, and likewise for $\hat{v}_t$. Finally, ϵ is included for numerical stability, with a typical value of $\epsilon = 1e - 8$ [35]. This algorithm allows the second order momentum to adapt the rate of change for each parameter. This makes the magnitude of the update to the parameters invariant to rescaling of the gradient. It creates a natural bound on the step size:

$$
\frac{\hat{m}_t}{\sqrt{\hat{v}_t}} \sim \frac{g_t}{g_t}
$$

ADAM while popular, shows the best performance in natural language tasks, whereas in image tasks ADAM is outperformed by SGD [43].

[6] As far as the author know whether ADAM manages to combine these advantages and always performs better than the previous two is not necessarily the case.

NAdam

We can introduce Nesterov momentum in Adam as in SGD [34]. Doing so will modify Adam to look like:

$$\theta_t = \theta_{t-1} - \frac{\alpha_t}{\sqrt{\hat{v}_t} + \epsilon} \cdot (\underbrace{\beta_1 \hat{m}_t}_{\text{The momentum term from t+1}} + \underbrace{(1 - \beta_1) g_t}_{\text{Gradient part}})$$

The motivation for introducing Nestorov momentum is the same as for SGD. It serves as an improved form of momentum [see 34, sec 2].

AdamW

If we use Adam as is, the regularization by L^2 also gets a adaptive learning rate. Thus weight decay and L^2 decouple, where by weight decay the meaning is the explicit penalizing of large parameters. This was noticed in [38], and they proposed AdamW to reintroduce the unmodified weight decay into AdamW. This entails removing the L^2 term from the loss function, and calculate the update it gives as a separate step in the optimizer.

RAdam

It is found empirically that it is important for Adam to have some form of warm-up of the learning rate. This means keeping the learning rate low for the first epoch, and slowing increasing afterswards. This is done in order to stabilize the estimates of the second momentums. In an attempt to incorporate warm-up more directly into Adam, Rectified Adam (RAdam) was created [40]. The difference is that we introduce a term, ρ_t:

$$\begin{aligned} \rho_\infty &= \frac{2}{1 - \beta_2} - 1, \\ \rho_t &= \rho_\infty - 2t \frac{\beta_2^t}{1 - \beta_2}, \end{aligned}$$

which we can use to calculate r_t to replace the warm-up heuristic as:

$$\begin{aligned}
&\textbf{if} \qquad\qquad \rho_t > 4 : \\
&\qquad\qquad l_t = \sqrt{\frac{(1-\beta_2^t)}{v_t}} \\
&\qquad\qquad r_t = \sqrt{\frac{(\rho_t - 4)(\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_t}} \\
&\qquad\qquad \theta_t = \theta_{t-1} - \alpha_t r_t \hat{m}_t l_t \\
&\textbf{else} \\
&\qquad\qquad \theta_t = \theta_{t-1} - \alpha_t \hat{m}_t
\end{aligned}$$

AdaBound

Now AdaBound [41] is another algorithm that tries to be the heir to ADAM. AdaBound tries to bridge the gap between ADAM and SGD by literally closing it as the optimizer iterates. This is done by clipping the parameter update to lie between two bounds, which you let converge. Thus, when the upper and lower bound converge there is no adaptability left in the learning rate, leaving one with SGD. The main changes can be written as:

$$\begin{aligned}
\hat{\eta}_t &= \text{Clip}(\frac{\alpha_t}{\sqrt{v_t}}, \eta_l(t), \eta_u(t)) \\
\eta &= \frac{\hat{\eta}_t}{\sqrt{t}} \\
\theta_{t+1} &= \theta_t - \eta \hat{m}_t
\end{aligned}$$

Where $Clip(x, \cdot)$ output forces x to lie between to two boundaries given. For boundaries they use:

$$\begin{aligned}
\eta_l(t) &= 0.1 - \frac{0.1}{(1-\beta_2)t + 1}, \\
\eta_u(t) &= 0.1 + \frac{0.1}{(1-\beta_2)t},
\end{aligned}$$

as the converging upper and lower bound.

AdaBelief

AdaBelief is a brand new version of ADAM [42]. The main change from ADAM is that here they use the centred second order moment. Thus we have:

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot (g_t - m_t)^2$$

They find that this simple change, with no extra parameters introduced, gives significantly better performance.

AdamRNW

Since all the changes mentioned in NAdam, AdamW and RAdam seemed well justified, and somewhat orthogonal, they were combined into AdamRNW, which was tested along the other algorithms. RAdam already had weight decoupling as an option, so my contribution was only to add Nesterov momentum to it.

Comparison

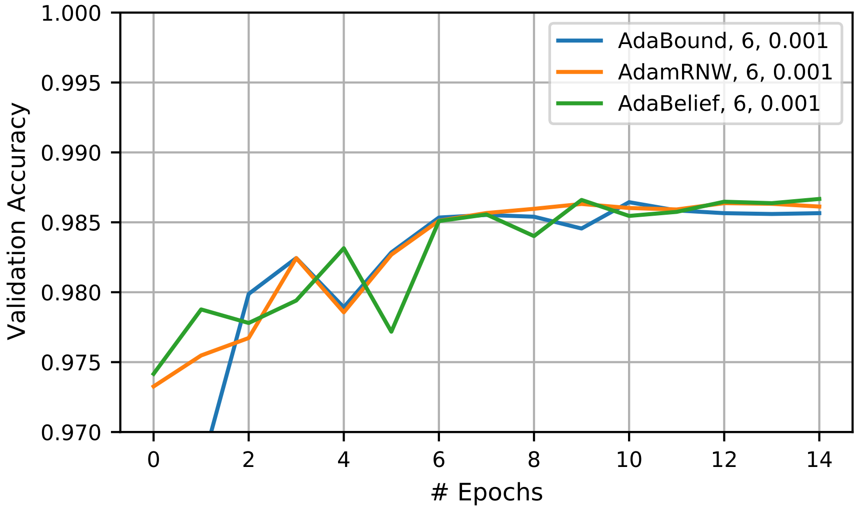


Figure 5.1: A comparison of the three versions Adam that I thought was in contention to be used. Plotted is the mean of a 5-fold validation accuracy as a function of epochs.

AdaBound, AdamRNW and AdaBelief is compared in Fig. 5.1. The learning rate used, 0.001, was the same for all, and as such this is not that conclusive. But it seems that they all have very similar performance, and as such I felt justified in using my own, AdamRNW. For these result I divided the learning rate by 5 every sixth epoch.

CONTENTS

NEURAL NETWORKS 6

The term model has been mentioned countless times by this point, but the reference is to some specific kinds of models. We have worked with variations of neural networks for this thesis. The model is built of consecutive layers which represents compositions of functions. Each layer consists of a number of nodes, or featuremaps, which is calculated based on the connections in our network.

In the first section neural networks are introduced with two different structures of networks, namely the Fully Connected Neural Network (FCNN), and the Convolutional Neural Network (CNN) (§ 6.1). Then we will look at the activation functions, which are crucial to the elevate the models from their linearity (§ 6.2). Important for larger models with many layers is the way one normalizes the inner workings. Thus we will look at batch normalization and weight normalization as two options (§ 6.3). Following last chapter we know that regularization plays an important role in the capacity of the model, one way to introduce regularization in the model is introduced in (§ 6.4). Finally the model used is presented (§ 6.5).

6.1 NETWORK CONNECTIONS

In some sense neural networks are an extension of modelling hyper-planes, with the addition of some non-linearity on top. The next issue is how to connect these linear functions. Below we will look at two ways, fully connected neural networks, and a special case of these which is the convolutional neural network. They are called neural since each output, of a linear sum with an activation function on top of it, resembles a neuron. For our purposes we will only look at feedforward networks. In feedforward the input is transformed through different layers to a layer giving a final output. Crucially, the connections can only go from an earlier layer to a later. Networks which allows backwards connections or recurrent connections exists and are called recurrent neural networks. In the model two types of connections between layers are used.

6.1.1 Fully connected neural network

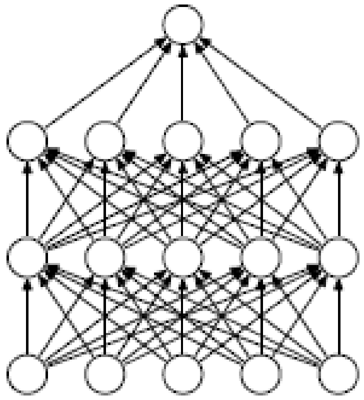


Figure 6.1: A FCNN with 2 hidden layers. The arrows represent a weight. We see that each neuron from each layer is connected to all the neurons from the previous. Illustration from [44]

The basic building block of a Fully Connected Neural Network (FCNN) is a linear combination of all inputs to all outputs. This can be represented for each node i as:

$$\begin{aligned} z_i^{(l+1)} &= \bar{w}_i^{(l+1)} y^l + b_i^{l+1}, \\ y_i^{(l+1)} &= \sigma(z_i^{(l+1)}), \end{aligned}$$

with l representing a layer, and y^0 being the inputs. The weights $w_i^{(l+1)}$, with the bias term b_i^{l+1} can be collected into one matrix $\mathbf{w}$. $\sigma(z_i)$ is the activation function which is responsible for introducing the non-linearity, and is something we will return to in the next section. The fully connected part is due to the fact that each element of y^k depends on all features of the preceding layer u^j, see Fig. 6.1.

6.1.2 Convolutional neural network

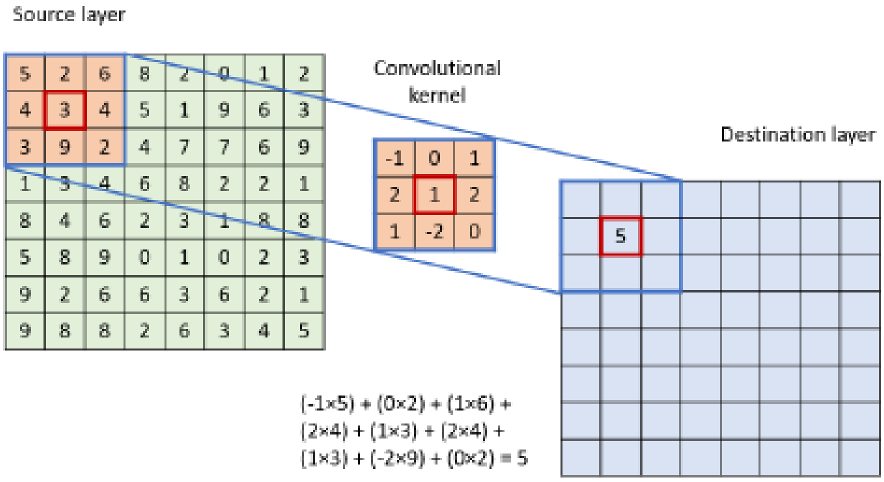


Figure 6.2: The result of a 3×3 convolution is shown. Illustration from [45].

Convolutions of images with a certain filters or kernel has been standard practice in computer vision for many years. Convolutions work by apply a filter as can be seen in Fig. 6.2. Formally, applying a convolution consisting of a given kernel, $K(k, l)$ to a image of some dimensionality, $I(i, j)$ one gets a new featuremap, $FM(i, j)$, of the size:

$$FM(i, j) = \sum_k \sum_l K(k, l) I(i + k, j + l),$$

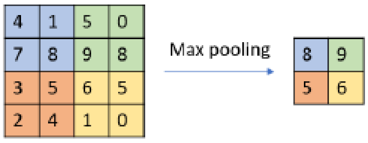
where the sum is over the kernel.[1] The kernel is applied to all areas of the input dependent on the stride and padding used. Stride determines the

[1] This is techically the cross-correlation, which is related to the convolution by flipping the kernel.

interval with which the kernel is used. Padding is the addition of zeros to the edges to possibly allow moving the center of a kernel along the edges. To get a CNN we stack layers of convolution with activation functions applied in between. With convolutions in the network you can learn the traditional feature engineering part of computer vision itself, without having to do it explicitly manually. Thus, using convolutions, one gets a new way of creating linear combinations of inputs, with the big difference being that each new feature created has some sense of restriction of neighbourhood of influence. Furthermore, each kernel is applied many times to the input in different places. In that sense it is a strong inductive bias to create nodes with weight sharing. [see 8, chapter 9]

6.1.3 Max-pooling

When applying max-pooling of a size 2, 2 with a stride of 2, which is what we use, to a matrix we divide it into 2, 2 squares and take the maximum of each as the result. An example can be seen in Fig. 6.3. The resulting dimensions size of the featuremaps is reduced by two in each direction. Thus, max-pooling is a way to reduce the dimensionality of the feature maps. Taking the maximum also introduces some kind of invariance to small local translations. This can of course be generalized to kernels of any size and stride.



4	1	5	0
7	8	9	8
3	5	6	5
2	4	1	0

8	9
5	6

Figure 6.3: An example of using max-pooling with a kernel of 2, 2 and a stride of 2. Illustration from [45].

6.1.4 Backpropagation

Backpropagation is basically the chain rule of derivatives applied at large. As mentioned, everything is implemented in PyTorch which handles the backpropagation automatically. Thus, I have chosen not to derive backpropagation for each module presented, but instead refer interested readers to [8], and the other cited sources of this chapter.

6.2 NON-LINEARITY

The non-linearity is introduced with the activation functions. Why activation functions? The activation functions are the key to the neural network. Without these, consecutive layers of linear combinations of features would only result in one grand linear combination. Thus the activation function, and more generally, the algorithms used between layers of linear combinations or filters, are the key to introducing non-linearity to the model. The activation functions have historically taken many different shapes and even to this day new functions are continually developed. We introduce the one we have been using, Rectified Linear Unit (ReLU) [46].

6.2.1 ReLU

In the earlier days of deep learning the activation functions used were the sigmoid, or tanh [47]. A simpler alternative exists which is the ReLU. The simplicity makes the network get sparse representations. Furthermore, the computations are cheaper with the ReLU, and it reduces the chance of vanishing gradient [48]. The ReLU can be formally written as:

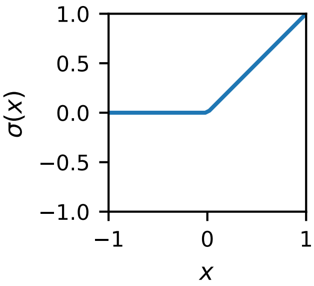


Figure 6.4: A plot of ReLU.

$$\text{ReLU}(x) = \sigma(x) = \max(x, 0).$$

A plot of the activation function can be seen in Fig. 6.4. The simplicity of ReLU makes it extremely efficient and it has all but replaced earlier activation functions such as the sigmoid. Since its inception, various new modifications to ReLU has been proposed. Nevertheless, I ended up using the ReLU and thus we will not take a deeper look at activation functions. One possible downside to this choice is readily apparent in the fact that the function can turn negative inputs off, thus leading neurons to become inactive. Thus for the last layer ReLU is not used.

6.3 NORMALIZATION

We have already seen that standardization of the inputs to the model improves the results (§ 4.2.1). When constructing models consisting of multiple layers, each layer serves as an input to the next, and one might naturally think that standardization might also have a role to play here. Furthermore, these inputs change during training so some running standardization might be needed. This is defined as Internal Covariate Shift in [49], and their proposed solution is the addition of Batch Normalization. Another way to normalize is to use Weight Normalization [50] which reparametrizes the weights in a smart way. Below we will explore both these approaches.

6.3.1 Batch Normalization

Batch Normalization [49] introduces another set of parameters to learn, namely the parameters to normalize each internal input. Typically applied directly after the convolutional operation, for each output x of a convolution we have for $BN_{\gamma,\beta}(x)$ [51]:

$$BN_{\gamma,\beta}(x) = \gamma \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

where γ, β are learned parameters, and ϵ needed for numerical stability. During training μ_B, σ_B^2 is the mean and variance estimated for the mini-batch. Thus Batch Normalization both normalizes outputs, and at the same time introduces a shift and scaling after the normalization, to keep the networks expressive power. For inference, the mean and variance are fixed, and are population wide estimates. This is done by keeping a running mean of μ_B, σ_B^2 during training as:

$$\begin{aligned} \mathbb{E}[x] &= \mathbb{E}_B[\mu_B] \\ \mathbb{V}[x] &= \frac{m}{m-1}\mathbb{E}_B[\sigma_B^2] \end{aligned}$$

Thus for inference we get:

$$BN_{\gamma,\beta}^{inf}(x) = \gamma \frac{x - \mathbb{E}[x]}{\sqrt{\mathbb{V}[x] + \epsilon}} + \beta$$

This is an enormously successful addition to deep models, and is used everywhere in deep convolutional neural networks [51] [52].[2]

6.3.2 Weight normalization

Weight normalization[50] can be viewed as an alternative to Batch Normalization[3]. We split the vectors resembling each weight into two components, namely a length and a direction. This means that for a weight $\mathbf{w}$ from some operation of our total weights θ we reparametrise it into:

$$\mathbf{w} = \frac{g}{||\mathbf{v}||}\mathbf{v}.$$

Thus we have increased the numbers of parameters by one by decoupling the length of weight, g, from the spatial orientation, $\mathbf{v}$. This is a normalization of the weights as well, and [50] find it to speed the convergence of the model up.

6.4 REGULARIZATION

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error by

[2] So successful that as of writing this [49] has been cited ≈ 22000 times since 2015.

[3] Both can be used at the same time, so not an exclusive alternative.

constraining the capacity. In § 5.2.2 we looked at how this could be done with the loss function. It is also possible to incorporate regularization directly in the model which is what we will look at now.

6.4.1 Dropout

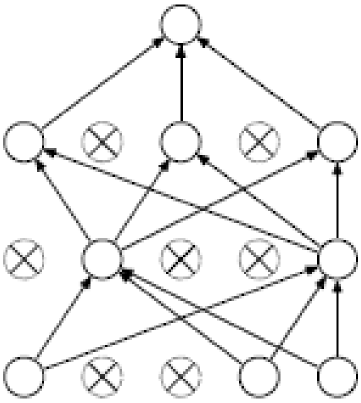


Figure 6.5: Dropout visualized. The arrows, representing weights, are randomly dropped during training. Illustration from [44]

Dropout [53] is a technique that aim to increase a neural networks ability to generalize. It is related to the method of adding noise to the hidden units of a network and can be considered as a form of stochastic regularization. Dropout was shown in [44] to improve generalization performance on many different tasks and it was thus ideal for use in our case as well.

It works by "dropping out" a percentage of the neurons during training as seen in Fig.6.5. This in turn ensures that no neurons can freeload, and that each neuron contribute. Furthermore, overfitting generally results from many neurons linking tighty, which is much harder since it is now random during training which links are active, and thus gets updated at the same time. In effect this lets one train multiple networks at the same time since different neurons will be turned off at each iteration. These networks are highly correlated with each other, but it let us think of it as an ensemble method. This makes the feedforward operation during training, and assuming a FCNN:

$$
\begin{aligned}
r_j^l &\sim \text{Bernoulli}(p) \\
\tilde{y}^l &= \bar{r}^l \cdot \bar{y}^l \\
z_i^{(l+1)} &= \bar{w}_i^{(l+1)} \tilde{y}^l + b_i^{l+1} \\
y_i^{(l+1)} &= f(z_i^{(l+1)})
\end{aligned}
$$

During testing, all the neurons are activated at the same time and their output is then scaled in order to reflect the increased number of connections compared to during training.[4] If the dropout percentage is set to p the output is scaled by p. Using Batch Normalization with Dropout at the same time creates problems due to both modifying the variance of the output [54]. A workaround is to use Dropout for FCNN, and use Batch Normalization for the convolutional layers.

6.5 MODEL

A visualization of the final model can be seen in Fig. 6.6. The model can be divided in two. There is a feature engineering part made of CNNs, and

[4] Extra in the sense that all connections are used.

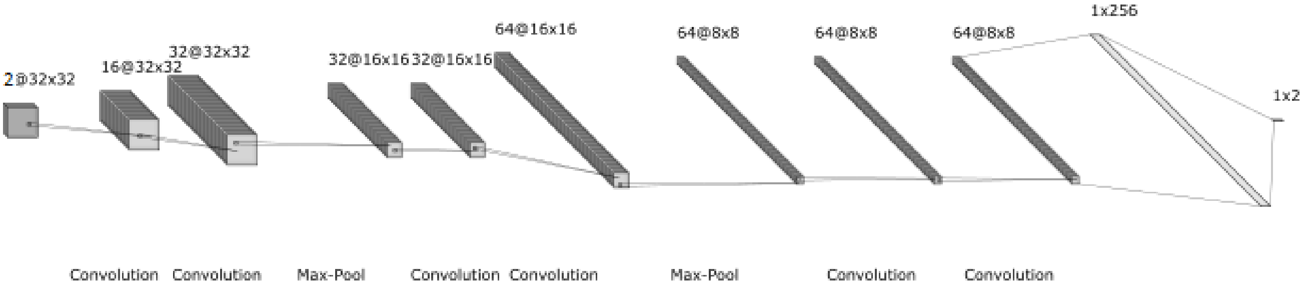


Figure 6.6: An overview of the final model. Created with the tool: https://alexlenail.me/NN-SVG/LeNet.html.

a classification part of FCNNs. Both used ReLU as the activation function except for the last layor which had none.

Feature engineering

I used three convolutional blocks, each consisting of two 3 × 3 convolutions with a stride of 1 and padding of 1. After each of the first two blocks I applied max-pool. This was to make the effective field of view of the last 8 × 8 featuremap cover the whole window. As max-pool decreased the size of the featuremaps I increased the number of channels from 2 → 16 → 32 → 64. I used Batch Normalization after Weight Normalization for the convolutional layers.

Classification

For the classification part I unrolled the 64 × 8 × 8 features into one vector to which I had two layers of FCNNs. I used Dropout after the first FCNN layer.

Hyper-parameters

The model was created using *AdamRNW* (§ 5.3.2) trained on **Mixed** (§ 3.1). I selected the model based on the best mean of 5-fold validation accuracies. Furthermore a grid search was made over the learning rate, $\eta \in [0.001, 0.002, 0.008, 0.016]$, and batch size $\in [64, 128, 256]$. For the final model I did not use either label smoothing or artificial foreign objects. Neither improved the validation accuracy. The case for artificial foreign objects is explored further in § 7.2.2.

CONTENTS

FOREIGN OBJECT DETECTION 7

This chapter presents the final results and insights gained from evaluation of our final model. In § 7.1 an evaluation of our Convolutional Neural Network (CNN) architecture is presented, with various ways to evaluate the models performance shown. In § 7.2, a more nuanced analysis of how the limited datasets might limit model generalization performance is presented.

7.1 EVALUATION OF MODEL

The model being evaluated in this section is specified in § 6.5. As a reminder, the overlapping sliding windows of $32x32$ are used to preprocess the images, meaning every pixel is evaluated anywhere from 1 to 4 times, with 4 being the norm for any pixel more than 16 pixels from any border.

A plot of the convergence of the final model can be seen in Fig. 7.1. The network was trained for 15 epochs. During training the loss was measured for each 50 mini-batches (plotted in blue), and after each epoch the loss and the accuracy on the validation set was calculated (plotted in orange and green respectively). The best accuracy on the validation set is achieved after 9 epochs of training, and the model corresponding to this epoch was chosen as the final model.

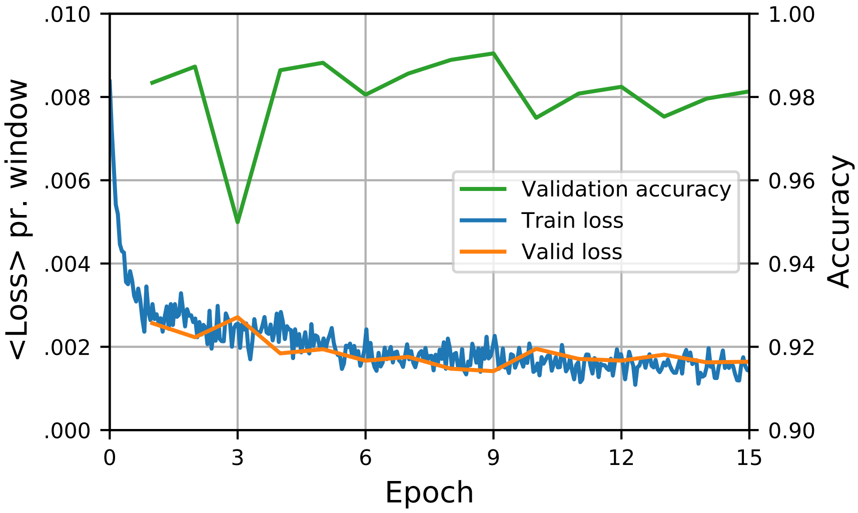


Figure 7.1: A figure showing training (Blue) and validation (Orange) loss as a function of the number of epochs trained. Superimposed is the validation accuracy (Green) again as a function of epochs. The final model was chosen to maximize validation accuracy, which was at epoch 9.

7.1.1 Test set evaluation

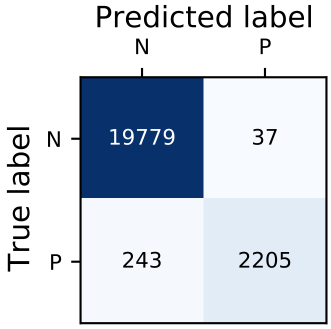


Figure 7.2: Confusion matrix for best model on test data. The final accuracy is 98.74%.

Testing on the full **Test Set** gave an accuracy of 98.74%. To summarise the overall binary performance on a window basis, we have also plotted the confusion matrix in Fig. 7.2. The confusion matrix reveals a large number of False Negatives, with a total of 243 windows. We know that the sliding window approach in my implementation is expected to give a number of cases where the window gets a False Positive label. This is due to the binary masks being square, and potentially used to cover round foreign objects, thus leaving the corners with the wrong truth value. This explains to some degree the inflated number of False Negatives, but to what exact degree is not explored here.

To illustrate the trade-off, inherent in binary prediction tasks, between false negatives and false positives, we introduce the Receiver Operating Characteristic Curve (ROC-CURVE). It is defined as the true positive rate as a function of the false positive rate. The model outputs a score for each binary category, and the final prediction is the maximum of the two outputs. This is equal to taking the difference of the two outputs, and having a decision threshold of 0. To create the full ROC-CURVE we will instead vary the this threshold, in order to go from a false positive rate of 0 to 1.[1] The result is seen in Fig. 7.3. The ROC-CURVE is plotted in blue, and the rates corresponding to a threshold of 0 is plotted as the orange dot. The Area Under the ROC-Curve (AUC) is 0.981, with a theoretical maximum of 1.0. In the bottom right corner is shown a close-up showing how the ROC-CURVE changes around the threshold of 0. We see that when we optimize for high accuracy we optimize for false positive rate as opposed to true positive rate.

[1] With a false positive rate = 0 corresponding to 0 false positives, which in this case means no positive predictions. This would return the accuracy to its' baseline of never predicting foreign objects.

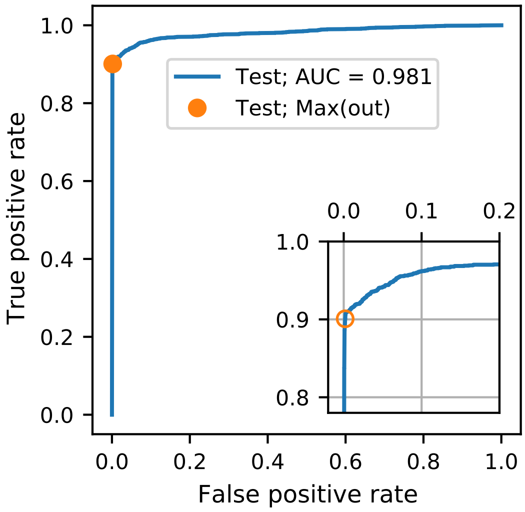


Figure 7.3: ROC-CURVE (Blue) for the outputs of the final model. The area under the curve (AUC) is 0.981. In the bottom right corner is a visualization of the top corner. The rates, when the prediction is the maximum of the two outputs, is plotted as the orange cirle.

Taking a more detailed look at the errors, we have plotted a logarithmic histogram over the different between the softmax of the output for both True predictions and False in Fig. 7.4. In the ideal case the True predictions (Blue) would be U-shaped, and the False predictions (Orange) would be centered around zero. [2] Taking into account the the histogram is logarithmic, the extremes of the True predictions is roughly a factor of 10 larger than their neighbours resulting in a U-shape. Unfortunately the errors does not seem to be centered around zero. As mentioned above, some of this might be due to the labelling, however, this has not been investigated further.

[2] Meaning that the model itself had a "hard time" deciding.

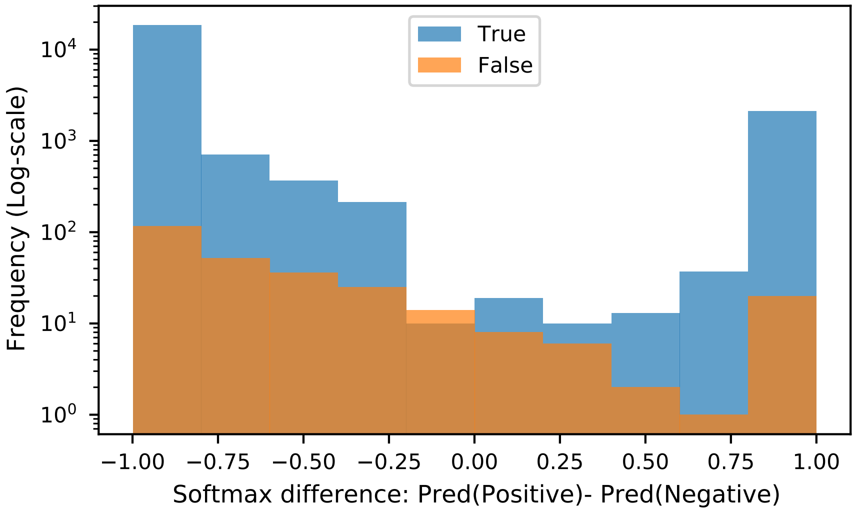


Figure 7.4: Histogram over True and False predictions as a function of the softmax normalized output difference.

7.1.2 *Occlusion test*

In order to visualize what the model actually uses to determine its predictions, an occlusion test following [55] was made. The idea is to occlude one area of the picture at a time, and then compare the prediction of this new image with the one from the un-occluded image. This gives you an intuition for which areas of an image that are important for the final prediction since these will potentially change the prediction. So in short, one generates a full test set from one single image containing all the possible combinations of occluded areas. Then for each pixel in the original image one visualizes which fraction of it being occluded predicts positive. In this way, each pixel gets a value from 0.0 to 1.0 which when plotted shows the importance of various areas.

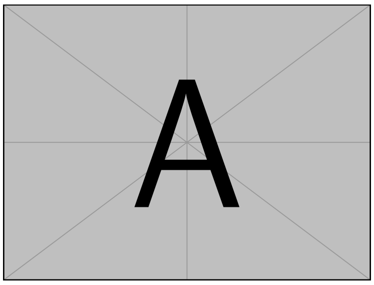


Figure 7.5: An example of one occluded image out of the 1849 images generated for the test.

In this case the occluded area was $12x12$ pixels. The size was chosen in order that the occlusion could cover any one foreign object, while still being small compared to the full image. Then $(32 + 12 - 1)^2 = 43^2 = 1849$ new windows were generated from one test image, each having a different placement of the occlusion square. An example of one of these can be seen in Fig. 7.5. We chose the occlusion to take the value of the local minimum in the window. The window fed to my algorithm is encased in the red square. The padding needed to have all variations of occlusions is shown in dark blue. Since the square is $12x12$, there is 144 predictions pr. pixel. In the ideal case we would see that only the foreign objects should change the resulting prediction when occluded. Three windows from the **Test Set**, shown in Fig. 7.6 in the upper row, on the same intensity scale across the row. The first (a) is a true positive prediction of the model, the second (b) a false positive, and the third (c) a false negative. On the second row is plotted the importance of each pixel, with a corresponding colorbar below. The importance being the fraction of times occluding the pixel changes the output.

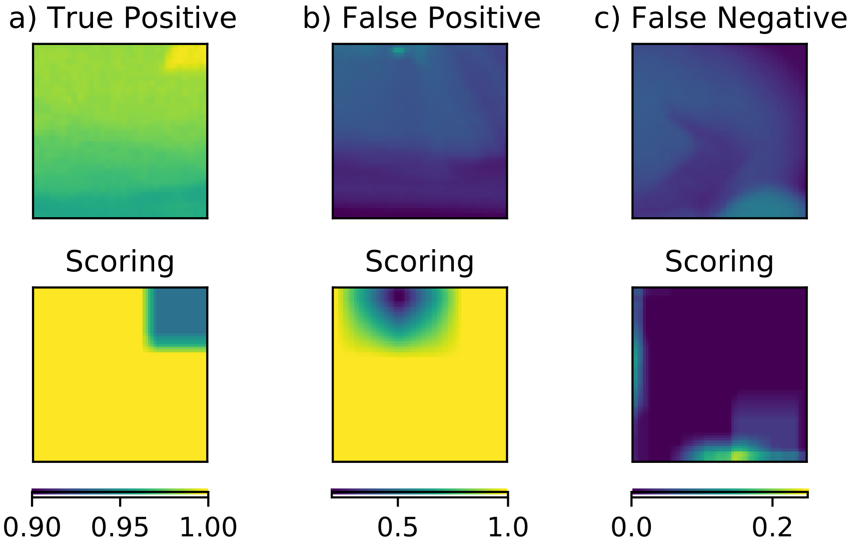


Figure 7.6: Three cases representing a true positive (a), false positive (b) and false negative (c) example is plotted in the top row. Below each is plotted the result of an occlusion test with an occlusion square of size (12,12) with the value of the minimum of each window.

We can see that the true positive case behaves ideally. Namely, the prediction only changes, from positive to negative, when the foreign object, in the upper right corner, is covered. For the false positive case we can also see clearly the pixels the model considers to be a foreign object. On the raw image, even to the naked eye, this could look like a foreign object. This is clearly a hard case to predict, and in some sense the model is justified in getting this wrong. Finally, for the false negative case, the picture is more muddled. First of all, the supposed existence of a foreign object is very hard to see with the eye. This could perhaps be a case of wrongly labelled data. Nevertheless, the occlusion test is perhaps the most informative in this case. We see that covering the upper right portion of the image leaves the prediction unchanged, which makes sense. What perhaps does not, is that covering either the left or the bottom part makes the model predict a foreign object. One explanation for this could be that the occluded square creates a strong artificial gradient, which perhaps is enough for the model to start changing predictions. Why this is more prevalent on the edges of the image is hard to say though, but

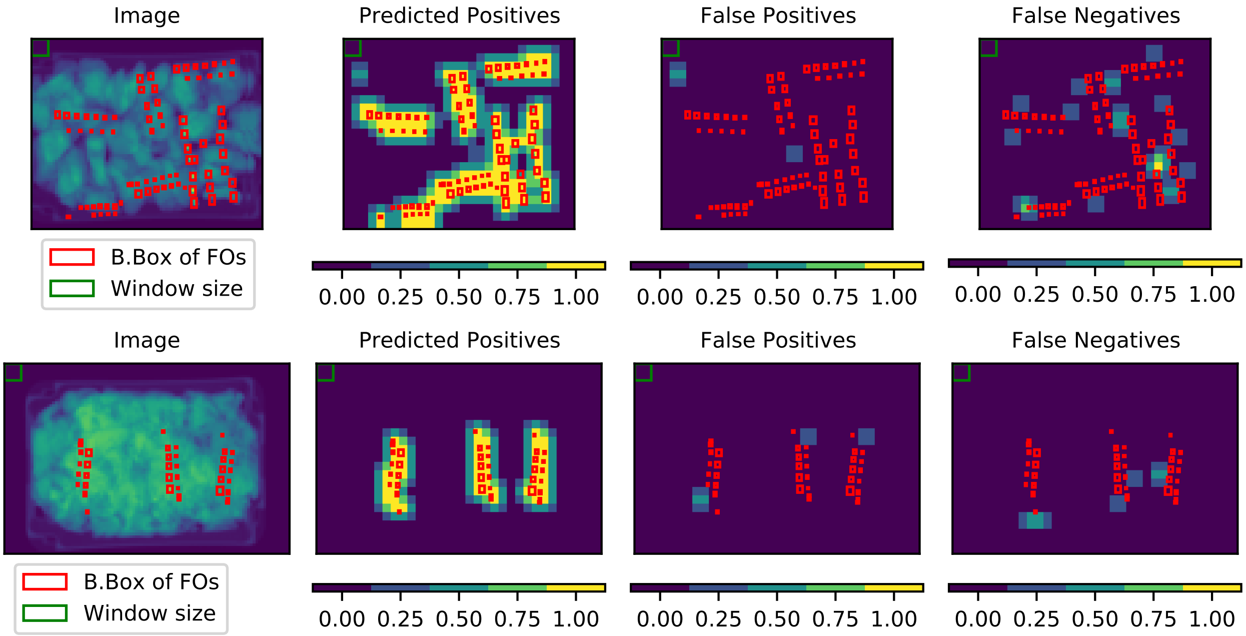


Figure 7.7: The model applied on the two full images. The first row is one the images where the performance was worst, the second row an image where the performance was one of the best. The images themselves are plotted in the first column, with my labelling of bounding boxes of foreign objects (red) on top. In the second column the windows predicting a foreign object is plotted, each contributing 0.25. Due to the overlap, up to 4 windows can predict positive for each area giving a maximum value of 1.0. In the third column we have visualized the windows resulting in false positives, and the fourth column we have the false negative predictions.

one could speculate that it is harder in general to predict on the edge, which shows that the model is vulnerable in these areas.

7.1.3 *Full picture evaluation*

So far, we have only looked at model predictions per window. The windows were generated using the sliding windows approach with 50% overlap, which in effect means that each area is in fact evaluated 4 times[3]. This implies that evaluating one window at a time is misleading, since they are not independent. In general, around 40% of the full pictures in the **Test Set** get a perfect score. These all have no foreign objects though, and mind that a couple of empty full pictures gets some false positives. One could imagine extending the model to combine the different predictions for each 16x16 square, potentially increasing the performance.

To investigate this, we visualize the performance of the model on two full pictures in Fig. 7.7. Each row corresponds to a different input picture. For the top row I have chosen one of the images with the worst performance, for the second row I have visualized one the images with foreign objects with best performance. The low energy channel of the images are plotted in

[3] In fact, only once for the corners, twice for the rest of the edges, and 5 times if there is some extra overlap in some areas due to the sliding windows size not dividing the full picture dimensions.

the first column. All the images have my labelling of the foreign objects, as bounding boxes, superimposed in red. In green I have shown the size of an $32x32$ window.

In the second column, we have plotted the windows which were predicted positive. Here the colorscale corresponds to the ratio of overlapping windows predictions being positive. In both rows, there seems to be a very good overlap between the positive predictions (yellow areas) and the foreign objects (red boxes). Furthermore, not a single foreign object is, in these two examples, not spotted by at least one of the windows.[4]

In the third column, the windows resulting in false positive predictions are shown. A clear example of a false positive prediction is seen in the upper left corner in the first row image. On the other hand, we can see that not a single time do we have more than two consecutive windows giving a false positive prediction. At least for these examples.

In the fourth column the false negative predictions are shown. We see that a fair amount of false predictions belongs to windows that just barely overlaps a bounding box. These are the False Positive labels that as mentioned would show up as "impossible" predictions. In the top row there are a few cases where a foreign object is overlapped centrally and not found. These can not be explained away, and are more troublesome. In general, there are very few areas which get all four predictions wrong. Thus, this shows that qualitatively the model seems to work reasonably well.

7.2 BOUNDS ON GENERALIZATION

This section explores some of the limits of the model. The idea is to see how the model performs outside of the training regime.[5], in some controlled fashion First, in § 7.2.1, the model is tested on **Uniform**. This is a dataset where it is easy spot the foreign objects with the naked eye, due to the background being uniform. Thus we would expect the model to perform well even though it is tested on data not represented in the training sets. Next, § 7.2.2 investigates the connection between training set and generalization ability. We look at the generalization by testing on datasets different from training[6], with and without the introduction of artificial foreign objects from § 4.3.

7.2.1 Uniform backgrounds

Uniform, introduced in § 3.1, is the dataset with a simple, almost uniform, background of varying thickness with two areas containing phantoms. The

[4] which would show as a red square on the darkest blue background.

[5] This is also called domain shift.[56]

[6] Using the three datasets that make up **Mixed**.

overall accuracy for the model evaluated on this dataset is 0.962 compared to a baseline of 0.870.[7] This is 0.025 lower than the score for the **Mixed Test Set**. This is mainly due to the fact that the model was not trained on these images. Nevertheless, the dataset itself was supposedly "easier" to predict on, given that the background is comparably uniform. One could imagine that this accuracy depends on the thickness of the background, since increasing thickness reduces the signal-to-noise ratio of the foreign elements. This relationship is plotted in Fig. 7.8. Shown is the full picture accuracy for the 47 pictures in **Uniform**, with the mean of the full picture accuracy as a function of the number of POM plates[8] plotted in blue and orange, respectively. The errorsbars represent the standard error on the mean. It initially seems hard to conclude that varying the thickness in itself affects the performance of the model, even though there is a trend showing degrading performance as the number of POM plates increase past 11 plates.

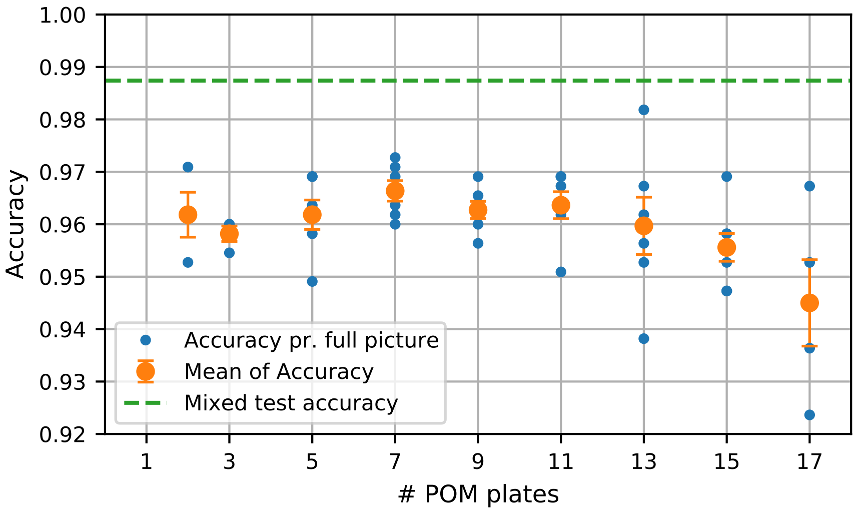


Figure 7.8: Thickness (# POM plates) vs Accuracy. In blue is shown the single picture accuracy (ratio of correctly predicted windows). The mean is plotted in orange, with the standard error on the mean shown as symmetrical errorbars. The green line is the accuracy, of 98.74%, on the full **Test Set**.

Taking a closer look at the drop in accuracy, we can repeat the full picture evaluation from last section, § 7.1. The full picture with the lowest accuracy, of 0.924, is chosen to visualize the performance of, in Fig. 7.9. As in the previous section, the low energy channel of the image is plotted in the first column. My labelling of the foreign objects, as bounding boxes, is superimposed in red. The green square shows the size of a 32×32 window.

In the second column, we have the windows which was predicted positive. The performance seems to have worsened, compared to Fig. 7.7. Many positive predictions do not overlap with any red squares. This is confirmed when looking at the false positive predictions, in the third column: The number of false positives have increased. Finally, roughly half of the false negative predictions, plotted in the fourth column, shows the same behaviour as in Fig. 7.7. These are the rightmost phantoms, and they barely overlap a

[7] The baseline from predicting no foreign object on every window.

[8] i.e. thickness of background.

bounding box. These could be artefacts of the sliding window approach. The other half, on the left side, shows the model missing some foreign objects.

The network does not make any mistakes on the background outside the box being scanned. This was also the case in the examples in Fig. 7.7, and in that sense, the network has learned to spot uncovered conveyor belt.

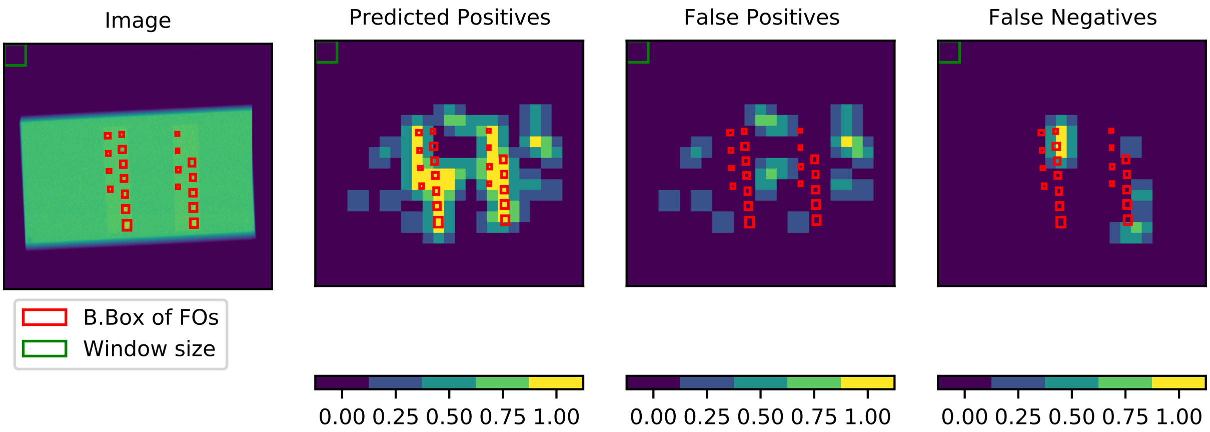


Figure 7.9: The model applied on an image belonging to **Uniform**. I have chosen the image where the model performed the worst. In the first column the low energy channel of the image is plotted, with my labelling of bounding boxes of foreign objects (red) on top. The green square in the corner indicates the sliding window size. The second column shows the windows predicting a foreign objects, each contributing 0.25. Due to the overlap, up to four windows can predict positive for each area giving a maximum value of 1.0. The last two columns show the false positives and false negatives, respectively.

7.2.2 *Dataset dependence*

This subsection studies the ability to generalize of the CNN approach trained on limited data. After all, all the available data is lab data, in the sense that it is created explicitly for developing and testing algorithms. This lab data has to be representative for the underlying distribution for the reported test accuracies to be trustworthy. Thus, it is naturally of interest to know whether your training sample is biased, and what the possible consequences would be. It is unknown to what extend the lab data is biased compared to the eventual use-cases of the Meat Master II. On the other hand, we have tried to analyse the possible consequences of bias by simulating the case of partial data.

We split the **Mixed** dataset into its constituent parts: **Cirles** (**Ci**), **Squares** (**Sq**) and **Pens** (**Pe**). Then, the model was trained on each partial dataset

Accuracy	Trained on **Ci**	Trained on **Sq**	Trained on **Pe**
Test **Ci** w/ AFO=0	0.9940	0.9897	**0.9522**
Test **Sq** w/ AFO=0	**0.9379**	0.9951	0.8246
Test **Pe** w/ AFO=0	**0.9594**	**0.9694**	**0.9778**
Test **Ci** w/ AFO=1	**0.9948**	**0.9917**	0.9433
Test **Sq** w/ AFO=1	0.9195	**0.9957**	**0.9357**
Test **Pe** w/ AFO=1	0.9579	0.9677	0.9772

Table 7.1: Accuracy for different combinations of training and test datasets. The columns represent the data trained on, with the rows being the data tested on. **Ci**, **Sq** and **Pe** are short for **Cirles**, **Squares** and **Pens**. The table is divided in two, with Artificial Foreign Objects added (AFO=1) and without (AFO=0). In the diagonals (blue background) the model is evaluated on the corresponding validation set. The best accuracy between the two tables with and without artificial foreign objects is written in bold font.

alone, and then evaluated the resulting model on all three datasets.[9] Thus, this is an attempt to approximate the potential real world case of training the model on data from a biased sample. Furthermore, this might give some insight into which dataset is the most powerful one to generalize from. Finally the addition artificial foreign objects is evaluated in this context as well. In order to make a fair comparison between the binary choice of whether to use artificial foreign objects, we optimized an extra hyperparameter in my runs. Namely, the weight given to the positive labels in the loss functions. This weight rebalance the ratio of positive to negative labels of the windows in the loss function. Introducing artificial foreign objects also moves this balance by turning 20% of the negatives to positives. This potentially makes the extra weighting of positives unnecessary for convergence. For each dataset we tested three different weights $1, 2, 4$, and chose the optimal one using cross-validation. Thus for the following we used a weight of 2 when training without artificial foreign objects. With artificial foreign objects the weight was 1 for training on **Squares** and **Pens**, and 2 when training on **Circles**.

The accuracy is presented first. Table 7.1 shows the $3 \times 3 \times 2 = 18$ different accuracies the models obtains.[10] The bold font marks the best performance with and without artificial foreign objects. We notice two things: First, in both tables, the accuracy drops off the diagonals. This shows that all performance drops when evaluated out of distribution. This drop in accuracy is especially pronounced when the model was trained on **Circles** or **Pens**. The smallest drop in accuracy is when the model is trained on **Squares**. This shows that the training dataset needs to be carefully selected to represent the full range of foreign objects likely in meat samples. Secondly, it does not seem like the addition of synthetic data changes the lowering of accuracy off the diagonal.

Introducing more positive windows in our training can potentially move the balance between false positives and false negatives, which the accuracy does not reflect. To explore this the ROC-curves are shown in Fig. 7.10. When not trained on **Squares** (Column 1 and 3) the addition of artificial foreign

[9] When evaluated on the dataset it was trained with, we report the result from the validation set. Where as evaluating on a different dataset the entire dataset was used a validation set.

[10] 3 datasets used for training, the same 3 datasets as possible test sets, and finally with and without artificial foreign objects, 2.

AUC	Trained on **Ci**	Trained on **Sq**	Trained on **Pe**
Test **Ci** w/ AFO=0	**0.9957**	0.9781	**0.9860**
Test **Sq** w/ AFO=0	0.8955	**0.9785**	0.9166
Test **Pe** w/ AFO=0	0.9531	0.9521	**0.9781**
Test **Ci** w/ AFO=1	0.9920	**0.9825**	0.9834
Test **Sq** w/ AFO=1	**0.9434**	0.9763	**0.9697**
Test **Pe** w/ AFO=1	**0.9557**	**0.9526**	0.9779

Table 7.2: AUC for different combinations of training and test datasets. The columns represent the data trained on, with the rows being the data tested on. **Ci**, **Sq** and **Pe** are short for **Cirles**, **Squares** and **Pens**. *AFO* indicates whether the model used artificial foreign objects to create synthetic data. In the diagonals (blue background) the model is evaluated on the corresponding validation set. The best AUC between the two tables with and without artificial foreign objects is written in bold font.

objects drastically improves the curves when tested on **Squares** (orange). Table 7.2 shows this improvement reflected in the AUCs in the second row of each table. The difference AUC is approximately increased by 0.05 when using artificial foreign objects. The lower left triangle is of special interest, since this area represents training on a comparatively simpler dataset to the ones tested on. That the AUC is increased here is encouraging.

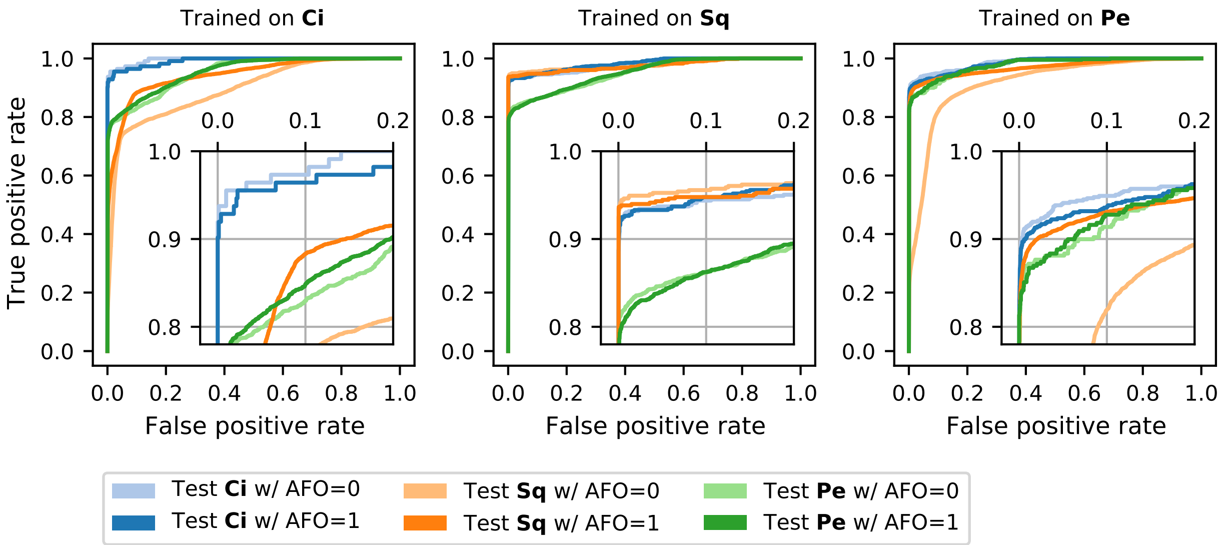


Figure 7.10: ROC-curves showing model performance. Each column corresponds to different training sets: **Cirles** (**Ci**), **Squares** (**Sq**) and **Pens** (**Pe**). The blue, orange and green curves correspond to testing on **Cirles**, **Squares** and **Pens**, respectively. The opaque curve corresponds to training without artificial foreign objects (AFO=0).

In summary, the performance drops whenever the model is applied outside of the training domain. This is the case when we apply it to some supposedly simple data such as **Uniform**. Furthermore, this is also seen for training on the constituent parts of **Mixed**. This is not overfitting since the model performs well on the dataset that it is trained on. The problem is instead that this performance does not translate to other datasets. We saw that measured on the accuracy we can not conclude the using artificial foreign objects improve our result. There was a clear effect on some of the ROC-CURVES though. These are in many ways a more thorough way to evaluate performance, and the increase here shows that Artificial Foreign Objects can help the modelling process.

contents

DISCUSSION AND CONCLUSION 8

This chapter is split into three sections: Discussion, Conclusion and Future Work.

8.1 DISCUSSION

We have not achieved the performance we had hoped for when starting the project. This can be due to limits of the data § 8.1.1 or of the model § 8.1.2.

8.1.1 Data

The central assumption of this work is that the available data represents the real world usage of the model. If this is not the case, the reported result will be severely biased. Some effects of this was explored in § 7.2. For the test on **Uniform** the drop in performance is not necessarily as bad as it seemed. These were the only background which was not meat. In that sense they were one large foreign object, and alternatively the algorithm should in fact predict all the windows of the whole image to be foreign objects. The real problem was discussed in section § 7.2.2. The performance, whether accuracy or Area Under the ROC-Curve (AUC), drops when the models are tested on unseen data. This shows that one has to be careful when collecting data to train on. An alternative to lab data would be a setup with online learning directly from the real world usage of the Meat Master II. This has its own set of challenges, but presumably the problem of domain shift is not one of them. For an overview of the general problems of domain shift I refer to [57].

Relabelling the foreign objects could increase the performance with three potential improvements. 1) The data should have been labelled using the polynomial method § 3.2. Our model managed to converge, but it was clear that the bounding boxes bordering foreign objects was a problem. 2) Another problem with the labelling was that I labelled all foreign objects equally. They were made out of different materials, steel and aluminium, since they were meant to represent different things, such as metal or bone. This extra information was lost. 3) Furthermore, three items were consistently left out of the labelling. Including these would have increased the potential of the algorithm.

The implementation of the artificial foreign objects could have been improved in at least three ways. 1) The idea with adding artificial foreign objects as a transformation was that it is possible to update this transformation on the fly. In that sense one could monitor for which parameters, size or shapes, the algorithm performed worst and then introduce more of these artificial foreign objects. I never got to implement this is in full, but this is one of the advantages of using the transformation approach. 2) To approximate the attenuation coefficients of metal we chose the four maximum values from each randomly drawn bounding box. This is rather arbitrary. Perhaps, it would have been better to model a distribution of attenuation coefficients to draw from. 3) We originally had a cut-off on the algorithm, to make sure that when the background was subtracted the lowest value would be around zero, otherwise the drawn bounding box would be discarded. This was removed since we normalized the images before creating examples without background.

8.1.2 Model

In machine learning it is common to test new results against baselines. The obvious baseline to use for this study would be FOSS' current implementation of their foreign object detection algorithm. Their algorithm is based on some clever thresholding. While, the algorithm would be nice to test against in theory, this would introduce new complications regarding secrecy. The lack of baseline makes it harder to interpret the accuracy of 0.9874. Based on my impression collaborating with FOSS I do still think it is safe to say that this accuracy is too low for real world usage. For FOSS' customers it is expensive to stop the production if the Meat Master II flags a crate of meat as containing a foreign object. Thus the false positive rate needs to be as low as possible from a cost standpoint. Off course the final consumer of the meat might be more concerned with the number of false negatives, and thus we are back at the classical trade-off problem.

The sliding window algorithm plays a large role in the shaping of the algorithm. But other methods could have been used to generate images for a Convolutional Neural Network (CNN). An alternative would be to use a random crop of the full images as input. This is then combined with rescaling to fix the size of the input to a common value. The number of available images was small, and the pixels have very consistent physical interpretation, i.e. the absorption of the beam. This made the sliding windows approach a good choice since there is no rescaling, and it lets us keep the images of a size corresponding to the foreign objects.

As a tangential project, the family of Adaptive Moment Estimations (Adams) was explored. My intial critia for choosing those to work with was that they should be orthogonal to each other, and not introduce too many new hyperparameters to tune. Unfortunately, it was not possible to incorporate

AdaBelief into AdamRNW due the time of publication of AdaBelief [42]. I chose a small subset of Adams to work with, and this choice was guided by what I perceived to be the most promising. This belief was informed by reading [42] and [41], but there are a few alternatives of special interest. The first is Yogi [39] which is similar to AdaBelief, and the second is Ranger [58] which is more like my mix of ideas in AdamRNW.

8.2 CONCLUSION

We have trained a model to predict foreign objects with an estimated accuracy of 98.74% on 32 × 32 windows. This, I believe, is not good enough for real world use, but some interesting observations have been made along the way. There is a very real detrimental effect of testing on data outside of our training distribution. Thus, in order to train a working model, both the labelling and the training sample has to represent the real world usage closely. It is possible to create simulated data of artificial foreign objects that increases performance as measured by the AUC.

8.3 FUTURE WORK

The preprocessing of the images from raw outputs to final absorption measurement was done entirely by FOSS. This step is crucial for the signal to noise ratio, and to remove systematic variations caused by imperfect detectors. I believe that implementing these models in a full setup should not be done independently of the preprocessing. Another master's thesis carried out at the eScience group [59] made a detailed pipeline for optimizing x-ray images for a similar test, spotting foreign objects in potatoes. The approach relied on a larger amount of preprocessing algorithms also in the context of applying a CNN at the end. Testing these results with FOSS' images seems like an obvious next step.

One could also use more advanced models with the pixelwide labelling such as YOLOv4 [60] or Faster R-CNN [61] or another one from [62]. These apply more advanced architectures with many more parameters, it would be of exciting to apply these here. It would go well with the polynomial labelling since they require pixel accuracy in the labels.

APPENDIX

A TABLE OF ADAM

If we define the bias corrections, and the non-forgetting as the following:

$$B_i(x,t) = \frac{x}{(1-\beta_i^t)},$$
$$M_i(v_1, ..., v_t) = \max(v_t, v_{t-1}).$$

We can write an overview of the different versions of Adaptive Moment Estimation (Adam) as in table A.1.

	v_t	$\theta_t - \theta_{t-1}$
Adam	$B_2(\beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2)$	$-\alpha_t \frac{m_t}{\sqrt{v_t}+\epsilon}$
NAdam	.	$-\frac{\alpha_t}{\sqrt{v_t}+\epsilon} \cdot (\beta_1 m_t + B_1((1-\beta_1)g_t))$
Amsgrad	$M(v_t)$	.
PAdam	$M(v_t)$	$-\alpha_t \frac{m_t}{(v_t)^p+\epsilon}$
Yogi	$B_2(v_{t-1} - (1-\beta_2)\text{sign}(\beta_2 v_{t-1} - {g_t}^2){g_t}^2)$	.
AdamW*	.	$-\alpha_t \frac{m_t}{\sqrt{v_t}+\epsilon} - w_d \cdot \theta_{t-1}$
RAdam**	.	$-\sqrt{\frac{(\rho_t-4)(\rho_t-2)\rho_\infty}{(\rho_\infty-4)(\rho_\infty-2)\rho_t}}\alpha_t \frac{m_t}{\sqrt{v_t}}$
AdaBound	.	$-\text{Clip}(\frac{\alpha_t}{\sqrt{V_t}}, \eta_l(t), \eta_u(t)) \cdot m_t$

Table A.1: * AdamW. ** RAdam.

BIBLIOGRAPHY

[1] "Wilhelm Conrad Röntgen". da. In: *Wikipedia, den frie encyklopædi* (Nov. 2020) (Cited on p. 2).

[2] *The Nobel Prize in Physics 1901-2000*. en-US. https://www.nobelprize.org/prizes/uncategorized/the-nobel-prize-in-physics-1901-2000-2 (Cited on p. 2).

[3] Ronald P. Haff and Natsuko Toyofuku. "X-Ray Detection of Defects and Contaminants in the Food Industry". en. In: *Sensing and Instrumentation for Food Quality and Safety* 2.4 (Dec. 2008), pp. 262–273. ISSN: 1932-7587, 1932-9954. DOI: 10.1007/s11694-008-9059-8 (Cited on p. 2).

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". en. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14539 (Cited on p. 2).

[5] FOSS. *MeatMaster_II_One_pager_GB* (Cited on p. 2).

[6] *100% Meat Analysis with FOSS X-Ray Analyser Installed Inline*. en. https://www.fossanalytics.com/en/products/meatmaster-ii (Cited on p. 2).

[7] CBC News · Posted: Nov 28, 2016 7:01 AM AT | Last Updated: November 28, and 2016. *Halifax Police Investigating after Needle Found in Potato | CBC News*. en. https://www.cbc.ca/news/canada/nova-scotia/halifax-pei-potato-needle-no-injuries-1.3870465. Nov. 2016 (Cited on p. 2).

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016 (Cited on pp. 3, 31, 32, 43).

[9] Glenn F. Knoll. *Radiation Detection and Measurement*. en. 3rd ed. New York: Wiley, 2000. ISBN: 978-0-471-07338-3 (Cited on p. 5).

[10] *X-Ray Optics Calculator*. http://purple.ipmt-hpm.ac.ru/xcalc/xcalc_mysql/ref_index.php (Cited on p. 7).

[11] *Lecture 3 : Accelerated Charges and Bremsstrahlung*. http://www.astro.utu.fi/~cflynn/astroII/l3.html (Cited on p. 8).

[12] Efrat Shefer et al. “State of the Art of CT Detectors and Sources: A Literature Review”. en. In: *Current Radiology Reports* 1.1 (Mar. 2013). Læs dual-energy del, pp. 76–91. ISSN: 2167-4825. DOI: 10.1007/s40134-012-0006-4 (Cited on pp. 9, 11).

[13] Thorbjørn Louring Koch. *Online Inspection of X-Ray Images*. Nov. 2017 (Cited on p. 9).

[14] darrenl. *Tzutalin/labelImg*. Oct. 2020 (Cited on p. 15).

[15] Kentaro Wada. *Wkentaro/Labelme*. Oct. 2020 (Cited on p. 15).

[16] Jeff Clune. “AI-GAs: AI-Generating Algorithms, an Alternate Paradigm for Producing General Artificial Intelligence”. en. In: *arXiv:1905.10985 [cs]* (Jan. 2020). arXiv: 1905.10985 [cs] (Cited on p. 17).

[17] Yann A. LeCun et al. “Efficient BackProp”. en. In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 9–48. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_3 (Cited on p. 21).

[18] *Torchvision.Transforms — PyTorch 1.7.0 Documentation*. https://pytorch.org/docs/stable/torchvision/transforms.html (Cited on p. 21).

[19] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. “Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection”. en. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Venice: IEEE, Oct. 2017, pp. 1310–1319. ISBN: 978-1-5386-1032-9. DOI: 10.1109/ICCV.2017.146 (Cited on pp. 22, 23).

[20] Ian Goodfellow et al. “Generative Adversarial Nets”. en. In: (), p. 9 (Cited on p. 22).

[21] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *arXiv:1312.6114 [cs, stat]* (May 2014). arXiv: 1312.6114 [cs, stat] (Cited on p. 22).

[22] Rasmus Skov Johannesson. *Generating Realistic Artificial Xray Images of Highly Homogeneous Food Items* (Cited on p. 22).

[23] *Python - Create Random Shape/Contour Using Matplotlib*. https://stackoverflow.com/questions/50731785/create-random-shape-contour-using-matplotlib. ImportanceOfBeingErnest (Cited on pp. 25, 26).

[24] “Bézier Curve”. en. In: *Wikipedia* (Nov. 2020) (Cited on p. 25).

[25] Jacek Cichoń and Zbigniew Gołębiewski.
"On Bernoulli Sums and Bernstein Polynomials". en. In: (), p. 13
(Cited on p. 26).

[26] Christian Szegedy et al.
"Rethinking the Inception Architecture for Computer Vision".
In: *arXiv:1512.00567 [cs]* (Dec. 2015). arXiv: 1512.00567 [cs]
(Cited on p. 33).

[27] Tong He et al. "Bag of Tricks for Image Classification with Convolutional Neural Networks". en. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*
Long Beach, CA, USA: IEEE, June 2019, pp. 558–567.
ISBN: 978-1-72813-293-8. DOI: 10.1109/CVPR.2019.00065 (Cited on p. 33).

[28] H. Robbins. "A Stochastic Approximation Method". In: (2007).
DOI: 10.1214/aoms/1177729586 (Cited on p. 35).

[29] J. Kiefer and J. Wolfowitz.
"Stochastic Estimation of the Maximum of a Regression Function". EN.
In: *Annals of Mathematical Statistics* 23.3 (Sept. 1952), pp. 462–466.
ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177729392
(Cited on p. 35).

[30] Sebastian Ruder.
"An Overview of Gradient Descent Optimization Algorithms". en.
In: *arXiv:1609.04747 [cs]* (June 2017). Comment: Added derivations of AdaMax and Nadam
Comment: Added derivations of AdaMax and Nadam.
arXiv: 1609.04747 [cs] (Cited on p. 35).

[31] Ning Qian. "On the Momentum Term in Gradient Descent Learning Algorithms". en. In: *Neural Networks* 12.1 (Jan. 1999), pp. 145–151.
ISSN: 0893-6080. DOI: 10.1016/S0893-6080(98)00116-6 (Cited on p. 35).

[32] Yurii E. Nesterov. "A Method for Solving the Convex Programming Problem with Convergence Rate O (1/K2̂)". In: *Dokl. Akad. Nauk Sssr.*
Vol. 269. 1983, pp. 543–547 (Cited on p. 35).

[33] Ilya Sutskever et al. "On the Importance of Initialization and Momentum in Deep Learning". en. In: (), p. 14 (Cited on p. 35).

[34] Timothy Dozat.
"INCORPORATING NESTEROV MOMENTUM INTO ADAM". en.
In: (2016), p. 4 (Cited on pp. 35–37).

[35] Diederik P. Kingma and Jimmy Ba.
"Adam: A Method for Stochastic Optimization".
In: *arXiv:1412.6980 [cs]* (Jan. 2017). Comment: Published as a conference paper at the 3rd International Conference for Learning

Representations, San Diego, 2015. arXiv: 1412.6980 [cs] (Cited on p. 36).

[36] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. "ON THE CONVERGENCE OF ADAM AND BEYOND". en. In: (2018), p. 23 (Cited on p. 36).

[37] Jinghui Chen and Quanquan Gu. "Padam: Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks". In: (Sept. 2018) (Cited on p. 36).

[38] Ilya Loshchilov and Frank Hutter. "Fixing Weight Decay Regularization in Adam". In: (Feb. 2018) (Cited on pp. 36, 37).

[39] Manzil Zaheer et al. "Adaptive Methods for Nonconvex Optimization". In: *Advances in Neural Information Processing Systems 31.* Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 9793–9803 (Cited on pp. 36, 63).

[40] Liyuan Liu et al. "On the Variance of the Adaptive Learning Rate and Beyond". In: *arXiv:1908.03265 [cs, stat]* (Apr. 2020). Comment: ICLR 2020. Fix several typos in the previous version. arXiv: 1908.03265 [cs, stat] (Cited on pp. 36, 37).

[41] Liangchen Luo et al. "Adaptive Gradient Methods with Dynamic Bound of Learning Rate". In: *International Conference on Learning Representations.* Sept. 2018 (Cited on pp. 36, 38, 63).

[42] Juntang Zhuang et al. "AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients". In: *arXiv:2010.07468 [cs, stat]* (Oct. 2020). arXiv: 2010.07468 [cs, stat] (Cited on pp. 36, 38, 63).

[43] Ashia C Wilson et al. "The Marginal Value of Adaptive Gradient Methods in Machine Learning". In: *Advances in Neural Information Processing Systems 30.* Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4148–4158 (Cited on p. 36).

[44] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *The Journal of Machine Learning Research* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435 (Cited on pp. 42, 46).

[45] Damian Podareanu et al. "Best Practice Guide - Deep Learning". en. In: *Deep Learning* (), p. 51 (Cited on pp. 42, 43).

[46] Vinod Nair and Geoffrey E Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". en. In: (), p. 8 (Cited on p. 44).

[47] Rikiya Yamashita et al. "Convolutional Neural Networks: An Overview and Application in Radiology". en. In: *Insights into Imaging* 9.4 (Aug. 2018), pp. 611–629. ISSN: 1869-4101. DOI: 10.1007/s13244-018-0639-9 (Cited on p. 44).

[48] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". en. In: *The Journal of Machine Learning Research* 15 (June 2011), p. 9 (Cited on p. 44).

[49] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". en. In: *International Conference on Machine Learning*. PMLR, June 2015, pp. 448–456 (Cited on pp. 44, 45).

[50] Tim Salimans and Durk P. Kingma. "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks". en. In: *Advances in Neural Information Processing Systems* 29 (2016), pp. 901–909 (Cited on pp. 44, 45).

[51] Shibani Santurkar et al. "How Does Batch Normalization Help Optimization?" In: *arXiv:1805.11604 [cs, stat]* (Apr. 2019). Comment: In NeurIPS'18. arXiv: 1805.11604 [cs, stat] (Cited on p. 45).

[52] Jonathan Frankle, David J. Schwab, and Ari S. Morcos. "Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs". In: *arXiv:2003.00152 [cs, stat]* (June 2020). Comment: NeurIPS submission. arXiv: 2003.00152 [cs, stat] (Cited on p. 45).

[53] Geoffrey E. Hinton et al. "Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors". en. In: *arXiv:1207.0580 [cs]* (July 2012). arXiv: 1207.0580 [cs] (Cited on p. 46).

[54] Xiang Li et al. "Understanding the Disharmony Between Dropout and Batch Normalization by Variance Shift". en. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, June 2019, pp. 2677–2685. ISBN: 978-1-72813-293-8. DOI: 10.1109/CVPR.2019.00279 (Cited on p. 46).

[55] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks".

In: *arXiv:1311.2901 [cs]* (Nov. 2013). arXiv: 1311.2901 [cs] (Cited on p. 51).

[56] "Domain Adaptation". en. In: *Wikipedia* (Nov. 2020) (Cited on p. 54).

[57] Alexander D'Amour et al. "Underspecification Presents Challenges for Credibility in Modern Machine Learning". In: *arXiv:2011.03395 [cs, stat]* (Nov. 2020). arXiv: 2011.03395 [cs, stat] (Cited on p. 61).

[58] Less Wright. *Lessw2020/Ranger-Deep-Learning-Optimizer*. Oct. 2020 (Cited on p. 63).

[59] Aleksandar Topic. *Adaptive X-Ray Inspection System (AXIS)*. en (Cited on p. 63).

[60] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". en. In: *arXiv:2004.10934 [cs, eess]* (Apr. 2020). arXiv: 2004.10934 [cs, eess] (Cited on p. 63).

[61] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (June 2017), pp. 1137–1149. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2016.2577031 (Cited on p. 63).

[62] *Papers with Code - COCO Test-Dev Benchmark (Object Detection)*. en. https://paperswithcode.com/sota/object-detection-on-coco (Cited on p. 63).

COLOPHON

This document was typeset using the custom $\LaTeX\ 2_\varepsilon$ document class `jespersthesis` which is almost identical to `dionsthesis` by Dion Haefner, based on `uiothesis` developed by Eivind Uggedal. It uses Minion Pro, developed at Adobe Systems, and Fira Sans, developed by the Mozilla Foundation, as body fonts.