**UiT** The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

# Improving automated underwater ship hull inspection through incremental learning & uncertainty quantification in deep learning models

Elias Estefano Gutierrez Riise

Master Thesis in Computer Science - INF-3981

Submitted on June 1st, 2023

*"I think we're on a new sigmoid curve, and I have no idea how far along that curve we are right now."*
–Thomas Scott

*"Sometimes lose, always win."*
–Old Estonian proverb

# Abstract

Rapid technological progress has recently eliminated the necessity of physically deploying humans for manual underwater ship inspections. Instead, Remotely Operated Vehicles (ROVs) can now capture and record videos, which humans can subsequently review. By leveraging annotated data, it becomes possible to train deep learning models to assist in data exploration. Nevertheless, the process of manually inspecting recorded videos remains labor-intensive and time-consuming.

This thesis investigates the potential of training a model on specific types of images to enhance learning efficiency, so the user has to annotate fewer images. Specifically, we investigate the two hypotheses. The first one is that a model's confidence value can be evaluated to give the user indications on which images should be annotated to yield the best performance. The second hypothesis is that a model doesn't have to be trained from scratch every time new training data is added, but only for a fraction of the original epochs.

To enhance accuracy, I plan to utilize a pre-trained deep learning model to identify uncertain images. These images will be reviewed by a human, and the model will be updated using incremental learning. I propose two distinct approaches for determining uncertainty. Separate deep learning models will use each approach to determine the images that will be used for training. I will compare these models to each other and to models trained on images that are classified as not uncertain.

By training on both the new and the old training data, I get results that support my second hypothesis. The findings from the experiments for the second hypothesis were inconclusive.

*This page was intentionally left blank.*

# Acknowledgements

I would like to express my gratitude to my supervisor, Einar J. Holsbø, for providing guidance and engaging in productive discussions during the course of this thesis.

Finally, I would like to thank the collaborators within the LIACi project, funded by the Research Council of Norway under project No 317854, for giving me this opportunity to work on the project.

# Contents

**Abstract** iii

**Acknowledgements** v

**List of Figures** ix

**List of Tables** xv

**1 Introduction** 1
- 1.1 Underwater ship inspections . . . 1
- 1.2 Background and related work . . . 2
  - 1.2.1 MobileNet . . . 2
  - 1.2.2 Multi-label classification . . . 2
  - 1.2.3 The LIACi data set . . . 2
  - 1.2.4 Monte Carlo Dropout . . . 3
- 1.3 Problem statement . . . 4
- 1.4 Hypothesis . . . 5
  - 1.4.1 Phase one . . . 5
  - 1.4.2 Phase Two . . . 6
  - 1.4.3 Phase Three . . . 6

**2 Method** 7
- 2.1 The LIACi data set . . . 7
- 2.2 Measuring performance with F1 score . . . 8
- 2.3 The MobileNet architecture adapted to multi-label classification 11
- 2.4 Picking uncertain images . . . 13
  - 2.4.1 Strategy One: Trusting the neuron outputs . . . 14
  - 2.4.2 Strategy Two: Using Monte Carlo Dropout [2] . . . 15

**3 Experiments** 17
- 3.1 Exploring incremental learning for the LIACi data set . . . 17
  - 3.1.1 Train to test data set ratio . . . 18
  - 3.1.2 Experimenting with incremental training . . . 18
- 3.2 Exploring Monte Carlo dropout to find a viable sample amount 20

3.3 Exploring incremental learning on uncertain or non-uncertain images for the LIACi data set . . . . . 21
3.3.1 Experiment basis . . . . . 21
3.3.2 Experimenting with incremental learning and uncertainty . . . . . 22
3.3.3 Experiment dependencies . . . . . 25

**4 Discussion & Results** **27**
4.1 Experiment results . . . . . 27
4.1.1 Exploring incremental learning for the LIACI data set 27
4.1.2 Exploring Monte Carlo dropout to find a viable sample amount . . . . . 40
4.1.3 Exploring incremental learning that also accounts for uncertainty . . . . . 50
4.2 General discussion . . . . . 77
4.2.1 Pros and cons by showing the user the model annotations before the user self annotates . . . . . 77
4.2.2 Using uncertainty for other purposes than training . . 78
4.2.3 No time measurement . . . . . 78
4.2.4 Future work/further improvements . . . . . 79

**5 Concluding remarks** **83**
5.1 Conclusion . . . . . 84

**A UI changes and additions** **87**
A.1 Storing the image on the server . . . . . 88
A.2 Live incremental learning . . . . . 89
A.2.1 Web application interface . . . . . 89
A.2.2 Changes done to the web application . . . . . 90

**B Incremental training with uncertainty** **91**

**C A closer look at F1 results** **97**

# List of Figures


2.1 The LIACi data set split into a train and test data set and their label distribution. A blue bar represents the total number of that label in the data set. An orange bar represents the number of the corresponding label in the train data set. A Green bar represents the number of the corresponding label in the test data set. . . . . 8

2.2 This figure shows the same as 2.1, but here the bars are normalized, so each set of colors sums up to one (100%). . . . . 9

2.3 An illustration to show how precision and recall are affected by the model's predictions. . . . . 9

2.4 The model's architecture illustrated on a high level. It consists of an input layer, a feature extraction part, and a fully connected part, before the outputs are run through a sigmoid activation function. The step function is later applied manually to classify images with labels. This model is reproduced from Riise [9]. . . . . 11

2.5 The left subplot shows the F1 score for different thresholds. The $x$-axis denotes the threshold value, and the $y$-axis denotes the F1 score. The right subplot has a $y$-axis that denotes the precision and an $x$-axis that denotes the recall. The graph shows the precision for different recall values. . . . . 12

2.6 Shows the precision and recall for each class. The left subplot measures the training data set, and the right subplot measures the test data set. . . . . 13

2.7 2D map of the model's annotations throughout the example video. . . . . 13

2.8 Shows how uncertain images are chosen. The red horizontal line is the classification threshold at 0.35. The two dashed lines represent the other interval of the point of uncertainty. The points are output values from a neuron. Point $A$ and $E$ is beyond the point of uncertainty, while point B, C, and D is inside. . . . . 14

3.1 An illustration of the different expected scenarios from using training variation one (figure 3.1b) and variation two (figure 3.1c) . . . . . 20

4.1 Test training summary for models trained with different fractions of the original epochs. New and old training data was used to train the models. . . . . 28
4.2 Train training summary for models trained with different fractions of the original epochs. New and old training data was used to train the models. . . . . 30
4.3 Train & test training summary for models trained with different fractions of the original epochs. New and old training data was used to train the models. Legend was not included as it was too big for the figure. Solid graphs represent training, and dashed lines represent test. . . . . 32
4.4 Test training summary for models trained with different fractions of the original epochs. Only new training data was used to train the models. . . . . 34
4.5 Train training summary for models trained with different fractions of the original epochs. Only new training data was used to train the models. . . . . 36
4.6 Train & test training summary for models trained with different fractions of the original epochs. Only new training data was used to train the models. Legend was not included as it was too big for the figure. Solid graphs represent training, and dashed lines represent test. . . . . 38
4.7 The four random images used to visualize uncertainty, all from the LIACi test data set. . . . . 41
4.8 Probability density function(s) computed by the mean and standard deviation from the results achieved with Monte Carlo dropout applied to figure 4.7a. The $\mu-2\sigma \approx 95\%$ is visualized as a cross. The two latter subplots show the change in $\sigma$ & $\mu$ as I increase the number of samples of Monte Carlo dropout. The true labels for the image are "sea_chest_grating" and "anode." . . . . 42
4.9 Probability density function(s) computed by the mean and standard deviation from the results achieved with Monte Carlo dropout applied to figure 4.7b. The $\mu-2\sigma \approx 95\%$ is visualized as a cross. The two latter subplots show the change in $\sigma$ & $\mu$ as I increase the number of samples of Monte Carlo dropout. The true labels for the image are "propeller" and "marine_growth." 44

4.10 Probability density function(s) computed by the mean and standard deviation from the results achieved with Monte Carlo dropout applied to figure 4.7c. The $\mu-2\sigma \approx 95\%$ is visualized as a cross. The two latter subplots show the change in $\sigma$ & $\mu$ as I increase the number of samples of Monte Carlo dropout. The true labels for the image are "anode" and "bilge_keel." . 46
4.11 Probability density function(s) computed by the mean and standard deviation from the results achieved with Monte Carlo dropout applied to figure 4.7d. The $\mu-2\sigma \approx 95\%$ is visualized as a cross. The two latter subplots show the change in $\sigma$ & $\mu$ as I increase the number of samples of Monte Carlo dropout. The true labels for the image are "defect" and "propeller." . . 48
4.12 Test loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. . . . . 51
4.13 Train loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. . . . . 53
4.14 Train & test, loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. . . . 55
4.15 Test, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. 57
4.16 Train, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. 59
4.17 Train & test, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. . . . . 61
4.18 Test loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. . . . . 63

4.19 Train loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. . . . . 65
4.20 Train & test loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. . . . 67
4.21 Test, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. The models have a budget of maximum thirty-five images to add to the training data per new image batch. . . . . 69
4.22 Train, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. The models have a budget of maximum thirty-five images to add to the training data per new image batch. . . . . 71
4.23 Train & test, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. The models have a budget of maximum thirty-five images to add to the training data per new image batch. 73
4.24 The F1 score evaluation for the four models. The left subplot shows the F1 score for different thresholds. The $x$-axis denotes the threshold value, and the $y$-axis denotes the F1 score. The right subplot has a $y$-axis that denotes the precision and an $x$-axis that denotes the recall. The graph shows the precision for different recall values. . . . . 75
4.25 Shows the same as 4.24, but for the models trained with a budget. . . . . 76
4.26 Illustrates how each image batch would have their image loss weighed. . . . . 79

A.1 2D map showing labels on the y-axis and frame numbers on the x-axis. A yellow tag means the model is uncertain . . . . 87
A.2 Switches added to the web application UI so the user can annotate images. . . . . 88
A.3 Shows how images submitted to the server are stored together with their annotated labels . . . . . 89

B.1 Training summary for loss and binary accuracy for the models. For each red vertical dashed line, 100 images are evaluated using Monte Carlo dropout to find the uncertain/non-uncertain ones, respectively. The images classified as uncertain/non-uncertain is then added to the training data set. The model then trains on the whole training data set for 30% of the original 35 training epochs. . . . . 92
B.2 Training summary for loss and binary accuracy for the models. For each red vertical dashed line, 100 images are evaluated using "Threshold interval" to find the uncertain/non-uncertain ones, respectively. The images classified as uncertain/non-uncertain is then added to the training data set. The model then trains on the whole training data set for 30% of the original 35 training epochs. . . . . 93
B.3 Training summary for loss and binary accuracy for the models. For each red vertical dashed line, 100 images are evaluated using Monte Carlo dropout to find the uncertain/non-uncertain ones, respectively. The images classified as uncertain/non-uncertain is then added to the training data set. The model then trains on the whole training data set for 30% of the original 35 training epochs. . . . . 94
B.4 Training summary for loss and binary accuracy for the models. For each red vertical dashed line, 100 images are evaluated using "Threshold interval" to find the uncertain/non-uncertain ones, respectively. The images classified as uncertain/non-uncertain is then added to the training data set. The model then trains on the whole training data set for 30% of the original 35 training epochs. . . . . 95

C.1 A closer, more detailed look at figure 4.24, from chapter 4 . 97
C.2 A closer, more detailed look at figure 4.25, from chapter 4 . 98

# List of Tables

2.1 The distribution of classes annotated in images compared to classes not annotated in images in the train and test data set. 8

3.1 The four variations of models trained. . . . . 24

B.1 The four variations of models trained. . . . . 91

# 1 Introduction

In this section, I will discuss what has been previously done in underwater ship inspections (including my previous project), some of the challenges still faced, and how I plan to contribute to machine-automated underwater ship inspections.

## 1.1 Underwater ship inspections

A ship's hull, propeller, bilge keel, and all other underwater components require regular inspections to ensure proper functioning. There are various inspection methods available. However, visual observation and human perception are still predominantly used. These inspections can be carried out underwater or in dry docks on land.

Research in the underwater ship inspection domain is an ongoing topic. Implementing computer vision technology, pre-processing images and videos, and multi-label classification can be highly beneficial to streamline the reporting process and interpret data efficiently obtained during underwater ship inspections. Such tools can assist inspectors in efficiently browsing data and generating comprehensive reports.

## 1.2 Background and related work

### 1.2.1 MobileNet

This subsection is paraphrased from my capstone project Riise [9].

Howard et al. [4] present a deep neural network called MobileNet. The model focuses on being lightweight while maintaining high performance. It does this mainly by using depthwise separable convolutions. When conducting benchmarks, different numbers are compared, including parameters on the ImageNet dataset for MobileNet against VGG-16 Simonyan and Zisserman [10] and GoogLeNet Szegedy et al. [11]. They find that GoogLeNet has 50% more parameters, and VGG-16 has almost 33 times more parameters. They both only perform about 1% better than MobileNet. Howard et al. [4] also compare their model to lightweight models, Squeezenet Iandola et al. [5] and AlexNet Krizhevsky, Sutskever, and Hinton [7], and find that MobileNet performs around 3% better.

### 1.2.2 Multi-label classification

This subsection is paraphrased from my capstone project Riise [9].

Wang, Raju, and Huang [12] use multi-label classification on videos of laparoscopic surgeries to recognize in which surgical phase they were by looking at the tools present. They use VGGNet and GoogLeNet for classification and ensemble learning to counteract overfitting in their relatively small data set. Sigmoid cross-entropy is used to calculate the loss, as it allows for multi-label classification. To split their data set, they use a validation split of 10%.

### 1.2.3 The LIACi data set

Computer vision and supervised training of models rely on labeled data, both the model's classification and feature extraction part. Today, one of the most popular datasets comes from the ImageNet database [1].

ImageNet contains 1000 classes of nouns and over a million images, spanning everything from animals to vehicles to various fruits and much more. Because of this, a network can learn many different features by training on such a broad dataset.

However, few underwater image data sets exist, and even fewer underwater ship data sets. Some of the underwater data sets that exist depict sea creatures

with unrealistic low amounts of noise compared to the data we have to work with. Because there is a lack of other data sets, I am using the LIACi data set from Waszak et al. [13], created and annotated by SINTEF. The data set consists of 1893 images of ships underwater, taken from videos filmed by an ROV. Each image in the dataset is labeled with up to ten labels and corresponding binary segmentation masks. SINTEF's underwater domain data set is disclosed publicly.

### 1.2.4 Monte Carlo Dropout

Monte Carlo methods, such as Monte Carlo integration, use random sampling and estimations instead of more common analytical methods to solve various problems. In this project, I will use Monte Carlo dropout to estimate uncertainty.

Gal and Ghahramani [2] state that using machine learning models for regression and classification has become quite popular. However, the model in itself has no way of measuring its uncertainty. Bayesian models provide ways to estimate prediction uncertainties in machine learning models. Bayesian methods provide uncertainty measures of predictions utilizing a prior probability distribution on prediction values. However, specifying priors is tricky, but we can approximate Bayesian uncertainty estimates by running models with dropout.

The uncertainties can be calculated with the existing model and weights without further training or manipulating weights as long as they consist of a dropout layer. This is beneficial as many popular networks already use dropout layers, and networks don't have to be redesigned at the expense of their complexity or accuracy. If the network has a dropout layer, Monte Carlo dropout can be performed at any time, stochastically through forward passes of the same data. This also allows for concurrent sampling (horizontal scaling).

Many models use datasets such as MNIST to evaluate their performance. Gal and Ghahramani [2] show improvements in RMSE and predictive log-likelihood and how to apply Monte Carlo dropout to deep reinforcement learning.

Later, Gal and Ghahramani [3], the appendix to Gal and Ghahramani [2], discusses the insights and applications for Monte Carlo dropout. They suggest it can show that a model isn't calibrated correctly if it contains a dropout layer. Larger datasets tend to have increased uncertainty. To fight this, the uncertainty can be scaled linearly to compensate for the large dataset.

Gal and Ghahramani [3] evaluate uncertainty by looking at the top and bottom

percentile to determine if it is an uncertainty of significance. To illustrate, if a model has a standard deviation ranging from 0.1 to 1 and a new data sample has a standard deviation from the mean of 3, then it's considered to be uncertain. However, if the model had a standard deviation ranging from 5 to 10, a data sample with a standard deviation of three would be regarded as non-uncertain, as it is in the lower percentile of uncertainties. There will (almost) always be *some* uncertainty. It's more of a matter of how significant the uncertainty is and how it compares to the other uncertainties.

They also propose Monte Carlo dropout as an application where the model predicts on an image with "high" uncertainty. It may be necessary for a human to look at the image and manually annotate it.

## 1.3 Problem statement

As I state in Riise [9], ROVs already record footage, and humans annotate them. In my last project, I created and trained a lightweight deep neural network that could multi-label classify images in a video and present it to the user. Although the model's performance was acceptable, the data set it was trained on can't be classified as "large." Therefore it would be interesting to see if making the dataset bigger, broader, and more balanced (as some classes are underrepresented compared to others) will positively affect the model. But retraining a model from scratch is time-consuming and tedious, especially on weaker hardware, and may also be costly if external cloud services are used instead of internal hardware. Getting a hold of annotated data for supervised learning is one of the more difficult tasks, as it is manual labor no machine can do. For every image annotated, a human has checked the labels as present. But among many possible images to annotate, are there images from which the model will benefit more than others? Can we use the already-trained model to help us classify these images?

In this project, I will explore these questions by retraining an already trained model on new data (from the model's point of view) and exploring different types of training data. To do this, I will first find images the model is uncertain of using various techniques. I will then use active learning to let humans annotate these images, and lastly, use incremental learning to retrain the model further. Since time is of the essence, I will not use actual new data, as that would require a human to annotate new images. I will instead split up the training data set I have to simulate newly annotated data.

## 1.4 Hypothesis

In this project, I investigate two hypotheses: The first is that the model's confidence value can be evaluated somehow, not by a human but by a machine, to give the user indications on which images should be annotated to yield the best learning per image ratio. The second hypothesis is that there is no need to train a model from scratch every time, as it can be trained for just a fraction of the original training epochs. To sum up, the nature of this project has three phases.

### 1.4.1 Phase one

Firstly, use the pre-trained model to classify which images it thinks would be valuable to be trained on. Since the model can't know which labels are true but only annotate images with labels it *thinks* to be true, I have to use the model in another way to help the user pick out new images to annotate. The hypothesis is based on the fact that the model doesn't simply spit out true or false for each label but gives a confidence value between zero and one (because of the sigmoid function). A step function then evaluates the output values to decide if an image is annotated with the respective labels. But since the step function is non-continuous and provides no helpful derivative, multiple sigmoid output values map to the same step function value. However, the model may not have the same confidence for one annotated label as for another.

Uncertainty in deep neural network models is a well-known concept. Uncertainty is measured per image and may also be higher for specific classes in general. And since uncertainty says something about the robustness of a model, it's natural that we want our model to be as robust as possible. If our model is uncertain of an image, it probably will be so for a similar image in the future. So if we could train the model on that image, our model will be more fit to handle similar images in the future, making it more robust as it fills in its knowledge gaps.

This is compared to doing the opposite, annotating images that the model is already certain of, which should make the model more sure of what it already knows, removing the focus of trying to improve robustness. Even though models have a specific set of classes they are trained to recognize, they don't want to learn how to identify every class in a particular way. They want to learn to identify each class in many ways. Not relying on a single feature type for a class makes a model robust to change in images. Training a model on non-uncertain images should only fit the model tighter to the features it already knows well. Training a model on uncertain images should make the model fit more loosely to the features it knows but fit better to a broader array of features, making it

more robust.

Images annotated as false negatives should stem from the model not picking up the class' feature. In contrast, false positives should stem from the model picking up features related to classes other than where they belong. False negatives should therefore be fixed by broadening the model's fit, and this will hopefully be achieved by only training on images the model finds uncertain. There should also be less overfitting in a model trained on uncertain images, as the model tries to learn from images with new features. A model trained on non-uncertain images indicates training on images in which the model already recognized features. This should lead to overfitting on these images/features and worsening in real-life cases where features vary.

### 1.4.2 Phase Two

The second phase involves the user annotating images manually. Optimally, the model guides the user to annotate images more beneficial to the model than others.

### 1.4.3 Phase Three

Lastly, the user wants to use the newly annotated images to improve the model. Of course, the model has to be trained, but how? The second hypothesis is that the model must not be trained from scratch. Acceptably equal test loss and accuracy results should be achieved as if the model was trained on the same images from scratch. Training the model on *only* the newly annotated images may lead to overfitting too much on the new data. However, simply adding them to the old training data should also preserve the old features. Adding new data to the old data also means the model is trained on old data many more times. This may probably lead to overfitting of the older training data.

# 2 Method

In this section, I will describe the various parts of the implementation and methods done in this master's thesis.

## 2.1 The LIACi data set

In this project, I will use the LIACi data set, but I will not use the binary segmentation masks provided in the data set, as segmentation is not what I am trying to achieve in this work. I will instead use the multi-labeled images in the data set, as in my previous project Riise [9].

The data set is split into a test and train data set. The validation split is done at 10%, leaving 190 images in the test data set and 1703 images in the training data set. As seen in table 2.1, the data set is unbalanced annotation-wise. It is skewed heavily toward "false" annotations, with 11883 (77.6%) being false of the possible 15327 annotations.

The label distribution used to train the model can be seen in figure 2.1, and the labels' ratio compared to each other can be seen in figure 2.2. The label "ship hull" has been removed as it's irrelevant to this model. It isn't interesting to know if the ship hull is in the frame, as it will be present for most frames.

| | Train | Test | Train & Test | Train (%) | Test (%) | Train & Test (%) |
|---|---|---|---|---|---|---|
| False | 11883 | 1335 | 13218 | 77,5% | 78,1% | 77,6% |
| True | 3444 | 375 | 3819 | 22,5% | 21,9% | 22,4% |
| **Sum** | 15327 | 1710 | 17037 | 100% | 100% | 100% |

**Table 2.1:** The distribution of classes annotated in images compared to classes not annotated in images in the train and test data set.

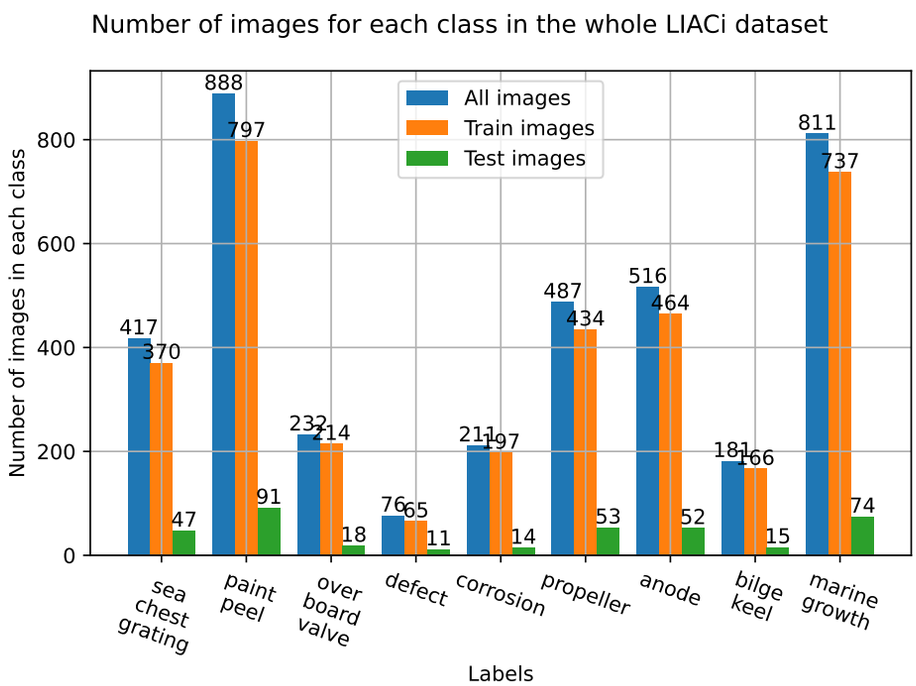



**Figure 2.1:** The LIACi data set split into a train and test data set and their label distribution. A blue bar represents the total number of that label in the data set. An orange bar represents the number of the corresponding label in the train data set. A Green bar represents the number of the corresponding label in the test data set.

## 2.2 Measuring performance with F1 score

The F1 score [8] is a performance measurement that measures a model's accuracy using precision and recall, illustrated in 2.3. Precision is measured by taking the ratio between the correctly predicted positives and the total positives predicted, as in eq. 2.1. Recall is measured by taking the ratio between the correctly predicted positives and the total true positives, eq. 2.2. Higher precision may indicate that the model has been very picky, only making a positive

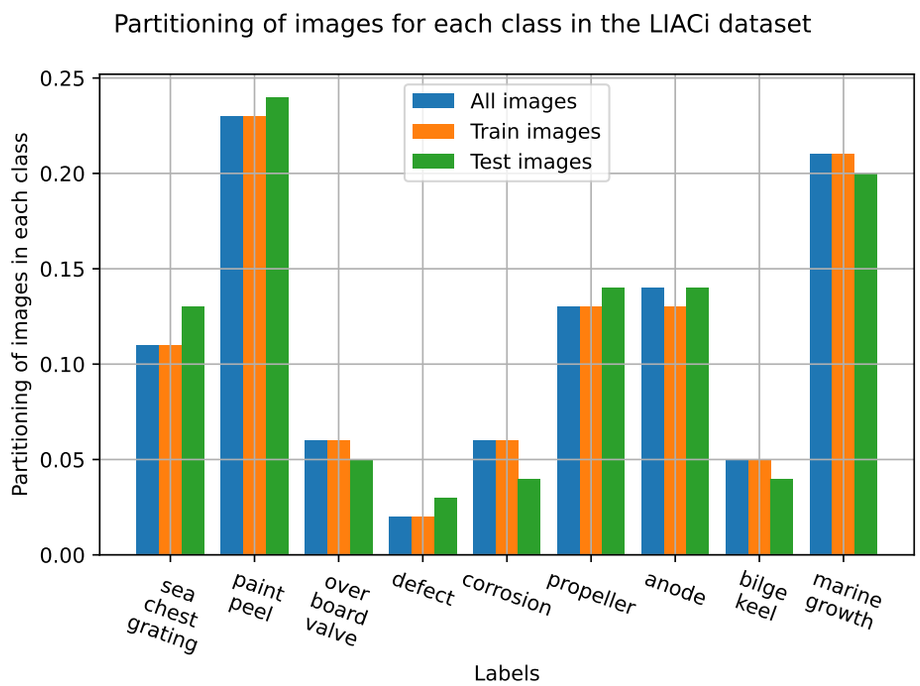
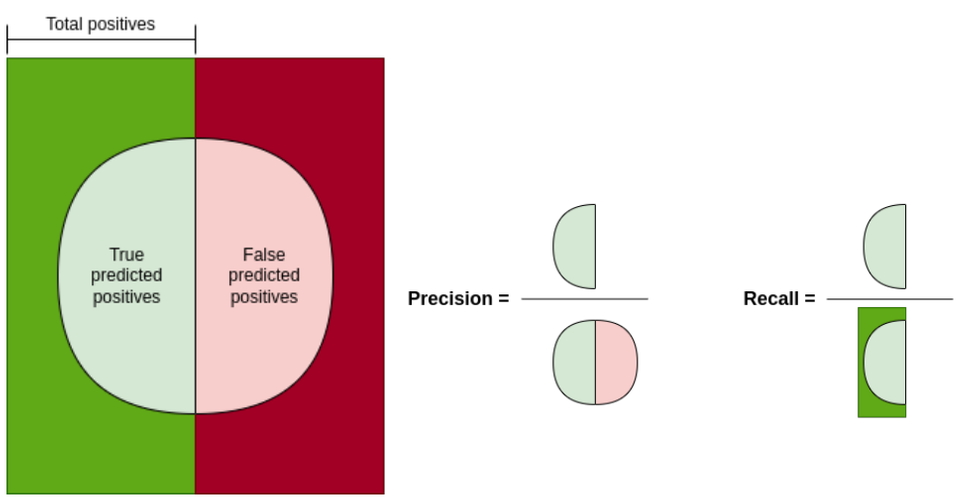



**Figure 2.2:** This figure shows the same as 2.1, but here the bars are normalized, so each set of colors sums up to one (100%).




**(a)** Model predictions illustrated. **(b)** Precision and recall's math illustrated.

**Figure 2.3:** An illustration to show how precision and recall are affected by the model's predictions.

prediction when the output neuron confidence is high, while recall is the opposite. A model's recall score isn't penalized for making false positive predictions,

only when missing out on making positive predictions. Having 100% precision and 100% recall means that you have a perfect model, always predicting positive when positives are presented and never when they're not. However, this is rarely the case, as with most machine learning models and measurements. The F1 score is the harmonic mean between these two measurements, and a higher F1 score is preferable (the score may be a maximum of one).

$$Precision = \frac{Correctly\ predicted\ positives}{Total\ positives\ predicted} \tag{2.1}$$

$$Recall = \frac{Correctly\ predicted\ positives}{Total\ true\ positives} \tag{2.2}$$

$$F1 = \frac{2}{Recall^{-1} + Recall^{-1}} = 2\frac{Precision \times Recall}{Precision + Recall} = \frac{2tp}{2tp + fp + fn} \tag{2.3}$$

The predictions the model makes do not have to be on images. It can be used for measuring a model's accuracy when placing bounding boxes. Higher precision for every pixel inside the bounding box that is the object and higher recall for every pixel of the object that is inside the bounding box. In the context of this project, it is used to evaluate a multi-label classification model. Since the model may predict anywhere from 0 to $n\ labels$ positives per image, the precision, recall, and F1 score may be measured for one or more images the model predicts on. The model uses a step function with a classification threshold $t$ at $0 < t < 1$ to classify a label as positive or negative. Setting $t$ closer to 1 effectively raises precision, and setting $t$ closer to 0 raises recall. To find the optimal F1 score, the classification threshold may be moved between $0 < t < 1$ to compute the different F1 scores. The classification threshold yielding the highest F1 score is then used to classify images further.

Measuring the F1 score may give deeper insight than simply measuring straight-up accuracy because the F1 score doesn't reward predicting positive falses. For example, if a model predicts everything as false on the LIACi data set, its accuracy will be 77,6%. But calculating precision and recall would compute the value zero, implying an F1 score of zero. A model predicting everything as true would yield an accuracy of 22,4%, a precision of 0,224, a recall of 1, and an F1 score of 0,366.

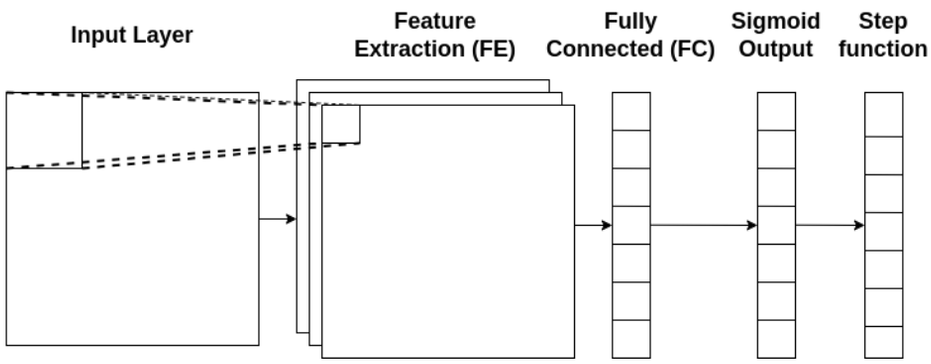



**Figure 2.4:** The model's architecture illustrated on a high level. It consists of an input layer, a feature extraction part, and a fully connected part, before the outputs are run through a sigmoid activation function. The step function is later applied manually to classify images with labels. This model is reproduced from Riise [9].

## 2.3 The MobileNet architecture adapted to multi-label classification

For this project, I needed a lightweight convolutional neural network model. I chose MobileNet as a base for this task, as it is fast and has fewer parameters than larger models, such as VGG16. MobileNet, right off the shelf, is a multi-class classification model, optionally pre-trained on the ImageNet data set. However, for my project, I need a multi-label classification model, as I need to be able to classify more than one label per image.

The solution was to switch out everything after the feature extraction part of the network with my implementation. My implementation, seen in figure 2.4, is designed much like the original MobileNet, with the main differences being in the output part of the network. MobileNet uses a softmax activation layer which is common among multi-class classification models. However, this will not work for a multi-label classification model as softmax creates a probability distribution among the possible outputs, not allowing multiple labels to have predictions close to one (100%).

Instead, I chose the sigmoid activation function as the output, allowing multiple neurons to have close to one as their output. Using sigmoid also allows every class to be (close to) zero, while softmax will always yield the same value if the output neuron values are almost equal. For example if a model predicts no labels in the image, the softmax value can easily be computed with eq. 2.4. My model has nine outputs, so softmax would give each output neuron a value of

1/9.

$$softmax(neurons) = \begin{cases} \frac{1}{n\_outputs}, & \text{if } var(neurons) \approx 0 \\ \frac{e^{z_i}}{\sum_{j=1}^{n\_ouputs} e^{z_j}}, & \text{otherwise} \end{cases} \tag{2.4}$$

As I've mentioned, MobileNet can opt-in for the pre-trained ImageNet weights. ImageNet is a very different data set (not the same classes and only a single class per image), but using its weights from the feature extraction part of the model is still beneficial, as it's highly trained to extract features. I could have also used the ImageNet weights from the fully connected part, but since I use a different data set, I want to learn how to classify these features from scratch. I also freeze the weights in the feature extraction part to avoid the overfitting the much smaller LIACi data set will eventually cause. Binary cross-entropy is used to compute the loss.

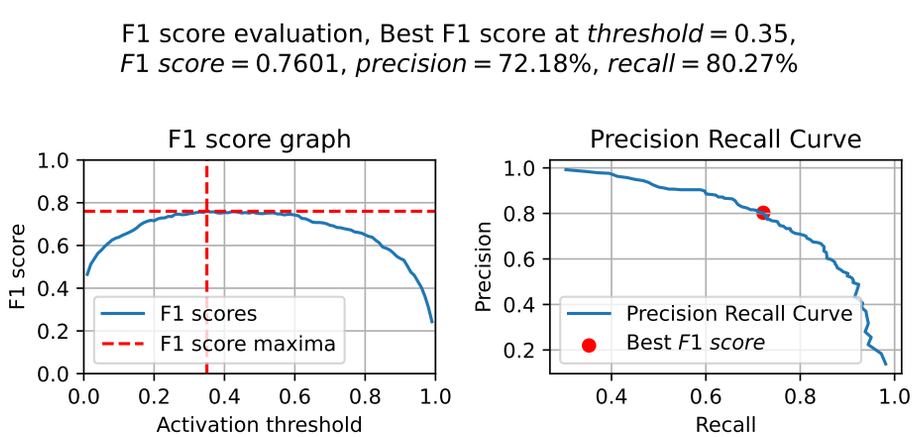



**Figure 2.5:** The left subplot shows the F1 score for different thresholds. The $x$-axis denotes the threshold value, and the $y$-axis denotes the F1 score. The right subplot has a $y$-axis that denotes the precision and an $x$-axis that denotes the recall. The graph shows the precision for different recall values.

After training the model, I compute the F1 score from the test data set to determine an optimal classification threshold. I, Riise [9], find that using $learning\,rate = 4 \times 10^4$ and $epochs = 35$ yield the best results. As shown in figure 2.5, the best F1 score on the test data set was achieved with a $threshold = 0.35$. The recall and precision per class with that threshold can be seen in figure 2.6.

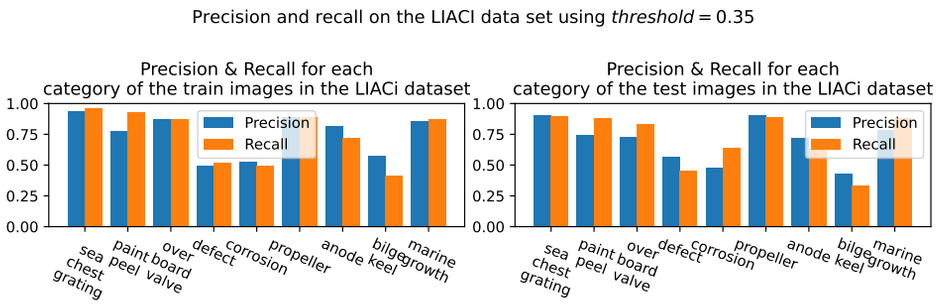



**Figure 2.6:** Shows the precision and recall for each class. The left subplot measures the training data set, and the right subplot measures the test data set.

## 2.4 Picking uncertain images

In my previous work Riise [9], the model proposed can already annotate images with multiple labels. The classification threshold was calculated using the F1 score and ended at 0.35.

The previous work includes a web server where the user can upload their underwater ship video to the web application, which the model then classifies. A 2D map of the model annotations throughout the video is created and displayed to the user, as shown in figure 2.7. However, this map only shows the annotations' boolean values, not the predictions' degree of uncertainty. The model does not give the user information about how uncertain it is. If the output neurons of an image are above or below the classification threshold ultimately determines if the image is annotated with the various labels.

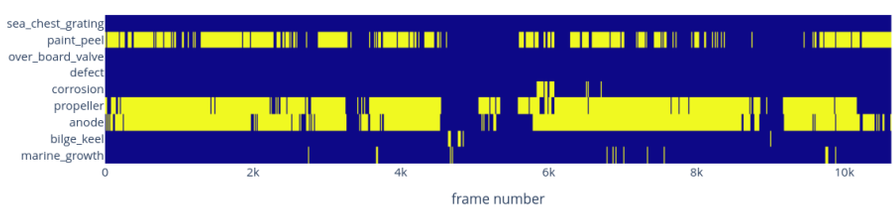



**Figure 2.7:** 2D map of the model's annotations throughout the example video.

The web application's back end uses the trained model to predict all the frames from the video. However, these images aren't kept in memory but discarded for new ones, and in the end, only the predictions are left. After this process is done, the task of finding uncertainties is issued.

In this work, I investigate two strategies to determine which images the model is uncertain of. Both strategies will have varying outputs dependent on where

the classification threshold is set.

### 2.4.1 Strategy One: Trusting the neuron outputs

Images are classified as uncertain if they contain one or more output neurons with their value within an interactively chosen interval, as demonstrated in figure 2.8. The pros of this method are that every image can be evaluated as uncertain (or not) with a simple arithmetic statement performed on every output neuron, shown in eq. 2.5. However, the neuron's output confidence value doesn't necessarily translate to how certain the model is.

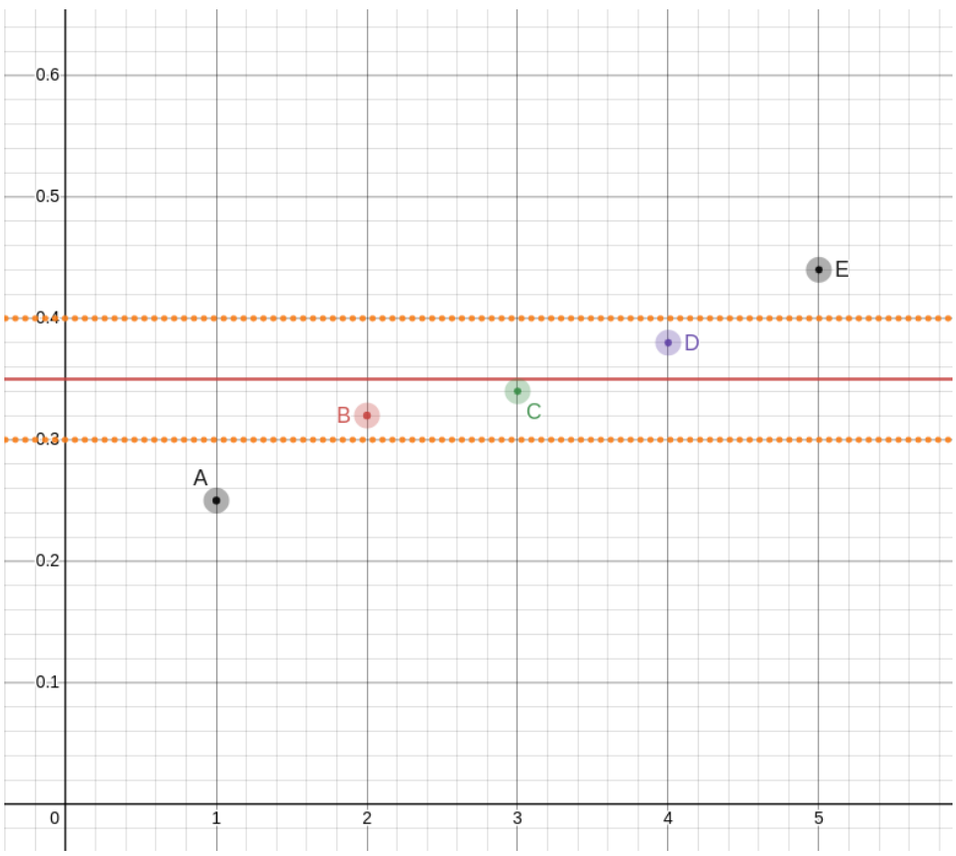



**Figure 2.8:** Shows how uncertain images are chosen. The red horizontal line is the classification threshold at 0.35. The two dashed lines represent the other interval of the point of uncertainty. The points are output values from a neuron. Point *A* and *E* is beyond the point of uncertainty, while point B, C, and D is inside.

$$\frac{interval\ size}{2} > |neuron\ output - threshold| \tag{2.5}$$

### 2.4.2 Strategy Two: Using Monte Carlo Dropout [2]

Monte Carlo dropout is a way of measuring uncertainty that also provides the degree of uncertainty and doesn't solely look at the output value of the last neurons.

OpenCV is again used to fetch frames from the video one by one. But in comparison to earlier, when the model annotates the video, now only one frame is kept in memory at a time. This is because the model has two different methods (as the model is a Python object) for making predictions. The one used in the video annotation part is optimized for GPU and larger batches of images, while the one used in this task is less scalable but easily allows for using "training mode."

Making predictions with the model's "training mode" turned on effectively activates the dropout layers even though the network isn't training (performing backward propagation). As discussed earlier, using dropout layers can be seen as using subnetworks of the entire network to make predictions. Predicting an image just went from deterministic to non-deterministic.

The different subnetwork predictions are sampled $n$ amount of times from an image. The predictions can then compute each class's mean, variance, and standard deviation for the classes annotated as present by the original prediction. The mean is sometimes lower (but rarely also higher) than the original prediction as the network can't use its "full compute power" to compute the output. However, it's important to note that this isn't always true.

Only classes annotated as true are classified for uncertainty measurements as neurons rather annotate a class as false than true. If the model annotates a class as true, it shows that weights in the model have picked up on something in the image, and I want to measure how uncertain it is in that prediction. Therefore, the original image prediction is required to know if an image is viable to be computed uncertainty for. The uncertainty cannot be computed when the image is fetched the first time, as predictions aren't computed before a whole batch of images is retrieved. And also the fact that another prediction method is used.

The mean and standard deviation can now estimate a probability density function (gaussian curve) for labels annotated in the original prediction. A similar approach to strategy one is currently utilized. If the threshold is below $\approx 95\%$ of the Gaussian distribution, the prediction is classified as "uncertain," and so is the whole frame, also shown in eq 2.6. Computing the probability density function isn't necessary, but it's an excellent way to visualize the mean and variance (standard deviation) among the samples.

$$\mu - 2 \times \sigma < threshold \tag{2.6}$$

Uncertainty may refer to either strategy unless otherwise stated.

# 3

# Experiments

For all experiments, 10% (190 of 1893 images) of the LIACi data set was used as test data, while the remaining 90% (1073 of 1893 images) was used as training data.

## 3.1 Exploring incremental learning for the LIACi data set

I performed experiments to check whether incremental learning works for the LIACi data set and to test my hypothesis that a model doesn't have to be trained from scratch. The model is trained initially on a data set. Still, as users annotate images from the video uploaded to the website, we want to use these images to enhance our model's accuracy further.

The naive solution to this problem is to add the newly annotated images to our data set and train a new model. However, training a "fresh" network presents multiple pain points. Both the code and the (LIACi) data set are needed to train a model from scratch. In the case of this project, where the user might have to host the web server locally, it means the user is running on inferior hardware, leaving for an even longer training time than if the service was hosted on an external server. The user's computer will most likely also be out of commission for the duration of the training as training a network is

resource-heavy, hardware-wise. Today many opt for cloud services for heavier computational duties, renting external hardware instead of buying their own internally. However, the price often scales with the number of arithmetics to be computed.

The solution I propose to solve these problems partially is to apply incremental training. I write partially because even though the model doesn't have to be trained from scratch, with my proposed solution, it still has to be trained for a couple of epochs. But it will certainly relieve some of the training if my hypothesis is true.

To simulate the scenario where a user annotates images, I use the LIACi data set and partition it into different image batches. The first image batch is the largest (800 images) and is initially used to train the model. The rest of the batches are smaller and of equal size (100 images) and fed to the program/model individually to simulate the user annotating new batches of images.

### 3.1.1 Train to test data set ratio

Instead of creating and training a new model from scratch, I add the newly annotated images to the old training data set to form an even larger one. However, I still keep the testing data set intact to have more comparable results. Changing the test data set as new images are added to the training data set may be a good idea when the model is set in a more production-like environment, where the model follows an exact method. However, in this project, the goal is to find out if there is a definitive better way to keep improving a model. If the already trained model can learn new knowledge without training it from scratch, that is also a plus.

### 3.1.2 Experimenting with incremental training

This experiment is done to see if incremental training is feasible on the LIACi data set, and if it is, is there an optimal way to do it? The test was done with two different methods. The first method only trains on the newly added data, while the second method trains on all the training data every time.

Since the experiment is supposed to simulate a trained model, a reasonably large portion (800 images) of the training data set is used to train the initial model. 800 images were used as it is enough to roughly train the model to an acceptable accuracy, while still having 900 images left to experiment with. Adding too many images to the first image chunk leaves little room for improvement and fewer images to improve from. Adding too few images to the first

image chunk will leave the model with little data to train on, leaving us again with a model that doesn't know anything before further training starts. As I only have a given set of images to work with, finding a good balance of initial and additional data added to the model's knowledge is essential.

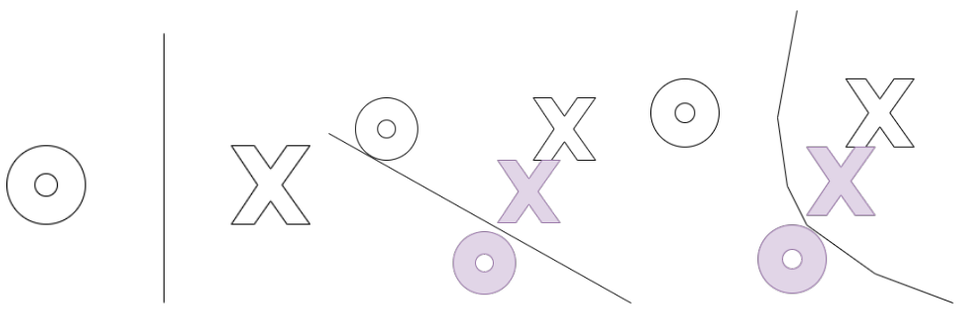
The model is trained for the same number of epochs as the original model. The amount is based on the number of epochs for the model to converge in Riise [9]. This is done to simulate having an already trained model fitted to the training data set before adding new knowledge to the model. The remaining original training data set is split into (close to) equal chunks so the model can incrementally train on them. The two methods decide *how* the model trains on these chunks. Since the model is already trained, it is only trained for a fraction of the original epochs. Training the model on all the data for the same amount of epochs gives no incentive to train on it incrementally, and we might as well have just trained a new model.

It's crucial to find the perfect balance when deciding the fraction of the initial epochs to train the model incrementally on. If too many epochs are used to train on the new data, more time will have elapsed, and overfitting may occur. If too few epochs are used, less time and processing power are necessary, but there is a risk of the model being unable to fit the new data properly, also known as underfitting.

### Method one

The model trains on significantly fewer images in method one than in method two. As time complexity is an aspect, it is valuable to find out if it is feasible to only train on the newly added images. Since it has fewer images to train on, it also suggests that it can be trained for a larger fraction of the original epochs while still keeping the time complexity low compared to regular training.

If method one is feasible, it would also mean that a model can be firstly trained by more powerful hardware and then shipped out to the user. The user would then only need the model itself and not all the training data it previously has been trained on. The LIACi dataset isn't the largest as of now, but it is still significantly larger than just the model. And if the data set grows, it is even more incentivized and convenient only to ship the model to the users. However, fitting a model to data is complicated, and tuning hyperparameters to get it right is just as tricky. Figure 3.1 shows a scenario where an already-fitted model (figure 3.1a) is overfitted on the new data (figure 3.1b) because it completely ignores the old training data. The best scenario (figure 3.1c) is if the new and the old training data model fittings are preserved throughout the training. This is why I also include method two.



**(a)** A separating plane fitted by a support vector machine.

**(b)** Worst case: an already trained support vector machine only trained on new data (colored data) to the point of overfitting.

**(c)** Best case: the same support vector machine trained properly on training data and achieves a good fit.

**Figure 3.1:** An illustration of the different expected scenarios from using training variation one (figure 3.1b) and variation two (figure 3.1c)

### Method two

Method two will train on a lot more images than method one. Every time new images are annotated, the data set grows, and the model has more images it must train on. The advantage of this method is that every image is trained on, and the situation illustrated in figure 3.1b is less likely to happen because not only the new training data is trained on, but also the old. The model will, therefore, not potentially be skewed all the way towards the new data points but instead try to fit every data point. The ideal case is if the model can adapt to both the new and the old data, as illustrated in figure 3.1c. However, the fit may be more difficult or even impossible. Sometimes data points aren't separable, at least not in the given dimension. Since all the old training data are trained on when adding new data, there may be more overfitting in the model towards the training data. And every time a new batch of data is added, the model trains on the old data again, causing more overfitting.

## 3.2 Exploring Monte Carlo dropout to find a viable sample amount

I did a small experiment on four random images from the LIACi data set to see how many samples were required to achieve a stable $\mu$ and $\sigma$. I sampled the four images 350 times each, with a dropout rate of 10%. The samples were done 350 times, as it's more than enough to ensure convergence.

## 3.3 Exploring incremental learning on uncertain or non-uncertain images for the LIACi data set

I did experiments to check if the model could help the user annotate more efficiently and to see whether my hypothesis that using the model's confidence value to help the user annotate images is valid. By efficiently, I mean having the model classify images the user may annotate to dividend the most in terms of performance per image annotated.

In a real-life scenario, the model is initially trained on a data set before it is set to use for classification. The model already has the purpose of helping the user browse the video by annotating the whole video and visualizing it to the user, as shown earlier in figure 2.7. But now, the functionality for the user to annotate frames from the video has also been added to open up the availability for some form of active learning. Having the user annotate new images is necessary because even though the model can classify and annotate videos, there is no reason to believe the model is 100% correct in its predictions. Like the human brain, the model can almost always be trained further on new data to achieve new knowledge. It's hard to say exactly when a model is sufficiently trained as it will likely never be perfect, but aiming for satisfactory performance on the test data set is a start. Since the model used in this project is a multi-label classification model, another measurement can be done by measuring its precision and recall to calculate its F1 score. Even though the F1 score cannot be directly used to train the model (as it has no direct gradient), it still provides valuable insight into its performance.

Annotating images requires humans and takes time, and time is always money. For every image in the data set, a human has inspected it and checked the correct boxes for the labels present in the image. In the LIACi data set, even segmentations are included making it an even more time-consuming task. Finding images worth labeling using these methods is only possible if a trained model or a data set exists to initially train the model. Since the LIACi data set exists, I can train a model that can be used to find new images to annotate. The challenge is finding images the model will benefit from the most.

### 3.3.1 Experiment basis

This experiment uses knowledge acquired from the earlier experiment, where I test different ways of utilizing incremental learning. In short, I tested incremental learning with training on the new training data and all the training data. In the later chapters, I will show the results from the testing and more thoroughly

discuss and elaborate on the results. For now, the reader needs to know that I will use the method where the model trains on all the data for 30% of the original epochs. This also goes for the experiment where I try to find a viable number of samples for Monte Carlo dropout. In later chapters, I explain the results, but for now, the reader needs to know it is a hundred samples.

Similar to the earlier experiment, a model is trained from scratch, with all the new data only being added to the training set. Again, in an environment where this is put into production, it is expected that more images will be added all the time. Adding some images to the test data set out of the annotated ones is probably a better idea to broaden the test data set. Still, in this experiment, the goal is to find out if there is a good way to use model inference to find images more beneficial for further model improvements.

### 3.3.2 Experimenting with incremental learning and uncertainty

This experiment is done to see if uncertainty and incremental learning can be tied to more effective learning on the LIACi data set. The experiment was done with two different strategies for measuring uncertainty. The strategies were also tested by only training on uncertain or non-uncertain images. Lastly, the same experiments were done with an image budget.

This experiment should also simulate that I start with an already trained model. Since I only have the LIACi data set, it is again split into different chunks, a larger one (800 images) and the rest equally small (100 images). The model is initially trained on the large chunk for the same number of epochs as it took the model to converge on the whole LIACi data set in [9]. This is to have a model somewhat fitted to the training data before classifying images as uncertain/non-uncertain and adding new data to the model's dataset. The remaining smaller chunks of data are now fed to the model to run inferences on them. They are fed to the model individually so that it may evaluate them. The two strategies for measuring uncertainty now dictate which images are classified as uncertain and non-uncertain.

#### Strategy one: Trusting the neuron outputs

I have already described the two ways of measuring uncertainty in earlier sections. However, I will reiterate them and describe how I've implemented them in this experiment. The first one measure uncertainty by defining an interval with the classification threshold in the middle. A single inference is run on the image to compute if an image is uncertain. If the absolute difference

between the model's output value and the classification threshold is lower than the interval value, as shown in eq. 2.5, the image is classified as uncertain. Note that this method does not say anything about the degree of uncertainty. It simply gives a binary yes or no if a class is uncertain in an image. However, it is possible to quantify uncertainties in an image as multiple classes may be within the threshold. Therefore uncertainties in an image may be $0 \leq uncertainties \leq n\ classes$.

### Strategy Two: Using Monte Carlo dropout

The second strategy computes uncertainty by using Monte Carlo dropout, which has also been described earlier. Monte Carlo dropout is a way of measuring uncertainty that also provides the degree of uncertainty. It activates the model's dropout layer while running inference, effectively sampling different sub-networks of the model. Different sub-networks yield different output values, giving each class its own Bayesian distribution. The variance in the distribution represents the uncertainty. Compared to the threshold interval method, this method says something about the degree of uncertainty.

As described earlier, a distribution is classified as uncertain if eq. 2.6 is true. The reason for not looking at only the variance is because the thought is that even if a class has a high variance, the statement eq. 2.6 is false, which means that in most cases ($\approx 95\%$), the model would still predict that class to be true. And if the class was to be annotated as present by the model in almost all cases, its uncertainty wasn't significant enough to be classified as uncertain. We don't want the user to annotate the image if the model is certain about the information in the image. If the variance is relatively small but eq. 2.6 is true, the class is too close to the threshold and, therefore, still classified as uncertain. This method may also provide several uncertainties in an image, $0 \leq uncertainties \leq n\ classes$.

This method is relatively slower than the first one, requiring $n = 100$ samples to provide a stable Gaussian curve, while strategy one only requires a single inference. The output for each sample isn't deterministic, but only a single part of the model causes this non-deterministic behavior, the dropout layer. As talked about before, this application aims to run on weaker hardware, but passing images through convolutional layers without a powerful GPU is often slow, especially when having to do multiple samples on many images. As I established, only a single part of the model yields different outcomes, and it is located at almost the very end of the model, at least computationally-wise. Therefore the image can be passed through the whole convolutional part of the model to extract all the features from it. These features will always be the same, so we can pass them through the remaining network for the $n$ samples, saving

| | **Threshold interval** | **Monte Carlo dropout** |
|---|---|---|
| **Uncertain** | Model 1 | Model 2 |
| **Non-uncertain** | Model 3 | Model 4 |

**Table 3.1:** The four variations of models trained.

precious CPU power and time. Adding this optimization made computing the mean and variance of each class in an image almost 30 times faster.

### Using uncertainties for further manual annotations

Whether we used strategy one or two, we should have an array where each index holds the number of uncertainties for its corresponding image. The implementation does not consider the degree of uncertainty because only one of the strategies has that measurement. Still, both of them have a number of uncertainties, which we will use later. For now, I will only use the fact that there is any value (besides zero) in the array, providing me with a binary mask. Inverting the mask now gives us true for all the images we want to remove (non-uncertain images) from the batch of images. After removing the non-uncertain images, the remaining images are added to the training data set. The model can now fit for a fraction of the original epochs on the new and old training data (with the test data set still the same). This process is repeated for every remaining batch of data from the LIACi data set. In a real-life scenario, the uncertain images would have to be annotated by a human. For testing, using the already annotated data in the LIACi data set is much easier and lets me simulate the active learning part.

### Comparing the models trained on uncertain images

We now have two models trained on uncertain images, but this experiment is trying to prove that training on uncertain images should improve the performance of a model. The natural thing to compare it to is training it on images classified as non-uncertain. To do this, we do not flip the bit mask, leaving us with a mask removing all the uncertain images. This is done for strategies one and two. We now have four models, two trained with uncertain images, one using Monte Carlo dropout, and one using the threshold interval, and two trained with non-uncertain images, again one using Monte Carlo dropout and one using the threshold interval. The combinations are in table 3.1.

### Balancing the experiment

The experiment above's flaw is that it may become rather unfair for either of the models if fewer images are classified as uncertain or vise-versa. If there are always more images classified as uncertain, the models training on uncertain images will have more images in their data set. This may yield falsely high performance from the models, not necessarily because training on the specific types of images works, but because the models have more data to fit on. To counteract this, a budget can be defined to set the maximum number of images a model can add to the data set to train on. The budget is set to an amount that is almost always entirely spent. Again, there is no degree of uncertainty, but the number of uncertainties is kept track of. An array of indices is sorted based on the number of uncertainties, and slicing is used to only keep $b = 35$ images, where $b$ is the budget size. The mask used to remove images will always remove every image not within the budget. The four models are otherwise trained as before. This was also done as a human won't annotate every uncertain/non-uncertain image, but only a subset.

### Minimizing luck from the equation of the experiment

As explored in [9], the LIACi is an unbalanced data set regarding class distribution. It might be the case that many of the images in the original test data set are already classified or resemble what the model views as non-uncertain. Training on only non-uncertain images will perhaps only tighten the already known fit, which also may happen to fit the test data. Training on uncertain images may therefore worsen the test data performance. To exclude this option from the equation of the experiment, the experiment is also performed multiple times. Every time the experiment is run, the whole LIACi data set is randomly shuffled around before splitting it into a train and test data set. In order to obtain average training curves, the experiments were conducted ten times. A regular model was also run ten times to measure its change on the ten different test and train data set distributions, the identical data set distributions as the other models.

## 3.3.3 Experiment dependencies

This experiment relies on the results from the experiment described in the earlier section 3.1, where I experiment with incremental learning to see if and what works best with the LIACi data set. I wanted to see if training on only new images added to the training data set was better than training on all the images in the data set. If incremental learning on the LIACi data set did not work, this experiment could still be done, but with some restructuring. Instead

of finding the wanted images and applying incremental learning, the model could be repeatedly retrained from scratch.

# 4

# Discussion & Results

## 4.1 Experiment results

In this section, I will discuss the results achieved from the experiments in the previous section.

### 4.1.1 Exploring incremental learning for the LIACI data set

These results come from the experiment described in section 3.1. In figures 4.2, 4.1, 4.3, 4.5, 4.4 and 4.6, the models are initially trained on 800 images and then 100 images are added for every new batch. A fraction of epochs is run for every new batch to fit the data. So, for example, f0.3 is $\frac{3}{10}$ of the original batches. Figures 4.2 and 4.1 contain the same graphs as figure 4.3, but split into each their graph to make it more clear. The same goes for figures 4.5, 4.4, 4.6. Since the total training epochs differ, the graphs are all normalized along the $x$-axis. The training curve of a regular trained model is also included for comparison. It is important to note that when graphs have been normalized, models with fewer epochs may seem to converge later than models with more training epochs. In reality, they might have faster convergence but just stopped training sooner as they converged.

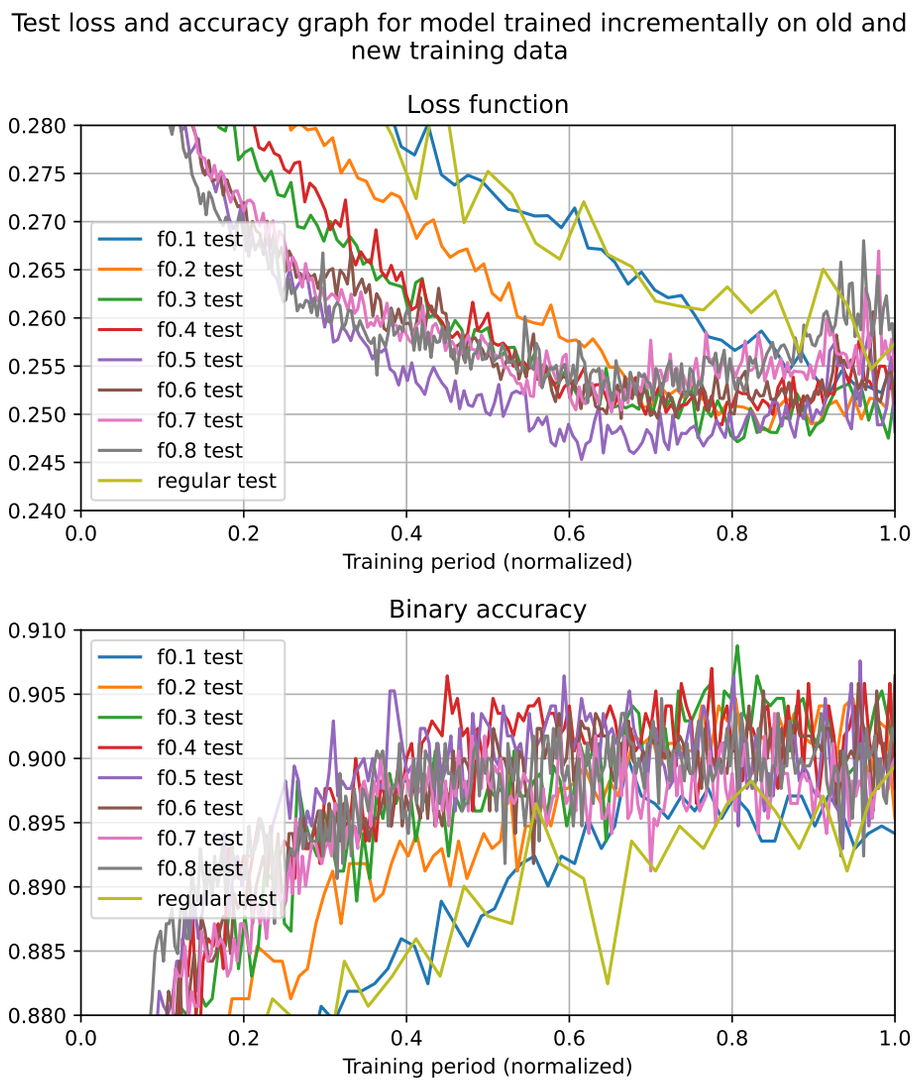



**Figure 4.1:** Test training summary for models trained with different fractions of the original epochs. New and old training data was used to train the models.

### Test graph figure for model trained on all training data (figure 4.1)

The test summary loss function graph for incremental training on all the training data shows that lower fractions of epochs yield slower convergence. We also see that higher fractions yield faster convergence but tend to cause overfitting. The worst case may be $f0.8$, where it's down to a loss of almost 0.25, around 60% through the training before rising to a peak of almost 0.27 but falling slightly in the end. $f0.4$ to $f0.7$ also start overfitting around 0.7-0.8. One might argue that $f0.5$ performs the best. It reaches the lowest point of

0.245 but starts rising again. Ultimately, it ends up as one of the better curves but has a lot of spiking. However, $f0.3$ converges relatively fast but doesn't show the same signs of overfitting as the other curves that also converge. It reaches a low of around 0.247 and ends at around that value. It is worth mentioning that $f0.2$ performs well but converges later. As this experiment was meant to simulate a real-life scenario where image batches are added incrementally, the training could have stopped at any time (if no more image batches were annotated by the user). If training were to stop earlier, the model should preferably perform as well as possible. $f0.3$ achieves this better than $f0.2$. The regular trained model spikes much towards the end and may have been unlucky with its training data distribution.

Looking at the binary accuracy graph, we can easily see that $f0.1$ peaks lowest and has overfitting tendencies towards the end. It is slower to fit with a fraction of 0.2, but in the end (after about 70% of the total training period) the loss is about the same as training with larger fractions. Again $f0.3$ performs well. It fits early and also has the highest peak at almost 91%.

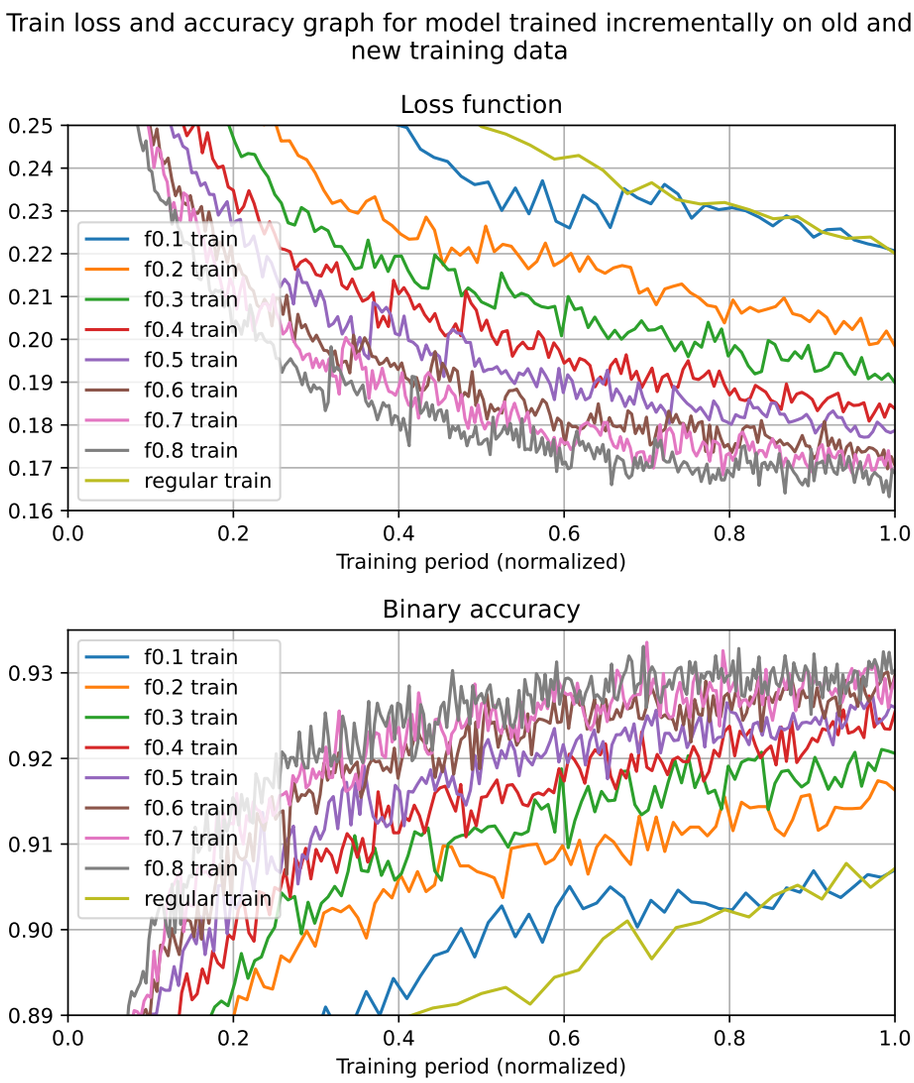



**Figure 4.2:** Train training summary for models trained with different fractions of the original epochs. New and old training data was used to train the models.

### Train graph figure for model trained on all training data (figure 4.2)

The train summary loss function graph clearly shows that the larger the fraction, the more overfitting occurs. $f0.1$ performs almost as well as the regular model.

The binary accuracy graphs also show that $f0.1$ performs the best in terms

of overfitting, with the lowest accuracy. But one could argue $f0.2$ is closer to $f0.3$ in this subplot. Again, the size of the fraction corresponds well with the degree of overfitting but $f0.4$ to $f0.8$ all end up with almost the same binary accuracy.

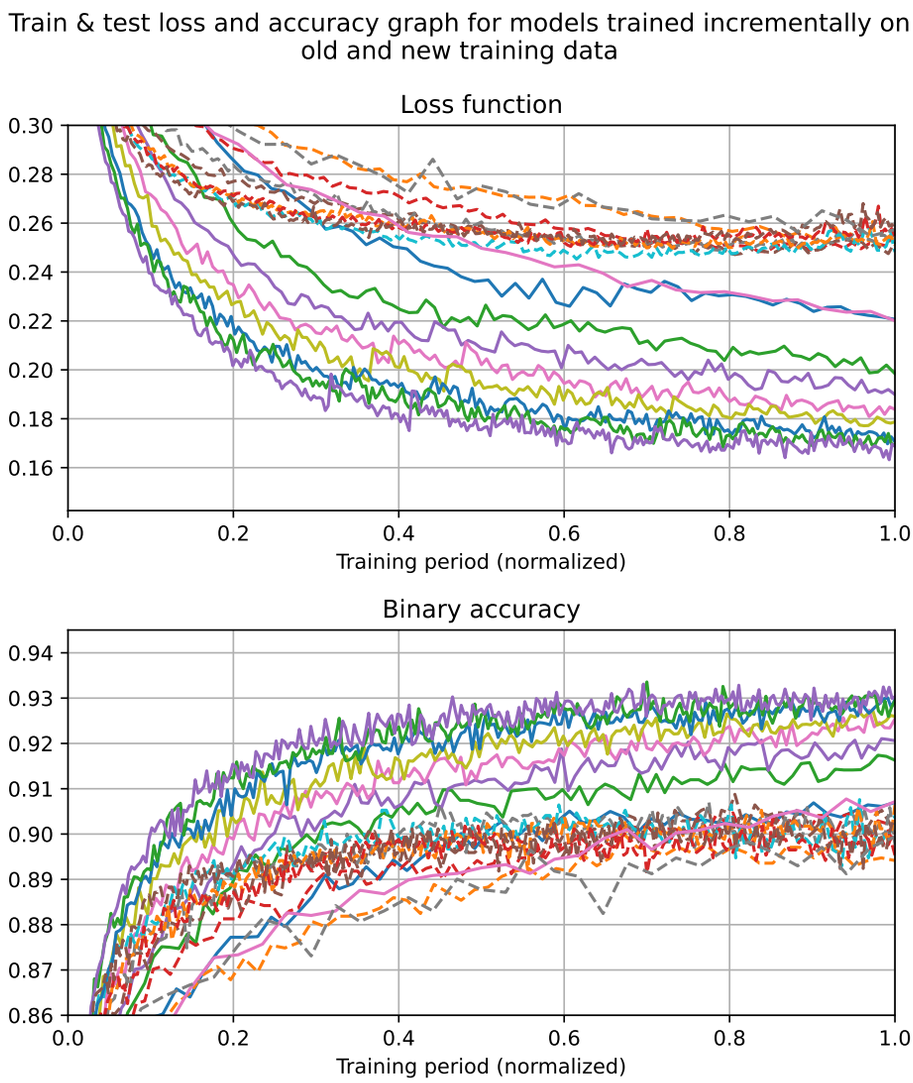



**Figure 4.3:** Train & test training summary for models trained with different fractions of the original epochs. New and old training data was used to train the models. Legend was not included as it was too big for the figure. Solid graphs represent training, and dashed lines represent test.

### Reviewing the training summary graphs for the model trained on all training data as a whole (figure 4.3)

Looking back at the whole training summary, we see that the more significant fraction is used, the more overfitting will inevitably occur. Training on a fraction too small leads to underfitting of the model. After evaluating the pros versus

the cons, I conclude that using $f0.3$ is the better option. It provides some overfitting on the train curves but not as much on the test curves, which is the most important. The overfitting is not as bad as some of the other more significant fractions. The advantage of $f0.3$ is that it converges well early if the model doesn't receive additional image batches, which the lesser fractions do not.

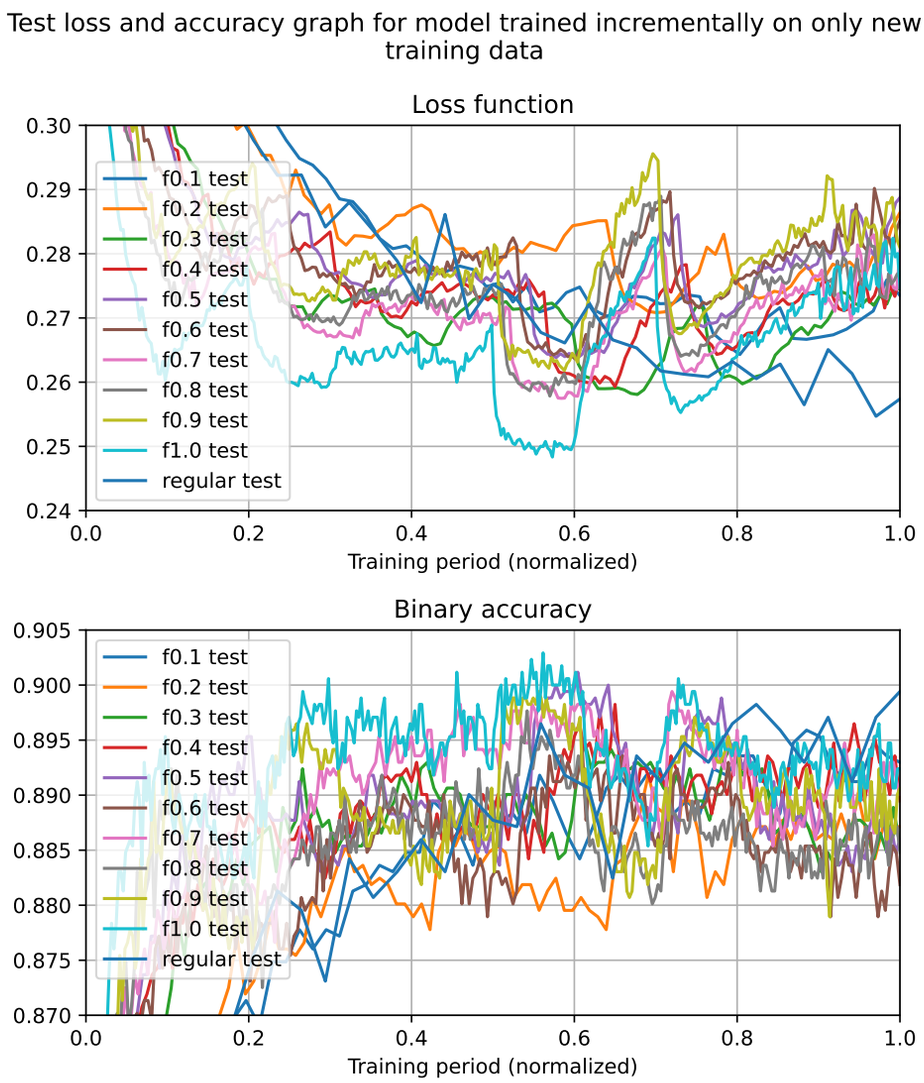



**Figure 4.4:** Test training summary for models trained with different fractions of the original epochs. Only new training data was used to train the models.

### Test graph figure for the model trained on only new training data (figure 4.4)

The loss function test graphs for the model trained on only the new data are much more volatile than the ones trained on all the training data. $f1.0$ is perhaps the clearest example of how much the loss changes for each new batch of images. It drops quickly and spikes shortly after, multiple times, but it is far from the only one. Almost none of the graphs stay anywhere near consistent and

predictable. $f0.1$ is closest to staying consistent, but that is probably because it affects the model's weights the least. At around 60% through the training, a batch of images is fitted on that directly contradicts the already trained weights in the model, losing accuracy for every epoch until the fitting is done.

The binary accuracy graphs are very much also unstable. Many of the graphs reach a high value quickly but continue to fall gradually, and eventually, they all end up with a worse accuracy than the regular model.

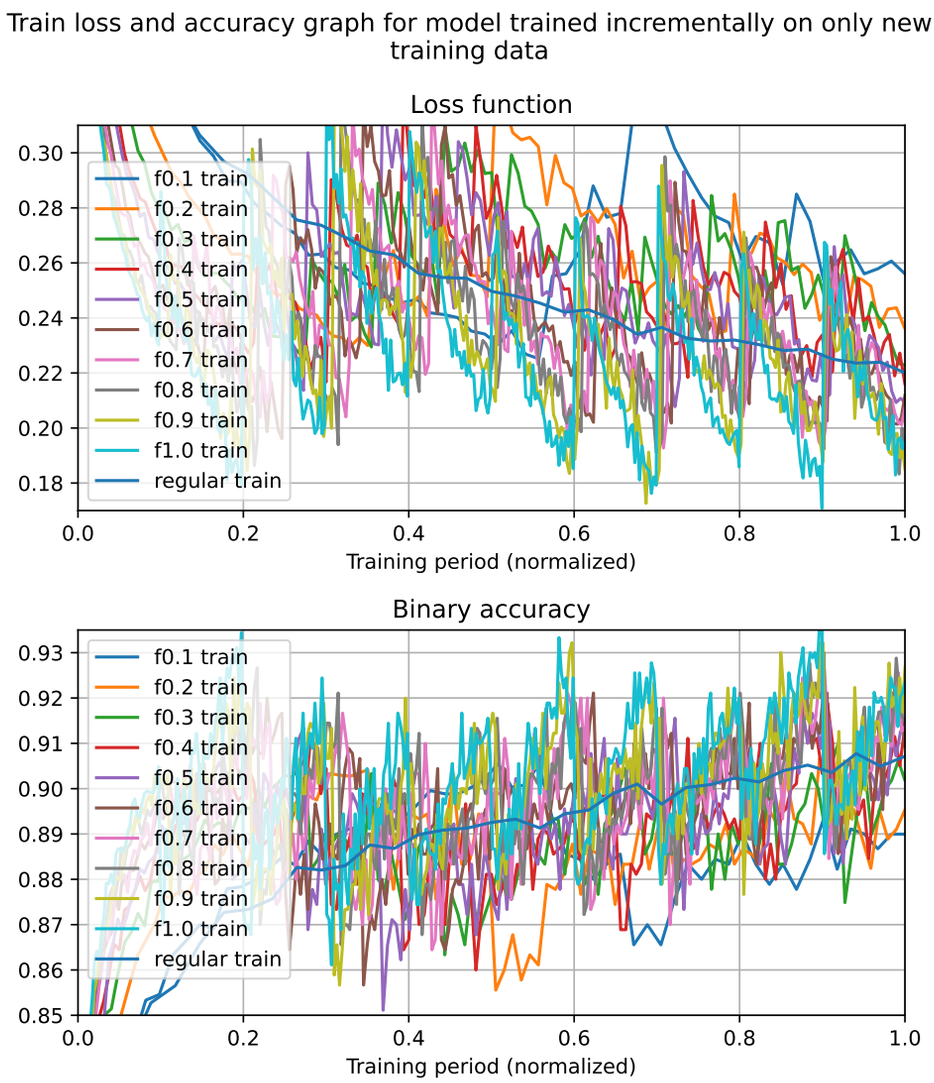



**Figure 4.5:** Train training summary for models trained with different fractions of the original epochs. Only new training data was used to train the models.

## Train graph figure for the model trained on only new training data (figure 4.5)

The loss function training graphs are also highly very volatile. Each new image batch can be seen by the loss value spiking through the roof and quickly falling again, in most cases, quickly overfitting. The loss value always spikes at new epochs because it will most likely be completely new images the model is introduced to, ignoring its previous "fit goal" in chase of the new one.

The binary accuracy graphs tell a similar story. The graphs all start well from their initial training, but as soon as they're introduced to new data, their accuracy spikes as they overfit the new data. Almost every single one of the graphs has a higher binary accuracy than the regular model.

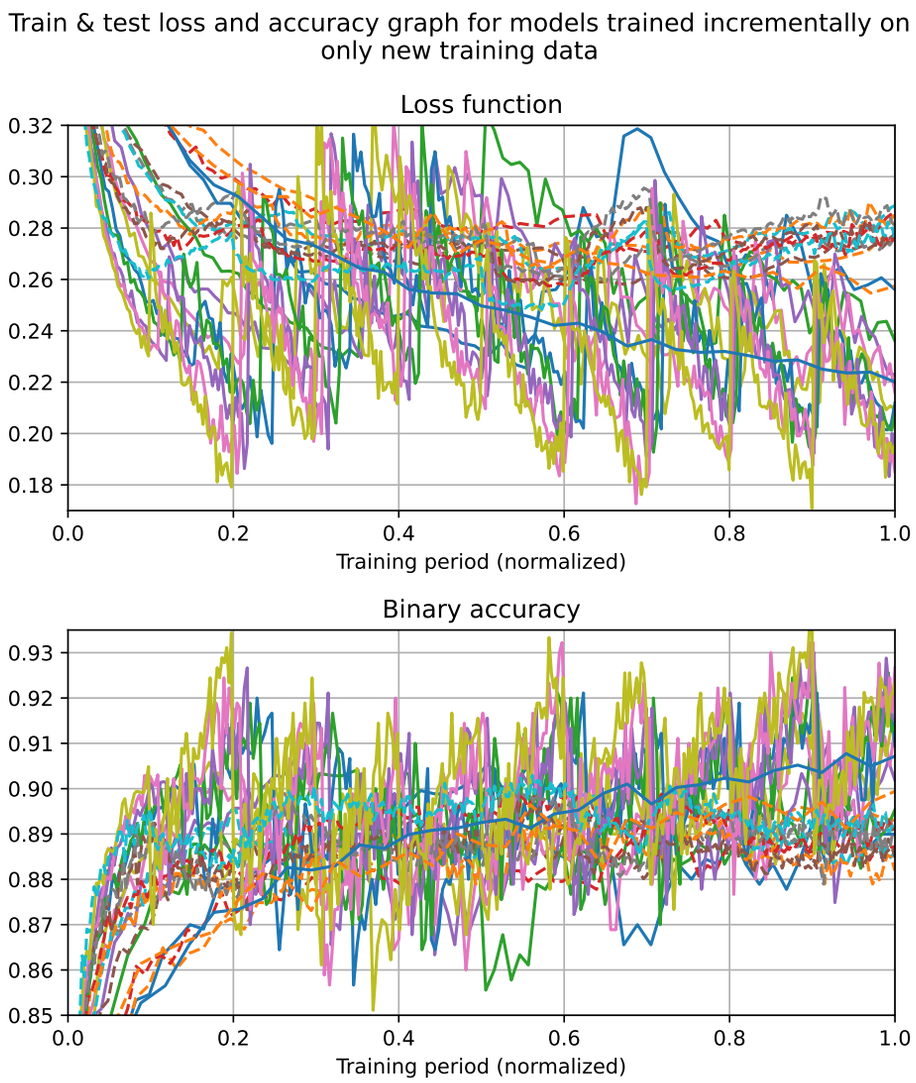



**Figure 4.6:** Train & test training summary for models trained with different fractions of the original epochs. Only new training data was used to train the models. Legend was not included as it was too big for the figure. Solid graphs represent training, and dashed lines represent test.

### Reviewing the training summary graphs for the model trained on only new training data as a whole (figure 4.6)

It is hard to draw any conclusions from evaluating the models' summaries. However, since none of the graphs give a satisfactory result, I would not recommend any of them. They all seem too unstable to use for other purposes

confidently.

### Afterthoughts

After reviewing both variations of incremental learning, some conclusions can be drawn. Using only the new data to train the model causes less overfitting and yields a wildly unstable train and test graph. Using the variation that trains on the old and the new training data causes more overfitting but achieves more stable and better test results. From the different fractions, $f0.3$ performed the best. It converged early but did not overfit as much as the larger fractions. Therefore, training on all the training data with $f0.3$ seems best for further experiments. The downside of using this variation of incremental training is that since all of the training data is trained on, it also requires all the training data to be present when the user wants to train their model incrementally. If only the new data were needed, the stand-alone model could be shipped to the user, but now they also need the (potentially) large data set if they want to train the model further.

My second hypothesis was that a model does not have to be trained from scratch to achieve acceptable test loss and accuracy results. Even though the model $f0.3$ overfits more than the regular model, it achieves slightly better test loss and binary accuracy. I, therefore, conclude that the results support my hypothesis.

### 4.1.2 Exploring Monte Carlo dropout to find a viable sample amount

These results come from the experiment described in section 3.2.

#### Visualizing Monte Carlo dropout uncertainty

As explained earlier, Monte Carlo dropout was used as an uncertainty measurement. While it's difficult to visualize Monte Carlo dropout for multiple images, I can still visualize it for a single image, as shown in figure 4.8. However, output neurons with mean values close to zero or one often have a low standard deviation, as no value can go beneath zero or above one (because the output neuron has a sigmoid activation function), skewing the standard deviation to lower values. The plot becomes more focused on these outlying low standard deviation distributions, while I want to focus more on the ones in the middle, as they have the potential to be classified as uncertain.

Removing the neurons with output closer to 0 and only including those with their original predictions above the classification threshold leaves the plot with more relevant curves, giving more insight into how the uncertainty classification works, as shown in figure 4.9. Even though, theoretically, output neurons with mean values closer to 1 also may have a low standard deviation. Their standard deviation is rarely low enough to take the focus away from the middle ones. They may instead provide some contrast to the other curves.

**(a)** The true labels for the image are "sea_chest_grating" and "anode." PDF and statistics can be found in figure 4.8

**(c)** The true labels for the image are "anode" and "bilge_keel." PDF and statistics can be found in figure 4.10


**(b)** The true labels for the image are "propeller" and "marine_growth." PDF and statistics can be found in figure 4.9


**(d)** The true labels for the image are "defect" and "propeller." PDF and statistics can be found in figure 4.11

**Figure 4.7:** The four random images used to visualize uncertainty, all from the LIACi test data set.

Figure 4.7 shows four different subfigures 4.7a, 4.7b, 4.7c and 4.7d. The four images are random images from the LIACi data set. Figure 4.7a shows a sea chest grating, the large grill in the center of the image, and an anode, the white vertical line in the upper left of the image. Figure 4.7b shows a propeller and marine growth on the propeller. Figure 4.7c shows the bilge keel with an anode on it, the white vertical line. Figure 4.7d shows a propeller with a defect. The defect is the notch at the bottom middle of the propeller. The images have been resized to the target size (224x224) but have not been pre-processed further.

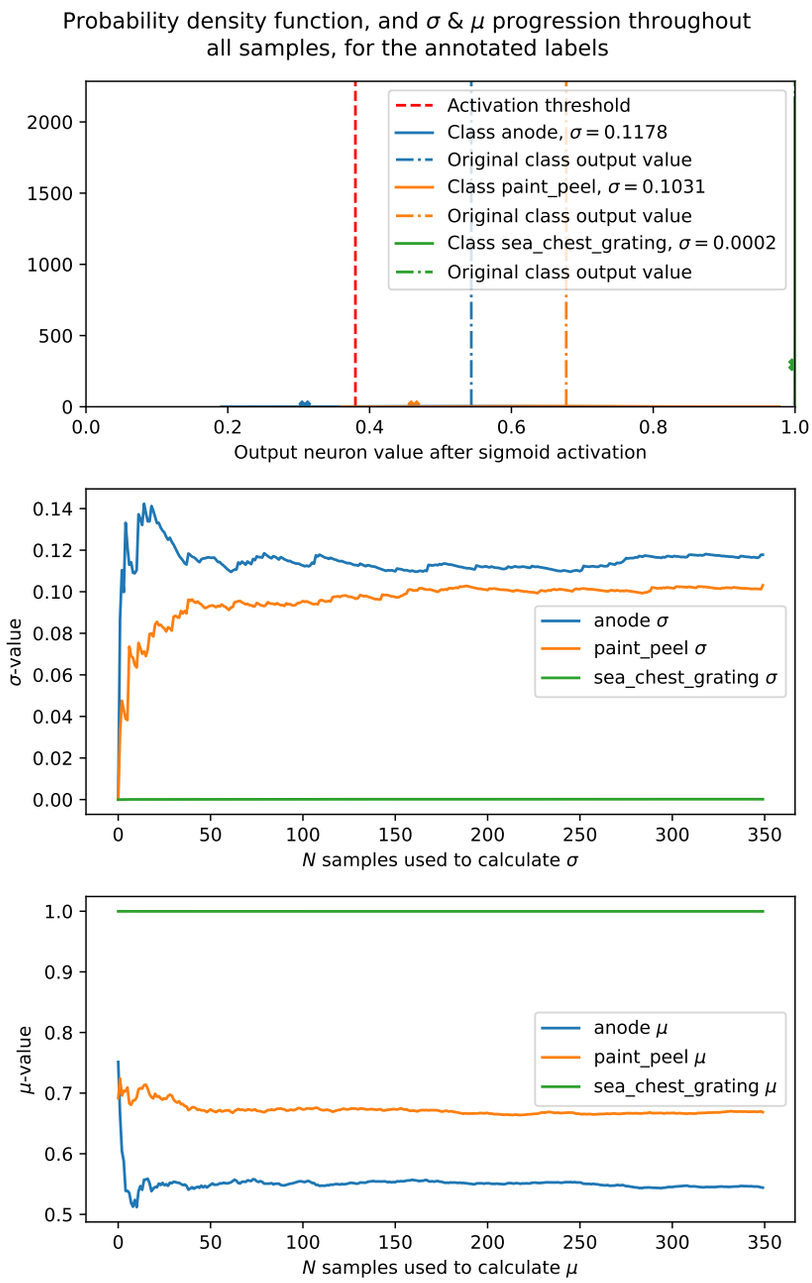



**Figure 4.8:** Probability density function(s) computed by the mean and standard deviation from the results achieved with Monte Carlo dropout applied to figure 4.7a. The $\mu - 2\sigma \approx 95\%$ is visualized as a cross. The two latter subplots show the change in $\sigma$ & $\mu$ as I increase the number of samples of Monte Carlo dropout. The true labels for the image are "sea_chest_grating" and "anode."

Figure 4.8 shows three subplots. The first subplot shows the uncertain classes viable to be classified as uncertain. Their original confidence value (denoted by a dotted dashed vertical line), probability density function (solid curve), and 95% mark (an x on the left side of each curve) are plotted for each class. The red dashed vertical line represents the classification threshold. The $x$-axis denotes the output neuron value. Each probability density function has an area underneath the curve equal to one. The variance (width) of the curve(s) represents the uncertainty. A broader curve means more uncertainty in the prediction for that class. The probability density function(s) consist(s) of hundred samples. The classes "anode" and "paint_peel" is the most uncertain ones in this subplot because they have the largest *sigma*-value.

Subplot two shows each class's progressive $\sigma$ (standard deviation) change as I increase the number of Monte Carlo dropout samples. The $y$-axis denotes the $\sigma$-value, and the $x$-axis denotes the $N$ samples used to compute $\sigma$. In total, 350 samples were collected. "sea_chest_grating" has $\sigma \approx 0$ from the first sample, but the two other classes stabilize at around sixty samples.

Subplot three shows each class's progressive $\mu$ (mean) change as I increase the number of Monte Carlo dropout samples. The $y$-axis denotes the $\mu$-value, and the $x$-axis denotes the $N$ samples used to compute $\mu$. In total, 350 samples were collected. "sea_chest_grating" has $\mu \approx 1$ from the first sample, but the two other classes stabilize around forty samples.

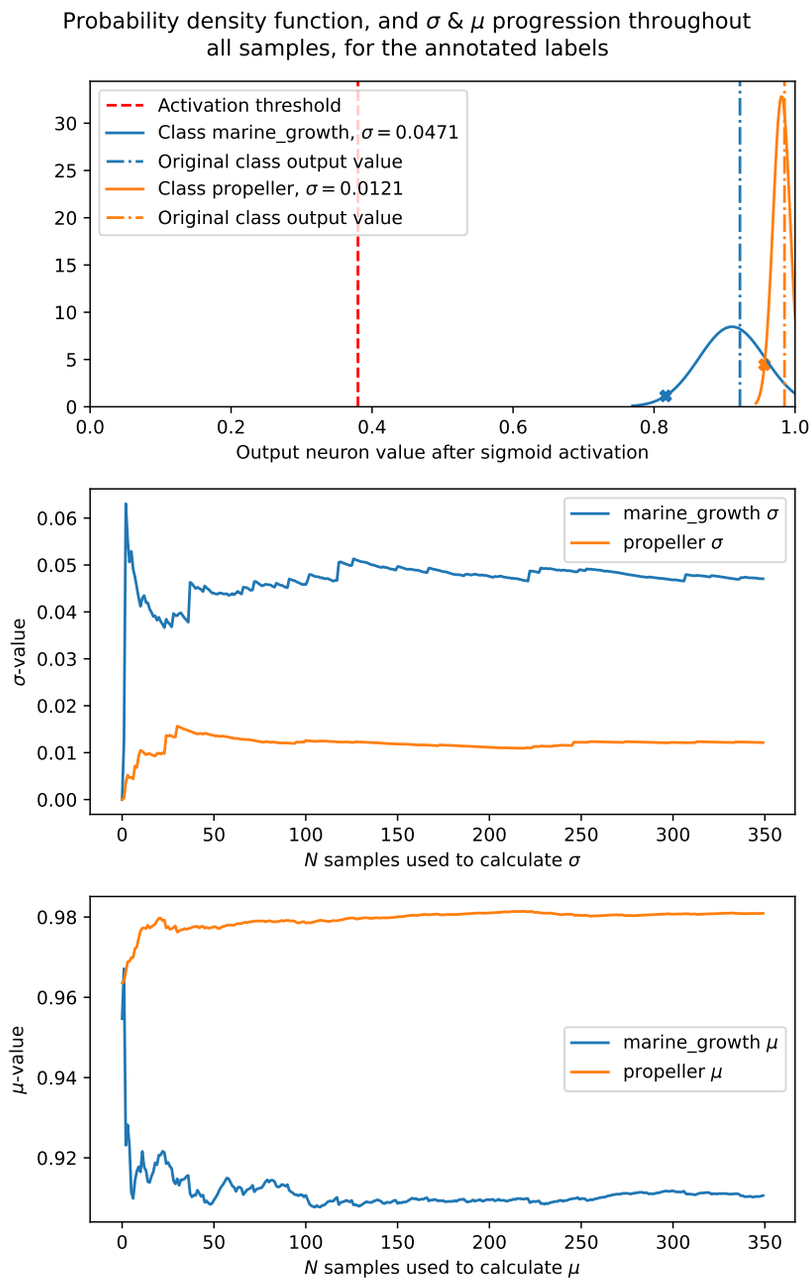



**Figure 4.9:** Probability density function(s) computed by the mean and standard deviation from the results achieved with Monte Carlo dropout applied to figure 4.7b. The $\mu - 2\sigma \approx 95\%$ is visualized as a cross. The two latter subplots show the change in $\sigma$ & $\mu$ as I increase the number of samples of Monte Carlo dropout. The true labels for the image are "propeller" and "marine_growth."

Figure 4.9 shows three subplots. The first subplot shows the uncertain classes viable to be classified as uncertain. Their original confidence value (denoted by a dotted dashed vertical line), probability density function (solid curve), and 95% mark (an x on the left side of each curve) are plotted for each class. The red dashed vertical line represents the classification threshold. The $x$-axis denotes the output neuron value. Each probability density function has an area underneath the curve equal to one. The variance (width) of the curve(s) represents the uncertainty. A broader curve means more uncertainty in the prediction for that class. The probability density function(s) consist(s) of hundred samples. The class "marine_growth" is the most uncertain one in this subplot because it has the broadest curve.

Subplot two shows each class's progressive $\sigma$ (standard deviation) change as I increase the number of Monte Carlo dropout samples. The $y$-axis denotes the $\sigma$-value, and the $x$-axis denotes the $N$ samples used to compute $\sigma$. In total, 350 samples were collected. Both classes seem to stabilize around fifty samples.

Subplot three shows each class's progressive $\mu$ (mean) change as I increase the number of Monte Carlo dropout samples. The $y$-axis denotes the $\mu$-value, and the $x$-axis denotes the $N$ samples used to compute $\mu$. In total, 350 samples were collected. Class "propeller" stabilizes around twenty-five samples, and class "marine_growth" stabilizes around a hundred.

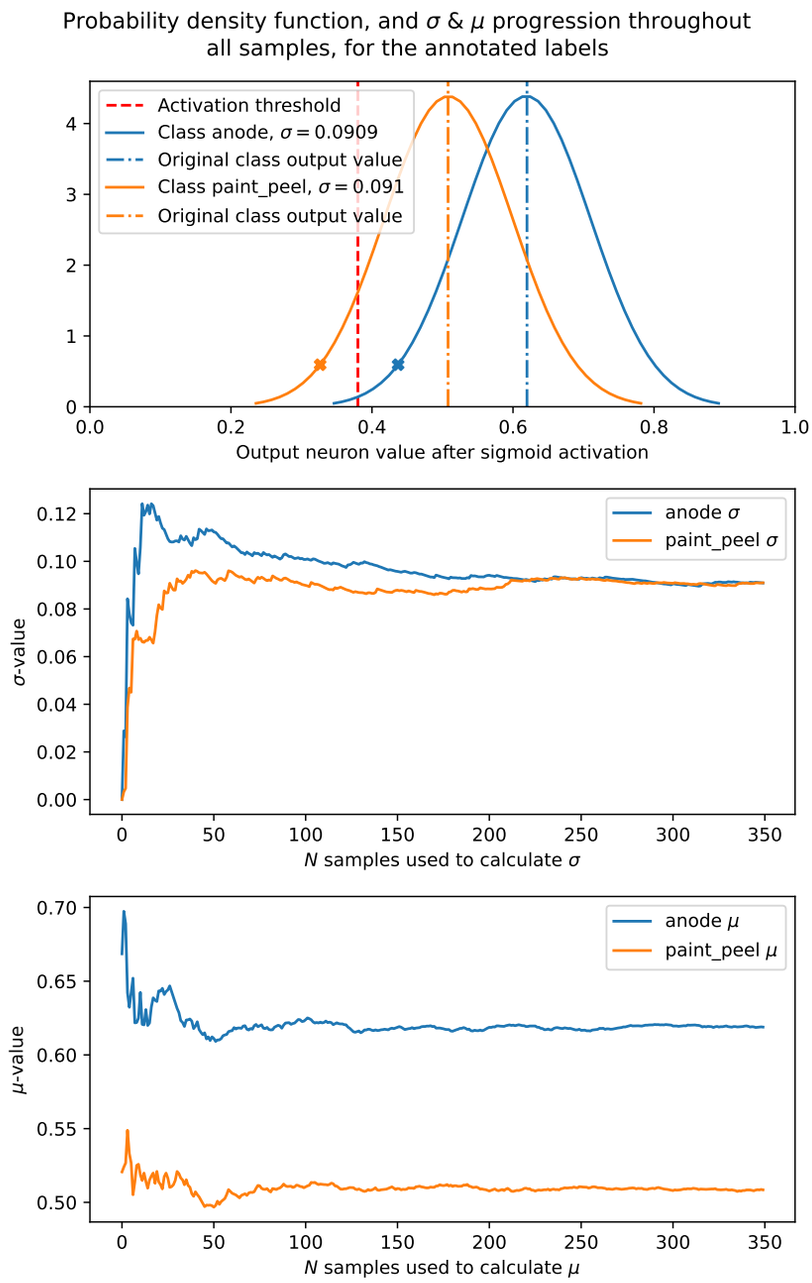



**Figure 4.10:** Probability density function(s) computed by the mean and standard deviation from the results achieved with Monte Carlo dropout applied to figure 4.7c. The $\mu - 2\sigma \approx 95\%$ is visualized as a cross. The two latter subplots show the change in $\sigma$ & $\mu$ as I increase the number of samples of Monte Carlo dropout. The true labels for the image are "anode" and "bilge_keel."

Figure 4.10 shows three subplots. The first subplot shows the uncertain classes viable to be classified as uncertain. Their original confidence value (denoted by a dotted dashed vertical line), probability density function (solid curve), and 95% mark (an x on the left side of each curve) are plotted for each class. The red dashed vertical line represents the classification threshold. The $x$-axis denotes the output neuron value. Each probability density function has an area underneath the curve equal to one. The variance (width) of the curve(s) represents the uncertainty. A broader curve means more uncertainty in the prediction for that class. The probability density function(s) consist(s) of hundred samples. The class "anode" is almost as uncertain as "paint_peel," but only the latter is classified as uncertain as its 95% threshold is below the classification threshold.

Subplot two shows each class's progressive $\sigma$ (standard deviation) change as I increase the number of Monte Carlo dropout samples. The $y$-axis denotes the $\sigma$-value, and the $x$-axis denotes the $N$ samples used to compute $\sigma$. In total, 350 samples were collected. Both classes seem to stabilize closer to a hundred samples.

Subplot three shows each class's progressive $\mu$ (mean) change as I increase the number of Monte Carlo dropout samples. The $y$-axis denotes the $\mu$-value, and the $x$-axis denotes the $N$ samples used to compute $\mu$. In total, 350 samples were collected. Both classes stabilize closer to seventy-five samples.

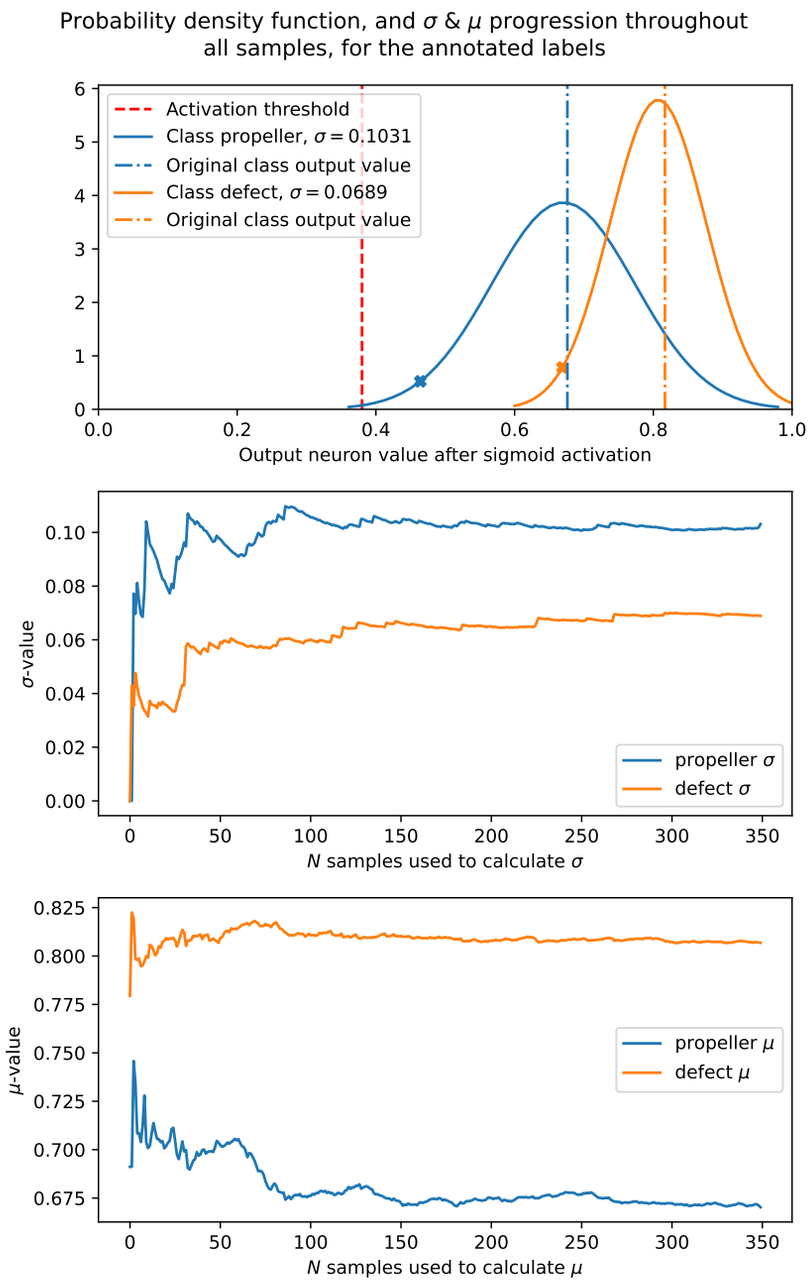



**Figure 4.11:** Probability density function(s) computed by the mean and standard deviation from the results achieved with Monte Carlo dropout applied to figure 4.7d. The $\mu - 2\sigma \approx 95\%$ is visualized as a cross. The two latter subplots show the change in $\sigma$ & $\mu$ as I increase the number of samples of Monte Carlo dropout. The true labels for the image are "defect" and "propeller."

Figure 4.11 shows three subplots. The first subplot shows the uncertain classes viable to be classified as uncertain. Their original confidence value (denoted by a dotted dashed vertical line), probability density function (solid curve), and 95% mark (an x on the left side of each curve) are plotted for each class. The red dashed vertical line represents the classification threshold. The $x$-axis denotes the output neuron value. Each probability density function has an area underneath the curve equal to one. The variance (width) of the curve(s) represents the uncertainty. A broader curve means more uncertainty in the prediction for that class. The probability density function(s) consist(s) of hundred samples. The class "propeller" is the most uncertain one as it has the broadest curve but doesn't breach the classification threshold.

Subplot two shows each class's progressive $\sigma$ (standard deviation) change as I increase the number of Monte Carlo dropout samples. The $y$-axis denotes the $\sigma$-value, and the $x$-axis denotes the $N$ samples used to compute $\sigma$. In total, 350 samples were collected. Both classes seem to stabilize around 120 samples.

Subplot three shows each class's progressive $\mu$ (mean) change as I increase the number of Monte Carlo dropout samples. The $y$-axis denotes the $\mu$-value, and the $x$-axis denotes the $N$ samples used to compute $\mu$. In total, 350 samples were collected. Both classes stabilize closer to eighty samples.

### Afterthoughts

After looking at four different images and watching their change in $\mu$ and $\sigma$ as I increase the number of Monte Carlo samples, we've observed a few things. Figure 4.8 gives a great example that for some annotations, there is little to no doubt from the model's side. While figure 4.10 shows us that even though the class "anode" has a higher degree of uncertainty, because I use eq. 2.6 to compute if a distribution is classified as uncertain, only the "paint_peel" distribution makes the image qualified to be classified as uncertain. It is also these cases that the equation where supposed to cover. Even if an image has a certain degree of uncertainty in it, it doesn't qualify it to be classified as uncertain. Since $\mu$ and $\sigma$ dictate if an image is classified as uncertain, the images must be sampled enough times to stabilize the change in $\mu$ and $\sigma$. Looking roughly at the graphs from the four images, I conclude that a hundred samples should suffice. It may be more than necessary, but as it can't overfit (like a model), it's better to ensure the distribution is stable. Having fewer samples shortens the computation time, but optimizing Monte Carlo dropout is not the goal of this thesis.

### 4.1.3 Exploring incremental learning that also accounts for uncertainty

These results come from the experiment described in section 3.2. Figures 4.14, 4.13, and 4.12 show all the model variations compared to each other. All three figures show the same models but test and train split into each of their figures for clarity. The same goes for 4.20, 4.19, and 4.18, but they show the graphs for the models trained with a budget of 35 images per batch.

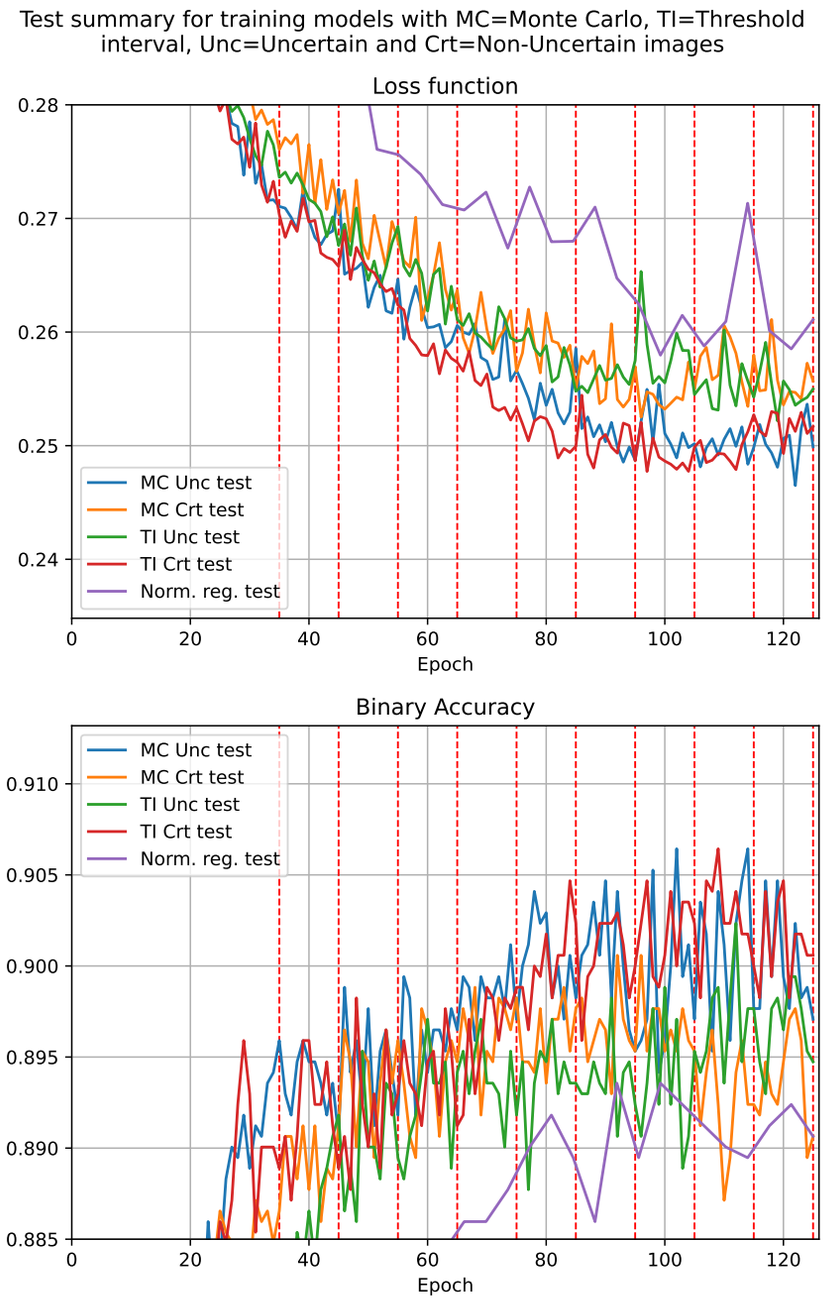



**Figure 4.12:** Test loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons.

### Test graphs for models that account for uncertainty (figure 4.12)

Looking at the loss function for each test graph shows that they all perform slightly better than the regular model in terms of performance. Even though the regular model spikes towards the end, it returns to one of its lower values, close to the global minimum. At its lowest, it's just below 0.26. The other graphs are not too far apart, but we see that towards the end, the two graphs (MC unc and TI Crt) slightly diverge from the other two (MC Crt and TI Unc). However, they encounter some overfitting and end up close to each other. Overall, MC Unc reaches the lowest loss out of the four. TI Unc and MC Crt perform the worst, but MC Crt fluctuates more than TI Unc, going both above and below TI Unc until around epoch 95. TI Crt follows MC Unc closely but performs better for the middle part of the training. The only real problem it seems to have is that it overfits harder than MC Unc from around epoch 110. While the overfitting for TI Crt doesn't stop, the MC Unc seems to have more potential to fluctuate up and down.

The test binary accuracy graph for the different models all end up close to each other, within a 1% accuracy difference. All the models follow each other closely until TI Unc falls off early at around epoch 75, but it picks up again later and peaks at almost the same height as MC Unc and TI Crt. MC Unc and TI Crt peak at the same height of around 90.6%, but MC Unc is twice up at the same peak value and performs arguably better than TI Crt until epoch 80. However, MC Unc loses a lot of performance during the last image batch.

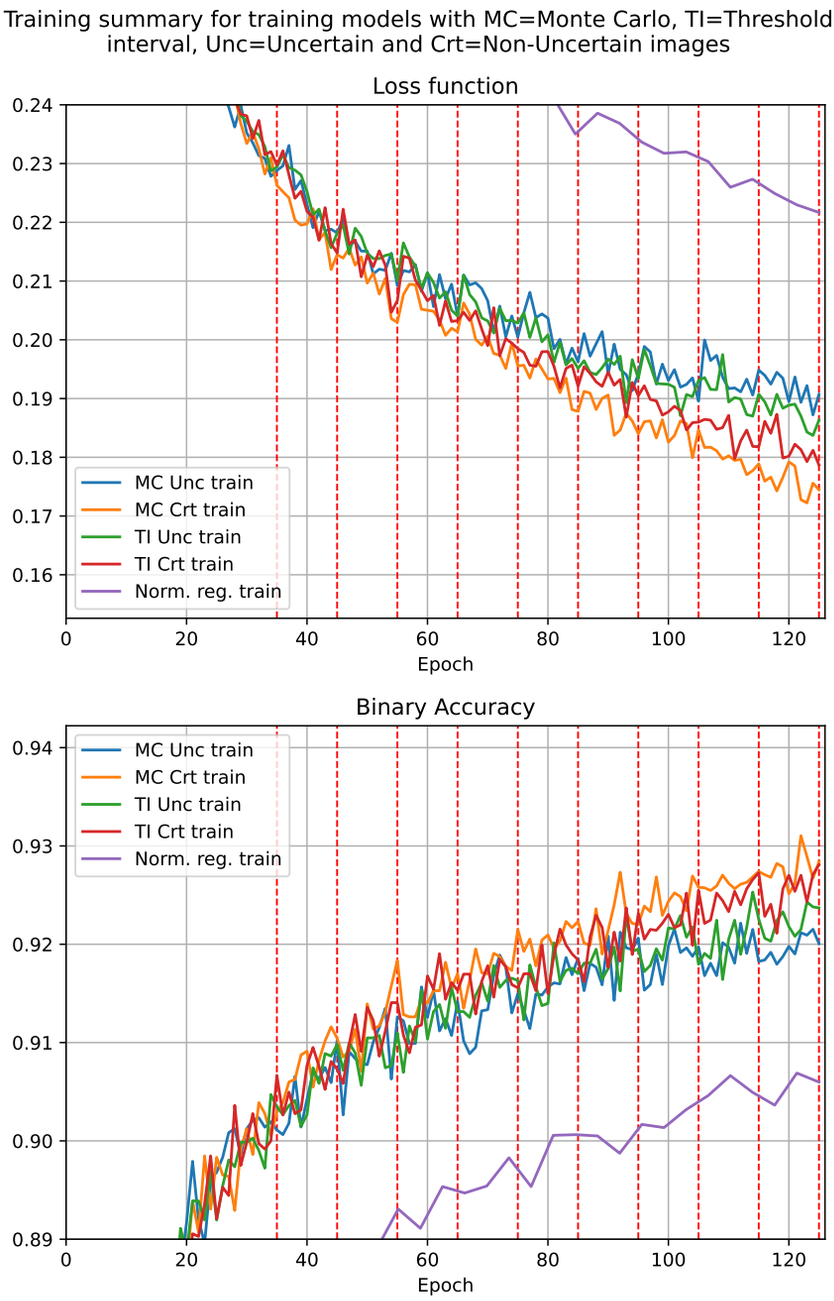



**Figure 4.13:** Train loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons.

### Train graphs for models trained with regards to uncertainty (figure 4.13)

It appears that the loss function for all the models has not yet converged. This is good as it suggests the training stopped before they had overfitted to their full potential. However, they all perform a lot worse than the regular training graph. This is because the model has trained for many more epochs on the same images than the regular model, a side effect of my incremental learning method. At around epoch 55, we also observe that two graphs (MC Unc and TI Unc) diverge from the other two (TI Crt and MC Crt). TI Crt also diverges from MC Crt at around epoch 80. MC Unc and TI Unc perform better than MC Crt and TI Crt.

The train binary accuracy graphs also show that they all perform worse than the regular model with almost 3%. Looking at how the other graphs compare, they are about the same. However, also here, MC Unc performs the best, followed by TI Unc, TI Crt, and then lastly, MC Crt. The exact order as the loss functions. The differences here are not as apparent as their loss functions, but the models trained on uncertain images perform better. MC Unc performs almost 1% better than MC Crt.

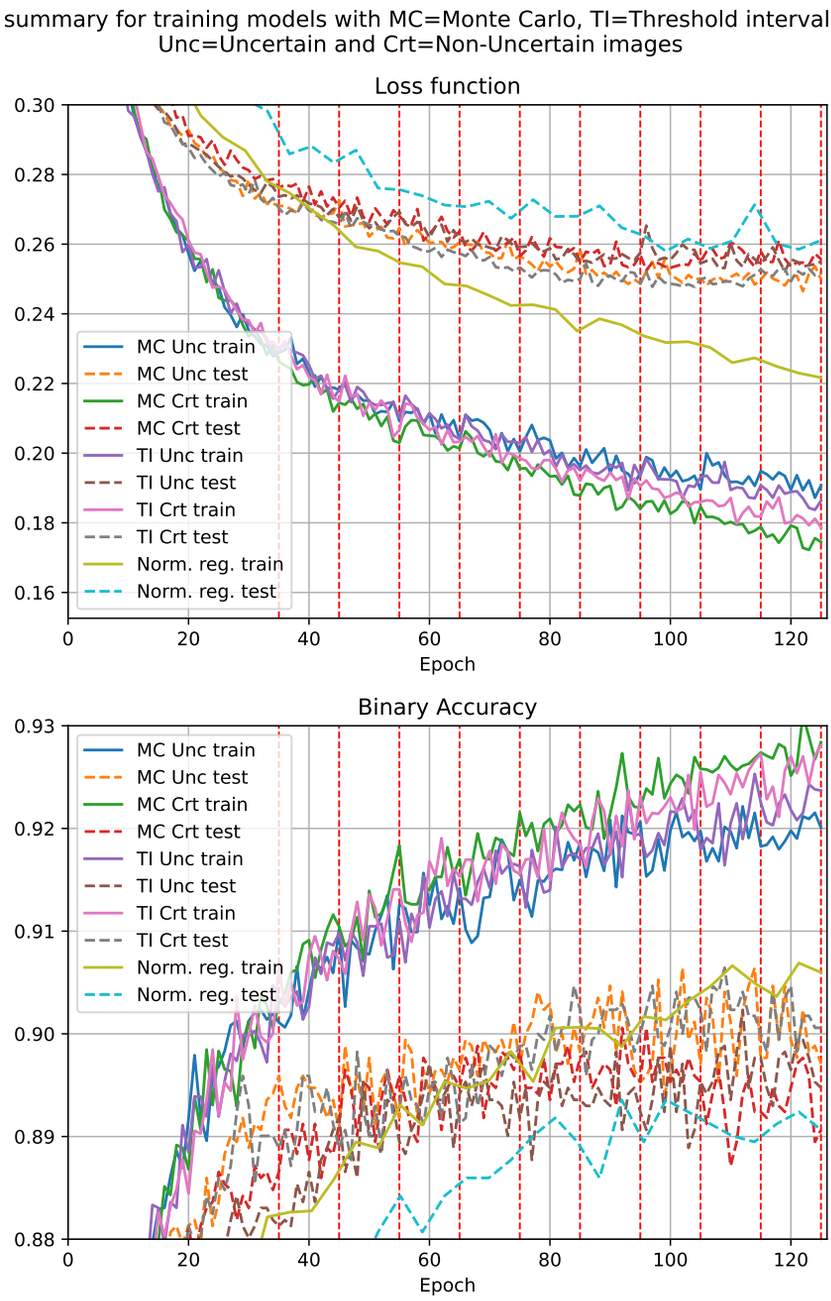



**Figure 4.14:** Train & test, loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons.

### Reviewing the training summary graphs for the models that account for uncertainty (figure 4.14)

We have learned a few things after looking at the test and train graphs individually. Both models trained on uncertain images have less overfitting on the training data than those trained on non-uncertain images. This is true in terms of both loss and binary accuracy. However, they still overfit quite a bit on the training data compared to the regular model. It's worth noting that most of the training after the initial 35 epochs will only cause the models to overfit more than the regular one (as it's only trained for 35 epochs in total).

I think the models trained on uncertain images perform the best on the training data overfit-wise (high loss and low binary accuracy) because the images later added to the training set are all images the model doesn't know. If the model does not recognize the images, it is forced to change so it can recognize them. This also means it probably has to sacrifice some of its weights trained on the old images to recognize the new ones better.

It is important to remember that the training loss and binary accuracy are not the primary objectives of the experiment but the test loss and binary accuracy. However, the test data performance of TI Crt and MC Unc is too similar to declare a definitive winner. A possible reason that these two perform the best on the test data may also be because, for this experiment, there is no cap for how large a portion of the image batch can be classified as uncertain/non-uncertain. So it is possible for a model to either get zero images or all hundred. The test data set is also "independent" because there is no guarantee that the images in the training data set also are a good representation of the features in the test data set. Therefore which images the test data set consists of plays a significant role in the test performance of the model.

### Mean model curves

The figures 4.17, 4.16 and 4.15 show the mean graph of ten different runs with randomized data ordering before the splitting of test and train data set. The models are still initially trained on 800 images before batches of 100 images are added for processing. The dashed vertical red lines separate the image batches. The training graph for a regular trained model is also included. It has been normalized to fit the $x$-axis. The same goes for 4.23, 4.22, and 4.21, which I will later discuss, but they show the graphs for the models trained with a budget of 35 images per batch.

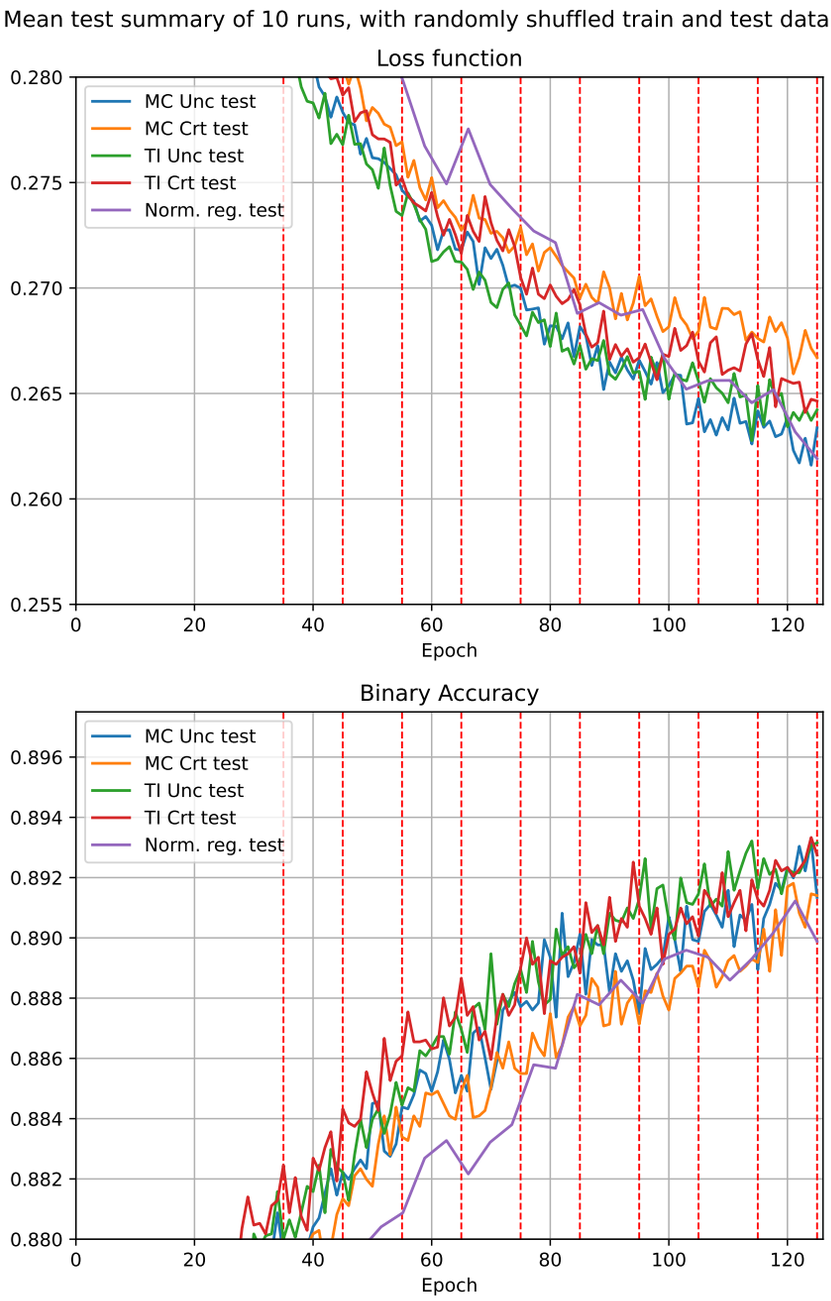



**Figure 4.15:** Test, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons.

### Mean test graphs for models that account for uncertainty (figure 4.15)

The mean test curves of the ten runs with randomized train and test data sets also provide insight. The test loss functions all perform worse than the regular model mean test loss function. However, we observe that both models training on uncertain data (MC Unc and TI Unc) perform better than those performing on non-uncertain data (MC Crt and TI Crt). Once again, MC Unc performs the best of the four, and MC Crt the worst. The regular model has almost the same test loss when looking at a single run versus the mean of ten runs (around 0.26 loss). While the models trained on uncertainty images have increased their test loss (from all under 0.26 to above 0.26). This indicates that the test data set distribution has more to say for these models. The regular model, however, is not affected to the same extent.

The binary accuracy graphs for the figure are very tight, and MC Crt is the only model that certainly is not performing the best. In figure 4.14, it can be observed that the performance of MC Crt was the lowest also when a single run was graphed. The other three models (MC Unc, TI Unc, and TI Crt) perform slightly worse here than in the single runs.

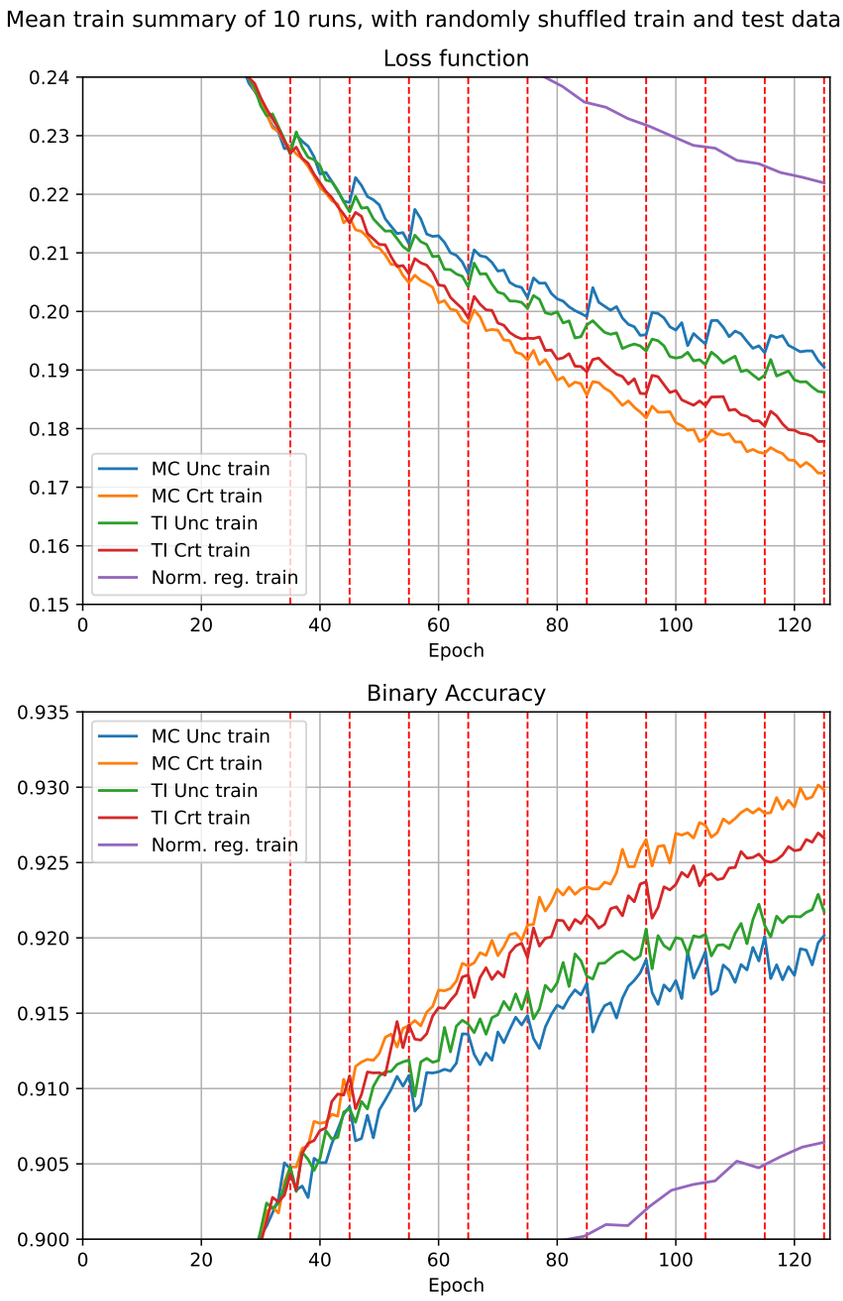



**Figure 4.16:** Train, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons.

### Mean train graphs for models that account for uncertainty (figure 4.16)

We observe from the loss function graphs that no matter the variation used to train the models, they take a hit to their training loss when a new image batch is introduced. This becomes very apparent when looking at the mean graphs compared to the single runs. Their end values are very similar to those when doing a single run, as seen in figure 4.13. This suggests that differing the training data set does not impact the training learning curve as much. It is clear that MC Unc performs the best and MC Crt performs the worst.

The binary accuracy graphs also mostly lose performance as new image batches are introduced. This is, however, not always true, especially for the two models trained on non-uncertain images. As new images are introduced, they will lose very little or even perform better. This seems to correlate to when the models trained on uncertain images (MC Unc and TI Unc) lose the most performance (see the transition to the fifth image batch around epoch 75). The trend is that the models trained on uncertain images still overfit the least, with MC Unc being the best.

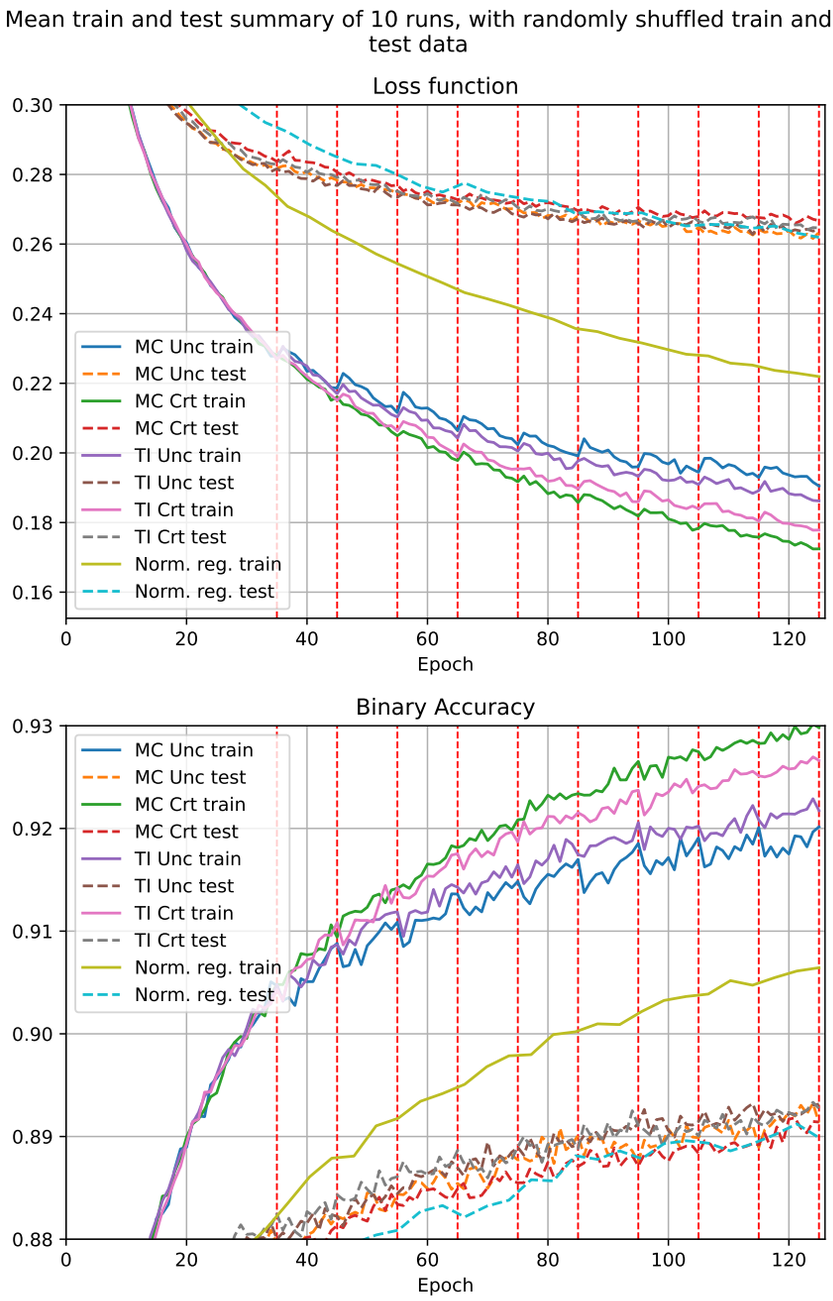



**Figure 4.17:** Train & test, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons.

### Reviewing the mean training summary graphs for the models that account for uncertainty (figure 4.17)

After running the various models ten times with different train and test data sets, some new information has been revealed. Training models on uncertain data yields a more significant rise/drop in training loss/binary accuracy, respectively, than training on non-uncertain data. This may also have something to do with the number of images each model trained on for each image batch. Although it is hard to say if MC Unc or TI Crt yields the best test results, it's clear that MC Unc yields the least overfitting on the training data.

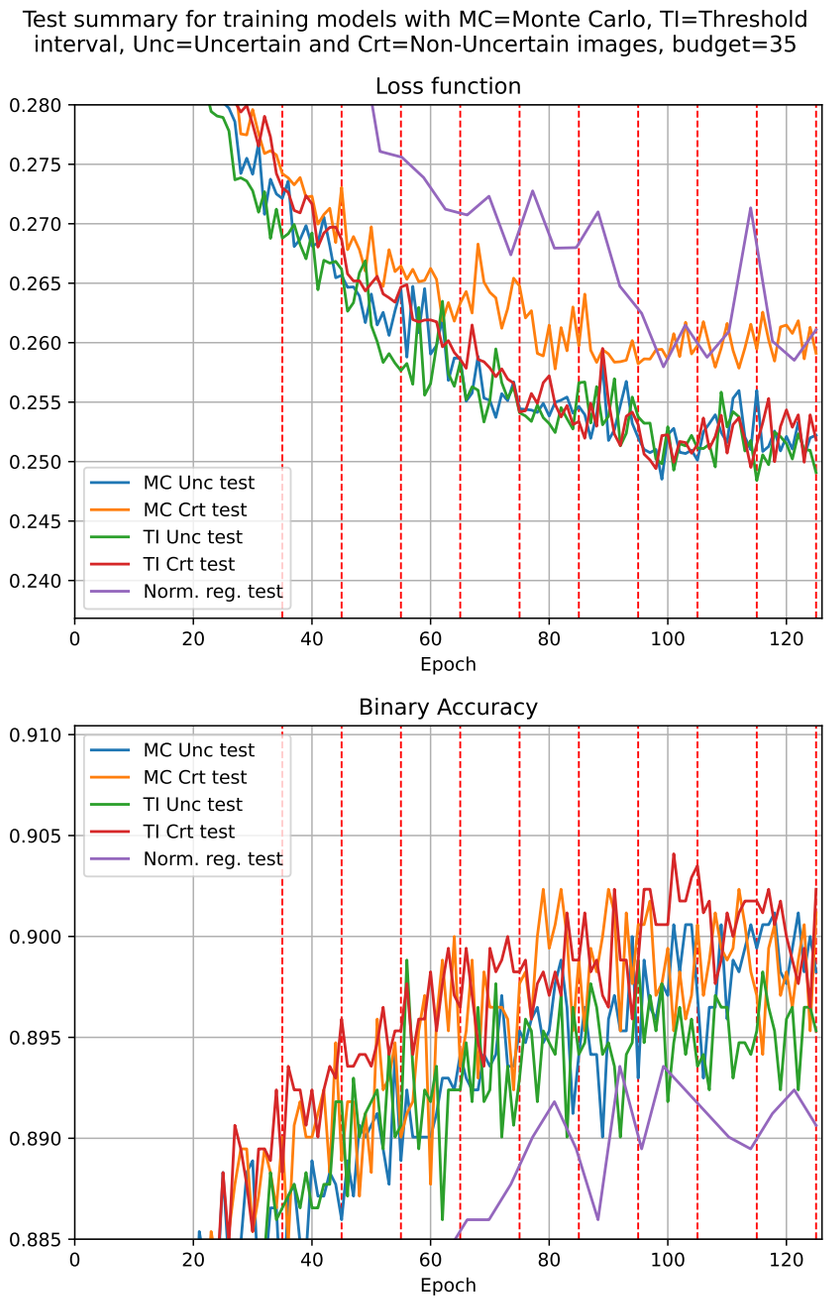



**Figure 4.18:** Test loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons.

### Test graphs for models trained that account for uncertainty, with a budget of thirty-five images 4.18

Looking at the test loss graph, we observe that MC Crt performs the worst of them all, diverging from the other three at epoch 60. Other than that, the three graphs (MC Unc, TI Unc, and TI Crt) are nearly dead even.

The binary accuracy subfigure neither has a clear, better-performing model. TI Crt peaks the highest but quickly falls in accuracy again. TI Unc is the only model that's almost consistently beneath the other models, it being the one with the lowest peak.

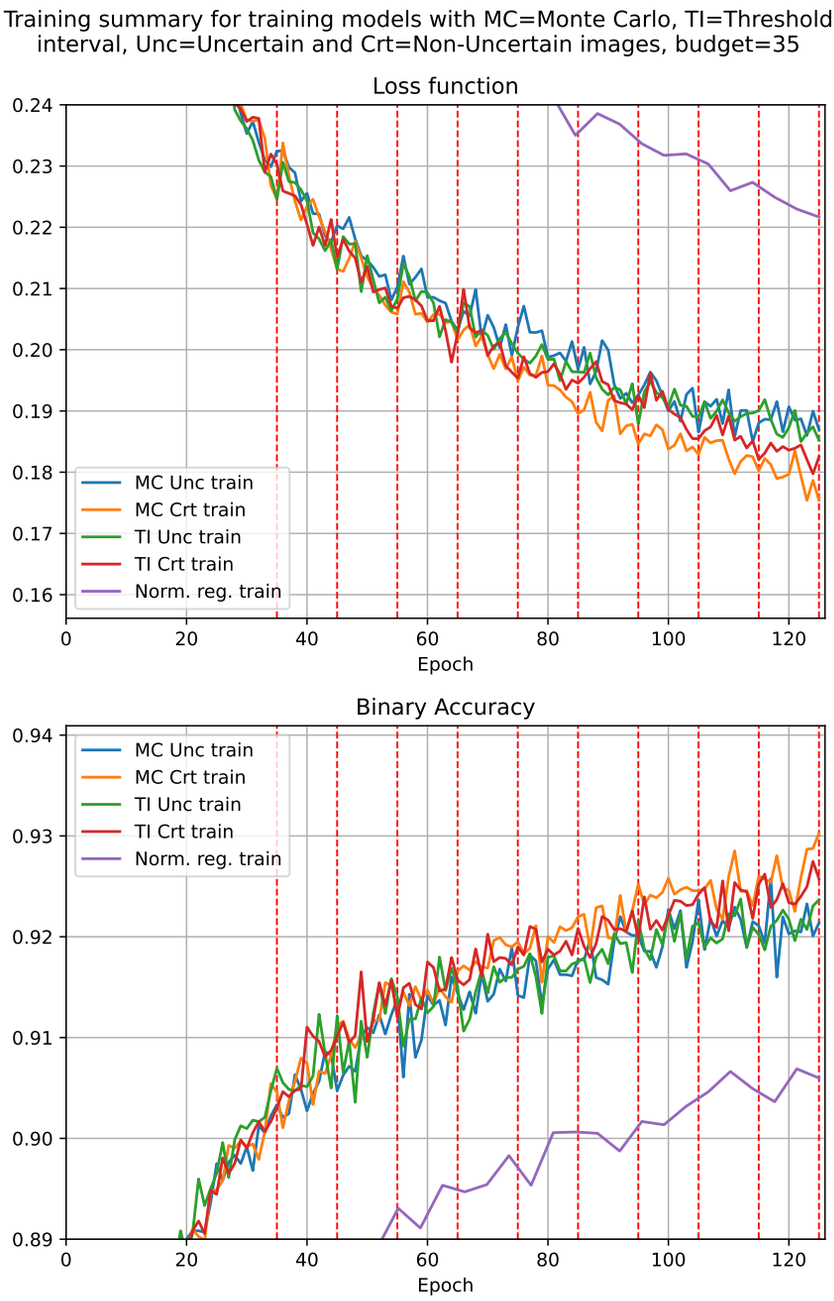



**Figure 4.19:** Train loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons.

### Train graphs for models that account for uncertainty, with a budget of thirty-five images (figure 4.19)

The train loss functions with a budget show that the models trained on uncertain images overfit the least. It is, however, not clear whether TI Unc or MC Unc performs better. But it is clear that MC Crt performs the worst.

The train binary accuracy graphs also now show that the models trained on uncertain images perform the best, as they have the least amount of overfitting. MC Crt is the model that performs the worst. The difference is not as clear anymore as in figure 4.13 because fewer images are trained on in general because of the budget cap, allowing for less room to diverge.

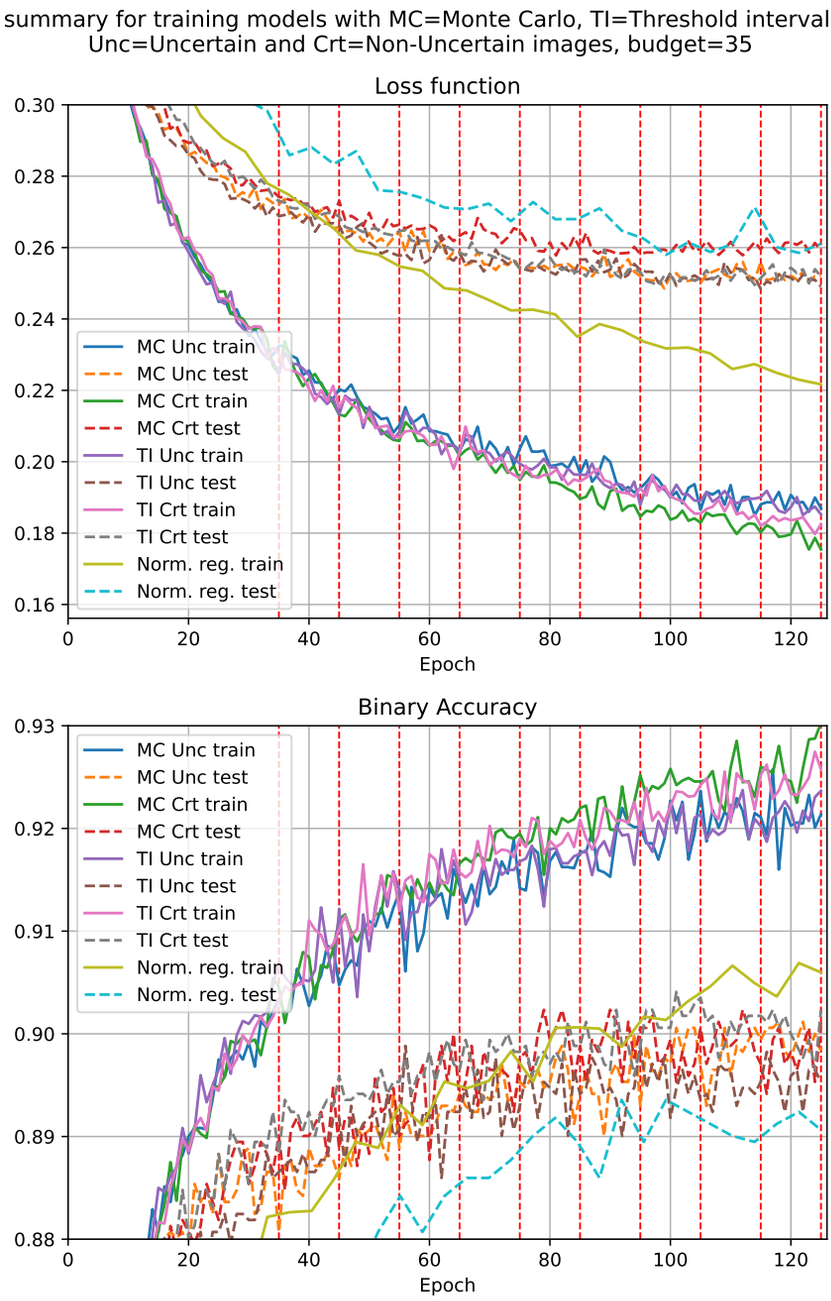



**Figure 4.20:** Train & test loss and binary accuracy graphs for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons.

### Reviewing the training summary graphs for the models that account for uncertainty, with a budget (figure 4.20)

The results from this experiment show us that models trained on uncertain images have less overfitting than models trained on non-uncertain images. Using a budget of thirty-five images affects the performance of the models by giving them fewer images to work with and therefore making them diverge less. Interestingly, MC Unc and TI Unc overfit less (in figure 4.14) when trained on more images (as it has no budget restrictions). This might suggest that the new training data actually contains knowledge the models don't have. When training on more data, it has more data to fit and therefore has to have a wider fit, sacrificing some knowledge. Compared to the budget model that receives less new data and therefore doesn't have to change its weights as much to fit the new data.

All of this is, of course, the opposite for the models trained on non-uncertain data. They already have knowledge about the data they train on and therefore overfit on the data. When they aren't on a budget and receive more images, they train more on the images they already know and overfit even more.

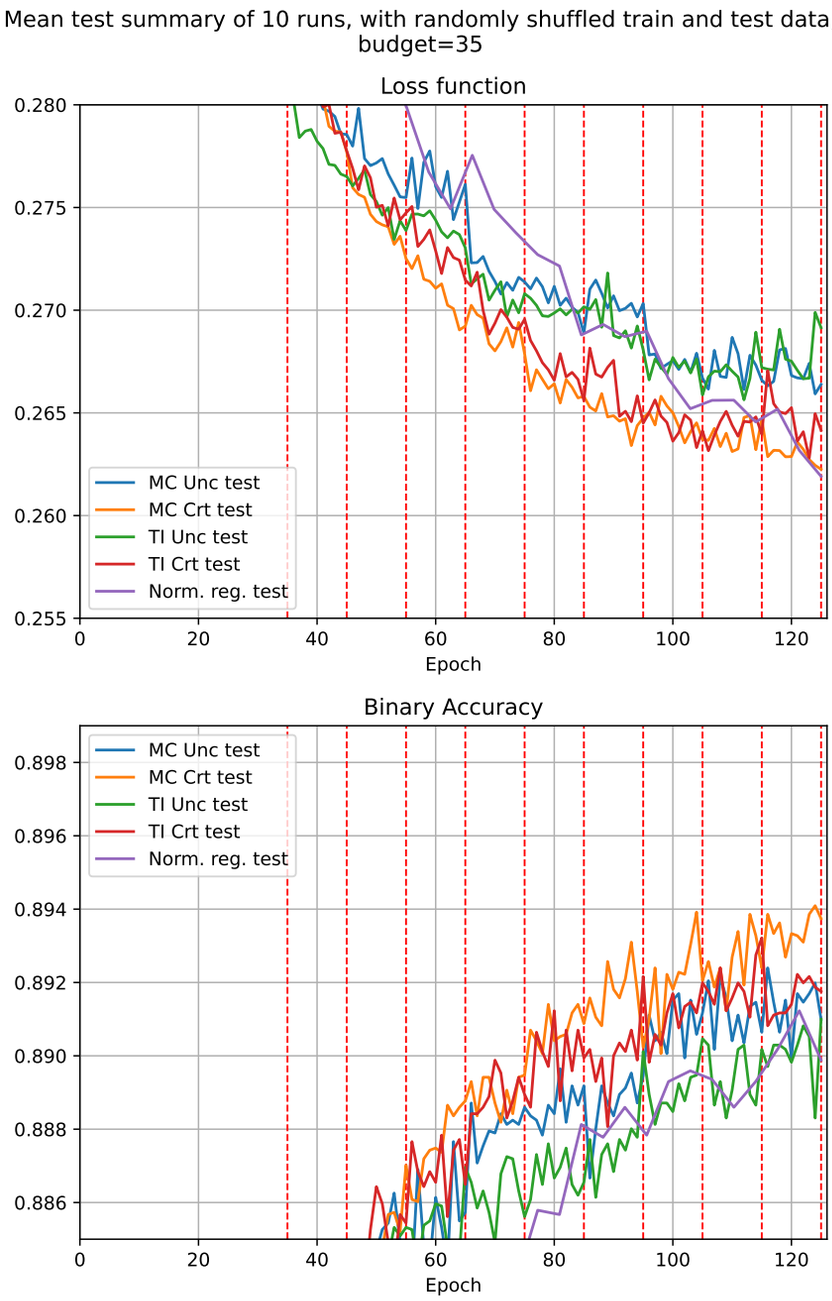



**Figure 4.21:** Test, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. The models have a budget of maximum thirty-five images to add to the training data per new image batch.

### Mean test graphs for models that account for uncertainty, with a budget (figure 4.21)

The test loss function mean graphs do not show the same as the single runs 4.18. For once, both models trained on uncertain images perform the worst. MC Unc and TI Unc perform equally bad until the last few epochs, and TI Unc spikes high (which is worse). The mean of each models' training summary shows that they all have worsened in terms of loss compared to their singular runs. MC Crt is perhaps the model that performs the best. TI Crt performs almost as well but fluctuates more towards the end.

Looking at the binary accuracy mean graphs, we see that also here, the worst model is a model utilizing uncertain images. MC Unc performs better and is almost at par with TI Crt, also towards the end. But TI Crt peaks taller than MC Unc, and the tallest of them all is MC Crt. It should be noted that the most significant difference between them is around 0.2%.

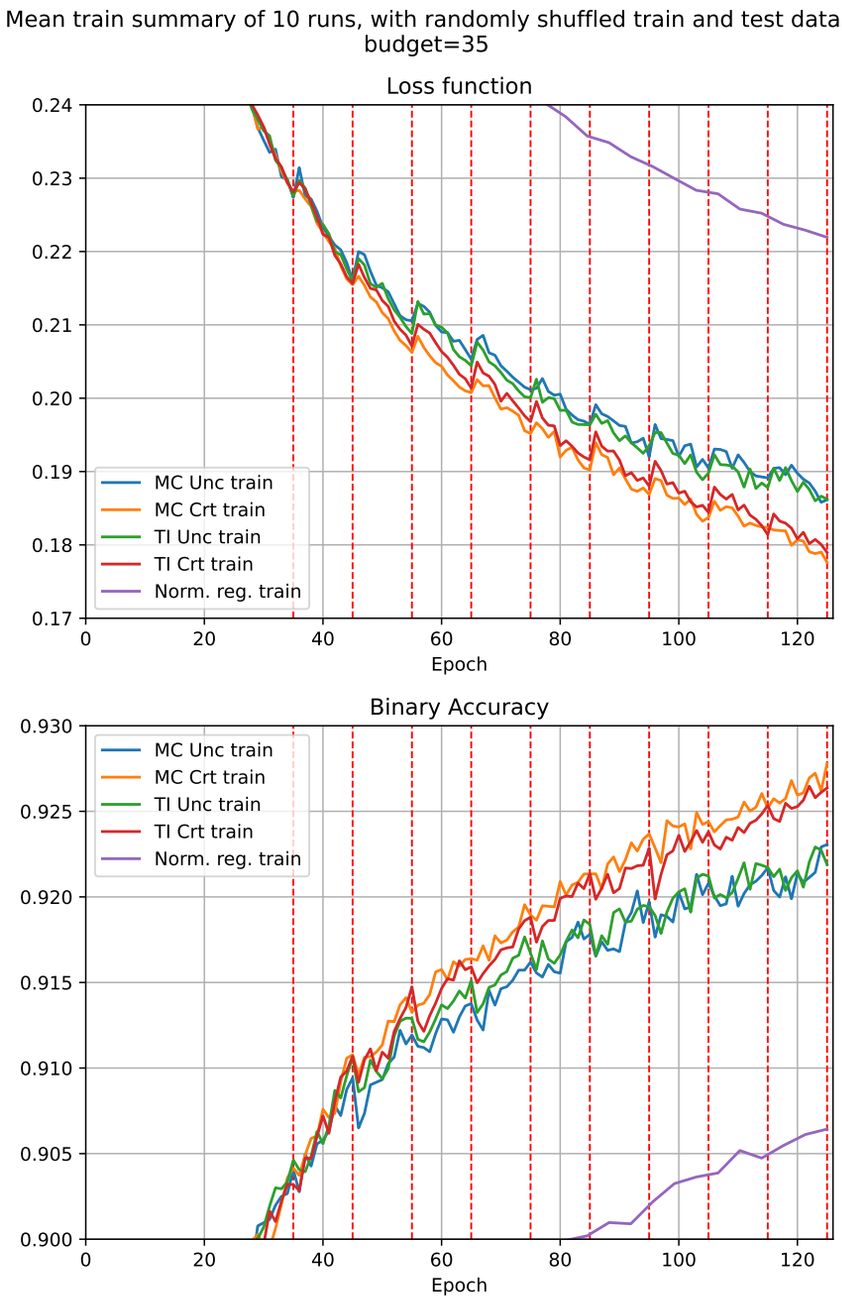



**Figure 4.22:** Train, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. The models have a budget of maximum thirty-five images to add to the training data per new image batch.

### Mean train graphs for models that account for uncertainty, with a budget (figure 4.21)

The training mean loss function, however, still shows that both the models trained on uncertain images perform better than those trained on non-uncertain images. MC Unc is performing slightly better than TI Unc, and TI Crt is performing slightly better than MC Crt. All the models' loss value rise as new image batches is introduced to the models. They all perform worse than the regular model's mean graph. The difference in loss value from the best to worst is about 0.01.

The binary accuracy mean graphs also show that the models trained on uncertain data perform better than those trained on non-uncertain data. MC Crt performs worse than TI Crt, but it's difficult to decide if TI Unc or MC Unc performs the best. All models lose some accuracy as new image batches are added.

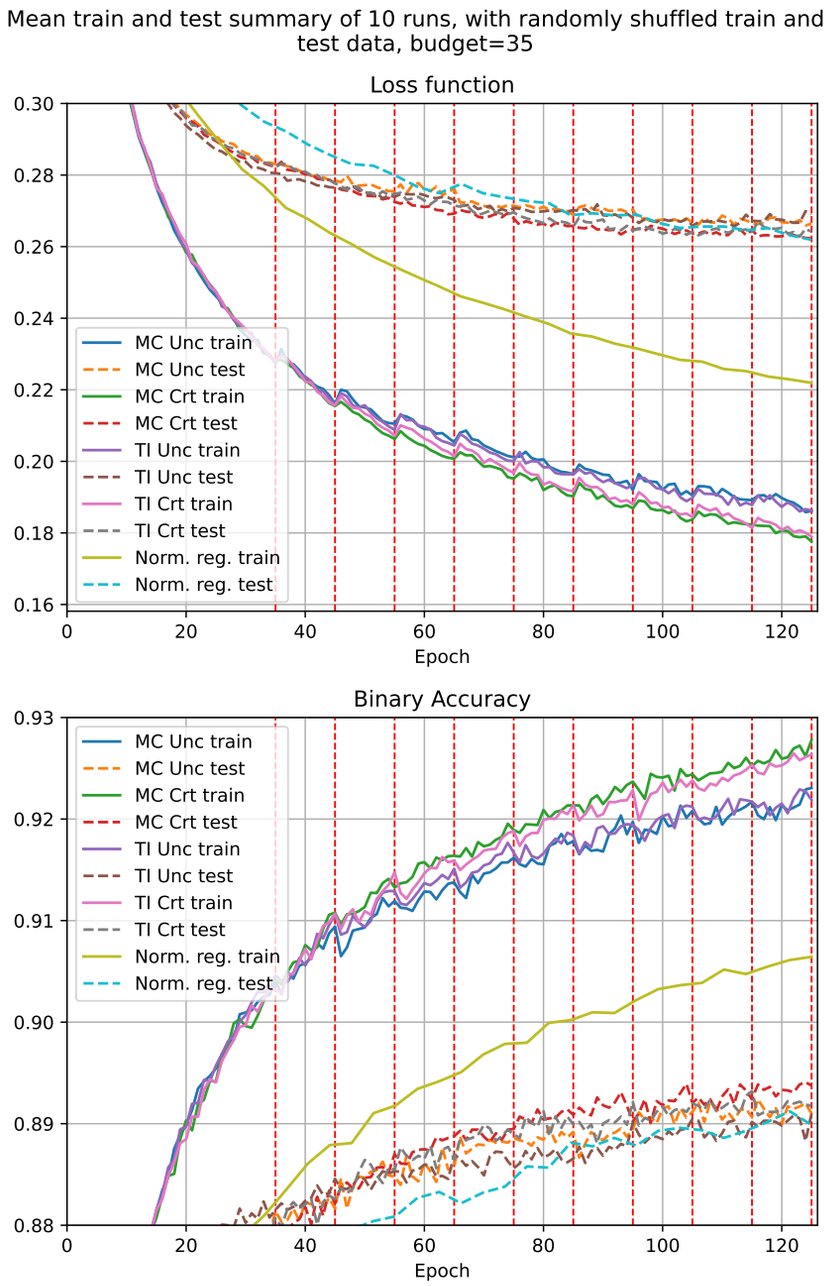



**Figure 4.23:** Train & test, mean loss and binary accuracy graphs of ten runs, for the four different combinations of models trained, together with a regular trained model's graph, normalized to the same training length as the four other models for easier comparisons. The models have a budget of maximum thirty-five images to add to the training data per new image batch.

### Reviewing the mean training summary graphs for the models that account for uncertainty, with a budget (figure 4.23)

The results from this experiment show that the models also overfit when having a budget, compared to the regular model. The models that consider uncertainty perform worse than the regular model on the training data. This is expected as they train for many more epochs on the data than the regular model. However, it was also expected that the four models would perform worse on the test data as they have only trained on a fraction of the entire data set. The number of images in the data set may be calculated by eq. 4.1. This gives the four models (a maximum of) $800 + 9 \times 35 = 1115$ images to train on, compared to the regular model that trains on 1703 images. We observe that all the models perform almost as almost well as the regular model when it comes to loss, and even better than the regular model in terms of binary accuracy.

$$Training\ images = start\ batch + n\ batches \times max(image\ batch\ size,\ budget\ size) \tag{4.1}$$

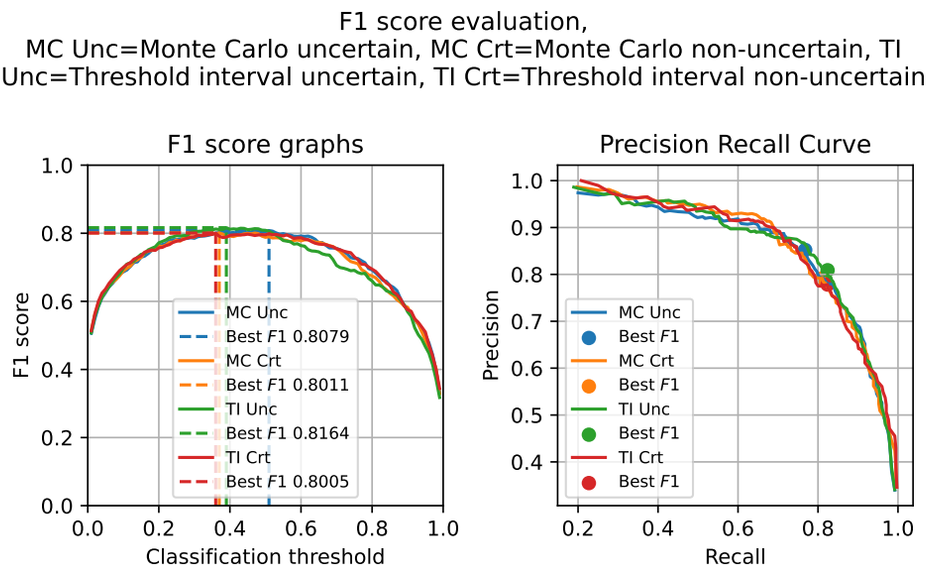



**Figure 4.24:** The F1 score evaluation for the four models. The left subplot shows the F1 score for different thresholds. The $x$-axis denotes the threshold value, and the $y$-axis denotes the F1 score. The right subplot has a $y$-axis that denotes the precision and an $x$-axis that denotes the recall. The graph shows the precision for different recall values.

### Evaluating the F1 scores for models that account for uncertainty

In figure 4.24, we observe that the model with the highest score is TI Unc, MC Unc, MC Crt, TI Crt, in that order. The noticeable difference between the four models is that MC Unc has its best F1 score at a threshold of around 0.5, while the other models have their best threshold just below 0.4 (which is close to the regular model). However, the F1 score is around 0.80 for all the models, with a threshold from around 0.3 to 0.5. There is only a 1,6 score difference in performance between the model that performs the worst and the one that performs the best.

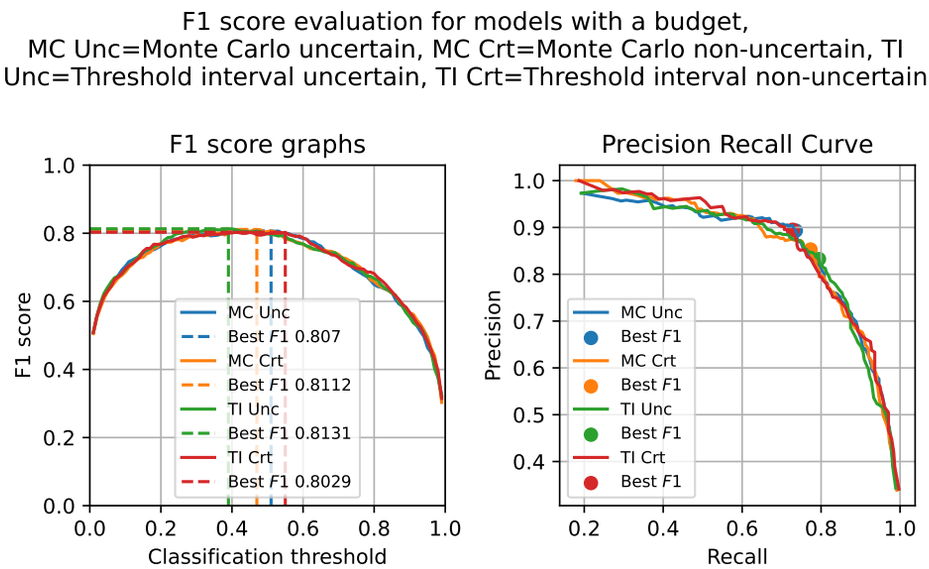



**Figure 4.25:** Shows the same as 4.24, but for the models trained with a budget.

### Evaluating the F1 scores for models that account for uncertainty, with a budget

In figure 4.25, we observe that the model with the highest score is TI Unc, MC Crt, MC Unc, and TI Crt, in that order. Compared to figure 4.24, the thresholds are more spread out, ranging from under 0.40 to about 0.55. After introducing a budget, both models trained on uncertain images' F1 scores fell. While both the models trained on non-uncertain images have boosted their F1 score to a degree. MC Unc is the model that stays the most consistent after introducing the budget, compared to the other three models.

#### Afterthoughts

After looking at all the different model results, it's not easy to draw a solid conclusion. The main differences between the budget and non-budget experiments would be that the differences are larger between the four models when they get to choose all the images from the image batch. However, this may be because of an imbalance in the number of images they each get to train on. The consistencies are that MC Unc and TI Unc overfit the models the least, while MC Crt and TI Crt overfit the most. This is true for loss and binary accuracy, with and without a budget.

Based on the F1 score graphs, we may adjust the classification threshold $t$ between $0.3 < t < 0.5$ to achieve slightly higher precision or recall without significant loss of performance. This allows for some flexibility in our approach if needed.

## 4.2 General discussion

### 4.2.1 Pros and cons by showing the user the model annotations before the user self annotates

Even though humans don't have to annotate images in these experiments, this work is meant to be applied in the web application I've also created, which is described in Riise [9], and in sections 2, A. As a video is uploaded, not only the uncertain images are shown but also which labels the model predicted in the image. As discussed in Riise [9], many images in the LIACI data set are noisy, which is most probably true for user-uploaded videos. The model annotating an image helps the user understand what's in the image, but it may also give the user a bias and make them see things that aren't in it. This is a common problem with users acquiring new knowledge from chatbot services. The AI model will give the user information, but naively the user may choose to blindly believe this information as a base truth, even if a closer look at it would prove faulty logic. This is why it is essential to be critical of the information provided by AI. However, using AI as a tool can save valuable time. As in this project, the model annotates the video, but certified personnel would still have to evaluate the images to make any actual conclusions about what is in the images. AI helps the user browse the images and gives the user baseline information to work with.

### 4.2.2 Using uncertainty for other purposes than training

Even if Monte Carlo dropout can't be concluded as the definitive best method to decide which images should be annotated by the user, it can still be used to inform the user that an image should be more carefully evaluated by the user. From the results of experimenting with uncertainty, we observed that the only image wrongly annotated was the image with an uncertainty in it. The class the model was uncertain of was also the class that was not present in the image.

### 4.2.3 No time measurement

No time measurements were made or included in this report, as this is not the project's goal. The model created in this project is meant to run on weak hardware commonly found in laptops. However, there has been a focus on the models used in this project should be lightweight to run on inferior hardware. I'm measuring the different methods' accuracy performance, not their time complexity. The time complexity will be more significant for the Monte Carlo dropout method, but if that method achieves the best performance, it's a valuable trade-off. The model's output will always be the same, independent of hardware.

#### Choosing the budget to be thirty-five

To balance the number of images each model gets, another experiment was run with a budget of thirty-five images, meaning a model could only use a maximum of thirty-five out of the hundred images in the batch. The number thirty-five was used because the models usually get at least thirty-five images after observing the regular experiment.

#### Insufficient data and hyperparameters

The reason the models performed too similarly to each other has maybe something to do with how the data was distributed in terms of image batches. There are many possibilities for how they could have been structured, both the initial batch and the "new" batches could have been larger and smaller. The budget size could also potentially have proved to play a more significant role with other image batch sizes. To tune the uncertainty classification threshold, the interval could have been made bigger/smaller for the "threshold interval" strategy, and for Monte Carlo dropout, the threshold could also have been manipulated either up or down.

### 4.2.4 Future work/further improvements

#### Improving the results of incremental learning

The variation of incremental learning I concluded with seems fine enough for this project. However, it could still use improvements. The main problem with the variation I landed on was that the method caused more overfitting than regular training. This is most likely because each existing image in the old training data set has already been trained on by the model before, and for every new image batch, the model trains on the old images even more times. To combat overfitting, better regularization methods could have been tested.

#### Vanishing gradients

Vanishing gradients are a common problem in very deep neural networks and recurrent neural networks. It is an unwanted effect that causes earlier layers (or gates in the case of recurrent neural networks), i.e., layers further away from where the backpropagation started, to learn less because the gradients start to vanish. I don't wish for my model to suffer from this, but an idea to fight overfitting would be to weigh older image batches less, halving their effect for each new image batch added, as illustrated in figure 4.26. This incremental learning method would be somewhere between the two implemented in this project. Instead of focusing as much on the older images as the newly added ones, the older ones would not have as big of an impact as the newly added images. A downside of this method is that it has to compute the loss for every image in each epoch but only gets a fraction of the total loss.

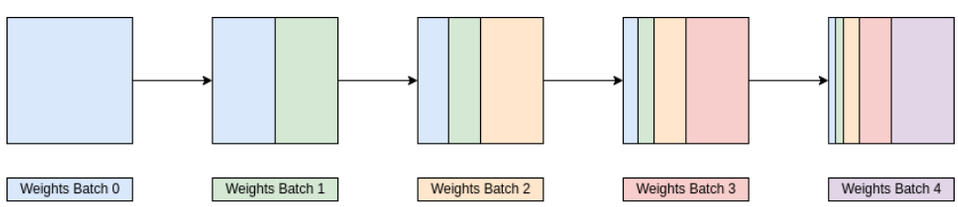



**Figure 4.26:** Illustrates how each image batch would have their image loss weighed.

A potential solution to this problem would be a more stochastic approach. Instead of using every image in the data set, only a random portion of the older data set is evaluated for each epoch. The model can therefore train on fewer images and use all the loss it computes. Figure 4.26 is still relevant as it would now represent the portion of images from each "generation" included in the epoch. The downside to both these methods is that extra values have to be stored in the image metadata and taken into account every time the

images are to be trained on. However, these methods could solve the problems that the methods in this project have. As seen from the experiments, method one diverges too far from the original data and has fluctuating results from the original data. Method two doesn't have the same fluctuating results but overfits the training data much more than the original model.

### PCA of images classified as uncertain vs. those classified as non-certain

An interesting experiment would be to conduct some form of feature analysis like PCA. Comparing the images that are classified as uncertain vs. those classified as non-uncertain. PCA could provide insight into why some images are classified as uncertain. It could also prove that humans can't observe any meaningful discrepancies. But it could prove that images with outlying features are classified as uncertain. If this was the case, the PCA distribution could be further evaluated with the model's original output to determine if an image is uncertain.

### Class-wise evaluation

As shown in earlier figures 2.6, every class has its own precision and recall score. An idea could be to implement a custom classification threshold for every class optimized by an F1 score most fit for the class. This would also yield different uncertainty classifications for each class (compared to now), potentially leading to better results for both methods of calculating uncertainty. Another idea could be continuously moving the classification thresholds relative to the latest F1 score. In this project, I simplify it by using the F1 score computed in Riise [9]. Bear in mind that moving the F1 score will not affect the training as the sigmoid output value is used to compute the loss.

### Using the degree of uncertainty

What I've done in this experiment is find if there is an uncertainty in the image, but I don't necessarily take into account the degree of uncertainty. What could have been interesting to experiment with would be only to evaluate the degree of uncertainty to classify an image as uncertain, independent of where it is in relation to the threshold. However, only the second strategy, Monte Carlo dropout, has the ability to measure the degree of uncertainty. But it could still be compared to the performance of the models in these experiments.

### Training the models from scratch every time

Lastly, it could also be interesting to see if the models would converge differently if they were trained from scratch every time. The training period in total would, of course, be much more extended, but the models would probably not overfit as much as they currently do.

# 5

# Concluding remarks

This thesis presents an exploration of incremental learning and uncertainty in deep learning, where the results are applied to a model utilized by a web application that SINTEF may deploy as part of the LIACi project as it is or may be further developed.

Chapter 1 presents the two hypotheses created to set more specific topics to explore.

1. A machine can evaluate a model's confidence value through uncertainty to indicate which images the user should annotate to yield the best learning per image ratio.

2. There is no need to train the model from scratch every time, as it only needs to be trained for a fraction of the original epochs.

Chapter 2 describes the different methods used to perform the experiments and the necessary website changes that were needed to show the user uncertainty. The experiments use data, models, and accuracy measurements used in Riise [9], but also new methods, mainly the two strategies to classify uncertainty in images. The first strategy for classifying uncertain images is threshold interval which only evaluates the sigmoid output (also known as the confidence) value of the model and uses it to classify an image as uncertain. The second strategy is Monte Carlo dropout which samples subnetworks of the fully connected layer in the model to compute a $\mu$ & $\sigma$, where $\sigma$ is the degree of uncertainty.

Chapter 3 demonstrates how the three experiments were performed. The first experiment is tied to hypothesis two and tried to determine if satisfactory performance could be achieved with incremental learning by training on only the new data training data or by training on all the training data for each increment of new training data. The second experiment was performed to determine how many times Monte Carlo dropout had to be sampled to achieve a stable $\mu$ & $\sigma$. The third experiment was tied to the first hypothesis and tried to determine if different uncertainty strategies could be used to choose images that would more effectively train the model.

Chapter 4 shows and discusses the results from the conducted experiments. Additionally, I suggest various future work that can be conducted in the form of experiments to improve the results possibly.

## 5.1 Conclusion

Based on the work and experiment results presented in this thesis, I can conclude the following:

The results from the first experiment support the second hypothesis that using the LIACi data set to train the model incrementally works and most optimally if the model is trained on all the training data for each increment. The trade-off for achieving the same test results (loss and binary accuracy) as the original model is more overfitting for both loss and binary accuracy. If solely the new data were trained on, the results were too unstable and fluctuating.

The second experiment tried to determine how many samples of Monte Carlo dropout would need to stabilize its $\mu$ & $\sigma$. The results showed that with a hundred samples, $\mu$ & $\sigma$ stabilized. Optimizations and more thorough testing could have been done, but it wasn't the focus of the thesis.

The last and most extensive experiment tried to prove the first hypothesis, but it proved overall inconclusive. However, some conclusions still came from the experiments. Although still more than the original model, the models trained on uncertain data always overfitted the least, both with and without a budget. The model trained with uncertainty classified by Monte Carlo dropout overfitted the least. However, I still believe that with the right hyperparameters and strategies, one can arrive at a conclusion to this hypothesis.

The future work section 4.2.4 suggests ways to conduct and improve the experiments.

The code package for this project can be found here `https://github.com/SINTEF/activeLearningForLIACI`

# A

# UI changes and additions

A new 2D map, shown in figure A.1, was created to show the user which labels and in which images the model is uncertain of in the uploaded video. As with the old 2D map, the user can click around in the figure to browse the video.

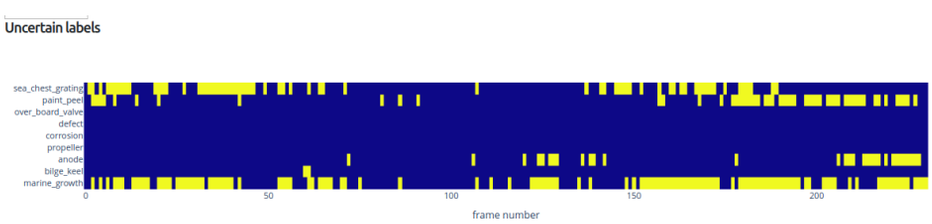



**Figure A.1:** 2D map showing labels on the y-axis and frame numbers on the x-axis. A yellow tag means the model is uncertain

Toggle switches have been added to the web application for each label, as seen in figure A.2. When the user clicks on a specific frame on the 2D map, the web application will jump to that frame and fetch the annotations the model has made on the specific image. The model annotations are used to set a starting point for the user. Hopefully, they won't have to toggle too many switches to correct the annotations. Users can hit the submit button when they are satisfied with the annotations.

The submit button takes the current frame number, turns the toggle switches

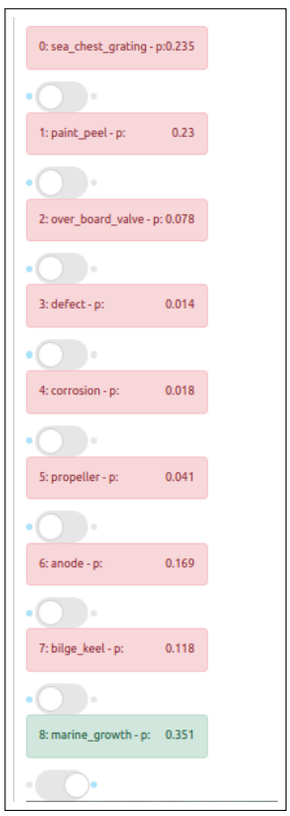



**Figure A.2:** Switches added to the web application UI so the user can annotate images.

into a list of boolean values, and sends the two to the server. The frame number is used to fetch the actual frame of the video, and it's stored as an image on the server.

## A.1 Storing the image on the server

When an image is submitted and stored on the server, the image is given a unique name and stored in a folder with other annotated images. The annotations are stored in a separate file using the common and easy-to-implement JSON format. Each (unique) image name is used as a key in a dictionary to access the multi-hot vector (annotations) that belong to the image. Figure A.3 shows roughly how the file system is designed.

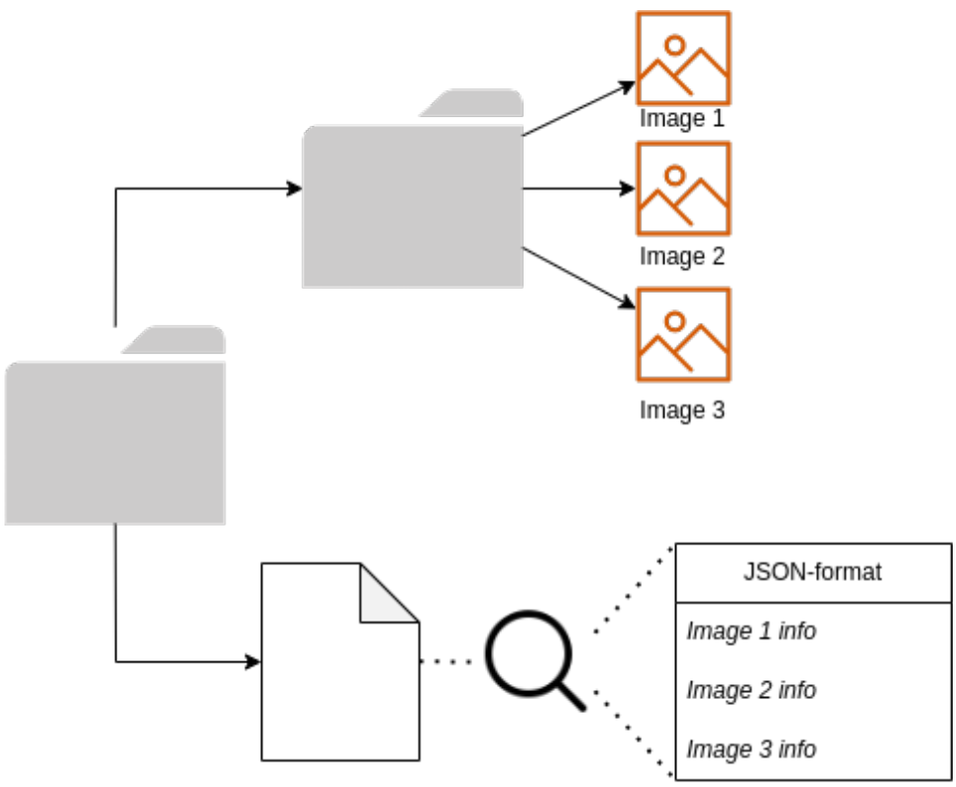



**Figure A.3:** Shows how images submitted to the server are stored together with their annotated labels

## A.2 Live incremental learning

A feature was also added to the app that lets the user incrementally train by using the strategy that was found most fitting in 4, training on the new and the old training data. The old training data is the LIACi data set, and the new training data is the images the user annotates. The test set is still the original test data from the LIACi data set. No feature allows for saving the updated model as of now.

### A.2.1 Web application interface

The web application is a further extension of the web application created in the capstone project Riise [9].

A couple of features have been added to accommodate the problem statement of the master thesis. However, as with the previous project (which this project is an offspring of), the design has not been a priority at all. I could have made

the website pretty and pleasing to use, but too much time would have been put into that when the website's purpose is solely to let the user use and interact with the "scientific" work I've done related to this project. So while the website is not aesthetically pleasing, it is very functional (which is the aim).

### A.2.2 Changes done to the web application

In the previous project Riise [9], the web application had some limitations, mainly related to too much data loaded into memory simultaneously.

The first case occurred after the video was uploaded to the server. OpenCV would extract every frame from the video and gather them in a sequential multi-dimensional array, scaling linearly in size with the video length. This was done to take advantage of TensorFlow scalable large-batch processing, although I didn't have a GPU to utilize this feature entirely.

Naively, a straightforward approach to solving this memory problem would be to predict every single individual frame, but as Keras's documentation [6] states, it's much more scalable to predict on batches.

The solution was to add a user-configurable constant to let the user choose a value that suits their hardware. The constant dictates how large each prediction batch will be, i.e., how many frames will be kept in memory at a time. A compromise between predicting every image simultaneously and keeping memory usage low.

$$mem\ size = channels \times height \times width \times batch\ size \times pixel\ size \tag{A.1}$$

The optimal batch size value wasn't found as it's not the focus of this master's thesis, and it's hardware dependent and won't be relevant for whichever hardware this will later be run on. Since frames are extracted independently, all video compression between frames is lost. The memory usage of a batch can, therefore, easily be calculated with eq. A.1, where each pixel is 1 byte.

No other features have been directly removed, so the site allows video browsing.

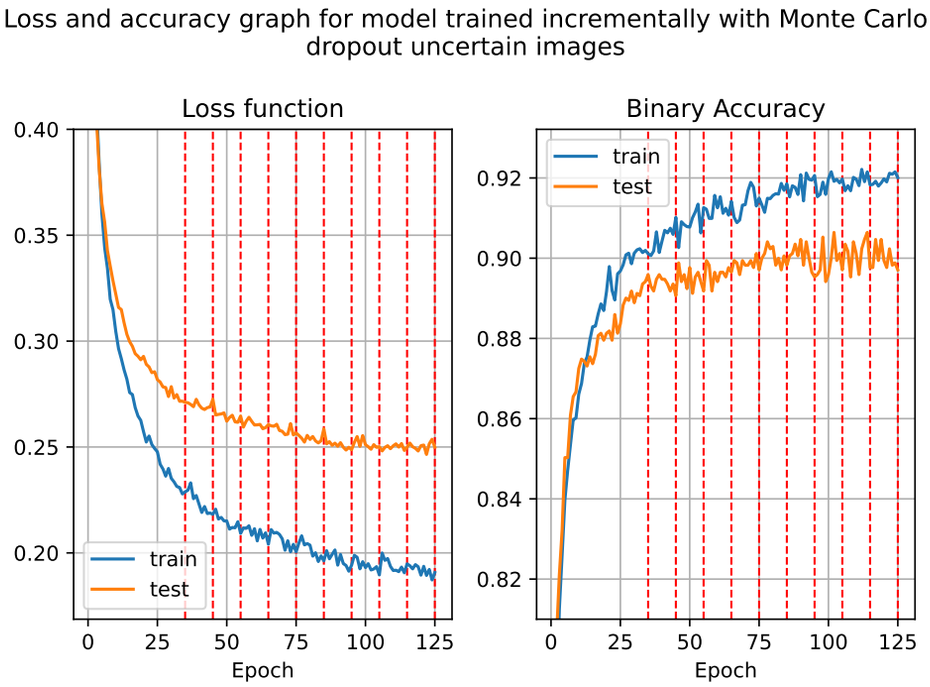
# B

# Incremental training with uncertainty

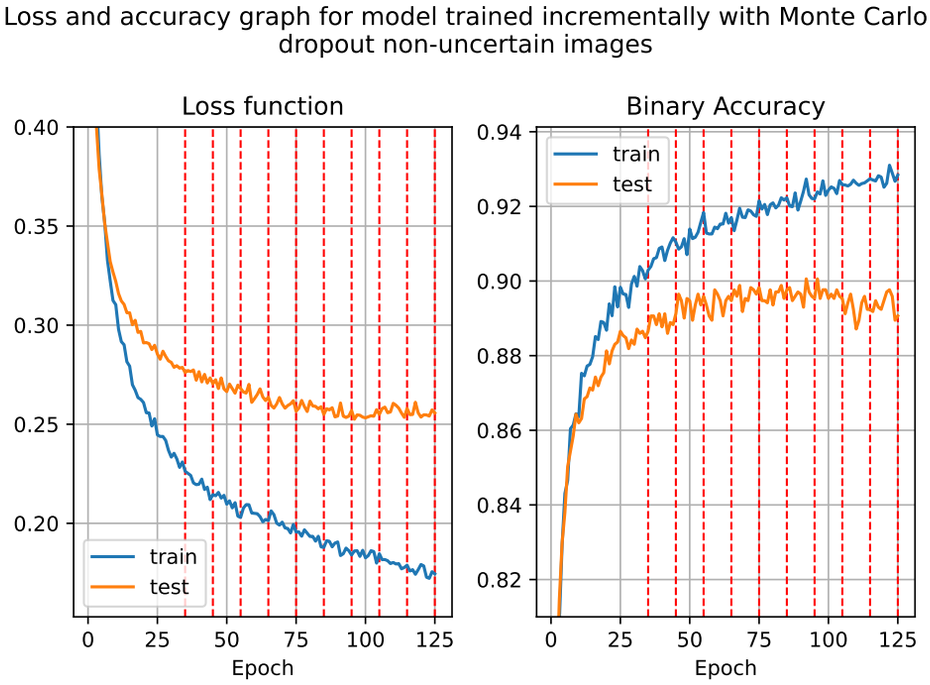
The four figures B.1a, B.1b, B.2a B.2b show the individual training summary for the four different models created in the previous chapter, as seen in table 3.1. Figures B.3a, B.3b, B.4a B.4b show the same, but for models trained with a budget of 35 images per batch. The dashed vertical red lines separate each image batch. The fraction of original epochs used is $\frac{3}{10}$.

| | **Threshold interval** | **Monte Carlo dropout** |
|---|---|---|
| **Uncertain** | Model 1 | Model 2 |
| **Non-uncertain** | Model 3 | Model 4 |

**Table B.1:** The four variations of models trained.

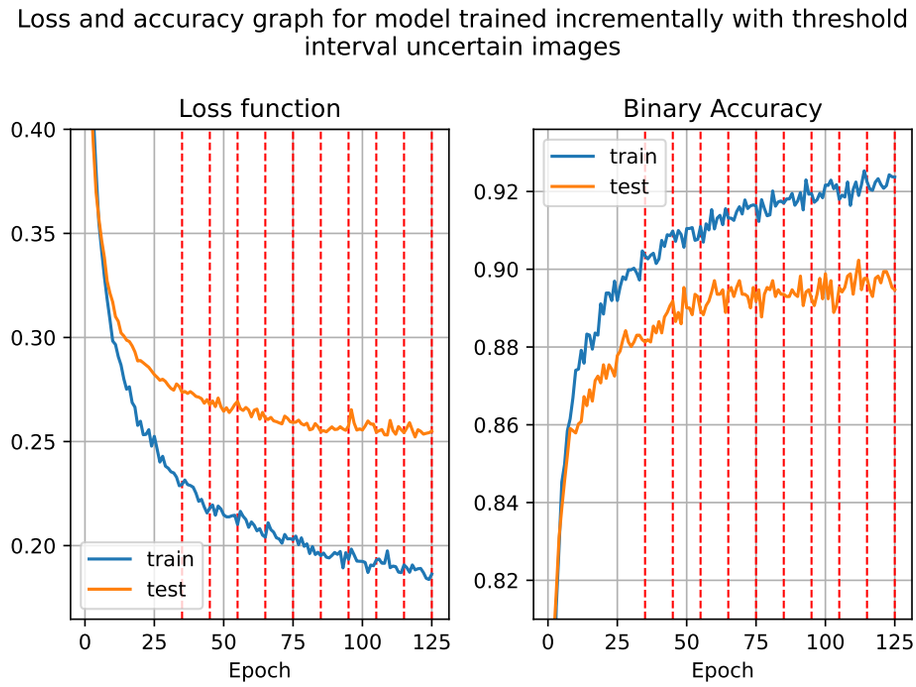



**(a)** Training summary for loss and binary accuracy for the model. For each red vertical dashed line, 100 images are evaluated using Monte Carlo dropout to find the ucertain ones.

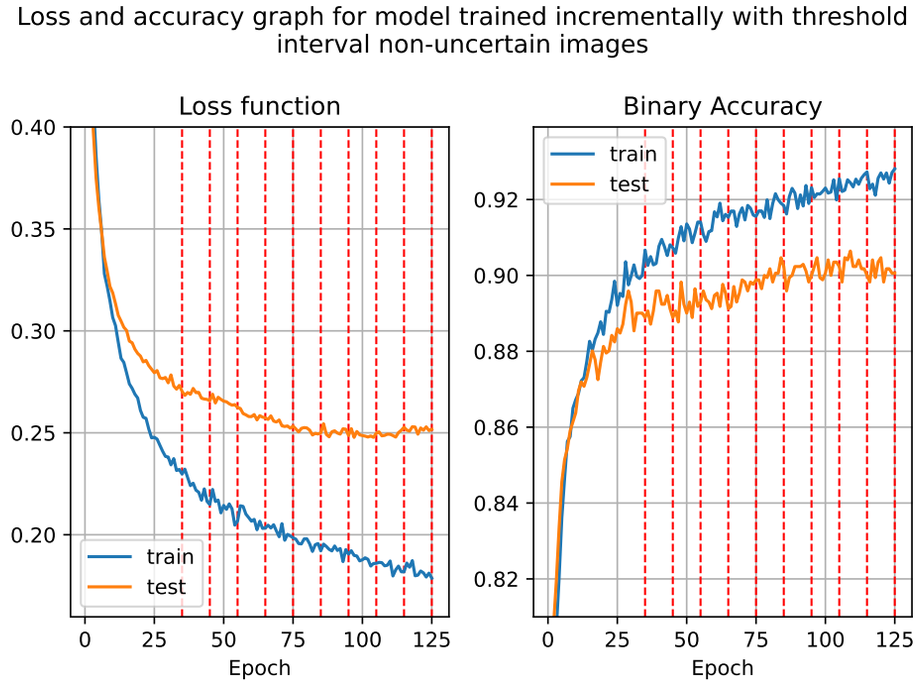



**(b)** Training summary for loss and binary accuracy for the model. For each red vertical dashed line, 100 images are evaluated using Monte Carlo dropout to find the non-uncertain ones.

**Figure B.1:** Training summary for loss and binary accuracy for the models. For each red vertical dashed line, 100 images are evaluated using Monte Carlo dropout to find the uncertain/non-uncertain ones, respectively. The images classified as uncertain/non-uncertain is then added to the training data set. The model then trains on the whole training data set for 30% of the original 35 training epochs.

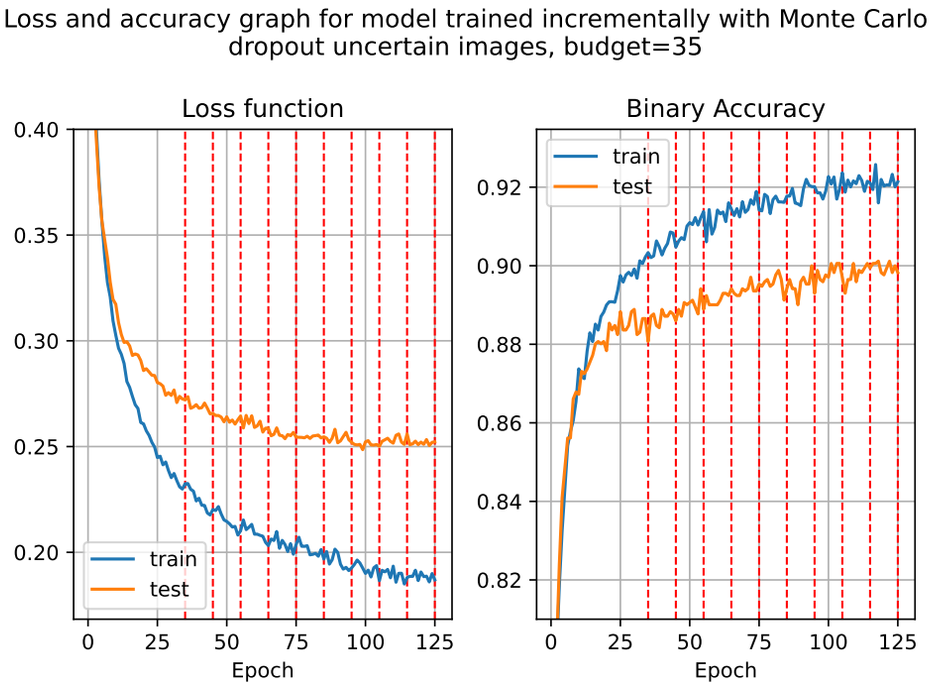



**(a)** Training summary for loss and binary accuracy for the model. For each red vertical dashed line, 100 images are evaluated using "Threshold interval" to find the ucertain ones.

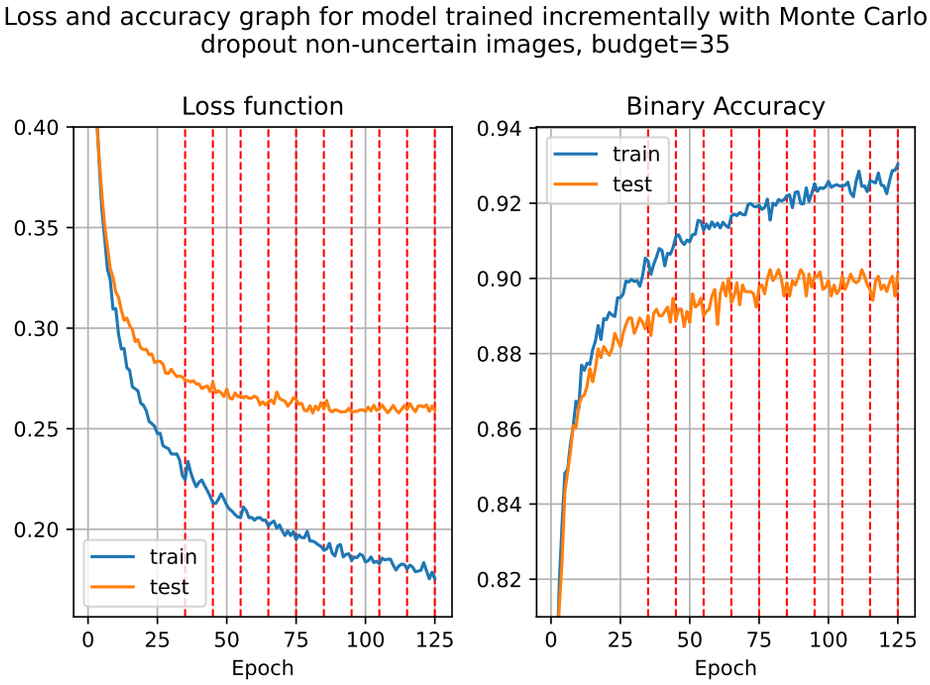



**(b)** Training summary for loss and binary accuracy for the model. For each red vertical dashed line, 100 images are evaluated using "Threshold interval" to find the non-uncertain ones.

**Figure B.2:** Training summary for loss and binary accuracy for the models. For each red vertical dashed line, 100 images are evaluated using "Threshold interval" to find the uncertain/non-uncertain ones, respectively. The images classified as uncertain/non-uncertain is then added to the training data set. The model then trains on the whole training data set for 30% of the original 35 training epochs.

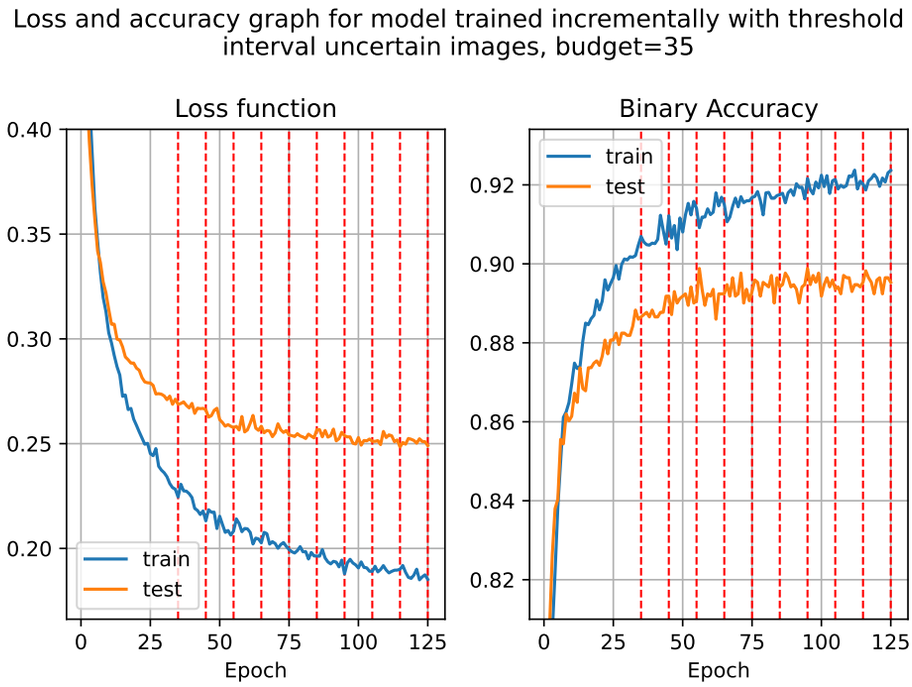



**(a)** Training summary for loss and binary accuracy for the model. For each red vertical dashed line, 100 images are evaluated using Monte Carlo dropout to find the ucertain ones.

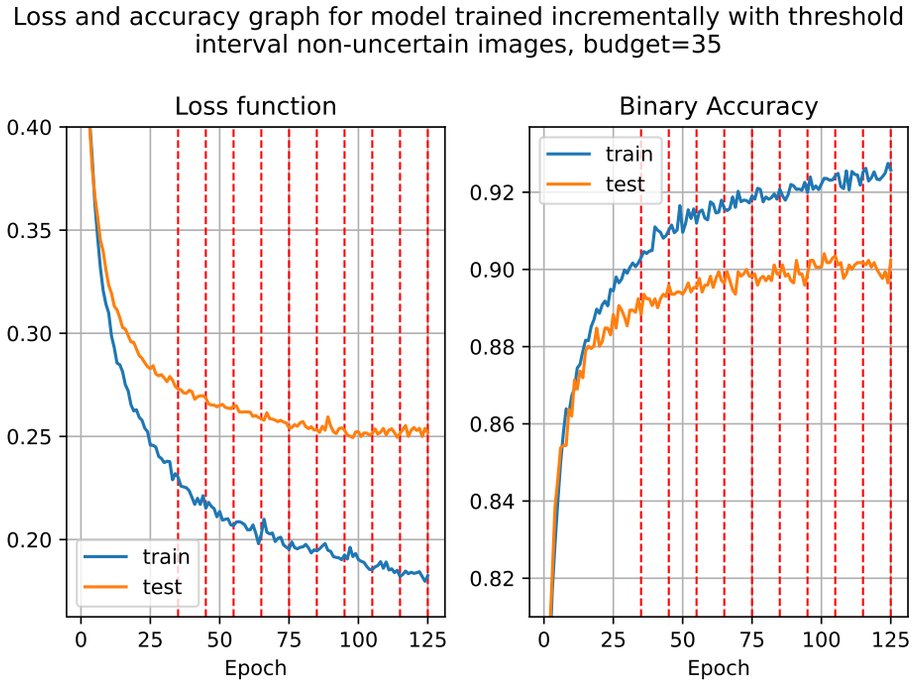



**(b)** Training summary for loss and binary accuracy for the model. For each red vertical dashed line, 100 images are evaluated using Monte Carlo dropout to find the non-uncertain ones.

**Figure B.3:** Training summary for loss and binary accuracy for the models. For each red vertical dashed line, 100 images are evaluated using Monte Carlo dropout to find the uncertain/non-uncertain ones, respectively. The images classified as uncertain/non-uncertain is then added to the training data set. The model then trains on the whole training data set for 30% of the original 35 training epochs.




**(a)** Training summary for loss and binary accuracy for the model. For each red vertical dashed line, 100 images are evaluated using "Threshold interval" to find the ucertain ones.




**(b)** Training summary for loss and binary accuracy for the model. For each red vertical dashed line, 100 images are evaluated using "Threshold interval" to find the non-uncertain ones.

**Figure B.4:** Training summary for loss and binary accuracy for the models. For each red vertical dashed line, 100 images are evaluated using "Threshold interval" to find the uncertain/non-uncertain ones, respectively. The images classified as uncertain/non-uncertain is then added to the training data set. The model then trains on the whole training data set for 30% of the original 35 training epochs.

# C

# A closer look at F1 results

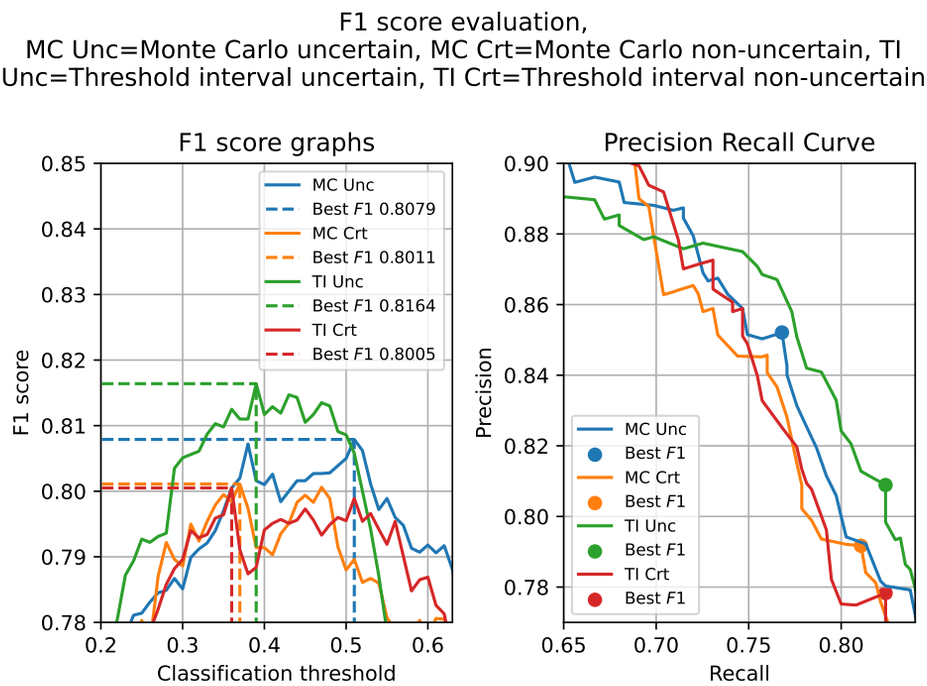



**Figure C.1:** A closer, more detailed look at figure 4.24, from chapter 4

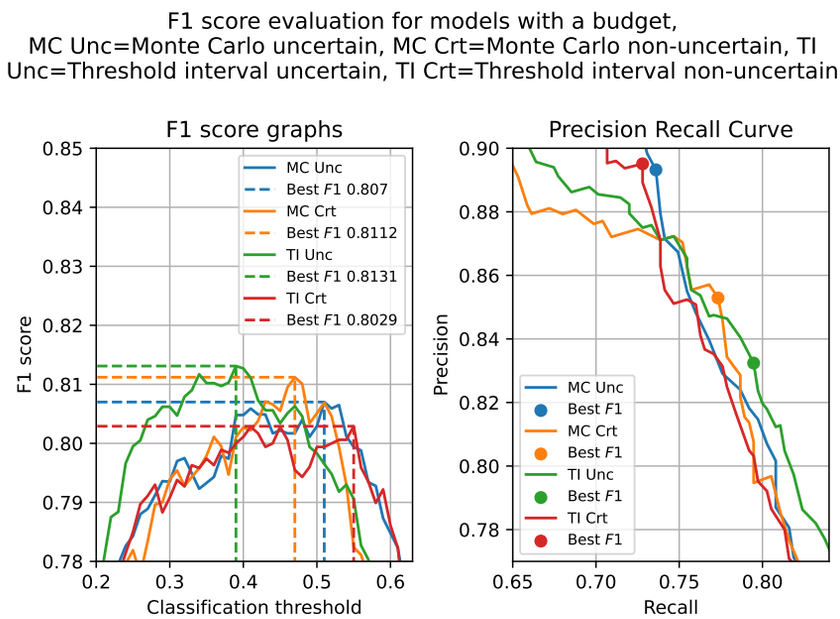



**Figure C.2:** A closer, more detailed look at figure 4.25, from chapter 4

# Bibliography


[1] Jia Deng et al. “ImageNet: A large-scale hierarchical image database.” In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009 IEEE Conference on Computer Vision and Pattern Recognition. ISSN: 1063-6919. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[2] Yarin Gal and Zoubin Ghahramani. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. arXiv:1506.02142 [cs, stat]. Oct. 2016. URL: http://arxiv.org/abs/1506.02142 (visited on 03/30/2023).

[3] Yarin Gal and Zoubin Ghahramani. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. arXiv:1506.02142 [cs, stat]. Oct. 2016. URL: http://arxiv.org/abs/1506.02142 (visited on 03/30/2023).

[4] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. arXiv:1704.04861 [cs]. Apr. 2017. DOI: 10.48550/arXiv.1704.04861. URL: http://arxiv.org/abs/1704.04861 (visited on 10/13/2022).

[5] Forrest N. Iandola et al. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*. Nov. 4, 2016. arXiv: 1602.07360 [cs]. URL: http://arxiv.org/abs/1602.07360 (visited on 11/25/2022).

[6] *Keras FAQ*. https://keras.io/getting_started/faq/#whats-the-difference-between-model-methods-predict-and-call. (Accessed on 03/21/2023).

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks.” In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (visited on 11/25/2022).

[8] Zachary Chase Lipton, Charles Elkan, and Balakrishnan Narayanaswamy. *Thresholding Classifiers to Maximize F1 Score*. May 13, 2014. arXiv: 1402.1892 [cs, stat]. URL: http://arxiv.org/abs/1402.1892 (visited on 12/12/2022).

[9] Elias Estefano Gutierrez Riise. “Underwater Ship Inspection Using Multi-label Image Classification.” Capstone project, University of Tromsø. 2022.

[10] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Apr. 10, 2015. DOI: 10.48550/arXiv.1409.1556. arXiv: 1409.1556 [cs]. URL: http://arxiv.org/abs/1409.1556 (visited on 11/24/2022).

[11] Christian Szegedy et al. "Going deeper with convolutions." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Boston, MA, USA: IEEE, June 2015, pp. 1–9. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298594. URL: http://ieeexplore.ieee.org/document/7298594/ (visited on 11/25/2022).

[12] Sheng Wang, Ashwin Raju, and Junzhou Huang. "Deep learning based multi-label classification for surgical tool presence detection in laparoscopic videos." In: *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*. ISSN: 1945-8452. Apr. 2017, pp. 620–623. DOI: 10.1109/ISBI.2017.7950597.

[13] Maryna Waszak et al. "Semantic Segmentation in Underwater Ship Inspections: Benchmark and Dataset." In: *IEEE Journal of Oceanic Engineering* (in press). DOI: 10.1109/JOE.2022.3219129.