# Project 2: Backpropagation and Gradient Descent

INF265 - University of Bergen

Vebjørn Sæten Skre Martin Flo Øfstaas

Spring 2024

***Division of labour***: We worked together side by side through the whole project.

## 1 Explanation of our approach and design choices

### 1.1 Approach

Our approach for this task project was to break down the problems into smaller problems. Then after finding out what problems needed to be solved, we tackled each of the problems head on. In part one of the project the general goal was to train 3 appropriate models for an object localization task, evaluate the models and choose which models performed best. A major importance for us was to keep the whole process of selecting the model bias free. This means that we would never touch the test set before actually testing the selected best model. This meant that we also needed an unbiased method to choose the model, which we implemented for both tasks. Although task 2 and 3 had different goals, they can still be seen as the same general task. Thus we followed the same approach for both tasks as far as it was possible.

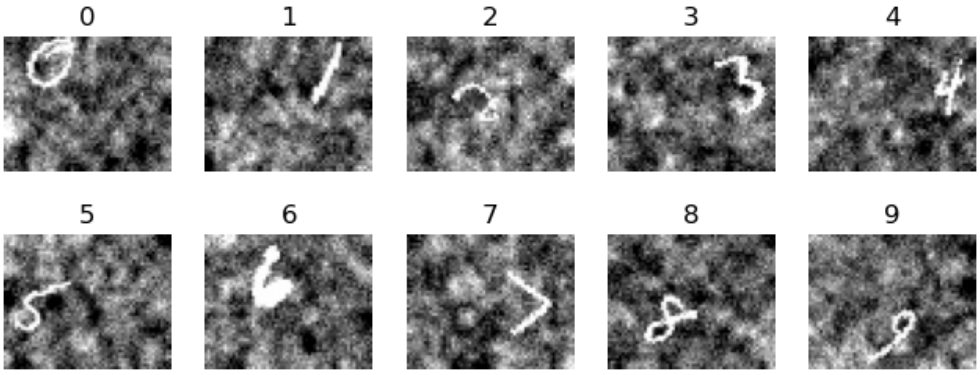
The general process was as follows. We read in the data. We used a preprocessor which we applied for all of the datasets which we generated from the mean and std value of the training set (the appropriate training set for the task). We then defined our models, training loop, loss function and performance measure (we will come back to these in just a minute). We made the loss function and performance function task agnostic, only needing to specify "loc" or "det" in the functions.

For running the models we quickly found out that our pc's was very slow training the models. We found the solution by using the limited time one have for free GPU use on google Colab, which in the end turned out great. Thus all of our models is trained on a NVIDIA V100 GPU. To not have to run everything in one training instance on colab we also stored our models and model performances so that this could be resused without having to rerun the training process. Since this is not mentioned in the project paper, will we not include this in the submission files, but we will gladly give them to you if you want to check them up.

For evaluating the model we focused on two things. The loss of the model and the validation accuracy. Our approach, which was equal on both of the tasks was to compute keep a track of the highest validation accuracy in the training process, and choose the epoch where the validation accuracy was the highest. Thus we ensured to choose a model wasn't overfitted. We also kept the track of the training loss and validation loss (plots that we will show later on), to further see how the model performed in terms of overfitting.

To verify that our progress through the project was correct we used our plot functions to verify on the training set that our model was somewhat correct. Again, we never used the test set before actually looking at the test set at the end of the model selection process.

For us this project was designed to be as general as possible. Thus we tried making all of our functions to work more or less as a library one could use for other tasks or run on different machines. Although we had at some times to be task specific, like specifying the links to the path files, we are satisfied with our approach. Below is the plot of the initial images when we first saw them in the beginning of the project.




One important thing that we noted about the data is that the images is perturbed, and thus some of the images would be hard even for us to detect. We were unsure of how our models would work, but it turns out that our models was quite good at detection even perturbed images. We will get back to this in the result page, but was important for us to mention this here, because if the image is transformed into the unreckognizeble, whats the value of our model to detect it?

### 1.2 Performance Measure

Both task in this assignment used the same performance measure, however in section 2 some alterations of the labels and predictions to make it work. We are aware that for part 2, object detection, the performance measure is not ideal as it does not use mean average performance (MAP) and instead uses average accuracy across grid cells, making it grid dependent. In the future we would like to implement a MAP performance to this part.

The performance function took in a model, a dataloader and a parameter specifying which assignment we are working with. If it was the second assignment we changed the predictions and labels of the data so that instead of a picture with 6 squares (as we used a 2x3 grid) it was simply changed to 6 pictures where each picture was one grid cell. This meant we could precede with our regular performance function with the only change being that the batch size was 6 times as big.

The rest of the performance measure followed the logic as stated in the assignment paper with accuracy being when the model classified both the detection and class label correctly if there was a number in the picture and if the model classified correctly when there was not a number in the picture. The boundary box accuracy was only calculated when it both thought it was a box in the picture, and when it was actually a box in the picture. We used pytorch's box_iou function for this. However this function compared every box in the picture with every other returning a square matrix. We solved this by taking the mean of the diagonal of the matrix. The reason we only calculated the box accuracy if the model thought it was a box there and it actually was a box there was that we only wanted to test the models box-drawing ability, not whether i failed on the detection thus getting a 0 box score. In the end we returned the box accuracy, detection & label accuracy, plus the mean of the two, that we use to select the best model.

### 1.3 Loss function

The loss function was, similarly to the performance measure, made to work for both part 1 and 2 of the project. The data transformation was done identically as in the performance measure, however we aslo changed the calculation of class labeling loss between the two assignments as there was only two classes in part two. To calculate the performance measure we simply followed the logic depicted in the assignment paper, calculating the binary cross entropy loss with logistic loss (BCEWithLogitsLoss) using the BCEWithLogitsLoss function from the torch module. We used mean squared error to calculate the box loss, and we used cross entropy for the classs label loss in part 1, when there was multiple classes and BCEWithLogitsLoss in part 2 when there was only two classes. We only calculated the box loss and class labels loss when there actually was a number present in the picture. We accomplished this by using a mask on the data with the boolean values of true detection.

### 1.4 Part 1 of the project: Object localization

The first task in the project was to load in the datasets. Here we applied a preprocessor to the images which normalized the data. We used the mean and standard deviation from the training set to do this. Then we defined the networks to be trained. Here we went for to more 3 different models with three different levels of complexity. When storing these models these the total size between them was:

1. SkreNet : 147mb
2. FloNet : 4.2mb
3. SimpleNet : 864kb

#### 1.4.1 SkreNet

SkreNet was designed with 4 convolutional layers with three fully connected layers in the end. For the convolutional layers we used a $3x3$ kernel with padding = 1 and stride = 1. We used maxpooling between the convolutional layers in layer 2 and 3, and 3 and 4. As for the activation function we used ReLu in all the layers.

The reason for why we wanted to have three fully connected layers in the end, instead of just one, was because we did not want to reduce the number of parameters too quickly before the output layer. As we will see in the performance section, the design choice of SkreNet worked well.

#### 1.4.2 Simplenet

SimpleNet was designed in a similar manner to SkreNet with 4 convolutional layers with $3x3$ kernels. However, this simpler model had a lot fewer channels and only one fully connected layer. It also had one maxpooling layer less than SkreNet as it did not have as many channels and thus was not so computationally expensive as it could have been. The Idea behind SimpleNet was to see how a relatively simple model with fewer parameters would handle the problem of object localization. All the activation functions was ReLU.

#### 1.4.3 FloNet

FloNet was our intermediate model in complexity. It had 5 convolutions layers, however it did have one maxpooling layer more than SkreNet to reduce dimensional. It had three maxpooling layers and

one fully connected layer. The reason we chose to only have one fully connected layer in spite of having a lot of parameters was that we wanted to see the effects of mapping thousands of parameters to a small number as 15 to see if it negatively affected performance. To be able to have the third maxpooling layer we used a $3x3$ kernel in the third convolutions layer. All the activation functions was ReLU.

#### 1.4.4 Hyperparameters

The project for this task was very computationally heavy. Thus we had to find a middle ground for how many hyperparameters we would choose, and the complexity of the models. We felt that this project was more focused on the architectures rather than how many hyperparameters we would choose. And since we had limited computational power we had to settle for what we felt the most important contributor for model performance was. Thus we ended up on choosing 3 models with varying model architecture complexity. As we will discuss in the model performance section, we felt satisfied by our results, but we could probably have gotton a little better by also doing hyperparameter testing. That said, we also believe that this wouldn't have been very noticeable compared to our results.

As for the hyperparameters, we ended up on SGD. We chose to use a learning rate for SGD to be lr = 0.05. We trained the models for 60 epochs (in both tasks), but we only used the model when it reached its validation accuracy. We did this implementation instead of early stopping, because then we could see how the model performed in the later stages, without having to use a model that over fitted to the data. For batch sizes we used batch size 64 for both tasks.

### 1.5 Object Localization

#### 1.5.1 SkreNet2

SkreNet2 is similar to SimpleNet2 but it has two convolutions layers between each maxpooling layer, and a lot more channels. We made this to be fairly deep and complex and since the output had so much less parameters than the fully connected layers in part 1, we could afford having more channels. The network had a total of 11 convolutional layers, 4 maxpooling layers and two batch normalizations. All activation functions was ReLU.

#### 1.5.2 SimpleNet2

SimpleNet2 is basically the SkreNet model from part 1 of the assignment with some non square kernels and a few more maxpooling layers to fit out desierd output. The Idea was to see if we could make a model architecture that could fit both problems. In this task we called it simple model as it had a lot less parameters than the original SkreNet due to the extra maxpooling layers. It had 4 maxpooling layers, 6 convolutional layers, and one batch normalization. All activation functions was ReLU.

#### 1.5.3 FloNet2

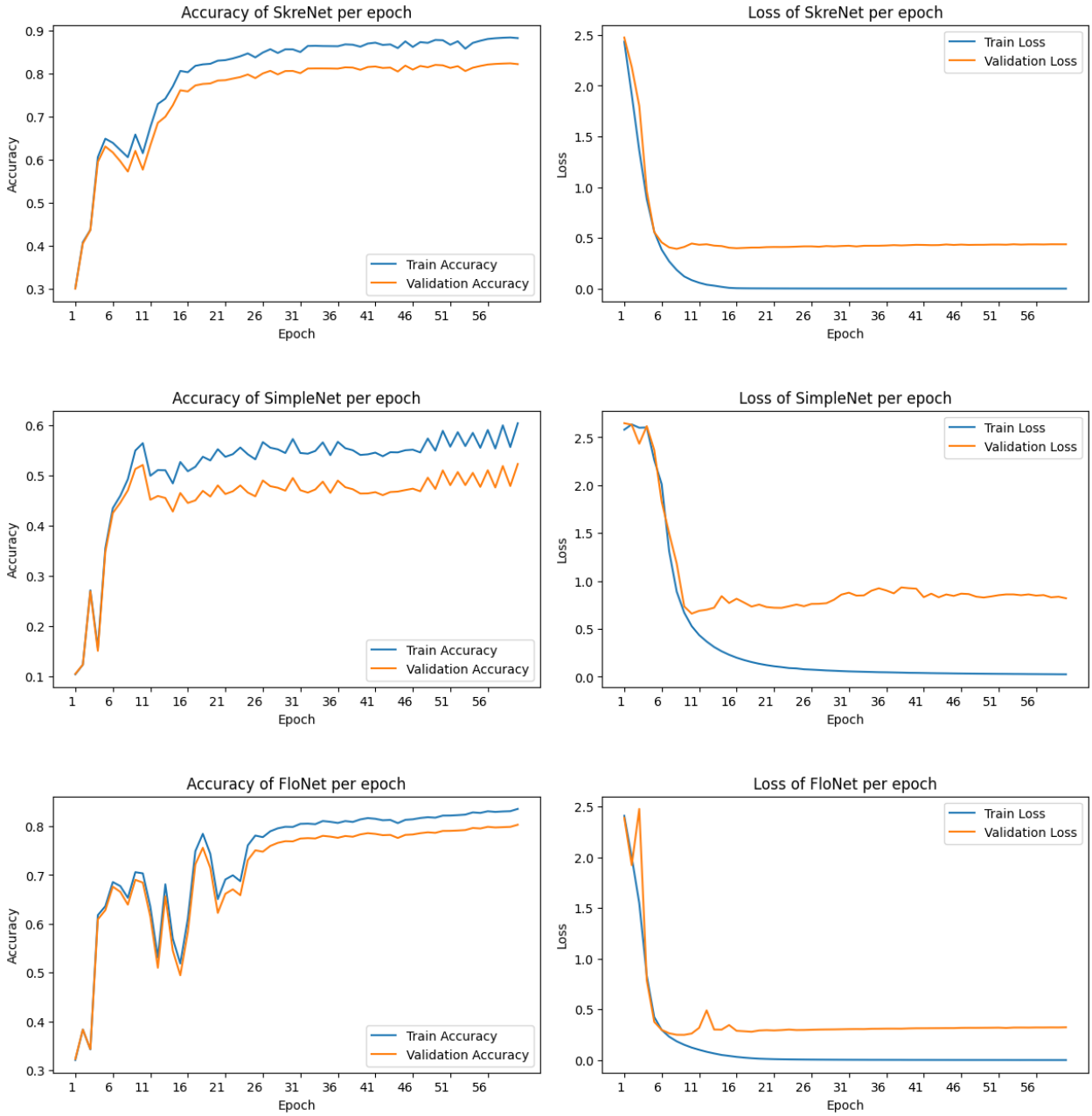
We created FloNet2 in a very disimilar way than the other models. We wanted to try something new, so we had a lot of convolutional layers in a row where i only increased the cannels. Then when we had 100 channels we started reducing the dimensions of the image with uneven kernels, padding=0 and maxpooling layers. This was the only model that used padding=0. It ended up with 11 convolutional layers, 4 maxpooling layers, 2 batch normalizations.

### 1.5.4 Hyperparameters

For this part we followed the same logic as in the previous section about hyperparameters.

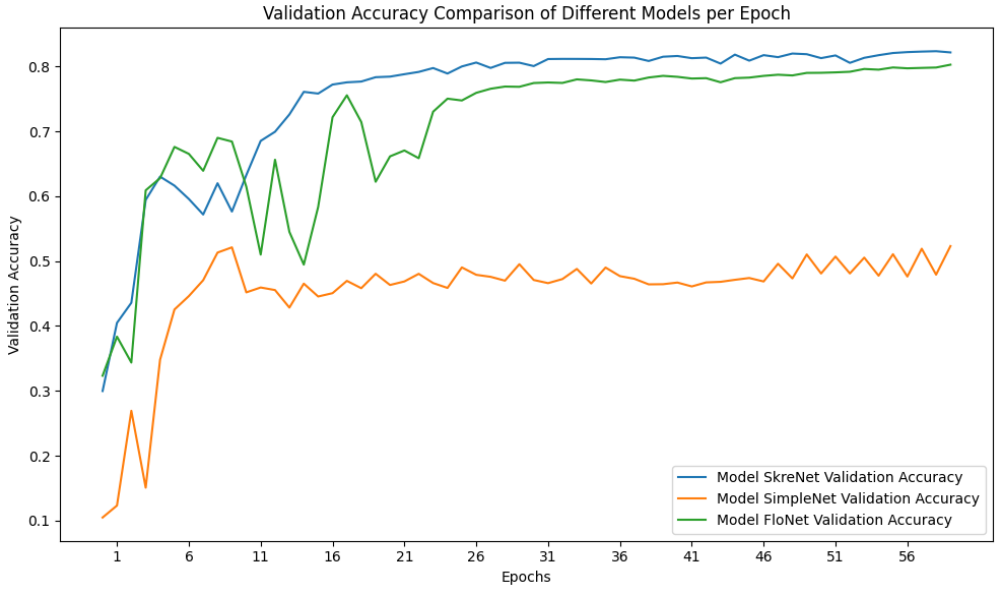
# 2 Plotting & results

### 2.0.1 Plotting for task 1: Object detection




Above is how our model performed over the 60 epochs it was trained on. We can see that all of our models quickly generalizes somewhat to the data. We can see that they start to hit a plateau after around 20 epochs, although SkreNet and Flo net is sightly still learning. We can see this from the best model being chosen at the 58th epoch. This suggest to us that more training alone would

have gotten us slightly better results. It's also interesting to see that Flo net at around 11 to 20 is very unstable is varying results on train and validation accuracy, but after a while picks up a more stable pattern. Looking at the loss it looks like the model starts to overfit at this point, but it quickly stabilizes. We also note that the simple model is performing much worse than the two others, and that it also has an unstable learning curve in terms of accuracy's on train and val. This suggest it us that the simpleness of the model makes it more random, it simply cannot pick up on the pattern in the data.



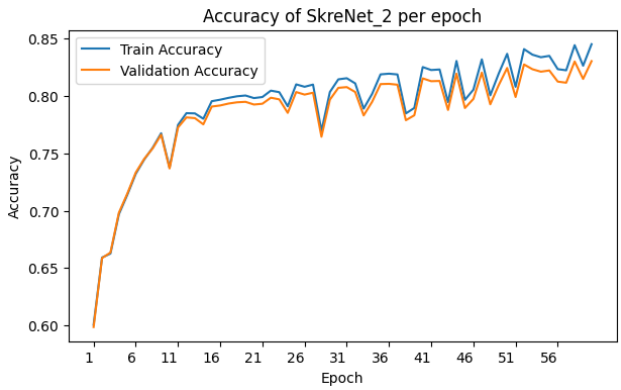

When comparing all of the models validation accuracies over each other we get a better representation of them compared to each other. As we noted in the earlier paragraph, FloNet performed unstable in the beginning, but we can see that all of the models did this to some degree. One interesting thing would be til continue the training on SkreNet and FloNet and see who would perform after 100 or 200 epochs and if they would eventually start to overfit. As it looks now, FloNet was creeping up on SkreNet in terms of validation accuracy, and the model also had lower loss. Although we can't say for certain, we suggest a more suited model would be something in between the two. While SkreNet might be too complex for the data, FloNet could be too simple. But this is all just speculation, I guess we have to try another time;)

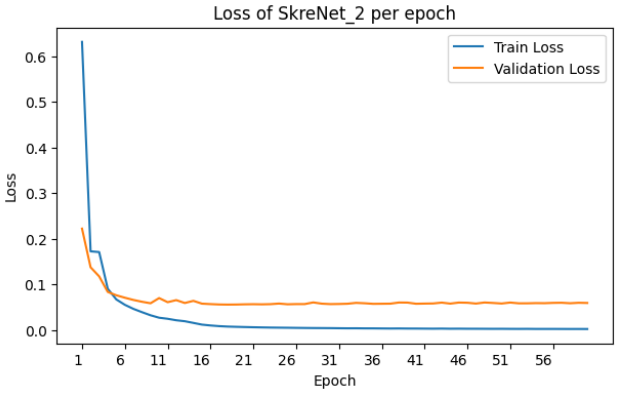
### 2.0.2 The best model on test data, part 1

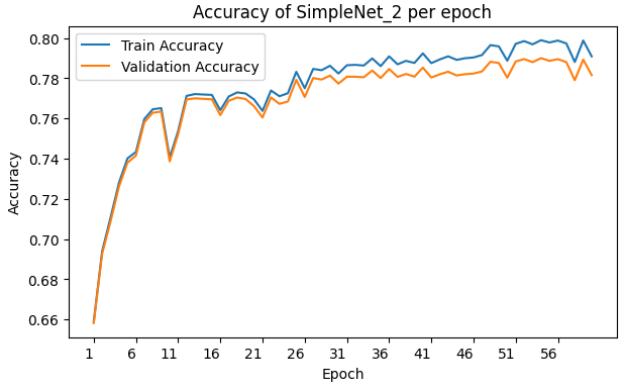
We automatically selected the best model base on the highest validation accuracy and tested it on the test dataset. The best model was SkreNet and it achieved a mean accuracy of 82,65%. The detection and class labeling accuracy was 92,63% and the box accuracy was 72,68%. As we anticipated the model did better on detection and labeling than it did on box drawing.

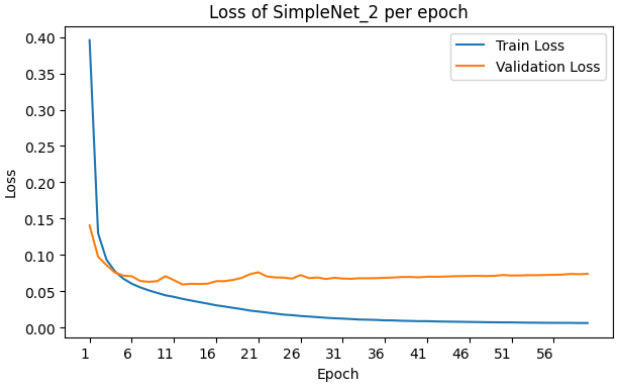
```
-------------Performance of the best model-----------------

Model: SkreNet at epoch 58
Validation average accuracy: 0.8237028121948242

-------------ACCURACIES ON TEST DATA-----------------

Detection & class labelig:
92.6364%

Box accuracy:
72.6757%

Average accuracy on test data:
82.656%
```

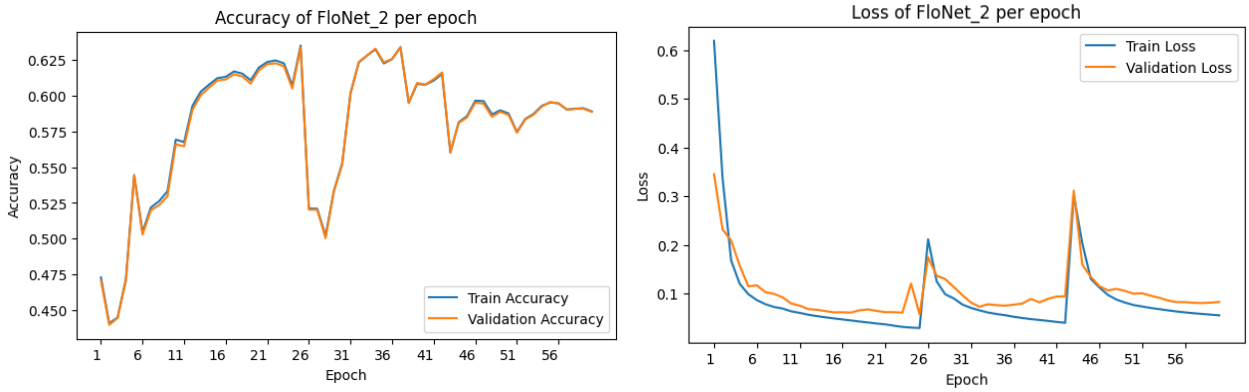
## 2.1 Object detection



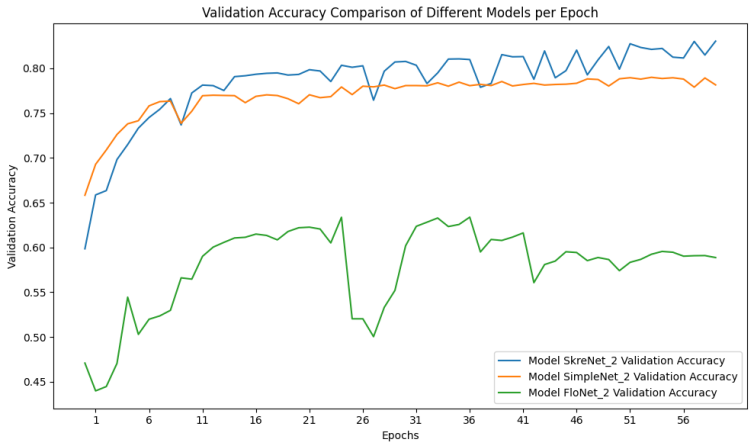






When evaluation the performance of our object detection models we got some unexpected results. Here our most complex model, FloNet2 got really bad results and never got itself to diverge towards a satisfying result. One could try to train the model for longer, but we might think that we created a too complex model that wasn't able to fit the data well. The other models performed much better. One interesting note is that while we in the task 1 was able to predict the correct label quite well, we did the opposite in this task. In the figure below we can see the results. Although the label prediction is better than box prediction (IoU), they are much closer than than in task 1. While we do not have any definite answer to this, we have thought that it could be that it is easier for the model to predict boxes when the "image" is smaller and boxes "bigger" due to how the prediction is done on the grid cells.



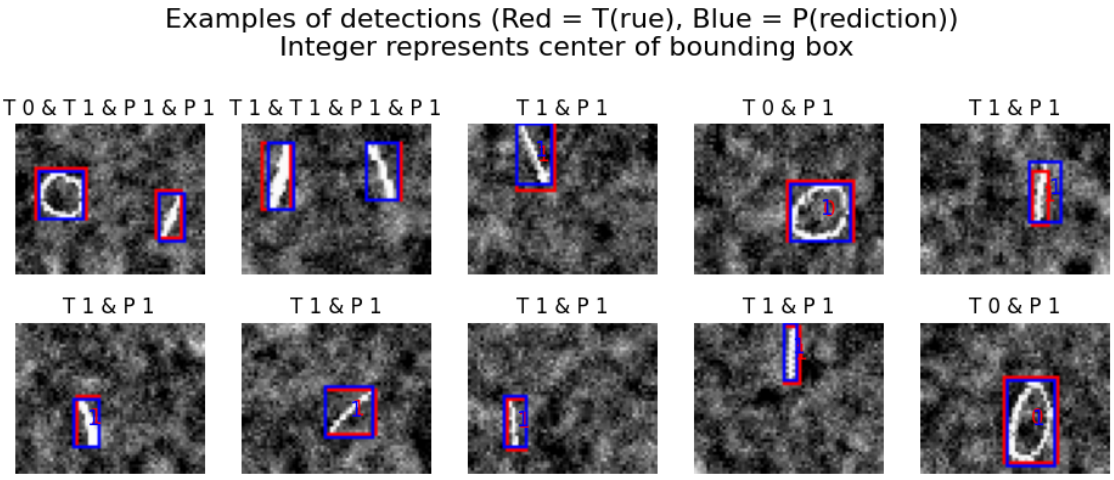

When comparing the three models side by side we got quite surprised by the sudden dip in the validation loss for FloNet2. We suspect that this model was too complex with too rapid max pooling at the end. SimpleNet2 is basically SkreNet from the last assignment, just with more maxpooling layers and non square kernels to match the expected output. One last interesting thing with this plot is that we can see a slight increase in the validation accuracy for SkreNet at the end later epochs. We also got the best epoch to be 59, which suggest that the network could have been trained for longer.

In Figure 5 we can see how our best model actually performs on the testing data. We can see that it predicts a 1 on a very obvious 0 which surprised us. On the hindsight we are very satisfied with the predicted boxes. Its difficult for us to say why the prediction for labels got worse, but we suggest that it could be due to the model finding lower loss with focusing more on boxes and more or less forgetting to also focus label prediction.

### 2.1.1 The best model on test data, part 2

The best model on was picked automatically by checking the validation accuracy of the different models. The best model on validation accuracy was SkreNet2, and it got a average score at 82,88%, the detection and class labeling accuracy was 87,72% and the box accuracy was 78,1%. We were very surprised by the low detection and class labeling accuracy. When comparing to task one we thought this should be even higher as it should be easier to detect only two class labels instead of 10. We can only speculate why this was the case, but we are happy with the overall accuracy.

```
-------------Performance of the best model-----------------

Model: SkreNet_2 at epoch 59
Validation average accuracy: 0.8303920030593872

-------------ACCURACIES ON TEST DATA-----------------

Detection & class labelig:
87.6732%

Box accuracy:
78.0962%

Average accuracy on test data:
82.8847%
```


Examples of detections (Red = T(rue), Blue = P(rediction))
Integer represents center of bounding box
T 0 & T 1 & P 1 & P 1
T 1 & T 1 & P 1 & P 1
T 1 & P 1
T 0 & P 1
T 1 & P 1
T 1 & P 1
T 1 & P 1
T 1 & P 1
T 1 & P 1
T 0 & P 1

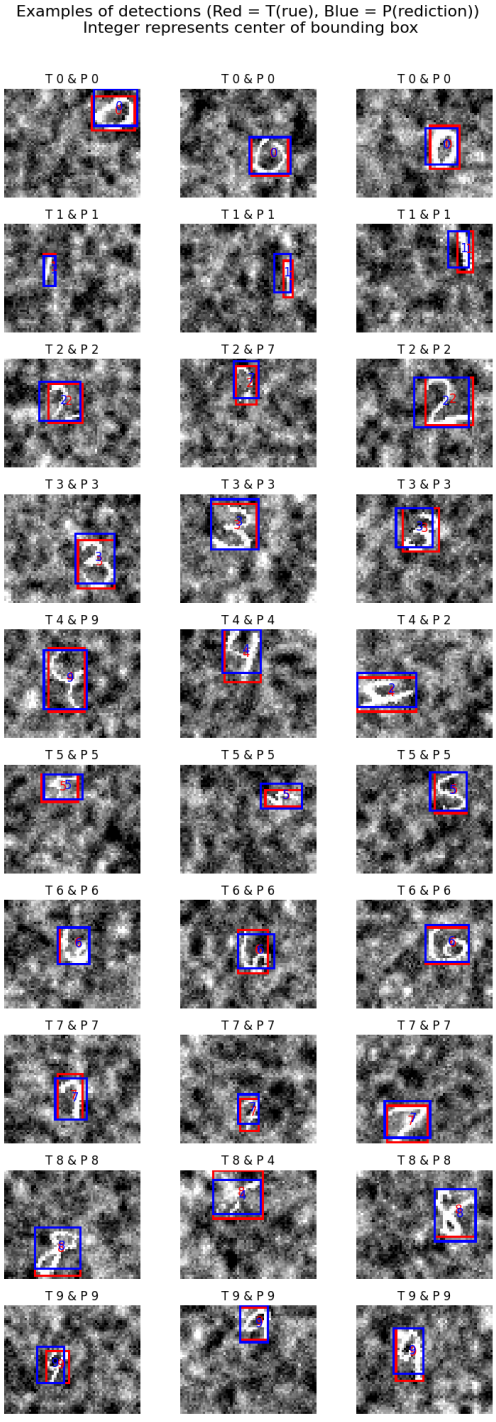

Examples of detections (Red = T(rue), Blue = P(rediction))
Integer represents center of bounding box
T 0 & P 0
T 0 & P 0
T 0 & P 0
T 1 & P 1
T 1 & P 1
T 1 & P 1
T 2 & P 2
T 2 & P 7
T 2 & P 2
T 3 & P 3
T 3 & P 3
T 3 & P 3
T 4 & P 9
T 4 & P 4
T 4 & P 2
T 5 & P 5
T 5 & P 5
T 5 & P 5
T 6 & P 6
T 6 & P 6
T 6 & P 6
T 7 & P 7
T 7 & P 7
T 7 & P 7
T 8 & P 8
T 8 & P 4
T 8 & P 8
T 9 & P 9
T 9 & P 9
T 9 & P 9

Figure 1: