Automatic EX-Inspection

Cim Stordal
e-mail: cim.stordal@cognite.com

Sammendrag av Bachelor Prosjekt

Tittel:	**Automatic EX-Inspection**
Dato:	16.05.18
Forfattere:	Cim Stordal
Veileder:	Ivar Farup
Arbeidsgiver:	Cognite AS
Kontakt person:	Stein Danielsen
Nøkkelord:	Kabel, Feil deteksjon, Maskin læring, Convolutional Neural Network, Attention, Datasyn, Maskinsyn
Sider:	59
Vedlegg:	4
Tilgjengelighet:	Åpen

Abstract: Skader i elektriske kabler medfører en stor risiko for både mennesker og utstyr, spesielt på oljerigger da elektriske kabler og brannfarlige gasser er separert bare ved hjelp av en tynn barriere. I denne oppgaven forsøker vi å detektere feil i kabler automatisk ved hjelp av hodemonterte kameraer og state-of-the-art bildegjenkjenning teknikker, i tillegg til vår egen teknikk, "Guided CNN", i en situasjon hvor det er lite treningsdata tilgjengelig. Resultatene viser at problemet kan løses med tilstrekkelig nøyaktighet ved bruk av state-of-the-art CNN teknikker. Videre viser vi at vår Guided CNN arkitektur gir bedre resultater enn state-of-the-art CNN nettverk med hele 2 prosent for dette problemet. Det største hinderet for å få dette systemet inn i produksjon, er bildekvaliteten på kameraet vi har testet, når bildet er tatt i bevegelse.

Summary of Graduate Project

Title:	**Automatic EX-Inspection**
Date:	16.05.18
Authors:	Cim Stordal
Supervisor:	Ivar Farup
Employer:	Cognite AS
Contact Person:	Stein Danielsen
Keywords:	Cable, Fault Detection, Machine learning, Convolutional Neural Network, Attention, Computer Vision
Pages:	59
Attachment:	4
Availability:	Open

Abstract: Faulty electrical cables pose a significant and severe risk to both people and equipment, especially on oil rigs where electrical cables and highly combustible gases are separated by thin barriers. This thesis describes an attempt to automatically detect faulty cables on oil rigs through images captured using head-mounted cameras and classified using state-of-the-art CNN techniques, as well as our own Guided CNN architecture. This is done in a situation where we have a limited set of available training data. This thesis shows how using state-of-the-art CNN techniques can result in sufficient accuracy also for a limited training set. From the results, it is also evident that my self-developed Guided CNN architecture outperforms the baseline, made up of a state-of-the-art CNN. As of now, the most pressing issue before pushing this system into production is that the camera hardware tested in this thesis does not perform well enough as images may turn out blurry.

Preface

I would like to thank my supervisor Ivar Farup for all the help and encouragement he have given me. I would also like to thank all the people at Cognite who have helped me whenever I have needed it.

Contents

Chapter 1

Introduction

Faulty electrical cables pose a significant and severe risk to both people and equipment. This is in particular the case for oil rigs where electrical cables and highly combustible gases are separated by thin barriers. In a worst case scenario, the combination of a faulty cable and a gas leak leads to an explosion causing massive damage to the rig and its crew. The problem is taken with the utmost seriousness also by Norwegian government officials [1].

To detect electric faults and prevent catastrophic events, oil rig operators routinely perform *EX-Inspections* of each area of the rig; at least once per 12 months. In practise, such inspections are visual inspections of all cables with human beings actively having to look for faults. This tedious task with normally no errors to report is obviously prone to human errors.

One way of removing human errors in such inspections is to automate the detection based on analysis of pictures or video. That is, instead of actively having to look for errors, the inspectors instead film all the cables and leave the detection up to a highly precise algorithm. However, in the literature, there seems to be extremely few studies on automatic fault detection in pictures or video. In fact, the author has been unable to locate any such specific studies. Neither does there seem to be many studies on similar type of problems. The problem of detecting a fault in a picture of a cable can be seen as a 'single point of max interest" classification problem. This means that a specific point in the image determines the classification, Most classification studies, however, concern several points of equal interest.

In this thesis, we shall make a modest, but not insignificant, step towards safer and more automated *Ex-inspections*. Our main result, will be a new Machine Learning algorithm capable of detecting electrical faults in pictures of cables, and moreover locate the position of the fault in the picture. The thesis and subsequent algorithm is not motivated by academic interests alone; it actually addresses the needs and concerns expressed to the author by the Norwegian Oil & Gas operator Aker BP. As a result, the end product is currently being tested by Aker BP at their Ivar Aasen platform in the Norwegian Continental Shelf, and it can be expected to enter widespread use in the near future if we

[1] Goliat akutt stengt: – for å si det litt banalt, det kan ikke være gnister i områder med olje og gass. `https://www.tu.no/artikler/goliat-akutt-stengt-ned-har-ikke-kontroll-pa-tennkilder-i-eksplosjonsfarlige-omrader/409007`, 2018.

can prove it working well in practice.

The new algorithm which we will develop in the upcoming material is based on Convolutional Neural Networks (CNN). During the past decade, CNNs have completely revolutionized image recognition and sits at the center of the current AI/Machine Learning hype. Across all disciplines and all industries, it has enabled a vast collection of breakthrough technologies from medical imaging and cancer detection [5], to self-driving cars, satellite surveillance, salmon farming, and many more areas.

One of the few arguments against CNNs as a technology for image classification has been its lack of explainability. That is, for a given CNN result, it can be extremely hard to explain why the model comes up with the given result. Often, an explanation is needed to build faith in the predicted result. To put it bluntly, no doctor is willing to set a cancer diagnosis on the basis of a black-box model without any means of verifying the underlying logic. Similarly, no human is willing to accept a cancer diagnosis without any explanation. To counteract this, researchers in CNN have in the past years developed a mechanism called *attention*[7] which in effect provides an explanation for which information the model has used to derive its conclusion.

In our upcoming algorithm for Ex-inspections, we shall also utilize the *attention* mechanism to provide explainability for the results. For our purposes, it not only provides an explanation as to why a picture of a cable has been labeled as faulty, it also locates the fault in the picture. However, this is not the only use of *attention* that we will rely on. In fact, we shall also use *attention* to *guide the CNN network*. As far as the author is aware of, our upcoming use of attention is completely novel and could *merit publication in its own right*. Whereas previous studies have used *attention* to locate objects in an image, there seems to be no research using it in combination with a classification network, to guide the model towards the correct place in the image, during training. We will come back to this point in Section 3.2.

Another distinctive feature of CNNs is that they require a considerable amount of training data. In our case, we do not have a large set of pictures of cables to use as training data. Luckily, the problem of scarce data can in many cases be circumvented using a pre-trained model as basis. The process of using a pre-trained generic model as a starting point for training a specialized model with limited data is often called *transfer learning*. As we will experience in the upcoming material, *transfer learning* is applicable in our case so that we will achieve remarkable accuracy even though our data set is limited to a thousand original images augmented with their synthetically generated counterparts (i.e., increasing the data set by adding various transformations).

In this thesis I would answer three research questions: One, what are the pre-processing steps and model architecture that is best suited for solving this problem?. In which we would look into the importance of the data set and the various prepossessing one can do. We would also look into how important the model architecture choice is and the effect fine tuning would have.

Further we would like to find out if there is a Specialized CNN architecture that works better for Single Point of Max Interest classification problems? Here we would develop and test out our own Guided CNN model architecture.

The last question we would look into is if stacking would improve the overall accuracy. Here we would try out a cluster of known ensembling techniques, and also one novel technique that would stack models using attention.

The remaining parts of this thesis are structured as follows: First, in the ensuing section, we review the background material we will need in the upcoming algorithmic development. Then, in Section 3, we develop our algorithm with its various steps. Next, in Section 4, we present the results of our algorithm applied to faulty electric cables. Later, in Section 5, we discuss our results, possible improvements, future work, and other ideas. Finally, in Section 6, we will talk about the whole system we designed for Aker BP to solve this problem.

Chapter 2

Background

2.1 Introduction to AI/CNN

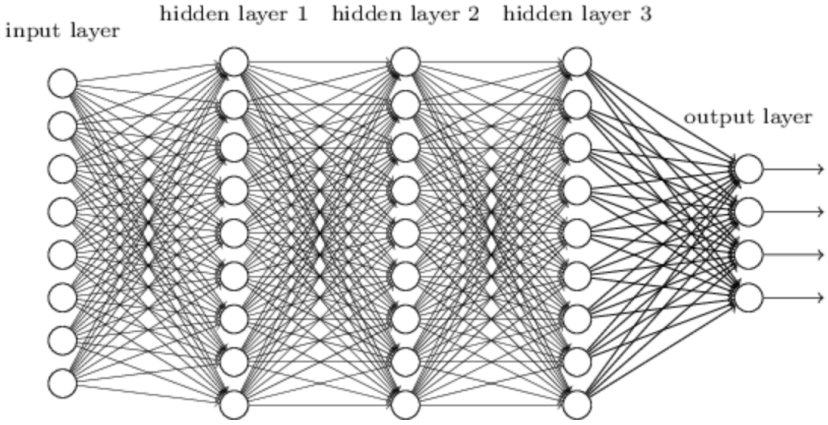


Figure 2.1: Neural Network [10]Creative Commons, free to share)

The purpose of a *Neural Network* is to recognize pattern in the data. It works because the network, as seen in Figure 2.1, adjusts its weights while it learns, such that it will minimize the loss. In simpler terms, it adjusts the weights to maximize the number of correct predictions. A neuron is a single node in a neural network. It takes any number of numerical inputs and outputs a weighted sum of these. The most common way a Neural Network will learn is through *backpropagation*.

In a normal *Neural Network* all the layers are fully connected. An example can be seen in Figure 2.1. However, in an image classification task, this is problematic. When all the layers are fully connected, you lose the spatial information in the image. The very same thing that is the important part of an image. That's why our research will use a specialized Neural Network architecture for image tasks, namely a *Convolutions Neural Network*.

2.2 CNN - Convolutional Neural Network

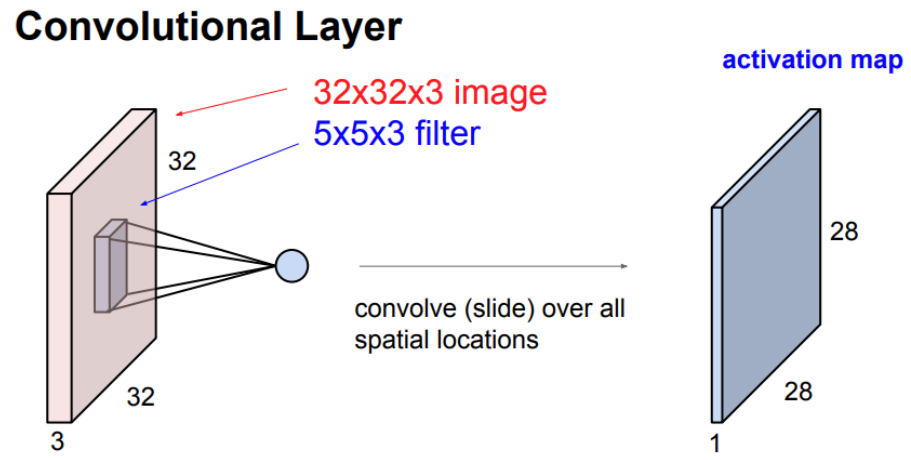


Figure 2.2: Convolution Layer [1](MIT)

There are four main parts of a CNN network. The convolution layer, the polling layer, the fully connected layer and the backpropagation training.

Convolution layer: Figure 2.2 show an input image of $32 \times 32 \times 3$. This means that the image size is 32px by 32px with a *depth* of 3, being the RGB. There is also a $5 \times 5 \times 3$ *filter* in the Figure. This *filter* slides over the image. How it will slide is determined by the *stride*. The *stride* is the step size it uses when it slides over the image. Often its also useful to add *zeropadding* to the sides, such that the filter can be run on all spatial locations. *Zeropadding* means that you will increase the size of the current feature map, by surrounding it with zeros. A *filter*, also referred to as a *kernel*, works as a feature detector. The filter is a set of weights/parameters that to begin with is randomly initialized, and through *backpropagation* updated. The fact that the weights are randomly initialized as opposed to zero initialization is important, as updating the weights is performed through multiplication, and anything multiplied with zero equals zero. If weights are initialized as the same value, that is also a problem. In such a situation, we would not get the variance in the various filters. Instead of having x different filters, we would have x identical filters. Random initialization allows them to learn different things.

When performing multiplication and summation against the *receptive field* of the network and the filter, we get a high value if it is a match, or a low value if it is not a match. The *receptive field* is the area the filter is currently tested against.

The result of the multiplication and summation will be a single point in the new feature map. When performing a convolution, the end result is a *feature Map*, also referred to as *Activation Map*. This is the 1x28x28 map in Figure 2.2. We get the size 1x28x28 from Equation 2.2.1, where N is the network size, P is the padding size, F is the filter size and S is the stride. This calculation has to be done for all three sides of the map.

$$\text{Output size} = \frac{(N + 2P - F)}{S} + 1 \tag{2.2.1}$$

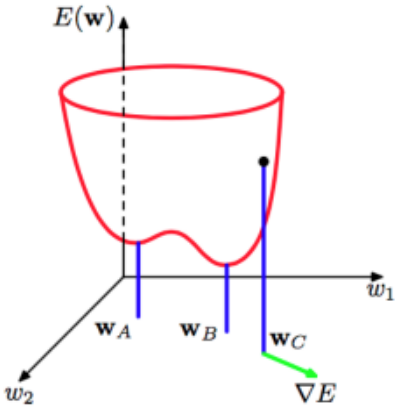


Figure 2.3: Error Function [4] used with permissions

Activation function: *ReLU*, Rectified Linear Unit is an activation function. The activation function would introduce none linearity, allowing the model to learn nonlinear prediction boundaries. The ReLU function is defined as:

$$\text{f}(x) = \max(0,x) \tag{2.2.2}$$

So that it eliminates any negative values. However reaching an activation of 0, is not good as that would mean the *neuron* dies. A method to circumvent it is called *Leaky ReLu*. It is similar to ReLu, with a small tweak.

$$\text{f}(x) = \max(0.01,x) \tag{2.2.3}$$

Instead of using 0, it uses a low value. In this case 0.01, while other values are also common.

Polling Layer: *Polling Layer* is used to down-sample the network. There are two main approaches to polling; *Max polling* and *Average Polling*. In this thesis our network will use Max Polling. In Figure 2.4 we see an example of max polling, using a 2x2 filter with a stride of 2.

Given the receptive field of the 2x2 filter, we select the max Value, then move by a Stride of 2, and do the same for the rest of the receptive fields. The output is the four values to the right in Figure 2.4. From this its clear that Max Polling lose interesting information in the image. This is something we will talk more about in Section 3.2.

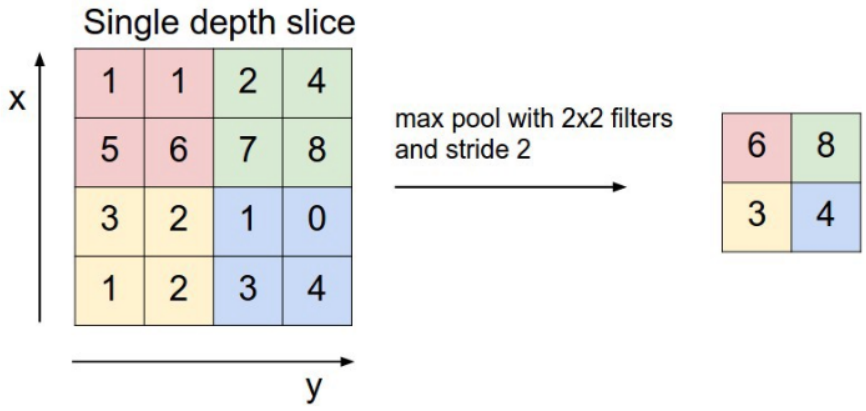


Figure 2.4: Max Polling [2] (MIT)

Fully connected – Prediction: The fully connected layer is a layer where all the neurons are connected to all the neurons in the previous layer. The prediction occurs on a fully connected layer with the same shape as the expected output. In a classification problem, this would be the number of different classes. For this fully connected layer, we use a softmax activation, or sigmoid activation, which produce the final class probabilities.

Training with backpropagation: Based on the prediction, the loss function returns a loss based on how wrong the estimate is. This loss is then backpropagated to the network, and used to adjust the weights.

When training, the goal is to reach the *Global Minima* for the given error function. In Figure 2.3 we can see Wa as a *Local Minima* and Wb as the *Global Minima*. If one sets the *Learning Rate* too high, one would potentially overstep and never reach it. A good visualization can be found in Figure 2.5. Here the red learning rate is too high, and as a result would never reach the global minima, in oppose to the green one, that is reducing the learning rate over time. The learning rate (lr) is the step size the network would take when attempting to minimize the loss. A common approach is to reduce the learning rate over time, such that it would be fast paced while still not getting stuck because of being too high. The problem with this is that you might end up in a local minimum, believing you have found the global minima.

Class Activation Map *Class Activation Map (CAM)*, is in this thesis used for visualizing how the model "thinks". It works by summing up the feature maps for a given class. For *GradCam*, Gradient weighted class activation map, we weight each feature map with the gradient. In this thesis, we will use the Gradient weighted approach, even when called *CAM* [12]. There are two main motivations for using Class Activation Maps. One is to help our understanding of the model, and from that information know what it struggles with. The second motivation is to minimize the gap to start trusting AI. Deep learning models, in particular, are not the best at explaining why they produce certain predictions. However, as the class activation map gives us its attention we know

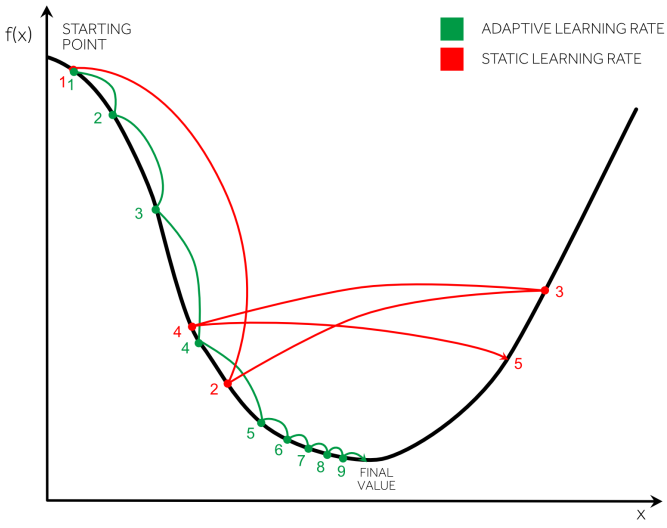


Figure 2.5: Static Learning Rate VS Adaptive Learning Rate

why it does certain things. And if humans see that the AI make predictions the same way they would, that might increase trust. This is very important in the oil and gas sector, where this product will be used, as their transition to AI so far, has been limited.

2.3 Word list

Stacking:

Stacking is the art of combining different models into an ensemble. There are multiple ways to do this. The naive approach is to take the average across the model outputs. However, there exist more sophisticated approaches. For example, if one knows that one of the models performs a lot better in poor lighting, or that one is better at detecting a certain type of the objects we could detect that situation and weight those models more heavily. This gives us a more adaptive and deterministic model.

A way to do this is to extract image features, and train a second layer classifier on top of it. There is a mathematical backing for using stacking. However, its efficiency relies on the correlation between the different models. More info can be found in the Kaggle Ensembling Guide [3].

[3] Kaggle ensembling guide, mlwave 2018. `https://mlwave.com/kaggle-ensembling-guide/`, 2018.

Model Correlation:

When training VGG16 with different parameters one might get both good and bad models, but combining them might end up not helping too much. The reason is that they are all strongly correlated. Two good ways to mitigate this are; to impose bootstrap aggregation or use different models, alternatively, you can do some augmentation that sets the models apart from each other.

Hyperparams

Hyperparameters are tunable parameters for an algorithm. For example, the dropout value, size of a given dense layer and learning rate are common hyper-parameters to tune in order to improve accuracy.

Test Time Augmentation

Test Time Augmentation is a technique applied during inference. For example, doing 10 predictions with various rotations, and averaging the result. This should be more deterministic than just doing a single prediction. In this thesis *Test Time Augmentation* will be referred to as *TTA*, along where we might specify *TTA(10)*. We may also specify a parameter x, where *TTA(x)* means x predictions were made before summation.

Here 10 refers to the number of times one did a prediction before summation.

Pre-trained Model

A pre-trained model already initialized with weights from a training run; often from imageNet. In this thesis, we use pre-trained ImageNet weights, and only *finetune* the last layers. This is done when there is not enough data to train the whole network. Fine-tuning is often used synonymously with *transfer learning*.

ImageNet

A multiclass classification challenge [4].

Image classification

Assigning the correct class label to a given image.

Data Set

The collection of images we have.

Model

The prediction algorithm.

---

[4] Imagenet large scale visual recognition competition. `http://www.image-net.org/challenges/LSVRC/`, 2018.

Overfitting

Overfitting is the process where a model learns a specific data set too well, and are not able to generalize the knowledge when presented with a new data set. One method for circumventing this, that we would use during our training is dropout.

Dropout

Dropout is a technique that would randomly select one or more neurons in the network that it would not train during that specific batch. This forces the network to create alternative paths through the network, hopefully turning it less deceptable to Overfitting.

Vanishing Gradient

Vanishing Gradient is a fundamental problem in machine learning, that happens when training with back propagation. During back propagation, the gradient decreases, as the gradient is the product of the previous gradients up to that point. The product of the earlier gradients would decrease as the gradient is in the range [0,1].

VGG Networks

The Architecture we use for most of the experiments in this thesis is a VGG network. It was created in 2014 by researchers at Visual Geometry Group at University of Oxford [13]. It has a linear network architecture that we will discuss further in Section 3.2.

YOLO

YOLO [11] is a real time Object detection algorithm that is based around the principle of You Only Look Once. We will during Section 3.2 discuss some of the implementation, to showcase how it differs from our approach.

2.4 Tools and Algorithms

Random forest

Random forest is a meta-estimator algorithm that we use for training our ensemble [5]. It uses Bagging (bootstrap aggregating), a method to randomly select a reduced feature set to operate on.

[5] Sklearn - sklearn.ensemble.randomforestclassifier. `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`, 2018.

Adaboost

Adaboost, also called Adaptive Boosting is a meta-estimator algorithm that we will use for training our ensemble [6]. It works by doing weighted average over a number of weak classifiers.

T-SNE:

T-SNE is a dimension reduction algorithm that can be applied to visualize highly dimensional data. In our example, it is used to visualize how similar the input images are, given the feature vector from an intermediate layer in the CNN architecture. The *t-SNE* algorithm works without class labels, and only when we are plotting the results, are we using the class label to color the points to show the clustering.

Tensorflow

Tensorflow is a software library created by researchers at Google, used when training AI models [7].

Tensorboard:

Tensorboard is a simple tool for visualization of TensorFlow models and training sessions [8]. In this paper, we use it for visualization of how the training runs went, compared to other training runs. The images from Tensorboard used in this thesis consist of four parts: Loss, Accuracy, Validation loss and Validation Accuracy. Loss And Accuracy are measured against the training set, while Validation Accuracy and Validation Loss are measured against the validation test set.

Keras

Keras is a high-level neural network API for Python. It can be built on top on Tensorflow, among other libraries, that removes a lot of the complexity of training a machine learning model. In this thesis, our research is built on top of the Keras Framework, with a Tensorflow backend [9].

Auxiliary Classifiers/Auxiliary Loss:

When you perform loss calculations not only at the last layer. This is a way to prevent the vanishing gradient problem. It also works as a way to include more spatial information in the image.

---

[6] Sklearn - sklearn.ensemble.adaboostclassifier. `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html`, 2018.

[7] Tensorflow - an open source machine learning framework for everyone. `https://www.tensorflow.org/`, 2018.

[8] Tensorboard: visualizing learning. `https://www.tensorflow.org/programmers\_guide/summaries\_and\_tensorboard`, 2018.

[9] Keras: The python deep learning library. `https://keras.io/`, 2018.

K-Means(Clustering):

K-Means is a clustering algorithm, that clusters using the Lloyd's algorithm [10].

Google ML Cloud

Google ML Cloud [11] is an online service allowing users to train their Tensorflow models on Google's infrastructure. In this thesis, our training would be conducted on a Google ML Cloud GPU instance, specifically on a "BASIC_GPU" instance.

2.5 Performance Evaluation Metrics

Accuracy is calculated as

$$A = \frac{N_{TP} + N_{TN}}{N} \tag{2.5.1}$$

where N is the whole data set, N_{TP} is the true positives, N_{FP} False positive, N_{FN} is the false negative and N_{TN} is the true negatives. *ValAccuracy* and *ValAcc* refer to the accuracy on the validation test set.

Recall is inherently problematic to talk about without more context as *recall* is defined as

$$R = \frac{N_{TP}}{N_{TP} + N_{FN}} \tag{2.5.2}$$

Here the only requirement is to have no *False negative* results. However, any model predicting only true would get a *recall* of 1.

Another way to see the probability is through precision.

$$P = \frac{N_{TP}}{N_{TP} + N_{FP}} \tag{2.5.3}$$

However, if we were to use this as a metric, we would get a precision score of 1 if our model predicted correctly that it was a fault, one time, and never predicted faulty any other times.

If one combines recall and precision as follow:

$$F_1 = 2 * \frac{R * P}{R + P} \tag{2.5.4}$$

one get the *F1 score*. F1 score is the harmonic mean between precision and recall. Accuracy works best when the output class is of equal importance. However because of the Accuracy Paradox [14] it might not be the most applicable to all situations.

In this thesis, we primarily use accuracy as a metric to determine the model's viability because we are interested in a good overall prediction. Not just that the few times it does a prediction, it is correct. We are also not after a model that just predicts faulty all the time. We will in the instance of our new guided

[10] sklearn.cluster.kmeans — scikit-learn 0.19.1 documentation. `http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`, 2018.

[11] Cloud - ai fast, large scale and easy to use ai services. `https://cloud.google.com/products/machine-learning/`, 2018.

CNN network validate our accuracy against the *F1 score*, and the *Recall* as these metrics contain some additional information that would be useful in determining if the training were successful.

Chapter 3

CNNs for Electric Cable Faults

In this chapter, we will discuss how to construct CNNs capable of detecting electric cable faults. Ultimately, this will allow us to implement an algorithm for automatic Ex-inspection in the ensuing Chapter.

In Section 3.1 we will go over the importance of the data set, and use this in combination with fine-tuning to create a very powerful model. The Section is organized to follow the same sequential lineup as a model training algorithm would.

In Section 3.2 we will explore a novel model architecture that is specialized to work on single point of max interest classification problems. The section will compare the architecture to the YOLO architecture, and explain how to implement this new architecture on top of a VGG network.

Section 3.3 will use the models from the two previous sections and attempt to combine them in order to create an ensemble model which outperforms any standalone model. This Section would use a multitude of known ensemble techniques, as well as our novel ensemble technique. Our new technique uses attention, to weight the various models in the ensemble differently.

3.1 Optimizing and Training CNN Models

In this subsection, we embark on the task of constructing a CNN algorithm for detecting cable faults. For completeness, we will evolve the construction in the form of a discussion around the various algorithm parts and how it fits with the data available to us. A notable challenge is that we have a very limited data set to use for both training and verification.

We will start our discussion by detailing how our available data was acquired. Next, in Section 3.1.2 we look into how important the choice of data splitting into training and verification data set actually is. In Section 3.1.4 we use the proposed data split and attempt to find the optimal image size to train on. Next, we will find the optimal model to use in Section 3.1.5. Afterward in Section 3.1.6, we will attempt to figure out the optimal number of layers to train on. Training on fewer layers while achieving the same level of accuracy should speed up training. Having found the best set-up, we then test out different

augmentations techniques that might help us see the images from a different "perspective". This will be the topic of Section 3.1.7.

3.1.1 Data Gathering

Gathering good data is one of the most important parts of machine learning, is it however often also one of the hardest. In this thesis, we use the Faulty-Cable-500 data-set. An in-house generated collection of 40+ faulty cables with a total data set consisting of 600 good and 600 faulty images. To improve the data quality, when generating the data set, we took the cables, divided them into about 0.5m pieces, and took 4 different images of each cable for each of the 4 backgrounds, before and after creating a fault. The images were taken with different lighting and bending of the cable. The cables were also in different sizes and colors. This was designed so it should be as easy as possible for the network to learn the cable, and not be interested in the background. If we were to take all the images with the faulty cable with one background, and all the good cables with one background, we would have an information leakage. This would make our model focus on the wrong details, and it would perform poorly on a real-life data-set.

As we are using the same fault, there is a potential data leakage if we were to just do a random split of this training set. So the validation set would be partly created with a random split, and partly consisting of new faults. There is also an inherit bias in the data-set as it is created by us, with our assumption of what a fault looks like. We may also have introduced some unconscious bias while creating the training set, for example, that images with fault might have been taken from another angle, or more closely. both of these issues would be mitigated by acquiring more real-life data and will be addressed if the validation split shows that the test set is biased.

Other ideas we had while trying to acquire the data set included talking to EX-Inspectors. However, they did not have a rule about taking images while doing an inspection. Another one was that we could use Amazon Mechanical Turk (MTURK) [12], and crowdsource this, and let people earn a few dollars for taking pictures of cables they otherwise would have thrown out. Because of our deadline for gathering data, we weren't able to do this.

For an AI project to be successful, it needs a plan for continuous collection of data. Our default plan is that all the employees will be equipped with a helmet camera with our application that would automatically capture images. This approach would allow us to build up the biggest database of images from oil platforms. The method does not have a good plan for acquiring data of faulty cables unless faulty cables are found on the oil rig. This means scalability when attempting to maintain a balanced data set would be slow.

3.1.2 Finding the Optimal Data Set Split

In situations where the data set has a poor split, one could risk overfitting the model to the training data. The model would then not be able to generalize to the test set, this is called overfitting. And in consequence neither to real-life data. Often this happens in situations where the image class distribution is

[12] Amazon mechanical turk - mturk.com. `https://www.mturk.com/`, 2018.

Table 3.1: Network architecture and settings used

Model = VGG19 Where we have replaced the top layer with: GlobalAveragePooling2D, Dense(256), Dropout(0.5),Dense(2) All layers = trainable Epochs = 40 SGD optimizer(lr=0.0001, momentum = 0.9)

highly uneven. A situation that is known as class imbalance. For our case, we will look into how data splitting behaves when we have the same amount of images per class, but is in a situation with a tiny data set.

First, we will look into performing multiple random splits instead of a single random split. Afterwards, we will look into our approach for structuring the data split, given our limited data set and the potential validation problem. During training, we will use the model architecture described in Table 3.1.

Multiple Random Splits

In situations where there is a limited amount of training data, the split between train and validation is inherently more important. One should assume there to be a few images in the data set that would be easy to classify, while some are hard to classify. In order to get an accurate validation of our results, and create a model that would generalize the best, we want an equal split among these. The current consensus is to perform a random data split. However if one only performs a single random split, it is hard to know if one actually got a good split. Given the limited data set, there is a higher chance for a bad split. Our approach is to perform multiple random data splits and perform a training run on each of them. Afterward, we can select the data split, which produces result closest to the average. As that is the data split that should contain a good amount of easy-to-detect and hard-to-detect images. If one of the data splits performs exceptionally well, that is probably an indication that something is wrong with the data split, that makes it easier for the algorithm to perform well. We select the model that produces result closest to the average to mitigate the fact that models can learn "fake" solutions to problems that humans wouldn't have imagined being possible. This is what "The Surprising Creativity of Digital Evolution" talks about [8]. To prove that a bad data split can occur we also handcrafted two data splits that we assumed would generalize poorly. As our training images are taken with different backgrounds we decided to create a split where there is a background not present in any of the training sets. For the second split, we had a background that is present in the training set for class X, while only being a part of the validation set for class Y. For both of these, we also tried to have some cable types only in the validation set that looked different from what it had trained on. In short, we are handcrafting an imbalance in the cable-types and backgrounds used for training and testing.

```
img_height = 800
img_width = 800
img_channels = 3
img_dim = (img_height, img_width, img_channels)

model = SimpleModel()
model.set_inputfolders(traindir,valdir)
model.set_model(get_model_func("vgg16")(weights=model.
                                    local_weights,
                                        input_shape=img_dim,
                                        include_top=False))

model.create_standard_top layer(model.base_model, activation="
                                    sigmoid")
model.compile()

model.set_epoch(40)
model.fit()
```

Figure 3.1: fine-tuning a VGG16 network with input size 800×800

3.1.3 Validation Problem

As previously mentioned, finding the correct data split for validation is a hard problem given the limited amount of data. There is also a potential for overfitting as the training set consist of only 50 faults. And the same faults would be a part of both the training and validation set. As a result, we have a tiny extra set of 100 images, based on 7 new faulty/not faulty cables, that would be added to the validation set. So that the models get to test their work against unseen faults.

We only performed a train/test split, instead of the industry norm of doing a train/test/validation split. This is because it would have meant that, given our limited data set, we would have almost no data left to train on. Our idea is rather to acquire a real-life data set that would act as a validation set.

3.1.4 Find the Optimal Size For the Images

A higher input dimension can lead to a higher accuracy on image detection as more information is retained. However, there is also a relationship between how far one can be away from the faulty point when taking the image, given the input size. This means that a good model should be trained on the optimal image size, where the parameters are the allowed distance from the point, and the prediction accuracy.

3.1.4.1 Does Size Matter?

One important feature that can be modified is the input size. How big should the image we train on be and does size really matter? These are the questions we shall answer in this section.

The code in Figure 3.1 is using our SimpleModel class to train a new model. Training on a higher image resolution is an idea taken from the winning entry

from a Kaggle competition [13]. Here we will feed the network an 800×800 image that will be split into smaller images that in turn being fed into a pre-trained VGG16 model, and finally passed through a sigmoid activation. We can use a sigmoid as we only have two classes during this experiment. Faulty/Not faulty. We then set the model to be a part of our SimpleModel object, and we will later call fit.

We want to use the same approach to find the relationship between input size and accuracy. We could then find the optimal relationship based on training time, accuracy, and what would work on real-life data.

3.1.5 Find the Optimal Model Architecture For This Project

There are a lot of CNN models available, some better suited for some problems, while others perform better for other. In this experiment, we are going to test some of the most known model architectures to see how they perform on our data. This is an important step for continuing our training session, as we want to do our research on the best possible model architecture for this problem. Some of the best-known architectures are VGG16,VGG19 and Inception. These can be used pre-trained with weights from the ImageNet competition [4], meaning that only the top layers have to be retrained.

The experiment is structured so all the models are tested on the same dataset, and some individual tuning of learning rate and the top layer will be done for each model. Because of this tuning is performed manually and not automatically through Google ML Clouds hyperparameter tuner [3], there is potential for our biases to favor one architecture over another.

3.1.6 Tuning the Model Through Hyperparameters

In subsection 3.1.5 we established the optimal model architecture. In this following section would we look into some research related to tuning the model to maximize performance.

3.1.6.1 How Many Layers Should Be Trainable

One of the hyperparameters that can be tested is how many of the layers in a network that should be trained. We want the number to be as low as possible as that would decrease the number of trainable parameters we would tune, and make our training faster. Because the network is pre-trained, it generally won't need to be tuned in the earlier layers, and in some cases doing so would be counterproductive because it would increase the chance for overfitting. Or in the worst case, destroy the pre-trained weights.

In Keras one can set layers to trainable/not trainable easily. The function in Figure 3.2 would freeze all except for the last "num" layers.

Our experiment is to try different amount of layers that are not trained and find the best amount, where best is defined as the smallest numbers of layers to still give us the same accuracy.

---

[13] Keras k-fold inception v3 (1st place lb 0.99770) — kaggle_2018. `https://www.kaggle.com/jamesrequa/keras-k-fold-inception-v3-1st-place-lb-0-99770`, 2018.

[4] Imagenet large scale visual recognition competition. `http://www.image-net.org/challenges/LSVRC/`, 2018.

```
def freeze_allExcept(self, num):
        for layer in self.finalModel.layers[:-num]:
            layer.trainable = False
```

Figure 3.2: Freeze all layers except the last x layers.

```
rotation_range=360,
shear_range=0.3,
vertical_flip=True,
horizontal_flip=True
```

Figure 3.3: Keras Augmentations Applied.

3.1.7 Preprocessing Application of Augmentation Techniques

As cables can be found in various different positions, rotations, and bendings we perform augmentation on the input images. All input images will have a random rotation R, where R ϵ [0,360>, applied to them. This is done to train the model into learning different orientations of a cable. Another possible approach could be to preprocess each input to be aligned horizontally. However, this is something that can be explored in future work.

The Keras augmentations performed on all the input data can be found in Figure 3.3.

We perform rotations and shearing [14] for two reasons, one is form factor. Given the limited data set its important to perform rotation so the model can recognize the that it is a cable, regardless of its bending and rotation. Second is that it works as a way to avoid overfitting. The more training images we have, the less chance there is for overfitting.

In an attempt to see the data from a new perspective, along with attempting to remove unnecessary data from the image, we decided to attempt to train models on other preprocessed versions of the images. The idea is that a combination of these models would be able to get a better accuracy than any single model.

3.1.7.1 Edge Detection

One idea is to train some of the networks on an edge detected version of the image.

In Figure 3.4, one can see that it is easy to detect the faulty cable from the different Canny edge detected images. There are three different images, as Canny expect the image to be a 2D array, and as our input image contains 3 colors we got a [224,224,3] array. We could either flatten it or create a different Canny for each of the color images. For Figure 3.4 we chose the latter, while for Figure 3.5 we use the flattened version. The implementation used can be seen in Figure 3.6. However if one wants to remove noise from the image, one can easily add the bilateral filter.

[14] Shear. `http://mathworld.wolfram.com/Shear.html`, 2018.

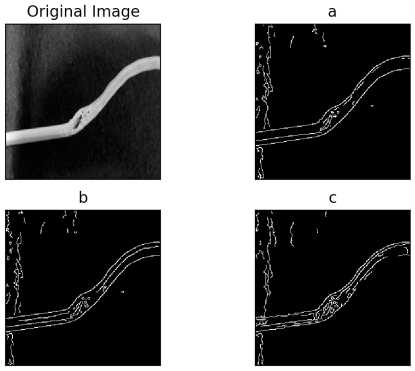
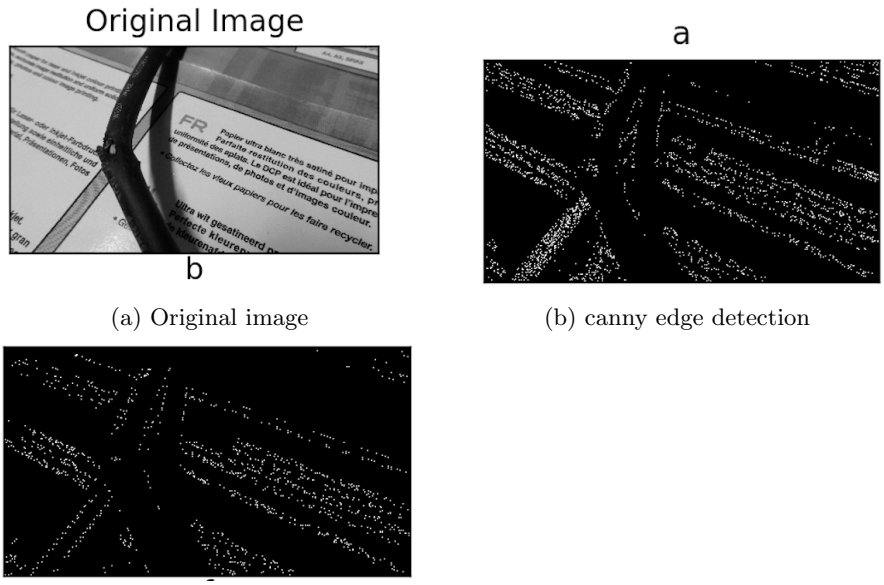


Figure 3.4: Canny edge detection working



(a) Original image

(b) canny edge detection

(c) Canny + Bilateral filter

Figure 3.5: Prepossessing of images

A bilateral filter is an edge-preserving filter. However looking at Figure 3.5c, compared to Figure 3.5b one can see that the fault is getting harder to notice. While we simultaneously remove the other noise, we also used Gaussian blur with 5 kernels. For this project, we decided to experiment with canny edge detection, canny + bilateral filter and canny + Gaussian blur. All images are transformed to the correct format prior to training and run with the same data split as determined in Section 3.1.2.

3.1.7.2 Segmentation by Color Quantization

Another good way to remove image noise is by implementing a segmentation algorithm, the simplest one is to use threshold. For example, every pixel darker

```
    from skimage import feature
    img = misc.imread(img)
    r = feature.canny(img[:,:,0])
    g = feature.canny(img[:,:,1])
    b = feature.canny(img[:,:,2])
```

Figure 3.6: Canny Edge detection

```
img = cv2.imread(sys.argv[1])

gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.
                                    THRESH_OTSU)

plt.subplot(221),plt.imshow(img)
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(222),plt.imshow(thresh)
plt.title('Image with threshold'), plt.xticks([]), plt.yticks([])
plt.show()
```

Figure 3.7: Code for plotting of threshold

than a given threshold will be yellow, and lighter would be purple.

An example of a good output can be seen in Figure 3.8. However this only works on images where the cable is in a color with a strong contrast to the background. If this is not the case, the output in Figure 3.9 will be the case.

Figure 3.8: cv2.Treshhold on a broken cable

There are other similar methods that can be used, for example, k-means. For this project, we decided to train a new network on the k-means version of the dataset images.

For our k-means research, we test using different values for K as part of our research to find a good number. We evaluate the k values manually, by seeing if we easily can find the cable and fault in the images. Then train a network on the best version.

3.1.8 Adding More Classes to the Data Set

In Figure 3.10, one can see from the Class Activation Map that the model has learned what a flaw is. The Class Activation Map outputs where in the picture, the model focused when doing its prediction. It focuses on the fault, however, it does not focus on the fault as part of the cable. In essence, we don't see that the light blue color follows the cable. This can lead to other problems in later situations where it believes to have found the flaw pattern, in a region with no a cable.

Figure 3.9: cv2.Treshhold on a broken cable - not working

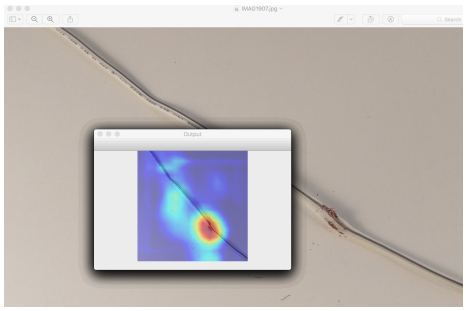


Figure 3.10: Class activation map of faulty wire test on the first VGG16 model

To mitigate this we decided to test creating another class consisting only of backgrounds in the data set, along with also creating another totally unrelated class, all in the hope that the model would understand that the cable artifact was important. The backgrounds used in these images are similar to the type of backgrounds used for our original images. Some are the same, while some are totally different.

3.2 Guided CNN Architecture

Guided-CNN is a novel idea that uses the Class Activation Map to guide the training towards a specific point of the image. A general CNN architecture is optimized against the class probability through backpropagation. In contrast, object localization and landmark detection problems calculate the loss against both the class probability and against either Bx,By,Bh,Bw, or only a Bx,By pair for each landmark. Where (Bx,By) is the center of the box, and Bh,Bw is the height and width.

For example, basic-Yolo-keras [15] implements their top layer as shown in Figure 3.11. Here they have a layer, here referenced as features, with the size $[14 \times 14 \times 512]$ that they feed into Conv2d, that they later reshape to $14 \times 14 \times 10 \times 7$. Assuming they have 10 different boxes, the grid is 14×14, and that they have two classes the final output is [p,Bx,By,Bh,Bw,p1,p2]. P is 1 if it thinks there is an object there and 0 otherwise, and p1 and p2 are the probability's for the respective classes, for each of the ten boxes in each of the grids. Here we have to train this conv2d layer the relationship between the features and this output shape.

The concept of a box exists as there could be multiple objects inside a grid window. The way YOLO allows for this is to have multiple different box shapes

[15] Basic-yolo-keras. `https://github.com/experiencor/basic-yolo-keras`, 2018.

```
output = Conv2D(self.nb_box * (4 + 1 + self.nb_class),
                        (1,1), strides=(1,1),
                        padding='same',
                        name='conv_23',
                        kernel_initializer='lecun_normal')(features
                                                           )
output = Reshape((self.grid_h, self.grid_w, self.nb_box, 4 + 1 +
                                  self.nb_class))(output)
```

Figure 3.11: Basic-Yolo-Keras [15] implementation of their top layer. Code is under MIT license.

and assign the object to the box that fits it the best. This means that with a grid size of 14×14 and 10 boxes, the maximum amount of object YOLO can find in an image is 1960, evenly distributed.

Our idea is to base our prediction on the same $14 \times 14 \times 512$ layer. However, instead of training a new layer, we get the coordinates directly from the gradient-weighted class activation map. We then return a loss based on how wrong the most activated point in the CAM is.

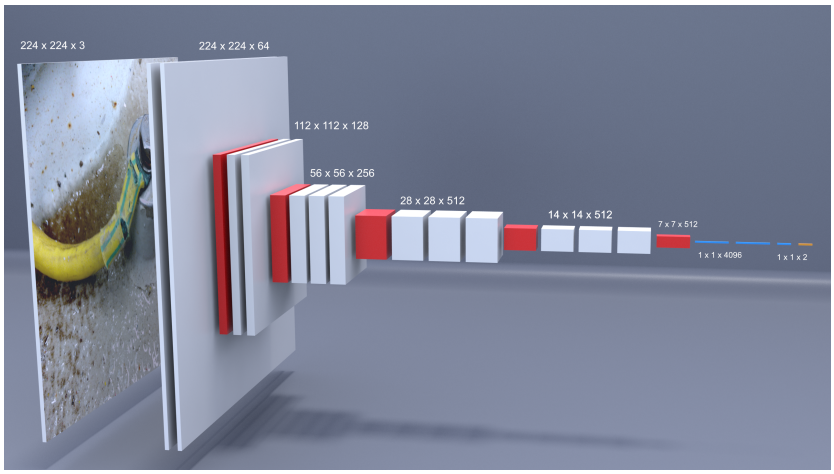
As we have classified this problem as a single point of max interest classification problem, there is already an inherent assumption that there is a single point we should move our focus towards. Previous research has already established that there is location information in the layers that can be extracted with a Class Activation Map [16]. In situations with limited data, our idea is that it would be beneficial to create an annotated dataset, where we annotate where the single point in a 14×14 grid is. This means we then can calculate the loss as a combination of class probability and how far away from our point the focus is. In theory, this implementation should act as a guided CNN network, where we are guiding the focus towards the point of interest.

3.2.1 Visualize the Change Between VGG16 and Our Implementation

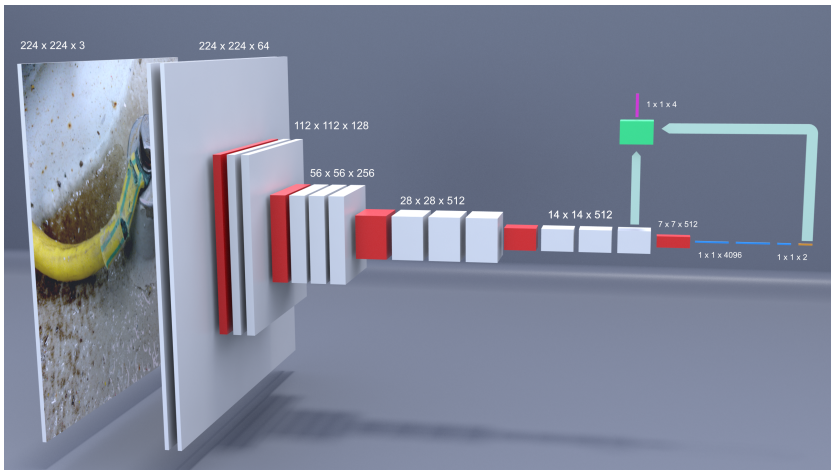
In Figure 3.12 one can see the model architecture before and after modifying it to use our guided CNN architecture. The example visualization uses VGG16, however, our model used in this test is based upon VGG19. The architecture change, however, is generic and should be applicable to most network architectures. From the visualization, we can see that the new architecture does it to take the output from an intermediate layer, and the final prediction layer, and feed them into our guided CNN layer. That outputs the prediction and the (x, y) coordinates.

3.2.2 Method

To test this hypothesis we will need to design and develop a new model architecture. The following sections present a detailed description of the important features of this new architecture and its implementation.



(a) VGG16



(b) VGG16 with Guided CNN

Figure 3.12: Visualization of change between VGG and VGG with Guided CNN. The red box is max polling, blue represents the fully connected layer, orange is softmax, Green is Guided CNN layer and pink is the final result in the Guided CNN Network.

Architecture:

Instead of adding a normal top-layer, our architecture has a split top-layer. VGG architecture is by design only a sequential single pipeline, while newer model architectures like Inception, perform things in parallel. Our VGG modification would be a split network where one part calculates the class prediction, while the other takes the output from the "block5_conv3" layer, and passes it through a series of new layers, that ends up giving us the coordinates of the point with the most activation. The "block5_conv3" is a $14 \times 14 \times 512$ sized layer.

The first step is to get the gradient, with respect to the class. We use that to calculate a weight for each of the 512 feature maps in the convolution layer.

```
class EpochCallback(Callback):
    """
        Updates the alpha value, based on the current epoch
    """
    def __init__(self, alpha):
        self.alpha = alpha
    def on_epoch_end(self, epoch, logs={}):
        value = F(epoch,50,50,1)
        K.set_value(self.alpha, value)
```

Figure 3.13: Guided CNN – Epoch Callback

After this, we multiply the 14×14 for each of the 512 feature maps, with the corresponding weights. Then we do a mean average of the $14 \times 14 \times 512$, and end up with only a single 14×14 map. We then need to normalize the values to be in the range [0,1]. First, we set all negative values to 0, and then we take all values and divide them by the highest value in the 14×14. Now that everything is in the range [0,1] we just need to get the coordinates for the block with the value 1.

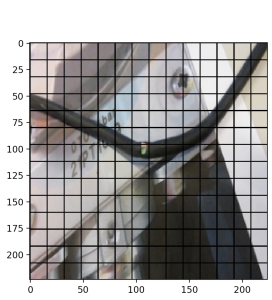
In the end, we concatenate the class prediction output and the coordinates of the point with the highest activation, and pass them to our custom loss function that returns a loss based on both the Euclidean distance and the class probability, in the hope that we would "guide" the model towards the point of interest.

In order to have polynomial loss importance, we have to be able to acquire the epoch number in our custom loss function. Loss importance denotes how much weight we should put on a loss. It is a polynomial loss importance if the importance was given by a polynomial function. In this thesis, we have two losses, class_loss, and XYLoss, and we use a constant loss importance for class_loss, while the importance of the XYLoss change over time. The reason for this is that we assume that its best if in the start, the most important thing is the class_loss, while the later we come in the training, the more important is the XYLoss. To do this we created a callback, that after each epoch, update a Keras variable, with the current epoch number. However as our polynomial function is an exponential function, and the result would not change for subsequential calls inside of an epoch, we instead perform the calculations for the $f(x)$ value in the callback such that it is calculated once per epoch. See Figure 3.13 for implementation details.

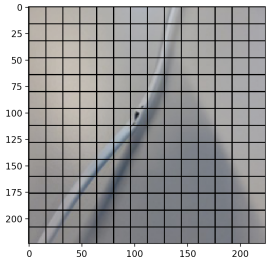
Loss Function:

The loss function is fed [p1,p2,x,y] as our prediction. Our annotated data is at this point, still in the range of Bx,By,Bh,Bw as we utilize the same annotations as for YOLO. This means that we do not have an error with one of the annotations being better than the other if we were to compare this method with YOLO. Also, it simplifies things as we don't have to maintain multiple similar annotation sets. The YOLO annotation is box annotation around the faulty part of the cable.

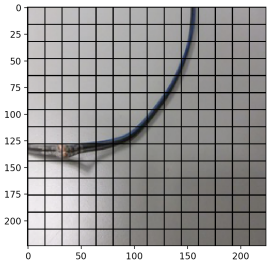
To turn a Yolo annotation into our grid annotation, we use the midpoint, and divide it by 16, as 224 divided by 16 is 14, our grid size. Then we cal-



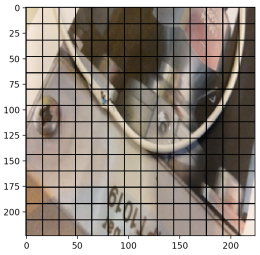
(a) Fault in the middle of a grid boundary



(b) Fault in the middle of a grid boundary



(c) Fault in the middle of two different grid boundaries



(d) Fault inside grid boundary

Figure 3.14: Cable faults with grid overlay

culate the Euclidean distance from the prediction and ground truth. Based on this we assign a loss. We call this loss the XYLoss. We also have a class annotation we should assign a loss to. We calculate the class_loss using categorical_crossentropy.

From this, we can just do a combined loss where we weight the distance-loss and class-loss differently, according to what performs the best. When combining the losses, they can be weighted with constants, so the relationship between the class loss and XYLoss are constant, or it could be something that changes over time.

Figure 3.15 show an example of the importance of the different loss functions being different over time, given by epoch. Importance can be seen as what value we multiply the current loss of that function with. The class loss would be stable, and therefore be 1 multiply itself. XYLoss would have almost no impact in the start, while over time would become more and more important. The function used for the polynomial class loss importance is defined in Figure 3.16.

Using a polynomial loss importance combination may be troublesome, as it would add "turbulence". Looking at the problem quite abstract, it seems to be the solution, however, if there is a lot of entropy in the error space, it may destroy your chances of even reaching something close to the Global Minima. The max_val argument is important to ensure that the value is in the correct range. If the value is too high, it would be to much focus on the (x, y), however, if it is too low, there would be not enough focus on it.

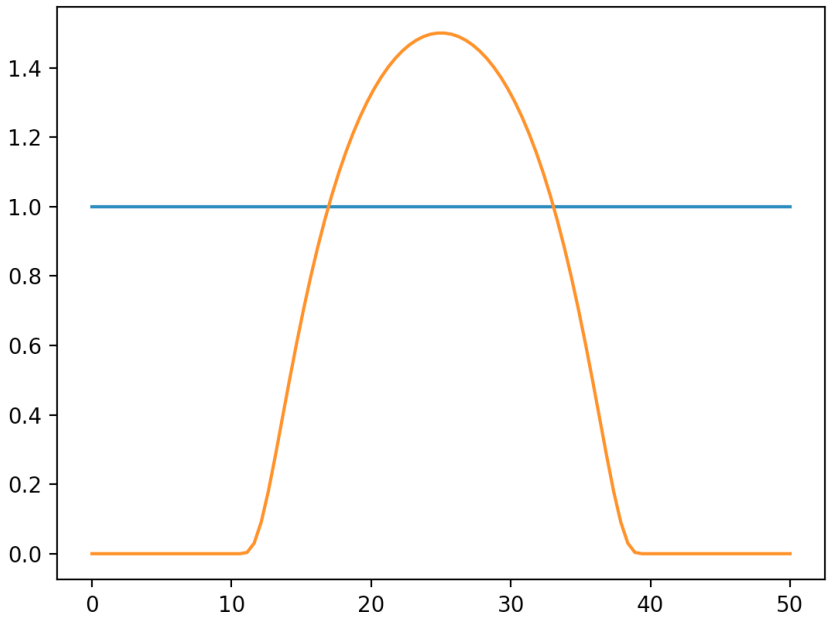


Figure 3.15: The importance of the different losses, over time. Blue is class loss, orange is XYLoss

Training Framework:

There are also other changes that would have to be made to test out this architecture. One is that we need to create a custom batch generator, and a custom augmentation function. As the normal Keras implementation is only for images, while we have images and annotation data, and if we rotate an image during the augmentation process, we need to be able to also rotate the annotation data. Basic-Yolo-Keras [15] has partly implemented this, however, it is not completely suited to our need as they do not handle rotation of images. However, this can be handled by the imgaug [16] library. In Figure 3.17 it is shown how one easily can apply the same augmentation to both the image and the key points, in our case the bounding box. Self.aug is defined as done in Figure 3.18.

3.2.3 Experiment

For the following research, we will perform multiple training runs attempting to beat a baseline accuracy score. The baseline is created by performing a run on the exact same architecture, just with XYLoss importance set to zero.

For our other experiments, we are attempting to tune the importance relationship between XYLoss and Class_loss. Both as the product of a polynomial function, and through constant loss importance relationships.

[15] Basic-yolo-keras. `https://github.com/experiencor/basic-yolo-keras`, 2018.
[16] Imgaug. `https://github.com/aleju/imgaug`, 2018.

```
def F(E, M, eps, max_val):
    """
        E - Epoch
        M - mid point
        eps - width
        max_val - top point
    """
    X = np.abs(1.0*(E-M)/eps)
    Y = np.exp(-1.0/np.maximum(np.abs(1-X**2), 0.000000000000001))*
                                        0.5*(np.sign(1-X)+1)
    Y = (max_val/np.max(Y))*Y

    return Y
```

Figure 3.16: Python implementation of Function F, used while changing the loss-importance over time

```
        keypoints = ia.KeypointsOnImage([
            ia.Keypoint(x=x1, y=y1),
            ia.Keypoint(x=x1, y=y2),
            ia.Keypoint(x=x2, y=x1),
            ia.Keypoint(x=x2, y=x2)
        ], shape=image.shape)

        self.aug.to_deterministic()

        image = self.aug.augment_image(np.array(image))
        keypoints_aug = self.aug.augment_keypoints([keypoints])
```

Figure 3.17: ImgAug handling of rotation

```
        self.aug = iaa.Affine(
                    rotate=(-180, 180))
```

Figure 3.18: Definition of what augmentations to apply

3.3 Creating a Model Ensemble using Stacking

Now that we have acquired a lot of models that are trained on various data, with various augmentation techniques that give us different scores, we can create an ensemble.

As part of the stacking, we take advantage of "Test Time Augmentation" to increase our accuracy, feature engineering to base our stacking on more than just the percentages, and in the end, random forest and AdaBoost compared to the normal and weighted average of the model predictions.

3.3.1 Test Time Augmentation

As rotation is a big form factor in cables, to get the best prediction we perform TTA - Test Time Augmentation to the image. Because the average on consecutive runs would give us a stable prediction that would tell more truthfully how well the model actually performs. For our model, we save both the mean, the one that won the most times, and the collected percentages for all combined.Test Time Augmentation might actually be a requirement as there are so many possible ways a cable can appear in an image, and given our limited data set there is an assumption that it would receive invariant training data that can make the model bias towards certain rotations and bendness of the cable. In order to get a good prediction, we should rotate it when testing. A problem with relying on this is the increased time it takes to do a prediction.

3.3.2 Feature Engineering

When training our second layer model, it is important to know what features we should train it on. If we only use the prediction per image, that's only one bit of information; faulty, not faulty. The next step is to use the percentages it believes per class. For example [0.4,0.6]. For this research, we would also use the (x, y) location of the point with the most activation in a Class Activation Map, as a method for automatically on-the-fly adjusting the weighting of the different models in our ensemble.

Explanation of this Method We want to group models that have similar prediction activation as more likely. An example prediction can be found in Table 3.2. In this case, a simple average would give us that not faulty won. However, if we include the (x, y) of the highest activation in the Class Activation Map, as shown in Table 3.3, we see that the faulty predictions really do know what they are doing. They are looking at the same places.

In a situation where the faulty predictions are looking at the same point. We weight the faulty predictions twice, allowing the minority a more powerful voice, as we believe they are correct. So the overall prediction would be 4 predicting faulty, while 3 predicting not faulty. So, in this case, faulty wins, even if more of the models said it was a "not faulty" prediction, as models focusing on the same point increases our trust in them.

Similarity with Deepface To the author's knowledge, the only method for finding correlation between models that is remotely similar to our approach

Table 3.2: Example of an ensemble prediction

Model	Predicts Faulty/Not Faulty Cable
Model a	Faulty
Model b	Faulty
Model c	Not Faulty
Model d	Not Faulty
Model e	Not Faulty

Table 3.3: Example of an ensemble prediction with (x, y) location information

Model	Predicts Faulty/Not Faulty Cable	(x, y) of point of max activation
Model a	Faulty	$(4, 3)$
Model b	Faulty	$(4, 3)$
Model c	Not Faulty	$(3, 0)$
Model d	Not Faulty	$(5, 0)$
Model e	Not Faulty	$(1, 0)$

herein is the face detection algorithm *DeepFace* [17]. The Deepface approach is to run two images through the same CNN, take the output from the activation layer, and calculate the loss difference between these two images. Small difference interpreted as similar, whereas big is different.

Note that the *Deepface* method is fundamentally different from our approach. We are evaluating multiple models for every image and calculate the difference using the maximum point in each activation map, whereas they use the same CNN-network for many images and measure the similarity between different images using the feature map. We can rely on using only the one with most activation as the most focus should be on the fault of the cable. And we can not be certain how the other activations would look as our models are trained on various forms of the input data, some are trained on edge-detected data and some on normal. However, there are problems with this approach. What if the cable has multiple flaws? And how do you handle for models not predicting a fault? As they can focus on different points. The solution to the last problem is to calculate loss with respect to only the class prediction for images without faults.

In Table 3.4 we can see the information we save for each model prediction. We only save p0 - the prediction for class 0, as also saving the prediction for the second class, when there is only two, is redundant.

The use of $(CAMX, CAMY)$ also has a problem. Given the limited grid size, of only 14×14, there are only 196 possibilities. This means that sometimes the models would predict the same $(CAMX, CAMY)$ location even when they should not. Because of the birthday paradox [18] the feasibility of this method

[17] Convolutional neural networks — coursera_2018. `https://www.coursera.org/learn/convolutional-neural-networks/home/week/4`, 2018.

[18] Understanding the birthday paradox. `https://betterexplained.com/articles/understanding-the-birthday-paradox/`, 2018.

decreases with each new model added to the stacking. With 5 models, there is about 5% chance of predicting the same $(CAMX, CAMY)$ location, at random. This means that our model assembly probably would see an increase in wrongly predicting things as faulty using this approach. We currently have no specialized method for mitigating this.

Table 3.4: Example on how a prediction from a single model can be broken down. P0 is the prediction for class 0, CAMX/Y is the (x, y) location to the place with the most activation in the Class Activation Map. CamP is the percentage of how activated that point is, compared to the rest.

ImageName	P0	CAMX	CAMY	CamP	Groundtruth
pi.png	0,40	3	4	0,40	1
pi2.png	0,9	2	1	0,80	0

3.3.3 Random Forest/Adaboost

For stacking the different models, we also want to train a second layer classifier on top of the models, to select the best model combination. Generally one wants to train a second layer classifier on top of unseen data. For example, by splitting the validation set into two, and performing training on one half, and validation on the second. However given the limited data set, we have decided to train on the data set from the test run, and perform validation on the same validation set. The reasoning behind this is that we have an increasingly small subset of data to use for validation if one would split the validation set. This would make us all the more open to "bad splitting" as previously mentioned. Bad splitting could make it seem like the model performs vastly superior to what it actually does.

In this experiment, we use Random Forest and Adaboost and compare the results of these with the weighted average, and the normal average across the different top models. With voting-average, there is a soft average and hard average. Aside from hard and soft averages, we would also perform median, max, min, min-max-median averages.

Chapter 4

Results

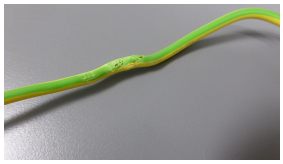
In the following chapter we will go over the results of the various experiments described in the last chapter.

4.1 Optimizing and Training CNN Models

4.1.1 Data Gathering

In Figure 4.1 there is depicted various images from the data set that we managed to craft. Figure 4.1a is an example of the bendness of a cable. While Figure 4.1c illustrates a more difficult background for the network to comprehend.

(a) Good Cable



(b) Bad Cable

(c) Good Cable

(d) Bad Cable

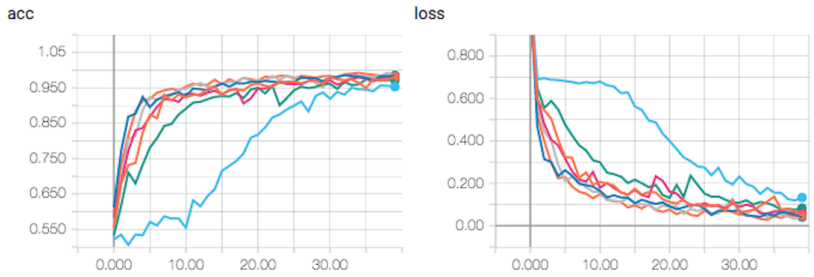
Figure 4.1: CableSet

4.1.2 Finding the Optimal Data Set Split

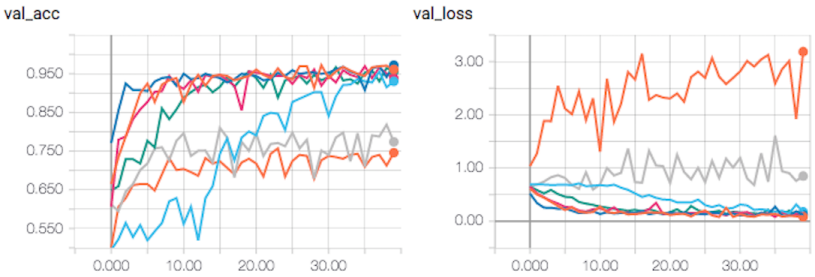
In Table 4.1, one can see that the runtime for the first two data splits was 3-4x as long as the last three. The reason for this is that the first two were fed full-size images, and resized on the fly, while the rest where fed images resized to 256×256, and only being resized to 224×224 on the fly. All runs are for 40 epochs.

Table 4.1: Data split: Split 5 and 6 are handcrafted data sets, while the rest are random splits with splitID as seed

Split ID	Validation Score at End	Runtime
0	0.9609	2 hr 52 min
1	0.9721	2 hr 54 min
2	0.9302	0 hr 37 min
3	0.9385	0 hr 36 min
4	**0.9553**	0hr 35 min
5	0.7737	0hr 39 min
6	0.7458	0hr 34 min



(a) Graph over data split runs



(b) Graph over data split runs - Validation

Figure 4.2: Tensorboard over data set split

The Tensorboard graph in Figure 4.2 also gives us some information. For one, that the outlier data, did reach about the same accuracy later on in the run. This means that the accuracy drop, to begin with, could be based on random chance, and if we were to run it for a longer time, it would converge.

A bad data split can be seen as a data split where images of a certain fault were not in the training set at all. Or where one background is only present for one class in the training set, while only for the other one in the validation. To test out this could impact our training result we handcrafted two bad data sets. In set 5, there is a background that is not present in any of the training sets, in set six that background is only present in the training set for class X, while only being a part of the validation set for class Y. Our handcrafted bad data split (still 25% split) reached validation score of 0.74 and 0.77 for data sets 6 and 5

respectively. This means that the split is important, however as our 5 totally random splits show us, the chances for such a bad split is probably low. The chance of a bad data split is dependent on the current data set you are working with. Thus a data split test should always be done to ensure a quality split.

The validation set consist of 311 images while the training set consists of 927. This means that the highest validation score is the result of classifying all but:

$$311 - (311 * 0.9721) = 8.6 \text{ images}$$

While the lowest:

$$311 - (311 * 0.9302) = 21.7 \text{ images}$$

Remark 1 *The results are not a whole number as this was calculated as an average of 10 consecutive runs, with different rotations of the images.*

This means there is a difference of over 13 images.

As we are interested in a good random split, we choose the split with ID 4, as that had the most average validation score at the end.

4.1.3 Find the Optimal Size For the Images

During initial tests we see that we can, by changing the input from 224×224 to 800×800, go from 92% accuracy on the validation set to 96%. However, there are two problems. For one, while training like this, we had to reduce the batch size, to not run out of memory. We would also be doing more calculations per image, leading to a slower performance. A network trained in 45 minutes on 224×224 took 5 hours with this setup when training on the same GPU. This is because the computation needed scales linearly with the number of image pixels [9].

Secondly, we want our model to be able to detect these faults from a camera at some distance. Increasing the input size would decrease the possible length we can be away from an object when taking a photo. This could lead to usability issues with the application.

We then concluded that we would not use this approach, and limit our self to images with size 224×224.

4.1.4 Find The Optimal Model Architecture For This Project

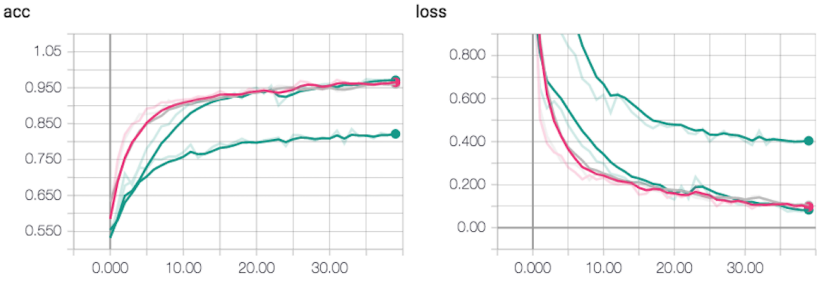
After playing around with various top layers, we were able to reach an accuracy of 80% on VGG16. Inception did not do as well, after setting all the layers to trainable I got inception up to 85% without a problem, and VGG16 to 91%. The next step is to attempt both. First train the whole model, and then only the top. Doing this didn't increase accuracy too much. Letting the training of all the layers stay for 60 epochs, where the learning rate was decremented after 25 and after 40 epochs, yielded better results. Inception hit 89% when it finished while VGG16 hit 90%, and had it's highest validation accuracy during training at 93%. However, both models suffered from overfitting. Obviously considering we are training all the layers, on only around 900 images. The best performing model was VGG19.

4.1.5 Tuning the Model Through Hyperparameters

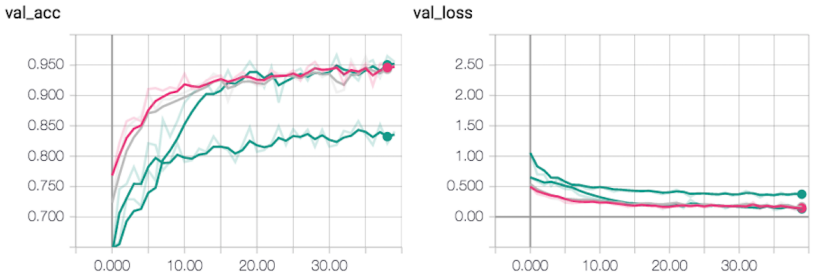
4.1.5.1 How Many Layers Should Be Trained?

Table 4.2: Trainable layers

Amount of trainable layers	Validation Score at End	Runtime
5	0.8356	18 min
8	**0.9525**	14 min
10	0.9474	22 min
All	0.9553	35 min



(a) Tensorboard of trainable layers



(b) Tensorboard of trainable layers - validation

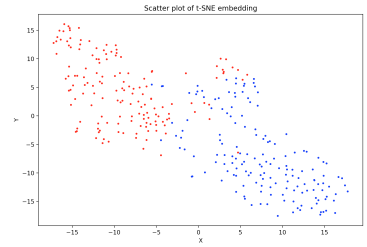
Figure 4.3: Tensorboard of trainable layers

The time parameter is not consistent at these low scales, as a job can take up to 10 minutes before starting on Google ML Cloud. A more appropriate time description can be found in Tensorboard. Shown in Figure 4.3.

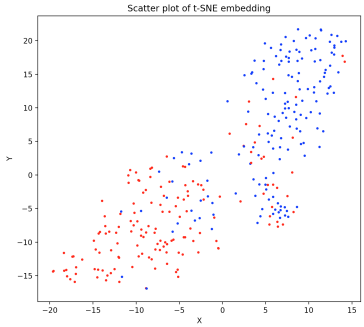
From Table 4.2 one can see that training all layers archive roughly the same accuracy as only the last 8. The minor difference can be repudiated to random chance. When only training the last 5 however, it would drop quite significantly. Given this drop we would, for the time being, train the last 8 layers.

4.1.5.2 Visualization of the Models

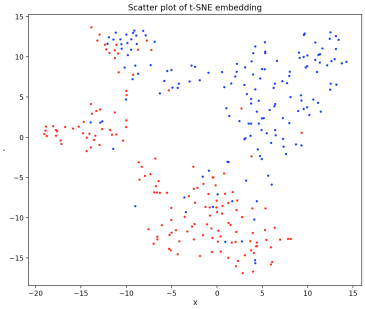
In Figure 4.4 one can see a visualization of how it would look if we clustered the images from the validation set. Blue objects are Cable_Good, while red is Cable_Faulty. This can give us some insight into how troublesome the data is.



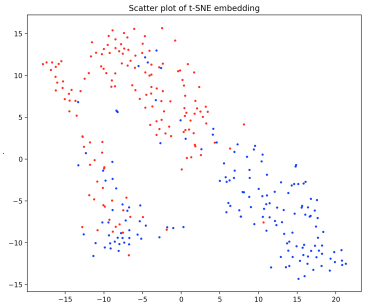
(a) Model 8



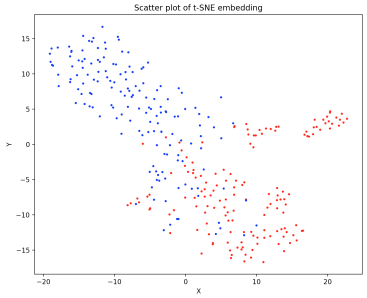
(b) Model Canny



(c) Model Canny Bileteral



(d) Model Canny GAUS



(e) Model Canny KMEAN

Figure 4.4: T-SNE scatter plot on the validation test set

Here we clearly can see that the model in Sub Figure 4.4a perform the best, as it has the least overlap between the two clusters.

4.1.5.3 Hyper-parameter Tuning

For fine-tuning a model, one can start playing with dropout, size of the various trainable layers, learning rate and other "hyper-parameters".

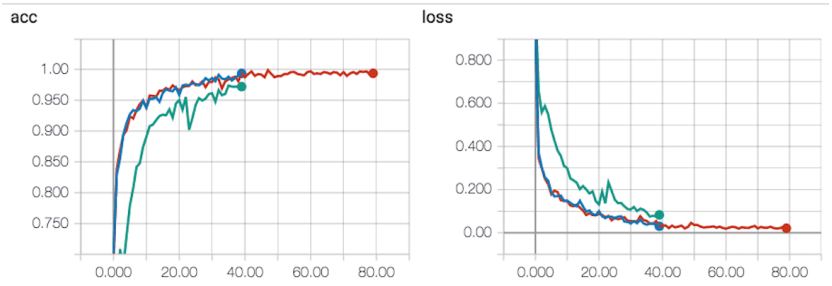
Based on the model trained with the last 8 layers as trainable, we set up a new test run. All the same, just without the dense(256) and dropout(0.5) at the top layer.

After this, we set up a second test. The same setup, just this time with lr drop at epoch 40, and 60. Starting at 0.0001, dropping to 0.00001, and then 0.000005, running for a total of 80 epochs.

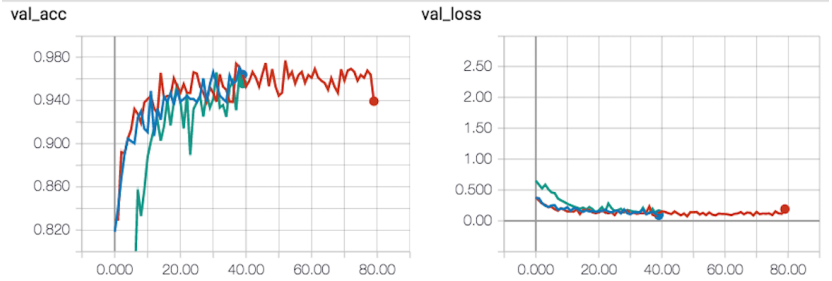
A third test was run with last 8 layers as trainable, without dense(256) and dropout(0.5) for 80 epochs.

Table 4.3: fine-tuning

ID	Type	epochs	Validation Score at End
1	split4 (all layers trainable)	40	0.9553
2	last 8, without dense(256),dropout(0.5)	40	**0.9637**
3	last 8, without dense(256),dropout(0.5)	80	0.9441
4	last 8, without dense256,dropout(0.5) and dropping lr	80	0.9393



(a) Tensorboard for fine-tuning



(b) Tensorboard for fine-tuning - Validation

Figure 4.5: Tensorboard for fine-tuning

As Table 4.3 shows, fine-tuning is not straightforward. Running for a longer period of time does not equal a better accuracy. It is important to note that the end val_acc is not the highest validation score for that training run. For example at epoch 73 of 80, model 3 had 0.9705 in val_acc while model 4 had, 0.9656. One can see this by looking at Figure 4.5.

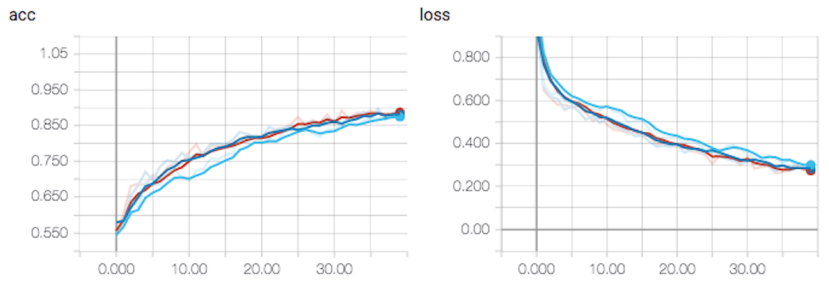
4.1.6 Preprocessing Application of Augmentation Techniques

4.1.7 Edge Detection

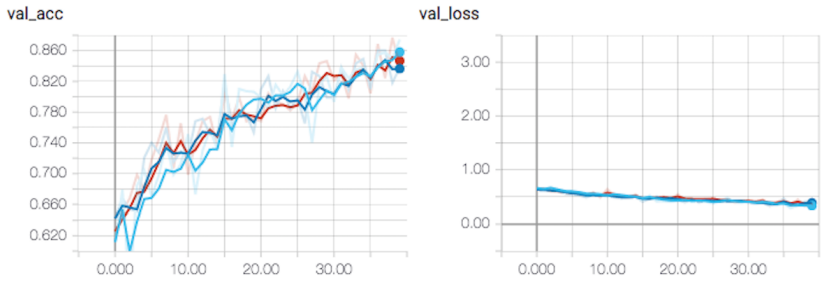
The highest validation score is Canny + gaus. At the end of epoch 38, it had 0.8768, a slight improvement over what Canny alone had at the end of the training session.

Table 4.4: Edge Detection

Type	Validation Score at End
canny	**0.8743**
canny + bilateral	0.8380
canny + gaus	0.8408



(a) Tensorboard of our three augmentation functions



(b) Tensorboard of our three augmentation functions - Validation

Figure 4.6: Tensorboard of our three augmentation functions

From the graph, one can see a relationship between the accuracy and the validation accuracy. This means that overfitting is not the problem. Its rather just a hard task. The accuracy of all three models was between .87 and .89, while validation accuracy where between .83 and .87 as shown in Table 4.4.

From the graph, one can also imagine that if we were to continue training the validation would increase. This is because the graph has an upwards direction, and does not seem to have flattened yet.

4.1.7.1 Segmentation By Color Quantization

Table 4.5: Segmentation

Type	Validation Score at End
K-Means(Clustering)(k=3)	**0.9050**

Color quantization is the process of reducing the number of colors in an image. In Table 4.5 you can see that we decided to train a network on K-

(a) Before K-Means(Clustering)

(b) After K-Means(Clustering)

(c) Before K-Means(Clustering)

(d) After K-Means(Clustering)

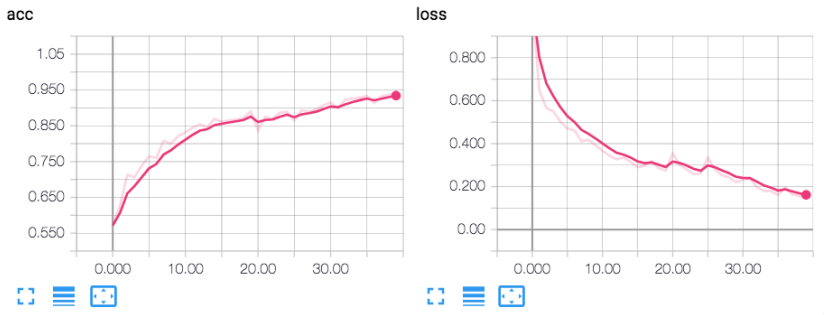
Figure 4.7: Images before and after being applied K-Means(Clustering)

Means(clustering) with k=3, meaning that it would be split it into three colors. In Figure 4.7 you can see the image before and after K-Means(clustering) is applied. You can see from the Tensorboard for the K-Means(clustering) training in Figure 4.8 that both validation accuracy and training accuracy seems quite similar, and also that they both seem to still be on an upward climb. This can means that if we were to run it for a longer period, the accuracy would increase. Potentially it could have reached a higher accuracy score than 0.9050.

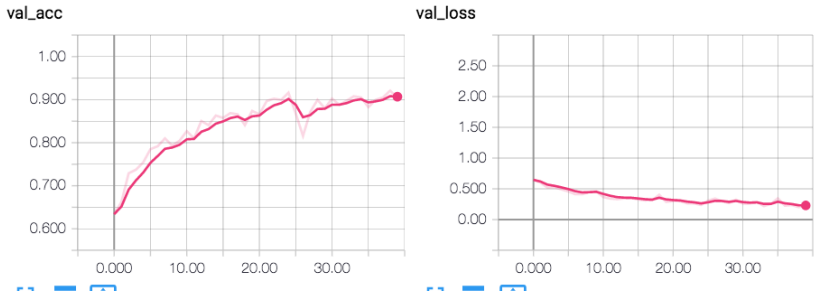
4.1.8 Adding More Classes to the Data Set

Initially, we gathered 93 background images, doing a 25% split. Adding to the old training set yield a slight improvement. Originally a VGG16 trained model on the same data set with same settings would yield about 92% accuracy. With this third data set, we got 96%. The increased percentage could be because the new class was easy to detect, and thus overall the classification accuracy would increase. Or because the model has inherently learned more about what qualifies as a cable.

To test this we did a new test run with all of the "nothing" images as part of the training set, and none as part of the validation set, as we are only interested in validation on the faulty/not faulty class. When adding only an empty directory for validation Google ML Cloud would crash. While it did work when attempting this on our local machine, it did not work while attempting it for Google ML Cloud. The solution was to add an empty directory with an empty file inside.



(a) Tensorboard for K-Means(Clustering)



(b) Tensorboard for K-Means(Clustering) Validation

Figure 4.8: Tensorboard for K-Means(Clustering)

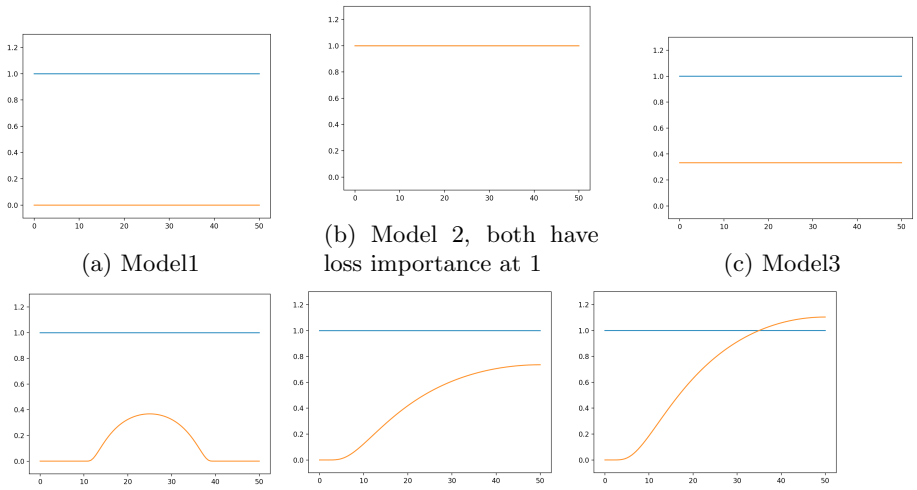
The result show that our assumption was correct with a validation score of 92%.

4.2 Guided CNN Architecture

For the guided CNN architecture testing, we tuned the relationship between the class_loss and XYLoss. In Table 4.6, constant x1 refers to a constant loss ratio where the XYLoss is multiplied with 1, before summation with the class_loss. The second way we concatenate loss is through a polynomial function, $f(x)$, where x is the current epoch number. In Table 4.6 one finds the function parameter passed to the $f(x)$ function, corresponding to that model training. A visual showcase of how the relationship between the loss for each of the models can be seen in Figure 4.9.

Table 4.7 shows the results of the experiments. We tested 3 different settings for a polynomial loss importance combination, and two models using a constant loss importance. From here we can see that in most of the instances, the results are worse than if we were only using class_loss. However, for Model 5, we see an increase of almost 2% from the baseline model. A discussion of the possible error sources for this result is given in Section 5.1. The results from model 4 are also interesting as it ends up at merely 50%. Meaning that something went horribly wrong. One can also see that Model 3 performs almost as well as the baseline model. However for model 3, we are greatly reducing the importance of the XYLoss, so it would almost be like we did not train with it at all.

In a situation where this architecture works as expected, one would expect



(a) Model1 (b) Model 2, both have loss importance at 1 (c) Model3

(d) Model4 (e) Model5 (f) Model6

Figure 4.9: Loss Importance

Table 4.6: Guided CNN models

Name	Learning rate	loss relationship
model1	1e-4	Baseline - only class loss
model2	1e-4	constant x1
model3	1e-4	constant x1 div 3
model4	1e-4	F(epoch,25,15,.35)
model5	1e-4	F(epoch,50,50,.35) x2
model6	1e-4	F(epoch,50,50,.35) x3

the recall to increase. Guiding training towards the correct place should yield a higher number of true positives. From Table 4.8 we can see that this is actually what happens. The Recall increases from 0.94 for Model 1 to 0.99 for Model 5.

In summary, our Guided CNN Architecture significantly outperforms the base-model using our polynomial loss importance combination.

4.3 Creating a Model Ensemble Using Stacking

4.3.1 Test Time Augmentation

The results from Table 4.9 only represent the results on the validation set. They do however not tell us how well the model would generalize to images taken in a different environment. The models for this experiment are the same as used for Table 4.4, Table 4.8 and Table 4.5. The result shows that the best model gets 98.37% accuracy with TTA(10) while the same model without TTA archived 0.9553 on the same data set. However, in this experiment, we conduct TTA on the model with the highest validation score during the training run. By looking at Table 4.9, Highest Validation score, and Validation Score with TTA(10), we

Table 4.7: Guided CNN architecture

Model	Validation Score with TTA(10)
model1	0.9512
model2	0.9383
model3	0.9480
model4	0.5129
model5	**0.9707**
model6	0.9155

Table 4.8: Precision/ Recall

Model	Average precision-Recall	F1Score	Precision	Recall
model1	0.94	0.95	0.96	0.94
model5	0.95	0.97	0.95	0.99

can see that it mostly decreases the score. The newly acquired score can be seen as a better accuracy metric for our model, as it attempts to mitigate the random chance element when a model performs better at certain rotations. There is also a problem with this experiment. As we conduct it on the model with the highest Validation Score, there is an information leakage, meaning that we can not trust the validation score to be an accurate measure of the accuracy, as we have used information from the validation test set when selecting when to stop our model.

4.3.2 Feature Engineering

In Table 4.10 modelCollection2 refers to the first 5 models from Table 4.9. When doing weighted average we performed it over the best performing fine-tuned model, and the 4 models trained with various augmentation techniques. The result of a normal average among them is an accuracy of 0.9025. However, if we assign 40% of the weights to the best performing model and 15% to each of the others, we get an accuracy of 0.9415. Interestingly we also archive 0.9415 when doing weighted-average based on doubling the percentages for the models where the Euclidean distance is zero. However, there are situations where a fault either stretches across grid indexes. If we change the max distance to 2, we get a accuracy on the validation test set of 0.9350. We can here see that it decreases.

Remark 2 *In Table 4.10 there is three results that are 0.9415. I currently can not give a appropriate reasoning for why, but my assumption is that it with the change from the model with dist 0 to the model with dist 1, there where no additional information leading the network to not adjust their weights. This is strange, but plausible because of the size of the data set.*

For the weighted average with 40 percent for the best model, the assumption is that the weighting would be quite similar to the distance-weighted averages. As there is one model that is a lot better than the rest. And such for all the distance adjustments, they would be done in regards to that model.

Table 4.9: Test Time Augmentation

Model	Highest Validation Score	Validation Score with TTA(10)
edge detection - Canny	0.8743	0.8474
edge detection - Bilateral	0.8603	0.8149
edge detection - CannyGaus	0.8768	0.8441
edge detection - K-Means(Clustering)(3)	0.9208	0.9025
VGG19 finetune - 1(8)	0.9636	**0.9837**
VGG19 finetune - 2(82)	0.9707	0.9805
VGG19 finetune - 3(83)	0.9777	0.9805
VGG19 finetune - 4(84)	0.9777	0.9480

Table 4.10: Using X and Y as a feature

Combination method	Model Collection	Accuracy with TTA(10)
Weighted avr(X,Y,dist = 0)	modelCollection2	**0.9415**
Weighted avr(X,Y,dist = 1)	modelCollection2	**0.9415**
Weighted avr(X,Y,dist = 2)	modelCollection2	0.9350
Weighted avr	modelCollection2	0.9025
Weighted avr(40% for best model)	modelCollection2	**0.9415**

These results should be looked at more closely, as the assumptions made in this remark have not been tested.

4.3.3 Random Forest/Adaboost

By using the same test split as we had for our training, we can achieve a score of 0.96 with AdaBoost, and a score of 0.96 with Random forest. However, these scores cant be trusted given the limited data set and needs to be further validated against a thirds data set.

Both Random Forest and AdaBoost perform worse than the best single model, and some reasons for that can be that our fine-tuning of these algorithms are not optimal. Another reason can be that we have too little data to train them on. There is also a high correlation between the results from all the 4 latest models, and this could impact the result.

Another way to combine models is using a weighted average. Generally there are two types, majority voting, also referred to as hard voting, and soft voting. In Table 4.11 modelCollection1 refers to all the models from Table 4.9.

In conclusion, none of the stacking methods outperformed the best single model on this test set. However, that does not prove that using a combination is a bad thing. An ensemble might be better at adapting to a new environment.

Table 4.11: Model Combinations

Combination Type	Models	Validation Score with TTA(10)
Median avr	modelCollection1	**0.9642**
Soft voting avr	modelCollection1	0.8279
Hard voting avr	modelCollection1	**0.9642**
Max avr	modelCollection1	0.9058
Min avr	modelCollection1	0.8279
Min-Max-median avr	modelCollection1	0.9058

Chapter 5

Discussion

In the discussion, we go over some of the potential problems with the results from this thesis. We also go over some potential ethical issues related to the usage of such a system. Lastly, we discuss future work and go into details on ideas for improvements of this research.

5.1 Specialized CNN Architecture For Single Point of Max Interest?

The new architecture yields promising results. However, the results have not been validated on a secondary data set and can be flawed in numerous ways. The result may be inherently dependent on this data set, and if that is the case, would not be applicable for other tasks. Secondly, when training the model we train them all for 50 epochs. Because of this, there is a possibility for the baseline to previously have had a higher score, than the new results with the guided CNN architecture. Not stopping at the highest accuracy means we don't have to bother with the information leakage that then would happen. If you stop the model when the validation score is high, you are using data from the validation test set to help train the model. However, for the test set to perform the best, it should be totally separate.

It is important to note that during this experiment, the result performed worse than our best model. A model trained only on class labels, gaining 98% accuracy. But these can not be compared as the new model architecture uses a new image generator and augmentation function. It is also using the Adam optimizer, as opposed to SGD used for the rest of the research. Furthermore, we removed the dropout layer when training the Guided CNN model.

5.2 Using X,Y as Features

The results of the research show that using (x, y) for model combination does not perform worse than normal weighted average. But given that it requires two very precise predictions before adjusting the weights, it would only be useful in a minority of the actual prediction cases. On this data set the gain was from

0.9025 to 0.9415. A major uptick. However, with a bigger image set, it is possible that the accuracy gain would decrease.

Given the limited size of the Class Activation Map, the 14×14 grid would only leave us with 196 possible points. This means that by chance, there would be situations where two models predict the same place. There is also a problem as shown in Figure 3.14d, that the fault is not always in the middle of a grid window. Sometimes it can span multiple different grids. In this case, we could have models predicting the neighbor laying points, while we, if we use a Euclidean distance of 0, would not see that they both found the flaw. Another problem is that this only works for faulty predictions, and not for "not faulty" predictions, as there are no clear parts of the image a Model should look when performing a "Not faulty" prediction.

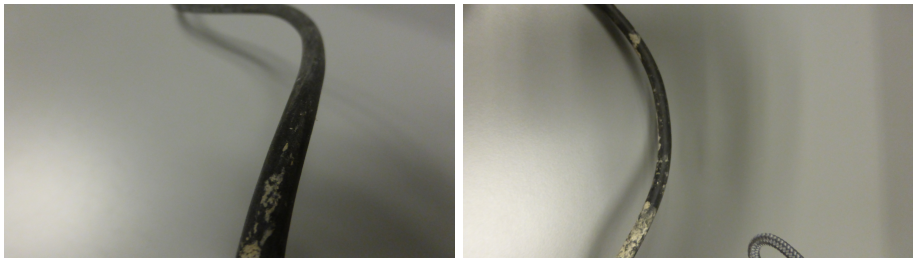
Even with these holes, we see this novel method as a promising way to combine models, as it more often than not gives us extra information about the performance of the models, given the data it is currently looking at, as opposed to only basing its combination on the model predictions on the training or validation test set.

5.3 Stacking Not Performing Better

The results in Table 4.11 and Table 4.10 show that neither of the new combinations of models outperforms the best single model. This does not necessarily mean that choosing the single model is the best method. The best model might be overfitting. And it might be that it would perform worse on real-life data. We might also be biased in our validation set, and thus not allow our other models to perform their best. It might be the case that they would supplement additional information under different light/environment conditions.

5.4 Problems

After performing Test Time Augmentation, and iterating half the validation directory to create a CSV we can use for the second stage layer, we started looking at the results.



(a) Good-Cable P1000733. Mislabeled (b) Good-Cable P1000643. Mislabeled

Figure 5.1: Mislabeled images from an old model

Figure 5.1 shows us that the model is performing inherently worse when the cables are dirty. And as all our models fail at this, it would continue to be a

problem unless we train on more similar data, or create models that operate on some preprocessed form of data where the concept of dirt is no longer that relevant. As of that, we can classify this as a training data problem, that inherently would be fixed once placed in a real-life environment as we would collect new data. these are results from a model trained on a previous data set split and using a different model architecture and does not seem to manifest itself with the current data split and current architecture.

The lack of correlation between real-life data and the in-house data also creates problems. Figure 5.2 is the edge detection model attempting to find a faulty cable, but in reality, ends up focusing on something else.

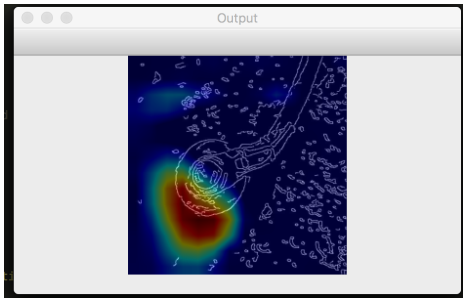


Figure 5.2: Faulty cable mislabeled without focus being on the cable

5.5 Potential Ethical Issues

When working with technology like artificial intelligence, you as a developer have an inherent ethical responsibility. The three main discussions that happen are "Stealing peoples jobs", "Surveillance" and if the machine learning is trained on biased data set, that would penalize a certain group of people. In this situation, the two relevant to discuss is "Stealing peoples jobs" and "Surveillance".

5.5.1 Stealing Peoples Jobs

An argument that comes up often when discussing AI is the "they are going to steal my job" argument. During development of this method, we made sure our focus was not to remove jobs, but rather to supplement them. And over time maybe act as a relief for the workers as EX-Inspection on easy to see areas(areas where we get a lot of pictures from), can be automated.

5.5.2 Surveillance

There is a possibility that this can become a surveillance problem, given that we would take images of what the workers are doing all the time. However, the platform is already heavily covered with surveillance cameras, so this project would not create a new problem. There is a notable difference in the use and storage routines for a surveillance system, and data used for machine learning. Because of this before such a system can be put out in production one would have to get it approved by the Norwegian Data Protection Authority. And

make sure to follow their rules for anonymization and securely storage of the information.

5.6 Preliminary Results

During a visit to the Ivar Aasen Platform. One of Cognite's employees discovered two cases of faulty cables. After sending images one of them back to us, we ran it against our algorithm and got this. Figure 5.3 shows the Class Activation Map for a VGG16 trained model, while Figure 5.4 shows a Class Activation Map for a VGG16 pretrained model with edge-preserving smoothing and edge detection.

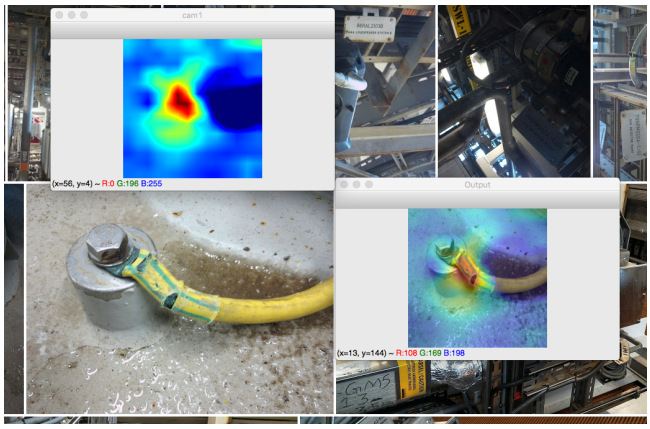


Figure 5.3: Algorithm detecting fault in a real cable from Ivar Aasen

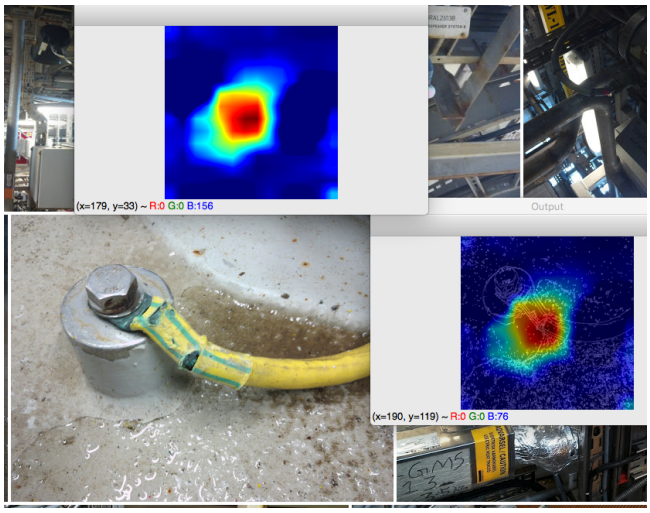


Figure 5.4: Algorithm detecting fault in a real cable from Ivar Aasen

Both models independently discover the fault, and the Class Activation Map is quite similar. From the Class Activation Map, we can see where the "atten-

tion" of the model is, during its prediction. This visualization is really important in making people trust the model. As if they see that the focus lay the same place as they themselves would focus, they are more likely to trust it.

5.7 Future Work

5.7.1 Guided CNN Architecture

Our Guided CNN architecture is an example of an attempt for a generic Domain specific network. This is because we used the domain-specific knowledge that a cable, has a specific point of max interest. Applying domain-specific knowledge to ML problems are what separates an OK model from a good model. Sometimes this work would be as part of the prepossessing step, or as here, a change in the model architecture itself.

We might want to attempt this with a network with auxiliary classifiers, such that we would not lose so much of the location information. Also having a network without all the max pooling and rather have GAP - Global Average Polling is also an interesting thing to test. As there is research showing that GAP networks contain more location information [16].

The best performing loss importance combination, as seen in Figure 4.9e, does resemble a linear function. Because of this, future work should look into whether using a linear function would yield better results.

As the Guided CNN architecture so far has only been tested on one specific data set, future work should validate the results on new data sets to see if this method is generic.

Future research should look into comparing this method with the state of art box predicting methods, in order to see if the XYLoss based on the Euclidean distance perform worse than equal loss for all pixels inside the faulty region.

Sergey Zagoruyko does in his thesis *PAYING MORE ATTENTION TO ATTENTION* [15] propose a similar solution, in which a student network would be trained based on the class activation map of its teacher. Where the use case of his research would be in parameter reduction of the model while still maintaining the same accuracy. This is quite useful for neural networks that should run on mobile devices. Comparing the results of that attention transfer learning with how attention transfer learning using the Euclidean distance is something that should be studied further.

For most data sets there exist no annotated data set over where the focus should be. Because of this future research should be attempted to use this without an annotated data set. The proposed implementation is that one would have multiple models training simultaneously. When one model correctly perform a fault prediction on an image, that some of the other models did not correctly classify, we can use the GRAD-CAM output of that prediction, to guide the other models. The models should be trained on subset on the data set, and on different augmentations of the data set, in order to reduce correlation between them. With this, they hopefully would gain a higher accuracy through helping each other.

5.7.2 CNN

Our method of calling the prediction function with test time augmentation multiple times increases the processing required to do a prediction. In the future, one should attempt to redo the research using a rotation invariant CNN model [6] instead.

Another method that would be interesting to research is to use key points in the image, to detect the rotation and bendness of the cable [2]. This can then be used to rotate the cable such that all cables are lined horizontally when training, and testing. However, this would need careful and time-consuming annotation of the data set.

In order to create a new data set with more different backgrounds, it seems interesting to test out segmentation of the images to remove the cable from the background such that new synthetic images with the same cable in various new backgrounds can be created. The new backgrounds can, for example, be from Google.

Chapter 6

Real World Application

In order to help Aker BP solve the problem the problem they have expressed concerns for, we have designed an automated prediction system with the goal of performing automated EX-Inspections. This system would use the algorithm developed in the previous chapters but would consider them a black box algorithm. In this Chapter, we will first go over the overall system, before looking into the details of one of our camera applications. Afterward, we would go over the missing pieces that are hindering us from pushing this into a production environment.

6.1 Overall Prediction System

The overall prediction system is the system that would take advantage of the already trained classification model. It is important to understand how the system is going to work, before going ahead training the model, as choices made during training affect the feasibility of certain things. For one, we are using Test Time Augmentation and possibly a combination of various models. Running this fast, without a GPU would be troublesome. Given the fact that all devices on an oil rig have to be EX-certified, they would not contain the latest and best hardware. As a result, having the prediction performed on the user's device is not optimal.

Secondly, we have to weight the use of running parts of the processing on the user's system or leave it all to our system. Leaving it on the user's side would help our system scale better. One thing we could do on the user's side is running the algorithm that would detect where in the image there is a cable, such that it could crop it out and only send that piece of the image to our backend server. This would allow us to do less processing on our side. The raw images contain additional information in the image that we at a later stage would want to use, so there are benefits of gathering the raw images. If we do not crop the image we quickly are able to gather the biggest data set of images from an oil rig. Allowing us to use parts of the same data set when expanding to other use cases.

The first step of the system is the image capture system. Here we developed two applications, with a clear difference in how they were implemented, based on the hardware they should work on. One designed to be a Voice Controlled

system, for the RealWear HMT-1 [19]. The second as a background service for the Orbit X [20]. The next step is the overall system that takes the new images, and tell our system to perform a prediction on them. The third step is the prediction function, this is the model we have previously trained.

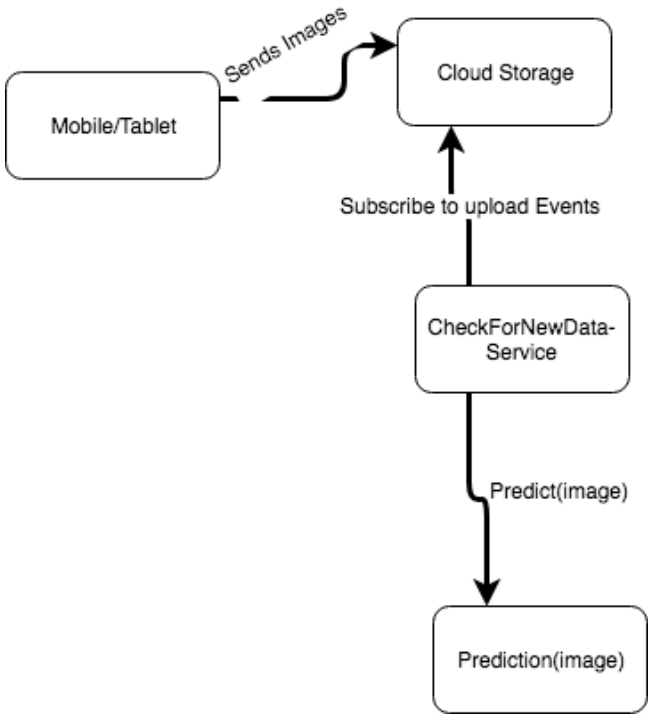


Figure 6.1: Overview of the architecture for the overall system

In Figure 6.1 one can see an overview of the system described in step two. Here the image capturing systems would push images to our cloud storage where our check "For New Data" service has subscribed to upload events and would be notified of uploads. Following the publish/subscribe pattern [21]. The service would so perform a prediction on the images. Currently, the checkForNewData-service have not been implemented.

6.2 Orbit X Camera Application

We created two different applications for automatically capturing images from mobile devices. This section is about the implementation of the Orbit X application. A head-mounted camera, using Android, without a screen. And the problems that arose because the developers of the camera made some questionable architectural decisions.

[19] The realwear hmt-1. `https://www.realwear.com/products`, 2018.

[20] Orbit x explosion proof wi-fi camera. `https://www.pixavi.com/product/mobile-devices/explosion-proof-wifi-camera/`, 2018.

[21] Publish/subscribe. `https://msdn.microsoft.com/en-us/library/ff649664.aspx`, 2018.

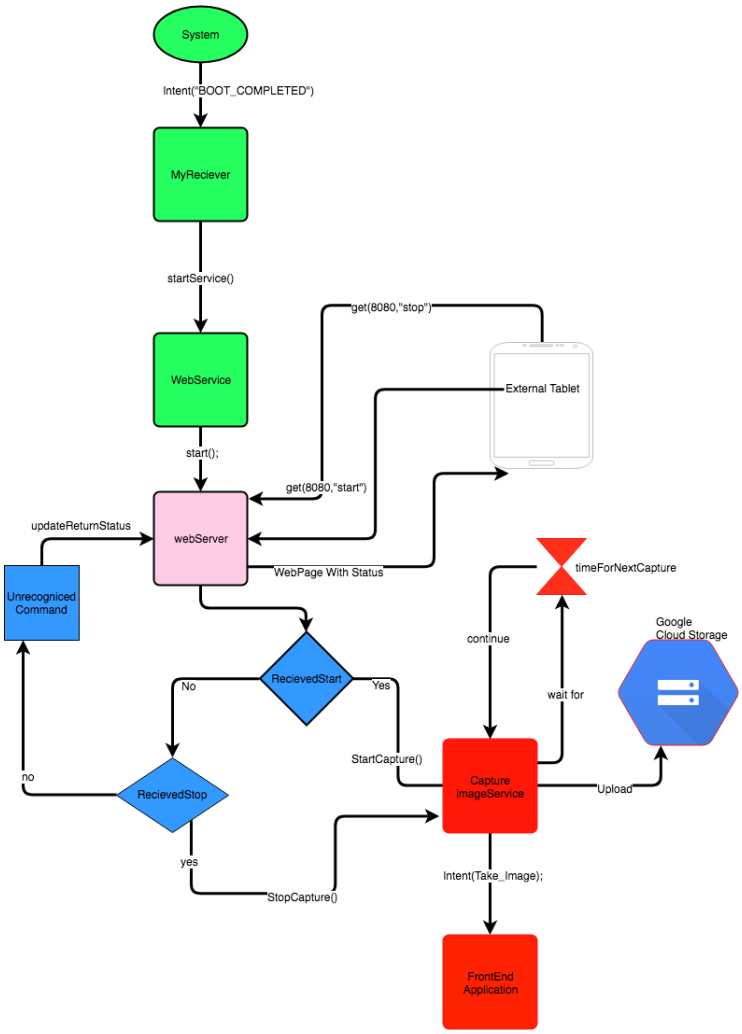


Figure 6.2: Overview of the Orbit x Camera application flow

There are three buttons on the camera, controlled by a front-end application called Sipido, that is the "brain" of the camera. Its also important to note that on Android, only the front end application is allowed to take a photo, as there has to be a preview window. This means that in order to add our custom image capturing, we generally would need to change out Sipido, and thus making the rest of the device obsolete. This would happen because their SDK does not export functions to re-implement the buttons that Sipido control.

A second method would be to create a backend service and trigger it to launch somehow. Given the nature that only the frontend application can receive intents about button clicks, we can not start it by a button click. The Orbit X, does talk to a phone/tablet over Bluetooth, but as of yet, this communication is also not possible for a third party app, to use.

Figure 6.3: Person with OrbitX camera on the head

The solution

The solution we came up with was to have a background service with a HTTP server, that would allow the users to turn the image capture feature on and off. The workflow diagram of this is visualized in Figure 6.2. The background service starts when the device turns on, and the customer would enable the image capturing by going to the website the HTTP server for the camera is exposing on the local network. From there one can check the status of the camera, and turn the image capturing on and off.

The Activity diagram showing the application flow is color-coded, such that red represents the capture image service, that would only be run if the active status is set to "start". When this service is running, the device does also take a wake lock to make sure that the system would not go to sleep. Green represents the bootup phase. Currently, this is done with a listener registered for boot_completed. Pink represents the main service, our HTTP server on port 8080, that would contain a web page where the user can see the status, and start/stop the automatic image capture service. The code for parsing the commands received from the client is shown in blue. Either start/stop the service, if neither of those, we tell the client that something wrong happened.

6.3 Data Problem

The Camera is designed to take images at a fixed interval. In Figure 6.3 one can see a person with the Orbit X camera mounted to the helmet. For the system to work, our system needs to parse these images and find out where in the image the object of interest is. In our case, where the cable is. While doing so we can also take a shortcut. As from talking to EX-Inspectors, we have learned that the flaws are almost always at the end of a cable. They personally had never

experienced a flaw to be in a Cable Tray. This means if we can annotate a data set based on finding the end parts of the cables, we can use this to perform a crop, and send it into our model classification ensemble.

To do this we need to craft a new data set, as we need to annotate the data set with boundary box information. Once this is done, one can try to train YOLO, or a similar network.

(a) Orbit x image from Ivar Aasen (b) Blurry Orbit x image from Ivar Aasen (c) Orbit x image from Ivar Aasen

Figure 6.4: Orbit x image from Ivar Aasen

In Figure 6.4 parts of the newly acquired data set is shown. From the new data set, we can manually crop out areas of interest and perform a prediction on them. However, all of them are without a flaw meaning that testing the recall for a faulty cable is not possible. We would only be able to check how well it performs at classifying images as good. And use it to train a yolo network for automatic cropping of the endpoints of a cable. However due to time limits, and limit of the new data set size, we were not able to do this in time.

6.4 Testing

The camera application we created for capturing images on oil platforms did not perform as expected when tested in a real-life situation. Many of the images, an example can be seen in Figure 6.4b had poor focus and got blurry. It might be possible to mitigate this by enabling "low light boost" or otherwise lowering the shutter speed. However, given the architectural implementation as discussed in SubSection 6.2, there is no feasible way to change this without having the developers of the camera release a new software update.

As a response to this, we might be forced to use another camera, and thus create a new application. As the camera application is, in this case, device specific.

6.5 What is Missing?

In order to push this into production, there are two things missing. With the first problem, it all comes down to data. There has to be an algorithm trained to extract the end point of a cable before our algorithm can then classify it. Currently, we do not have enough data to train an algorithm to do this. This is a pressing issue that is attempted to be solved now, by actively gathering more data.

There is also not a plan in place for automatically gathering of new Faulty Cable data, aside from the few instances of faulty Cables that our algorithm would find. This limit the accuracy gain the system will get, even when it is in production. In Figure 5.3, that will be discussed more in the next Chapter, we see a real-life fault discovered on the Ivar Aasen platform. Proving that the classification technology does work on new data.

The second problem is that we need the system to be able to tell where the image was taken. There are three main ideas being proposed. The first idea is to expect that the crew is able to determine where the photo was taken, based on seeing the photo. Because the oil rig is quite big, and the image can be quite close up, this is prone to failure and could in some instances be very costly, as they might have to check multiple places before finding out the location of the faulty cable. Next idea is to rely on GPS location. However, based on tests previously performed on the oil rig, the accuracy would not be good enough. In some instances, there would be a fault margin of a meter, while in other situations the fault margin would increase to 20 meters. The best solution is to use a technology developed by Google called VPS [22]. VPS stands for Virtual Positioning Service. However, this method is currently not publicly available yet. For the time being, our system will have to rely on GPS in combination with a humans assumption about where the image is from. While in the future we hope to utilize Google's Virtual Positioning Service.

[22] Google has an indoor positioning tech in the works, called vps. `https://techcrunch.com/2017/05/17/google-has-an-indoor-positioning-tech-in-the-works-called-vps/`, 2018.

Chapter 7

Conclusions

In this thesis, we have explored state-of-the-art image classification algorithms applied to automatic EX-Inspection on oil platforms, where a key additional challenge has been the limited amount of data. In addition to developing and discussing various Machine Learning models, we have also developed a pipeline that takes care of everything from automatically capturing images on the platform, to the prediction stage. With the various parts actively being tested on the Ivar Aasen oil platform.

Moreover, By improving the existing state-of-the-art, we have developed a new highly accurate model. This new model is capable of detecting real faults on the Ivar Aasen oil platform, while also locating the position of the fault in the image. Furthermore, we have shown how important the data split is, in situations with low data, as a poorly performed split, can greatly impact the training. We have also demonstrated the importance and usefulness of pre-processing the data. Finally, we have shown that off-the-shelf model architectures trained using fine-tuning are sufficient to create a solid model.

Recall the three research questions put forth in the introduction as primary objectives in the thesis. The second of these questions posed the problem of going beyond the current state-of-the-art and try to use attention to actually improve the result. The underlying intuition was that a specialized CNN architecture would be more suitable for single point of max interest classification problems. Herein, we have shown that our novel model architecture, Guided CNN, which utilizes attention during training indeed beats the baseline model. The author would like to stress the innovative nature of this result.

Our third question was to find out if stacking would improve the overall accuracy. We discovered that for these models on this data set we were not able to improve beyond the best single model. However, we still maintain that stacking can be useful. We also showed that our novel idea of stacking using attention provides an increase of 4 percent over model averaging, where all models are weighted equally.

In conclusion, we have shown the usefulness of a CNN network for automatic EX-Inspection, and come up with two novel methods, guided CNN and model stacking using attention, that deserves publications on their own. The fully automated system is not completely ready for being pushed to production yet; however, this is something that should happen in the near future.

Bibliography

[1] Cs231n: Convolutional neural networks for visual recognition. `http://cs231n.stanford.edu/slides/2017/cs231n\_2017\_lecture6.pdf`, 2018.

[2] Keypoint detection - stanford.edu. `https://web.stanford.edu/class/ee368/Handouts/Lectures/2014\_Spring/Combined\_Slides/12-Keypoint-Detection-Combined.pdf`, 2018.

[3] Overview of hyperparameter tuning cloud ml engine for tensorflow. `https://cloud.google.com/ml-engine/docs/hyperparameter-tuning-overview`, 2018.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning - Page 236, chapter 5*. Springer, 2006.

[5] Mehmet Ufuk Dalmış, Suzan Vreemann, Thijs Kooi, Ritse M Mann, Nico Karssemeijer, and Albert Gubern-Mérida. Fully automated detection of breast cancer in screening mri using convolutional neural networks. *Journal of Medical Imaging*, 5(1):014502, 2018.

[6] Sander Dieleman, Kyle W Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notices of the royal astronomical society*, 450(2):1441–1459, 2015.

[7] Qin Huang, Chunyang Xia, Chihao Wu, Siyang Li, Ye Wang, Yuhang Song, and C-C Jay Kuo. Semantic segmentation with reverse attention. *arXiv preprint arXiv:1707.06426*, 2017.

[8] Qin Huang, Chunyang Xia, Chihao Wu, Siyang Li, Ye Wang, Yuhang Song, and C-C Jay Kuo. Semantic segmentation with reverse attention. *arXiv preprint arXiv:1707.06426*, 2017.

[9] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[10] Michael A. Nielsen. Neural networks and deep learning, chapter 6, 2018.

[11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[12] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *See https://arxiv. org/abs/1610.02391 v3*, 7(8), 2016.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] Francisco J Valverde-Albacete and Carmen Peláez-Moreno. 100% classification accuracy considered harmful: The normalized information transfer factor explains the accuracy paradox. *PloS one*, 9(1):e84217, 2014.

[15] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*, 2016.

[16] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2921–2929. IEEE, 2016.

Appendix A

Process

Knowledge

During this thesis, the lack of knowledge for the topic in question was a big problem. It meant I would not be able to estimate how big tasks would be, and would force me to spend a lot of time on reading up on the basic. To mitigate this I participated in weekly internal seminars on Machine learning. Another way I worked on mitigating this was through taking an online course on Coursera on RNN, to improve my overal knowledge on AI. As well as the math behind it. RNN, Recurrent Neural Network instead of CNN, as at that time I where already fairly comfortable with the math behind CNN. We also performed daily standup meetings among the members of the machine learning team at Cognite. This allowed me to easily get input from people that knew more about the subject than I.

System Development Model

During this thesis I used a Kanban inspired system development model. As mentioned in the pre-project, the use of Kanban is based around the fact that we did lack a lot of knowledge during this thesis. In the start we did not know anything about AI and we did not have a data set to work on. These things forces us to choose a highly adaptive agile methodology.

By choosing a Kanban inspired structure over a Scrum inspired for our one man project, we would be allowed to skip the overhead of retrospective meetings and lock-in.

WIP

A big part of Kanban, is limiting the Work In Process, the amount of active work objects. However for this thesis, we had to look partly way from this. During the thesis we would work on writing the thesis, and developing a convolutional neural network simultaneously. Training a CNN network takes time, so to increase efficiency, we would be allowed to work on many tasks simultaneously. Because if not, there would be downtime where we basically would just wait for a training run to finish.

There where also not performed any lock-on of the WIP. Such that if one started on a task, that was deemed harder than expected, one where free to put it back in the backlog. There where also no sprints, where one would look in the backlog as with scrum, as the goal was for the most agile methodology, such that we could change our focus, as often as we learned new things.

Kanban Board

We used a Kanban/scrum board on Trello to monitor the backlog and current active tasks.

Standup

There was a daily standup meeting before lunch. From this we where able to explain our current focus, and brainstorm potentially ideas for troubling parts. The standup is often used as a way to let the rest of the team know what the others are working on, to avoid work collision, however in this instance this was not used. The reason for this is that it was only I working on this specific project.

Product Owner

During the pre-project, we planned to have a meeting every Friday with the product owner. Because of trouble allocating time for this meeting, this was reduced to the occasional ad-hock meeting when met in the hallway. The product owner where also showed working prototypes of the solution, but only when one was done. There where no obligation to produce a new version at a certain interval.

A.1 Tools Used

In Figure A.1 the development tools is shown. It is split into Version control, Task Tracking, External Cloud Services, IDEs and Code Quality Tools.

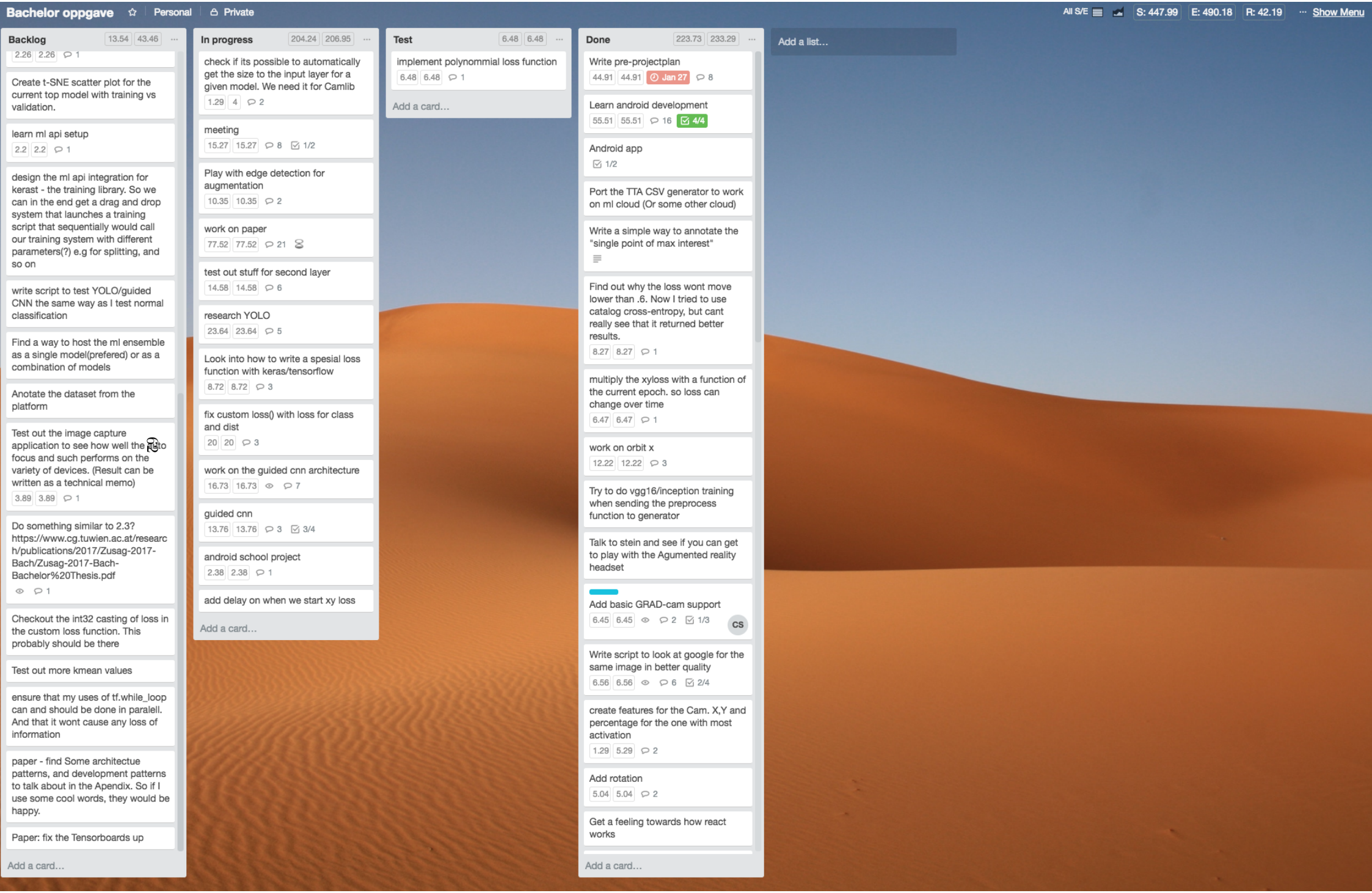
For IDEs, we used Sublime Text and Xcode mostly for the python specific code. Android Studio was used for our two Android Applications. We also did some development inside Google Colab notebook. That is a Notebook where one can easily test out code, while having it run on one of Google's GPU instances, such that one could perform training runs much faster than on a CPU.

For quality assurance tools, we used Pylint to make sure we followed the pip8 standard, while we used SonarCube for Android to control the quality.

All Code was version controlled through git, and saved on bitbucket. Trained models would be stored on Google cloud storage. And for external processing power, we would use Google Colab and Google ML Cloud.

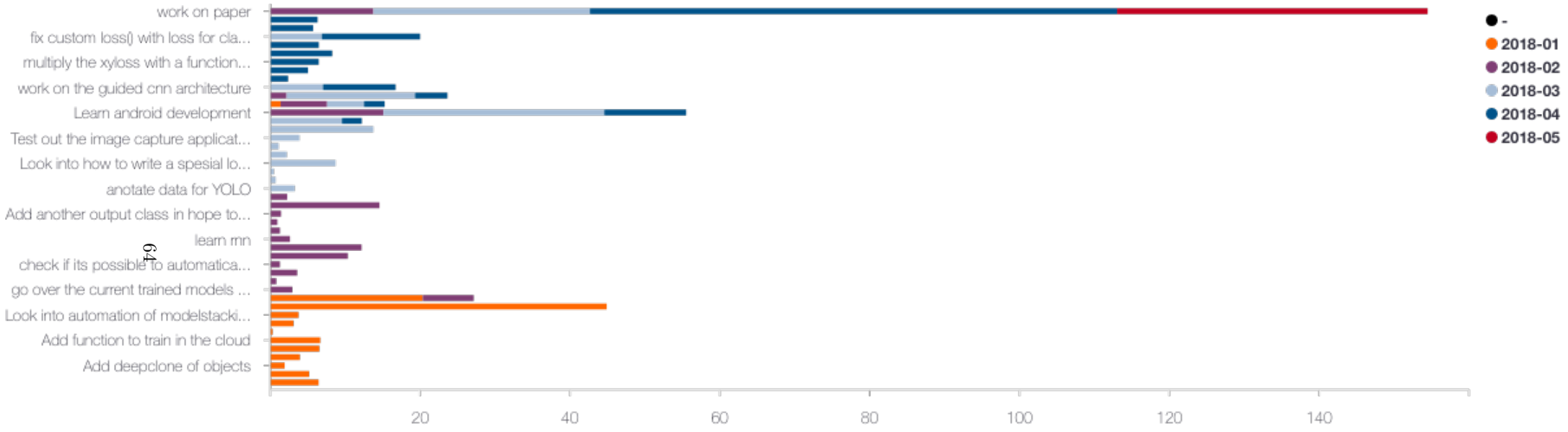
During this thesis task progress was tracked through Trello. We also took advantage of the Chrome plugin Plus for Trello to have integrated time tracking. As mentioned in the pre-paper.

A.2 Trello Board


Bachelor oppgave
Personal
Private
All S/E
S: 447.99
E: 490.18
R: 42.19
Show Menu
Backlog
13.54
43.46
2.26
2.26
1
Create t-SNE scatter plot for the current top model with training vs validation.
learn ml api setup
2.2
2.2
1
design the ml api integration for kerast - the training library. So we can in the end get a drag and drop system that launches a training script that sequentially would call our training system with different parameters(?) e.g for splitting, and so on
write script to test YOLO/guided CNN the same way as I test normal classification
Find a way to host the ml ensemble as a single model(prefered) or as a combination of models
Anotate the dataset from the platform
Test out the image capture application to see how well the auto focus and such performs on the variety of devices. (Result can be written as a technical memo)
3.89
3.89
1
Do something similar to 2.3? https://www.cg.tuwien.ac.at/research/publications/2017/Zusag-2017-Bach/Zusag-2017-Bach-Bachelor%20Thesis.pdf
1
Checkout the int32 casting of loss in the custom loss function. This probably should be there
Test out more kmean values
ensure that my uses of tf.while_loop can and should be done in paralell. And that it wont cause any loss of information
paper - find Some architectue patterns, and development patterns to talk about in the Apendix. So if I use some cool words, they would be happy.
Paper: fix the Tensorboards up
Add a card...
In progress
204.24
206.95
check if its possible to automatically get the size to the input layer for a given model. We need it for Camlib
1.29
4
2
meeting
15.27
15.27
8
1/2
Play with edge detection for augmentation
10.35
10.35
2
work on paper
77.52
77.52
21
test out stuff for second layer
14.58
14.58
6
research YOLO
23.64
23.64
5
Look into how to write a spesial loss function with keras/tensorflow
8.72
8.72
3
fix custom loss() with loss for class and dist
20
20
3
work on the guided cnn architecture
16.73
16.73
7
guided cnn
13.76
13.76
3
3/4
android school project
2.38
2.38
1
add delay on when we start xy loss
Add a card...
Test
6.48
6.48
implement polynommial loss function
6.48
6.48
1
Add a card...
Done
223.73
233.29
Write pre-projectplan
44.91
44.91
Jan 27
8
Learn android development
55.51
55.51
16
4/4
Android app
1/2
Port the TTA CSV generator to work on ml cloud (Or some other cloud)
Write a simple way to annotate the "single point of max interest"
Find out why the loss wont move lower than .6. Now I tried to use catalog cross-entropy, but cant really see that it returned better results.
8.27
8.27
1
multiply the xyloss with a function of the current epoch. so loss can change over time
6.47
6.47
1
work on orbit x
12.22
12.22
3
Try to do vgg16/inception training when sending the preprocess function to generator
Talk to stein and see if you can get to play with the Agumented reality headset
Add basic GRAD-cam support
6.45
6.45
2
1/3
CS
Write script to look at google for the same image in better quality
6.56
6.56
6
2/4
create features for the Cam. X,Y and percentage for the one with most activation
1.29
5.29
2
Add rotation
5.04
5.04
2
Get a feeling towards how react works
Add a card...
Add a list...

Throughout this project we have used Trello, to keep track on active tasks, and the workload that is left. There is a lot of tasks in the backlog as this is an ever changing project where, because of the lack of lock-in on active tasks, we would adjust our focus to the most pressing issue. This means that when more pressing issues, or other ways change in focus, the backlog would just fill up.

A.3 Time Spent on Task Diagram



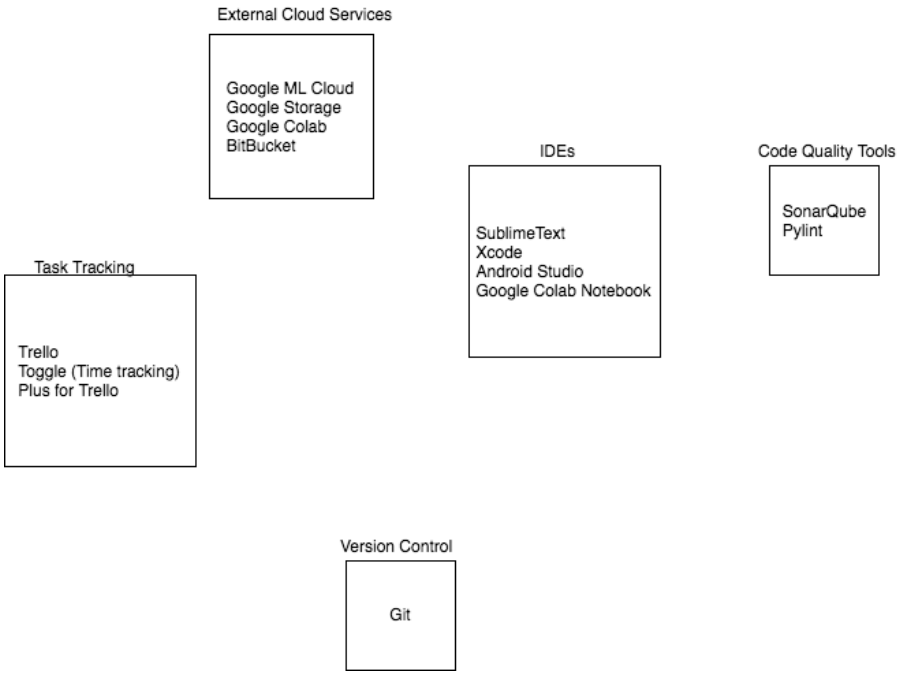


Figure A.1: Development toolchain tools

From this one can see that my biggest task was working on the thesis, while secondary was working on android development. Where I developed two applications. The timer for Android Development also do take into account some time used for developing school android applications. The biggest single task in reality is probably the Guided CNN implementation, but a lot of that work was done while simultaneously working on other tasks. Therefor measuring the exact time is hard. The time-used chart, created using "Plus for Trello" is resized to fit the this page, and therefor the name of many of the tasks is not visible.

Time is measured from I get to the office til I leave. With lunch assigned to the current active task. - The accuracy of this time chart is questionable. For one, a lot of this tasks is write something, and wait while you are testing it. Because of this one would work on multiple projects at the same time. Some tasks are also interconnected and one would complete parts of it without thinking about changing card. Another reason for inaccuracies is that changing would be forgotten, or that one would not care to make a new item for small things. E.g Every second week there is a one hour long meeting where people explain what they have done/working on. However I have assigned this time to the task I where currently working on.

The charts also does not take into account any time spent during weekends or after I leave the office.

If the chart had been grouped by week one would see that during Easter I did not "work". However I did work on both my paper and my Guided CNN Architecture during that time. There is also a 3 day period where I did not work. During in which I where sick.

A.4 Milestones

Four deadlines where mentioned in the preproject. This is a summary as to how they went.

1. 20.1.18 Delivery of the pre-project plan.
    - Delivered before without a problem.
2. 31.1.18 Deadline for initial training data.
    - This deadline was almost overstepped. We got a data set version one, about the 15th of January, however that did not have the quality we where looking for. We then set a deadline for the 27th of January, and if we did not have acquired it, or had a solid plan for acquiring it, we would change use-case. The 27th we decided to hand make the data set, as we did not find a better solution. The data set where handmade the 30th of January.
3. 15.3.18 Preliminary prediction model.
    - Already 09.02.18 did we have a preliminary working model, that had been tested on real life images. The result of that test can be seen in Figure 5.4 and Figure 5.3. We where able to do this so quickly after gathering the data set, as we had already implemented the basic functionality for all of it, and testing it on a public data set. Such that we would not fail to meet this deliverable, in case of problems with gathering of the data set.
4. 01.05.18 Almost all coding should be done and focus should be on the report.
    - We changed the deadline to 15th of April, such that I would have one month of time to work on the thesis. I finished the CNN research the 16th of April, one day over time.

A.5 Meeting Notes

Attached is a subset of the meeting notes with Ivar Farup. This notes are selected to demonstrate that we had a problem with data, that not including the architecture in the thesis itself was a planned decision, and that the results of the thesis, in the end, ends up being really interesting.

Meeting with Ivar - Date January 12th

Participants
Ivar Farup
Cim Stordal

Agenda

1. Introduction to the project, and who each of us are.
2. Tools to use during this project.
    - Git.
    - Python Code Quality Standard.
    - Trelloboard.
    - Timer for how long I work.

Results

1. Agreement.
    - To use pip8 standard without maxline of 79.
    - Skype meeting every Wednesday at 1 PM.
2. Before next Meeting.
    - Ivar would read my preproject.
    - Ivar would ask a PHD student if he where interested in helping me with Tensorflow related questions.

Meeting with Ivar - Date January 24th

Participants
Ivar Farup
Cim Stordal

Agenda

1. The lack of data.
2. Mechanical Turk idea for acquiring data.
3. My backup project in case we where unable to acquire data.

Results

1. Agreement.
    - Hard limit for acquiring data set for Friday. However also have to define what is "having data".
    - Write about my backup use-case and data problem in the preproject.
2. Before next Meeting.
    - Cim - Acquire data.

Meeting with Ivar - Date April 18th

Participants
Ivar Farup
Cim Stordal

Agenda

1. Only have architecture in the Apendix?
2. Review of the current version of the thesis.
3. Results from Guided CNN Training

Results

1. Agreement.
    - We agreed that my proposal to only write about architecture in the apendix is allowed as to the parts that would be evaluated on. However it is against the normal procedure of a bachelor paper. So its unsure if I would be penalized for it.
    - Section 2 and 3 in the thesis is okay, while 4 and 5 are a bit worse.
    - The results from the Guided CNN training seemed really interesting. Ivar also instructed Cim to consider publishing that as a standalone academic conference paper.
2. Before next Meeting.
    - Cim - Fixup thesis according to the remarks Ivar had.

A.6 General Notes

During the pre-project we determent that it was a lot of unsure elements to this project, and that it was beneficial to work close with the costumer so we could get feedback, and have really fast iterations. As the resulting architecture show, this was a good decision. The application is no longer a part of the Operation Support App, but rather shifted towards being a Camera. The start/stop button, of the camera application, could at a later stage be put as part of the Operation Support App. The original idea was a application where the Inspector would take a picture while conducting an inspection, for so to have it evaluated, simultaneously as he would look at it. This was based on a assumption that the EX-Inspector would potentially do a bad job. However as many of the potentially faulty places is hard to see from one angle, this method is hardly the best. We also discovered that in some situations it could go u to 12 months between inspections. And our current method tries to address this issue by focusing on gathering the best coverage of the oil rig. Such that we can perform predictions everywhere, everyday. Instead of attempting to double check the work of an EX-Inspector.

Appendix B

Technical Memo

B.1 Architecture decision for the Orbit X application

Given the architecture of the Orbit x, where there is a frontend application that is responsible for the buttons, and there is no easy way to re-implement their behaviour, we created a background service that starts a web service listing on port 8080. by connecting to that service we can start/stop our image capturing, and capture the images only by sending an intent to the frontend application.

During testing we had technical issues, regarding the saving of the images, as they sometimes would be saved as the same name. The developers where helpful and managed to patch the bug withing 48 hours. See Figure B.1 for parts of the email conversation I had with the developers. We also had a live test on the Ivar Aasen platform the day after they patched the bug, so that they managed to patch it in 48 hours was of utmost importance for us.

B.2 Language for the Project

Choosing the correct language for the project is an important task. In machine learning the most prominent languages are Python and R. For multiple reasons I ended up choosing Python as my language. For one, this was a language I previously had worked quite a bit with. Secondly that's what the team at

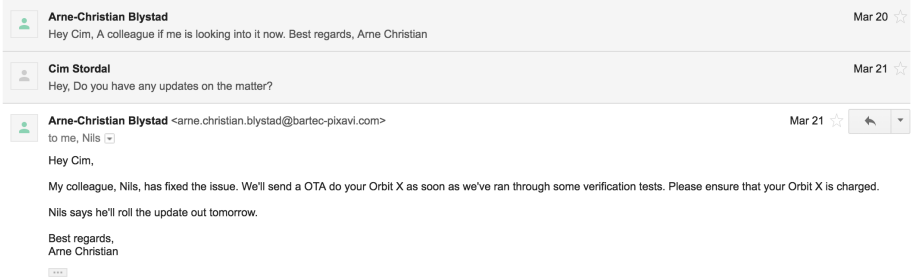


Figure B.1: Orbit X bug - email conversation with developer

Cognite uses, and I would need a good reason to not follow them.

B.3 Change in how the Image Application Works

Taking an image before the inspection, vs what we are doing now. The reason why we are not going to focus on taking an image before an inspection, for so to analyze it, and act as a double checker for the work of an EX-Inspector is that the points they would access is often behind objects. Such as it would be unfeasible to have a single image capture all the necessary information. The more apt implementation is to focus on gathering data from across the whole platform. This boils down to a problem with the EX-Inspection routine. It can go up to 12 months between certain areas being checked. Our method by capturing images, all the time, from all persons on the platform, allow us to build an image database that can analyze big parts of the platform, everyday. Also it would take into account that certain flaws are a result of human errors, e.g the cable fault from Ivar Aasen earlier is most likely the fault of a human stepping on the cable.

B.4 Deciding Between API Integration and Guided CNN Research

In the second week of March I had to plan the upcoming workload. I had two big projects that I had to choose between. It was hard to estimate them, but both where put at clearly above two weeks of work. Either making this system production ready, and get it all into our API. Or focus on the research and test out my idea for a Guided CNN Architecture. To select between these two, I first had to find out if the idea was even viable.

After searching through paper after paper, it was not obvious that anyone had tried it before, or atleast published it. However given that there is a publication bias, favoring positive results, someone might have tried it and already found out that it wont work. The rest of the pre-research can be divided into three steps.

Step one was to talk to the researchers at the office, who all found the idea to be interesting, but lack domain knowledge in computer vision to say if it would be viable. Second step was to reach out to someone with domain knowledge in computer vision. Here I reached out to a PHD student, and another Norwegian company focusing on computer vision. Who both said that they did not know if it would work or not.

Some of the problems that was raised is the loss of information due to the max polling layers. However they where interested in knowing the result.

For further guidance I consulted Ivar Farup, my supervisor. To discuss what would be the most interesting for the paper. The same question where asked to my internal supervisor at Cognite. Both said that focusing on new research would be the more interesting part.

The last part of this decision making, was to confirm with Stein Danielsen, the product owner about what they would be most interested in. He replied that both would be interesting, but in the near future the API implementation would have the biggest impact. However he also noted that I should choose the

thing that I would be the most passionate about, as that would lead to a better result.

In the end I decided to focus on the Guided CNN Architecture, in the pursuit for something unique to add to this thesis.

B.5 How to Know Where The Image Was Taken?

Ideas: GPS, last found Tag in an image, or just assume they would know the platform well enough from the image. GPS works quite well for certain areas of the platform, but for other not so well. Last tag in image is a bad method as it might be a long time since the last time it found a tag. Its also naive to believe that the workers know every part of the platform, just by being shown an image.

The solution: The most promising solution to this problem seems to be the use of Google's research on Augmented Reality(Visual Positioning Service(VPS)).

Appendix C

Pre-project Plan

Project plan

Automatic EX-Inspection

Written by
Cim Stordal 471174
For
Cognite AS

1. Background and Goals

1.1. Background

Today, before any operation can start on an oil rig, a full EX-inspection has to be carried out. The EX-inspection is a visual inspection of all electronic equipment on the rig (or at least in the particular zone) that checks that all cables, connectors, cameras, smoke detectors, electrical cabinets, etc don’t have visible defects that could cause a spark or heat to occur. The spark or heat could ignite gases in the air, which could be likely to emerge in explosive concentrations in some areas of the rig. The EX-inspection is done manually today, with visual inspection of all the aforementioned points. This is very time consuming and causes less production and more downtime, as it has to be performed after any planned or unplanned stop before production can be restarted.

1.2. Project Goal

This bachelor thesis will build a completely automatic visual inspection tool using a computer camera and machine learning to automatically detect anomalies on electrical equipment like frayed cables, missing insulation, corrosion, broken or damaged seals on ex-proof cabinets and other problems that could trigger an explosion.

2. Scope

2.1. Field of Study

The interest is to create a product that would help to minimize the workload of an EX-inspector. And in the end could work as a substitute for them in the case of visual inspection, however, in the start it would work more as a “failsafe”. Double checking the work of an EX-Inspector.

2.2. Planned Limitations - Scope

Everything related to the running of the application would be handled by Cognite AS.

The mobile application(Tablet frontend) would be created as a simple webpage application as part of the operation support app Cognite AS already have. This allows us to not write any native dependent code as we will then not have to maintain multiple implementations. It's also important that the application follows the same principles as other Cognite AS applications.

The detection algorithm are limited to detecting only what we manage to acquire a good dataset for. Currently the hope is that we can gather an imageset for “cables with hole” and “car battery corrosion”. However this might end up being harder than we expect, and if we only acquire for one of them, that's okay.

2.3 Time and Resource - Scope and Limitations

As this is a bachelor, there is an implicit time limit to completing the application and paper within the 16th of May. However, most of the application should be done sometime before that, as during May, the main focus should be on the paper that should be written.

As machine learning, especially on images, is highly computing expensive Cognite AS would give me access to Google Cloud ML so that the project could be training on a GPU cloud.

2.4. Assignment Description

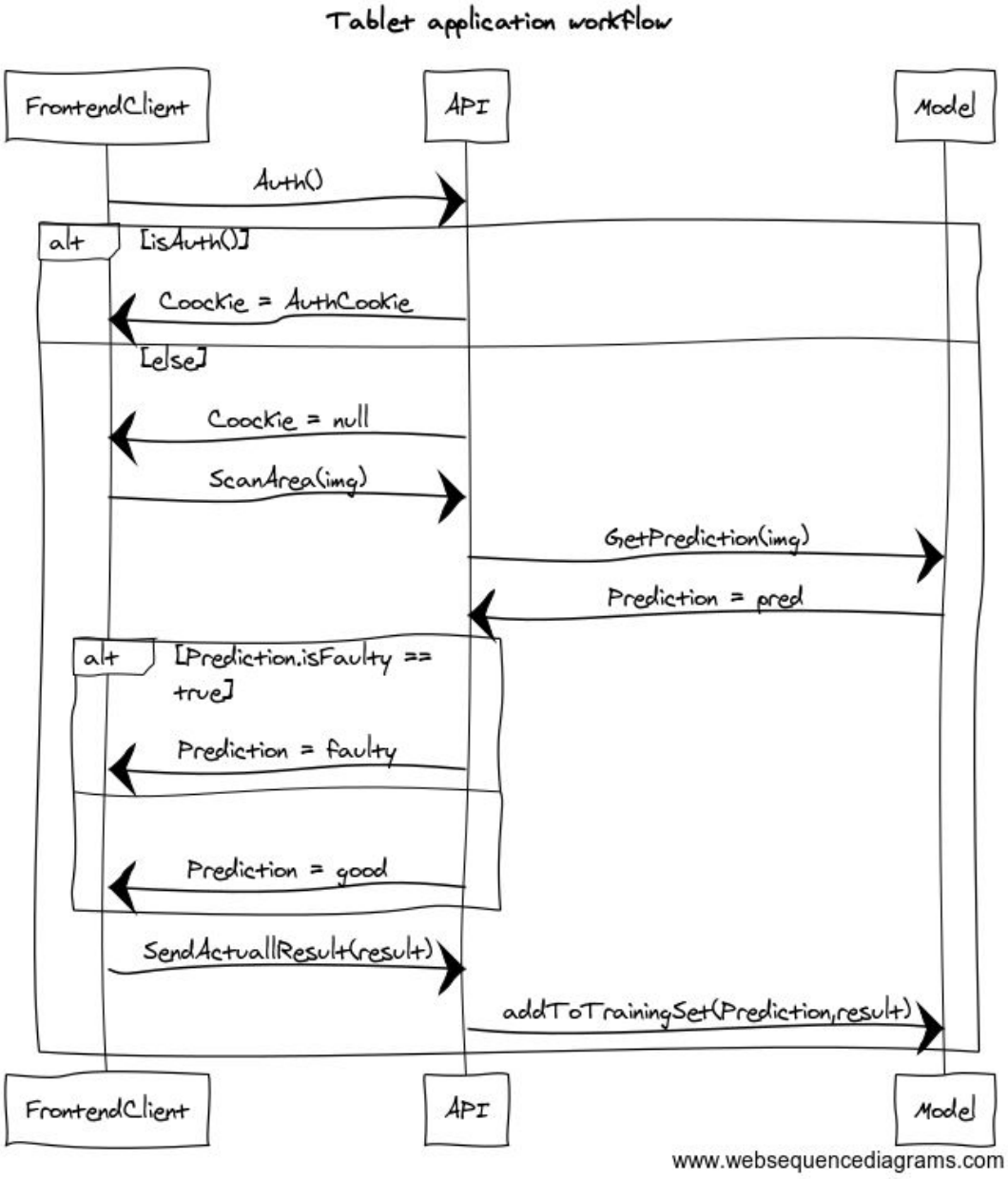


Image 1: Sequence diagram of proposed system.

Image 1 shows a brief outline of the proposed system. A frontend client that talks with an API, that again uses a backend model to do predictions on the input images.

The final product is divided into four separate parts, all working together as shown in image 1.

1. Frontend client
    a. “Scan” / Take an image of what the EX-Inspector is inspecting, and send it to the backend API.

    b. Receive a message in case the backend fault-predictor, predicted the image to contain something faulty.
2. Backend API
    a. Receive image from the client, and return a fault prediction on it.
    b. Should also store the image, together with what the ex-inspector said about the object.(Faulty, if so what type of fault)
3. The prediction model
    a. A model trained with tensorflow that decides if the object is faulty or not.
    b. Here we need a reproducible script to retrain the model, or model-assembly. And we do this by utilising code from the library we also create.
4. A library with useful helper scripts for image recognition built around Keras and TensorFlow
    a. The library containing a collection of small scripts used while creating the prediction model, that can be reused. Some examples on expected scripts are:
        i. Class Activation Map / visualisation of what the convolutional neural network bases it prediction on
        ii. GPU cloud training
        iii. Split training set
        iv. Script to find similar images on Google. (This can be used for pseudo labeling)

There is no need to create a database for storing login information as this would be an addition to an already working API.

3. Project Organizing

3.1. Responsibilities

As I am the only member, it's generally all up to me. However, to avoid being in sole control, I work closely with the company, allowing me to showcase and get feedback on new updates and ideas instantly.

3.2. Rules and Routines

Working 9 - 17 Monday to Friday at the office, skype meeting with supervisor every Wednesday 1300.

The school also have imposed some rules on us, one being that we need to count hours. This would happen using trello + the chrome plugin “plus for trello” [3] that implements time tracking for a trello board using Toggl. [4] This should allow me to, in the end, create a graph with

feedback on time used per task. Tagging the different tasks by type - planning / framework / mobile and backend should allow me to showcase time spent sorted by type.

All code should also be uploaded to git. And the best version of our models should be uploaded as well. Cognite does impose some rules here towards backup and storing duplicates, as models are of high value.

4. Planning

4.1. System Development Model

The Waterfall model would not be a good suit as there is a lot of not known entities to the project, defined in the risk section. That can force us to pivot. This forces us to adopt a more dynamic, and agile development model.

The full implementation of Scrum would be overkill, and we would rather use a Kanban inspired approach. However not limited to only be working on one task at a time, as some tasks have to be done simultaneously. We do however implement the daily standup meeting right before lunch to get input from other people at the office about potential problems. A five minutes standup meeting, to discuss what I am working on, and a quick brainstorm around ideas.

As there are many uncertainties for this project, and with the possibility of the constant influence from the product owner and potentially customers, there would be no looking of the backlog. This allow the project to move fluidly in any direction, at a fast pace.

It's also required that the participates write daily log at the end of the day, explaining in short what they worked on.

4.2. Meetings

Supervisor:

- Status meeting with supervisor after each milestone.
- Skype meeting with supervisor Wednesdays at 1300.

Product owner:

- Daily update on my progress as part of the standup meeting
- A status meeting every Friday

5. Quality Assurance

5.1. Documentation and Code Quality

It does not seem like the firm impose any restrictions on python code, however, there are some people running pylint as a quality assurance tool. To have the same standard as most other companies, we have imposed on ourself to use at least part of the pep8 standard.[1] with pylint to check it. Some parts of it however, for example the line size limit of 79 chars might be too restrictive and would be avoided.

All documentation for the python code should be written as “docstrings” as per the pep8 standard. So that the documentation can be generated with “pydoc”. All code should be committed to the clouds as per Cognites own guidelines on the matter. Generally we would also like code to be reviewed, however as I am only one, this is problematic. Reviewing your own code is not a viable way as you would miss obvious mistakes unless you wait a few weeks before reviewing it again. Our solution here is rather to use “lints” more specific, pylint for python to aid. Later in the project, it might be beneficial to go over the old code.

Tests:
The fault prediction model we create would act as a good test for the library. The model itself would be tested against a unique imageset it has never seen before to measure its quality. Generally we would not implement test for the various subtasks in the library as often the tests would be hard to automatically determine as good or bad, and often they can be really time consuming to run. As the subtasks are only implemented when needed, they would implicitly be testen when attempting to utilize them.

The backend API tests should be created as this is reachable from the outside, and is considered critical.

5.2. Risk Assessment

ID	Risk	Probability	Consequence	Countermeasures?
1.	Not hitting the deadline	middle	high	no
2.	Sickness/illness making me incapable of working	middle	high	no

3.	Not being able to acquire enough data	high	critical	yes
4.	Image recognition technology haven't gotten far enough	Quite high	high	yes
5.	Missing knowledge	Quite high	high	yes
6.	Decreased code quality as I am only one	high	high	yes

Tabel 1: List of potential risks

		Consequence				
		Low	Middle	High	Critical	
PROBABILITY	HIGH			6	3.	
	Quite high			4, 5		
	Middle			2, 1		
	Low					

Tabel 2: Consequences of the different risks

In table 1 we can see a list of the possible risks for this project. In table 2 its plotted into a grid so one can easily see which is critical. Below is the countermeasures for the 4 risks that ranked in the red section in table 2.

5.3 Countermeasures:

3. " Not being able to acquire enough data"
   - We already have decided to use Transfer learning to help mitigate the problem with the low amount of data. However, if this still is not sufficient, one might have to pivot to another related problem. It is not possible to do AI without data. Ideas that have been

circulating as ways to acquire data have been Google images, talking to EX engineers, cable manufacturers, using GAN to generate images, creating faulty cables inhouse, or potentially attempting to gather it using crowdsourcing. For example Amazon's Mechanical turk.

To mitigate it we try to find out if its possible to acquire it as fast as possible, with a hard deadline proposed in the Gantt diagram.

4. “Image recognition technology haven't gotten far enough”
   - Image recognition has gotten far and previous research have already used it to detect corrosion. However as no previous research seems to have done fault detection related to cables, we can not be certain that there will not be some technical challenges that will stop us. We have divided the technology aspect into two parts:

     1. That Convolutional Neural Networks cannot detect a faulty cable from a good cable, something that given enough data should be more than doable.

     2. That the project will fail to detect all the instances of cables in a given image. Given limitations of the Faster rcnn and the YOLO implementation when things get too close to each other there is a real potential problem that it would not see all instances. To address this multiple ideas have surfaced: for example using other spatial information to know where to look, like a tag on the system will give us information on how far away the camera is, and in what direction it's facing From that we can know where the cables should be so we could split the image and look at those locations. Another being that one would rather go for a slower approach, for example linear sweep, however, this comes at a performance penalty that would force us to do the parsing at an external GPU cluster, and not on the tablets. However as we already have decided to do the prediction processing externally, this is not a problem. It's also worth noting that this may end up being a no-problem as the objects where faulty cables can occur might only include one cable. So more information has to be acquired before we can quantify this statement.

5. “Missing knowledge”
   - As AI is a totally new field for me, the problem of missing knowledge is ever present. To overcome it I have decided to work at the office together with some of Norway's best AI researchers, as well as spending the first weeks working on testing out different image recognition techniques and reading research on the subject. The daily standup meeting also helps to mitigate this, as this allows me to get easy input from others with more knowledge on the subject than I.
6 “Decreased code quality “
   - To avoid decreased code quality, I will take advantage of “lint” as previously written. Presumably, at some point someone at the office will also go over my code and challenge the work I have done, to give me insights into what I can improve.

6. Project Plan

Gantt Chart

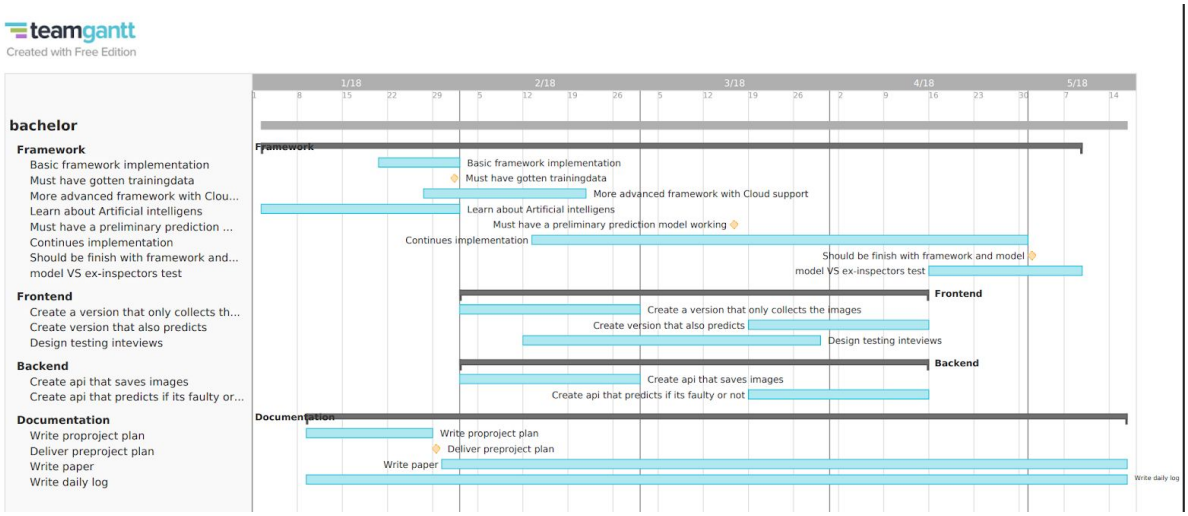


Image 2: GANTT overview of the project. Created with TeamGantt [2]

In image 2 one can see the proposed GANTT diagram for this project. The first month have a big focus on writing the preproject and learning the state of art related to image recognition and image detection. There is sat up a few different milestones for the project, that we would try our best to hit:

6.1 Milestones:

1. 29.1.18 Delivery of the pre-project plan.
2. 31.1.18 Deadline for initial training data
3. 15.3.18 Preliminary prediction model(1 and a half month after getting initial training data)
4. 01.5.18 Almost all coding should be done by now, and focus should be on completing the report and documentation.

6.2 Time and Resource Plan

The project is divided into a planning phase, a development phase, and documentation phase. In the start, the planning and development phase would go hand in hand while it slowly converges into more development. The same pattern would come again during the end of the development phase as May ends up being almost just testing and documentation. However, as the Gantt scheme from image 2 shows, the development and documentation would always go hand in hand. This is as it is important to document,

both in development documentation and also in what is going to be a part of the final paper.
Every Monday, before lunch, the current backlog should be estimated, as an attempt to have some sort of idea about the amount of work currently in the backlog. This should also allow me to, in the paper, show an overview of how off the estimates were. Hopefully showing that we over time learn to do better estimates.

References:

1. Legacy.python.org. (2018). PEP 8 -- Style Guide for Python Code. [online] Available at: http://legacy.python.org/dev/peps/pep-0008/ [Accessed 26 Jan. 2018].
2. Teamgantt.com. (2018). Online Gantt Chart Software | TeamGantt. [online] Available at: https://www.teamgantt.com/ [Accessed 26 Jan. 2018].
3. Chrome.google.com. (2018). Plus for Trello (time track, reports, scrum). [online] Available at: https://chrome.google.com/webstore/detail/plus-for-trello-time-trac/gjjpophepkbhejnglcmkdnncmaanojkf?hl=en [Accessed 26 Jan. 2018].
4. Toggl.com. (2018). Toggl - Free Time Tracking Software. [online] Available at: https://toggl.com/ [Accessed 26 Jan. 2018].

Appendix D

Image Permissions

Figure 2.3 Figure 2.3 is Figure 5.5 from the "Pattern Recognition and Machine Learning" book by Christopher Bishop. The figure is copyright to Christopher M. Bishop, used with permission as stated on the books webpage. [4]

Figures

Below are all of the figures from Pattern Recognition and Machine Learning (except for the photographs in Figures 4.8 and A.4). Copyright in these figures is owned by Christopher M. Bishop. Permission is hereby given to download and reproduce the figures for non-commercial purposes including education and research, provided the source of the figures is acknowledged.

I am very grateful to Markus Svensén who has prepared these figures.

The figures are available in JPG, PNG, PDF and EPS formats. Please note that many of the EPS figures have been created using MetaPost, which give them special properties, as described below.

All figures are available in single zipped folders, one for each format.