Project 1.1 - Hotdog or not hotdog

Authors: *Tobias Konradsen (s144077), Helena Hansen (s153178) and Christian Dandanell Glissov (s146996)*

Architecture

The final Convolutional Neural Network model use 4 convolution layers utilizing a kernel size of 3, each followed by a ReLU and max pooling layer. A kernel size of 3 is used, as in Simonyan and Zisserman, 2014 it was observed that stacking convolution layers with 3x3 kernels allows for a larger effective receptive field, while utilizing less parameters.

Pooling layers are utilized to reduce the spatial size of the representation by downsampling, which results in a lower number of parameters, memory footprint and computation. To prevent the amount of parameters to become too low, we increase the number of channels when max-pooling. There is only a single convolutional layer between each pooling layer, as by experimentation were no noticeable increase in accuracy by including more or using max pool less often. We did see a reduction in performance when decreasing the amount of convolutional layers. At the end 3 linear layers are used, due to the high amount of activations from the final convolutional layer (8096) each layer gradually reduces the amount of parameters to get the desired output of 2 classes.

Optimizers

The convolutional neural network was tested with the optimizers: SGD, SGD with momentum, ADAM, ADAM with fixed weight-decay, ADAM with a decreasing weight-decay and annealing inspired by Loshchilov and Hutter, 2017, which can be seen in Figure 1. The best of these optimizers were found to be RMS and Adam and its derivatives. RMSprop does seem to be at the higher end, but based on the experiments there does not seem to be a significant difference between Adam optimizers and the RMSprop.

Data augmentation

The training set is small, only consisting of 2047 images. To add additional samples we chose to use two types of affine transformations, a horizontal flip and cropping of the images of limited scale. It is important to adhere to the human representation, e.g. we should be careful to not scale too much, as some images will no longer look like a hotdog, but be targeted as it. More augmentations could have been implemented, such as rotations, brightness, crop and pad and shearing, but this was omitted.

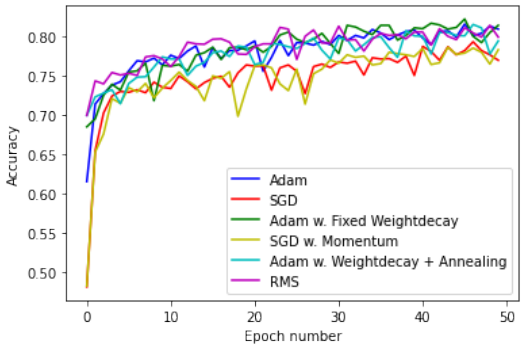

Figure 1: Accuracy of all tested optimizers

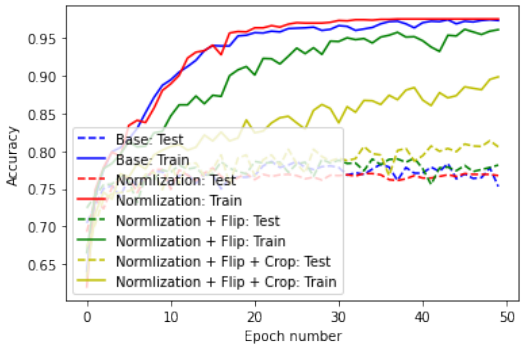

Figure 2: Accuracy depending on Data Augmentation

From Figure 2 it is seen that there does not seem to be a significant difference in test performance, but the accuracy of the training is a lot higher when not using augmentations, this indicates most of the information has been captured by the model and we risk overfitting.

Batch normalization and dropout

Our CNN makes use of one batch normalization after its first convolutional layer, which in theory should allow for faster convergence and more stability. In order to test whether this had any effect,

we modified the network, so could see the difference between the base case, no batch normalization and batch normalization for all layers, while also looking at the effect from dropout. Based on results from 20 epochs the batchnorm did not seem to provide any significant improvement to the accuracy, but there does seem to be less fluctuation in the accuracy, but with 20 epochs it is difficult to conclude anything. Default dropout did not improve the model, hence not included in the architecture.

Performance

The performance of the network reached an accuracy of 82.2% when trained on 150 epochs. From Figure 3 the accuracy and loss can be seen of the training and validation set. It is seen from the loss that the model overfits. A less complex model was implemented to see if complexity of the model is the culprit, but overfitting persisted with the accuracy being slightly less, indicating that more data or regularization is required. The model overfitting will cause a gap between the confidence of the model outputs given by the Softmax probabilities and the classifications, meaning the outputs are not very reliable, Guo et al., 2017. Another approach is to do early-stopping as regularization, but this was not done.

The model has slightly more false-positives (the data sets are balanced) as seen in in the confusion matrix, Figure 3, this means the the neural network is biased to one subgroup of the data set, perhaps due to some excessive scaling in the transformations, making hotdog images look like "not hotdogs". Another reason might be that there is a lot of different hotdogs, making it a tougher task for the model.

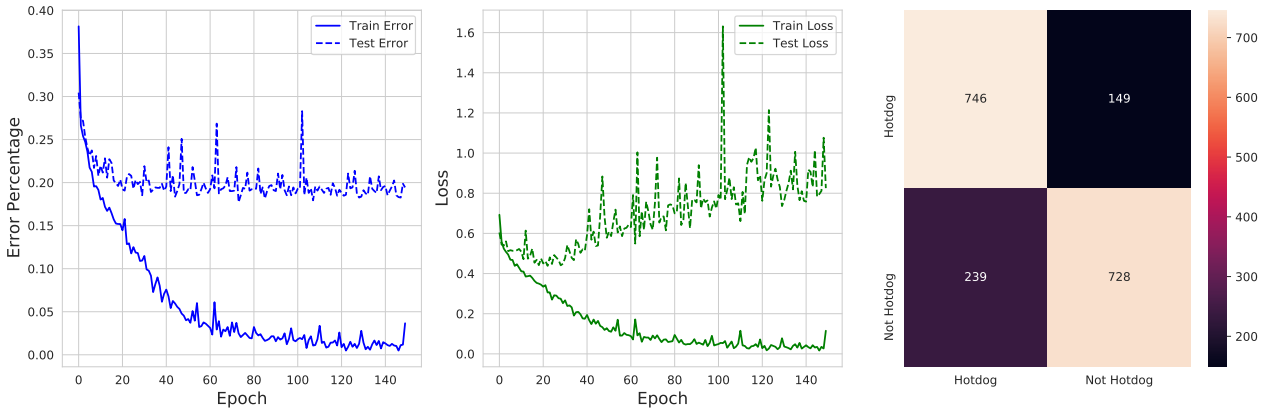

Figure 3: The best accuracy was found to be 0.823. The model seems to overfit the loss.

An example of 6 random miss-classified images can be seen in Figure 4. It can be seen that the model is sensitive to colour intensity, as images including shapes with brown and reddish nuances are often miss-classified. Furthermore, the data seems to have a few errors in the manual classification, as we found few objects, such as some red sushi, classified as a hotdog.

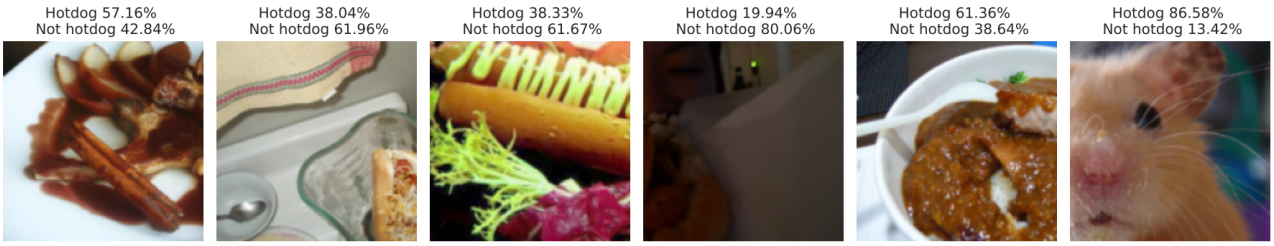

Figure 4: Examples of different miss-classified images

White noise to hotdog

The pixels of a noise image was iteratively optimized for the network to classify it as a hotdog, the resulting image is seen in figure 5. This was done by minimizing the loss of the network by optimizing the pixel-values of the noise image w.r.t. the target value of a hotdog. The human eye have a hard

time recognizing anything in the resulting image of wildly varying pixel values, which could be fixed if a weighted neighbourhood penalty was added for smoothing and is clear future improvement that could be made.

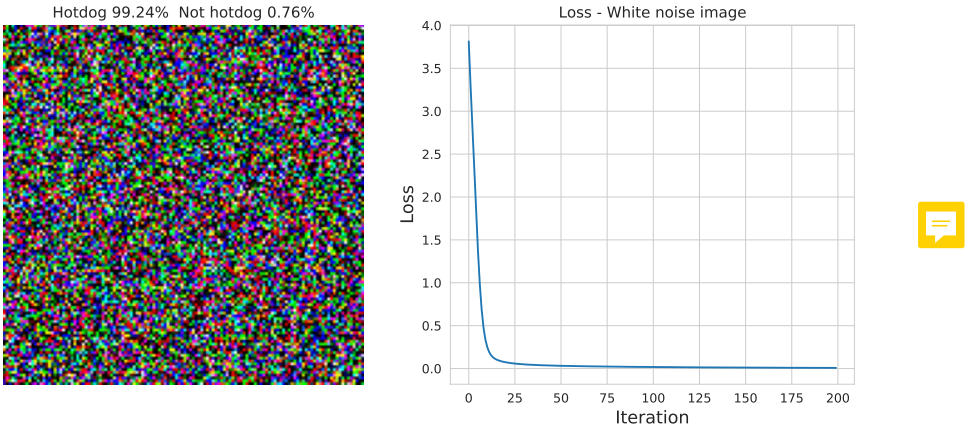

Figure 5: An image of noise optimized for the network to classify it as a hotdog (right) and the corresponding loss of the network from the noise image (left).

Project 1.2 - SVHN boundary boxes

In the following project, object detection is achieved by training a CNN on 11 different classes, 10 classes for different number ranging from 0 to 9 based on the cropped digits from the SVHN data-set, and one to denote no number as a way to classify backgrounds with no objectives. Cifar-10 was first utilized as the class denoting a non-number, but was changed to images containing the mean of the boxes for the full numbers set from the SVHN data-set, as this allowed for more plausible data which improved the following number detection.

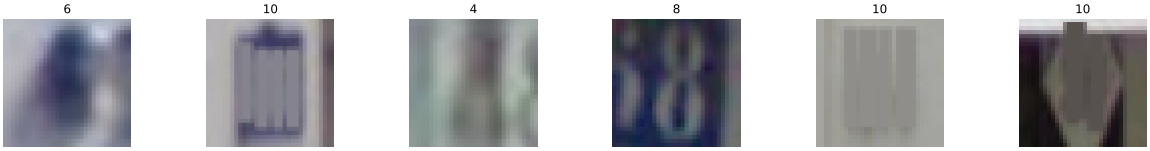

Figure 6: For the images with no objectives a mean was used to cover the numbers and the target was set to 10.

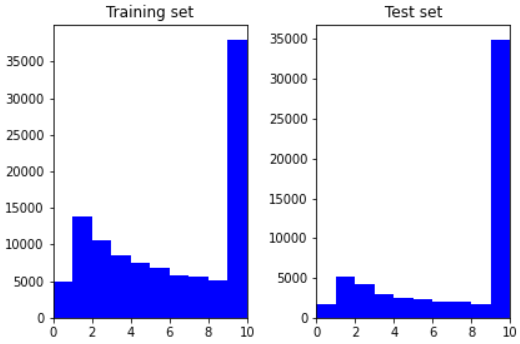

Figure 7: Distribution of classes

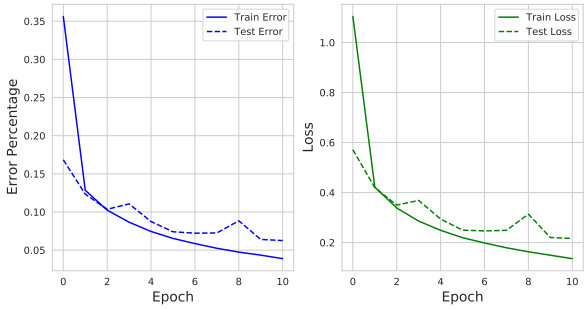

Figure 8: Performance of CNN

We had 106659 images for training, whereof 33402 is images depicting no numbers, and 59434 for testing, whereof 33402 is of no numbers. The distribution of numbers in training and test is quite equal, as can be seen in Figure 7, while there is more images not featuring numbers in the test than training. Both the inequality between classes and the in-equal ratio between train and test sets concerning images outside the SVHN data-set can influence our results. Further improvement could potentially be found by dealing with these, but we did not have the time to investigate.

From the confusion matrix (which is not shown, due to space limits), we saw that the model predicted equally well for each of the classes, hinting that the different sizes between the classes is not an issue.

Architecture

The CNN for object detection does not make use of linear layers, so the number of channels in the final convolutional layer in the model have to correspond to the number of classes, in this case 11. Furthermore each channel only consists of one value. Training ensures that the response for each of these channels correspond to the probability of the associated class.

Our chosen CNN is inspired of the VGG Neural Network. We utilize kernel size 3 with padding and stride 1 for the first two convolutions layers followed by a max pool layer, and the next 3 more convolutions layers followed by another max pool layer. In these layers we go from 3 to 128 channels, thus we try to learn as much structure from the training as possible. The last three convolutions layers ensures, that we end up with a output of 11 1-dimensional channels and functions as our classifier.

The performance of our CNN can be seen in Figure 8 after being trained for 11 epochs it reached an accuracy of 96.1 % for training and 93.7 % for test. Allowing it to run for more epochs could improve the accuracy further, but we may also risk over-fitting. We deemed the accuracy good enough for this case.

Predictions

The trained CNN can then be utilized on larger images as a Sliding Window algorithm, where it will give a prediction for what object is located in each area. Each pixel in the output will have a sum of 1 over the 11 channels, thus when trying to detect objects we may perform the non maximum suppression for each channel separately.

Running the Sliding Window algorithm will give a number of different possible bounding boxes that tries to encapsulate a certain feature. Since only a single CNN is trained for images of size 32x32, it will require a transformation of any inputs when utilizing it for detection in order to change the bounding box. Consideration is needed to be given to both its sizes, and the ratio between width and height as the final bounding boxes depend entirely on this process.

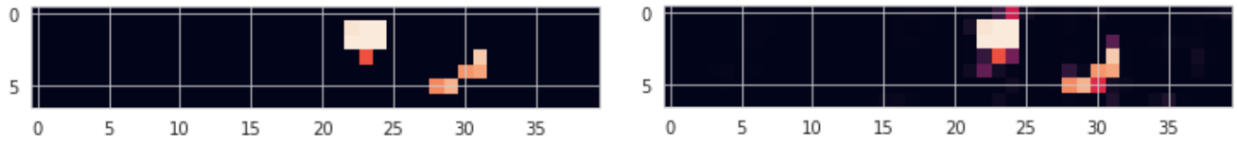

Figure 9: Example of predictions of class 3 for a random image before (right) and after (left) threshold of p_{class}

Non Maximum Suppression

The CNN outputs a number of probabilities for each pixel area in an image belonging to a certain class, and depending on the size of the input we may construct different bounding boxes. These may have several overlaps, as our sliding window algorithm may not be the perfect size for the detected feature, thus, we need a method for selecting the best bounding boxes, which takes into account the probability of a feature occurring twice.

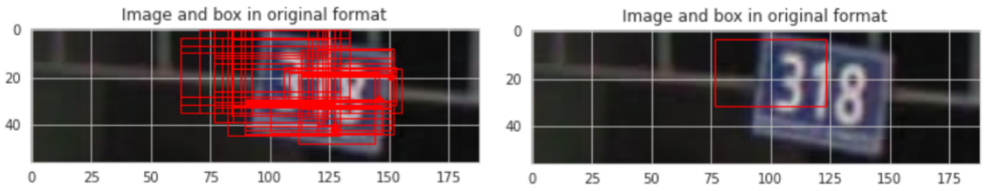

Figure 10: Bounding boxes before (left) and after (right) Non Maximum Suppression for random image for class 3

Non Maximum Suppression is utilized to select one entity out of several overlapping entities, such as in this case, one bounding box. It is performed by first discarding all boxes below a certain class probability, p_{class}. We had to choose a high p_{class} of 0.9 as we generally had high confidence of the predictions. An illustration of the prediction before and after a threshold can be seen in Figure 9.

The remaining boxes are run through a while loop, that discard any boxes that have too much of an overlap. The selection criterion is IoU (Intersection over Union), which is a measure of how similar two bounding boxes are, and will be more aggressive the closer to zero it is. We choose an IoU threshold of 0.6 since this removed most of the overlap, while still trying to allow for two close objects of same class to appear. An illustration of the bounding boxes before and after performing Non Maximum Suppression can be seen in Figure 10.

Results and reflections

Figure 11 shows an example of an image, where the sliding windows and on NMS has been performed for all classes. It can be seen that the classes predicted are 5,6 and 0, which is almost equal to the true labels of 6,6 and 0. The error may be due to the two objects depicting 6 are quite close together, and the number 5 and 6 being very similar. Further improvements could also be made by including more possible bounding box sizes and optimizing the NMS, as our current does not contain their associated object completely and seems generally too wide.

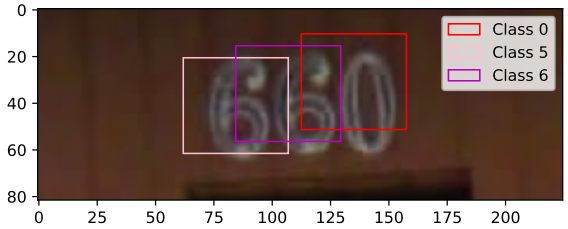

Figure 11: Predicted classes for random image

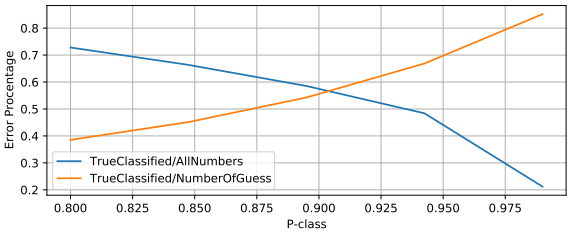

Figure 12: Error scores for 100 different random images, as a function of the threshold p_{class}

In order to evaluate our model and code we calculated two different error scores measured for different p_{class} over 100 different random images, as can be seen in Figure 12. "TrueClassified" referring to the number of correctly guessed labels, "AllNumbers" being the total amount of labels, and "NumberOfGuess" the total amount of guesses of labels. Then the percentage of "TrueClassified"/"AllNumbers",will be an estimate of true positives, and "TrueClassified"/"NumberOfGuess" will be an estimate of false positives.

For the optimal model both of the error measures will converge to one. As the optimal classifier will then classify all the images correctly without doing any false-positives. So as seen in the plot, the optimal value for our model will be around a p_{class} of 0.9, as this threshold is the best trade-off between the two error measures.

Some other interesting ideas, we did not have time to test is also data augmentation. We noticed some edges being predicted as numbers from the bounding box, data augmentation might have made the model more robust to this. Finally, a proper error measure, such as mean average precision would have been interesting, but due to time constraints, we did not have time to understand and test the method properly.

1 References

Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On calibration of modern neural networks.
Loshchilov, I., & Hutter, F. (2017). Decoupled weight decay regularization.
Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

Diary

08-06-2020

On the first day experimenting with optimal architectures for the network was the main focus. Working on this was Helena and Tobias. Christian later modified the transformations and architecture.

09-06-2020

The performance of the model had to be verified, with plots, Tobias found miss-classifications and did confusion matrices. Helena tested optimal augmentations, optimisers, batchnorm and max-pooling. Christian adjusted the model architecture, found different optimizers to test, looked into papers, trained the final model and generated loss plots. To classify a white noise generated image to a "hotdog", all members of the team participated and the same goes for the writing of the report.

10-06-2020

All members worked on data download, pre-processing and modelling of the CNN. Tobias and Christian made algorithms for plotting bounding boxes. Christian worked in the evening to make it possible to use the train and test loaders with the mean of the bbox'es instead of cifar-10.

11-06-2020

Tobias and Christian worked on implementing different sizes of bounding boxes and saving them for later use. Cleaning of the bounding boxes with non-maximum suppression was worked on by all members. Helena wrote for project 1.2, and the section was later looked over and edited for the final results by all group members.