Machine Intelligence

Project 1c: Deep Learning
COMP6380

University of Newcastle

Semester 1, 2019

Supervising professor:

Prof. Dr. Stephen Chalup

Submitted by:

Filip Heikkila [3322424]

Beytullah Ince [3324091]

Janina Mattes [3321632]

Ryan Stiff [3202269]

Q1 Detecting Empty Plastic Bottles and Aluminium Cans

The aim of this question was to develop an object detection model for plastic bottles and cans. This part is broken into two sections that show how the use of different training data sets can affect the results of an object detecting algorithm.

The object detecting algorithm used in this project comes from the Tensor Flow Object detection API, https://github.com/tensorflow/models/tree/master/research/object_detection. This is a commonly used algorithm for hobbyists developed by GOOGLE Inc. It has a range of online tutorials allowing it to be quickly implemented, issues stem from the use of different versions of needed packages, as well as the development of the training and testing data sets. These data sets are one of the most important parts of the object detector.

These data sets can be made in different programs, but all involve outlining the object's to be detected with a box or silhouette and noting what class the object is, i.e. bottle or can. These data sets can be any size and be set in different lighting and background layouts. The orientation of the objects and if they are covered can also affect the training. Initial training sets used had several images with half of a bottle or can visible from the side of the image, or a bottle covering part of another bottle. These fragments make the algorithm harder to train and if they are ignored and not marked as the correct class than they can cause the algorithm to ignore parts of bottles going against the rest of the dataset.

Object detection program 1

The first model successfully trained was run on a CPU because of issues running the program on computers with GPU's due to the version issues mentioned earlier. Due to the slow running time of the algorithm on CPU the data set was minimised with several objects per image and a total data set of 50 images. Initially the data set was going to have several versions of each sample image in colour, rotated and flipped. This was in the hope of training the image so that a wider range of drinking cans could be detected without relying on the colour, and cans and bottles orientated different ways wold still be detected.

The dataset was altered so that every object highlighted is complete and not covered by any other object. Several grey scale images, different lightings, and background and orientated bottles were used.

The architecture and pre-trained data sets were chosen as SSD_mobilenet_v1_COCO. Initially a model that was a lot slower but had a much higher increase in accuracy when it came to the COCO data set was used. However due to the use of CPU this was a very slow learning model, when the batch size was decreased this sped up but produced very unsteady loss values and did not seem to train at a usable rate. The SSD_mobilenet_v1_COCO model was significantly faster and still had a higher accuracy compared to the faster models so it was used. This model allowed a batch size of 12 images to be used with a reasonable learning speed for a CPU, ~15-20 seconds per step.

The learning rate used was 0.004 with 12 images per batch. This was trained for 2896 steps which gave a loss of 1.04. At this point the loss was still fluctuating and could be trained for significantly longer but since the model saved the weights regularly this minimum point was recorded. This can be seen in Figure 1, where the dark line is the smoothed loss data to show that there is still some decrease in loss, and the fainter line is the actual loss which is quite spiky.

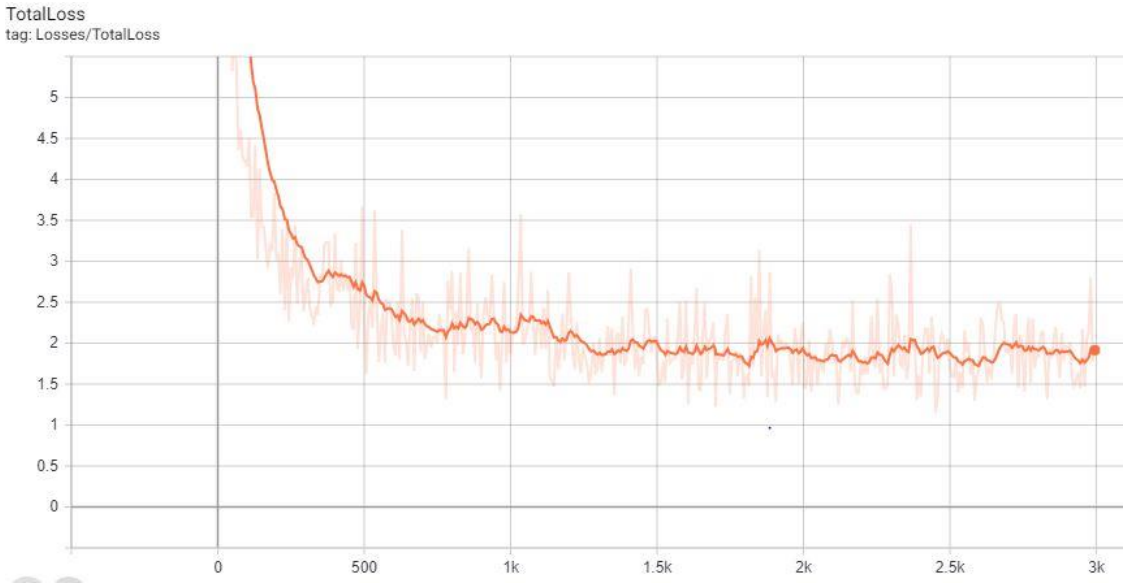

Figure 1 Total loss of model no. 1

Results

The smaller data set has allowed the quick training of this model, while still producing usable results, which can be seen below.

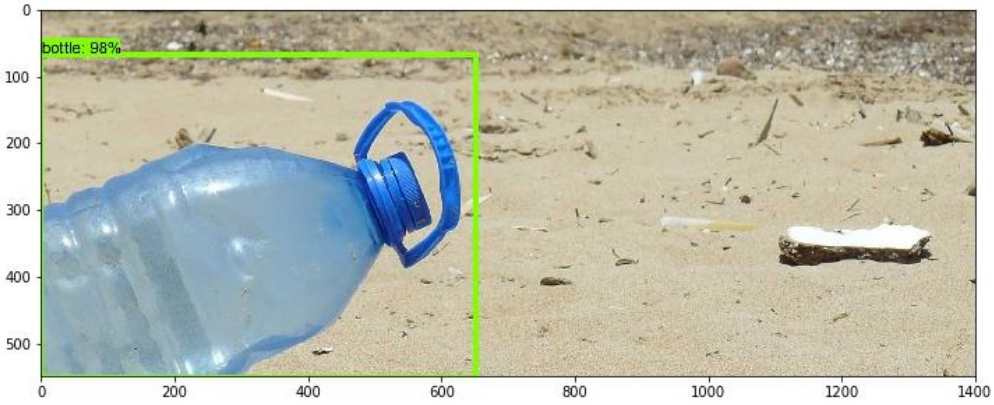

Figure 2 example no. 1

Figure 3 example no. 2

Figure 4 example no. 3

However there are still cases where the objects are not detected. As can be seen in Figure 1, the loss isn't decreasing steadily. This can come down to the lack of variation in the data set, since most bottles were labelled and cans not a wide enough range since they were used in colour causing it to learn specific colours better. Increasing the batch size further would also help, as it would calculate the batch error over more samples potentially decreasing the sharp spikes seen in the loss graph.

Object detection program 2

The second model's training was run on a GPU in order to perform computations faster. After solving the version issues and tweaking the sample config for the used model, there were not any issues left with the usage of a GPU. Together with the self-taken images and images from various search engines the combined number of images is 667. The number of bottles and cans are approximately equally distributed.

The pre-trained data model in use was faster_rcnn_inception_v2_coco, which also has been trained with the coco data set. This model is the fastest among the Region Convolutional Neural Network (R-CNN), while still being more accurate than the previous model. R-CNN's are used for object detection, rather than image classification, that's why it is was necessary to have images with outlined boxed again.

The initial learning rate was 0.0002 with a batch size of 6. The model was trained for 43046 steps, which ended up with a total loss of 0.043. As figure 4 shows, it was not necessary to train the model for that long. The same or even lower loss-value could have been achieved with much fewer steps. For example, after 5807 steps the loss-value was already at 0.033.

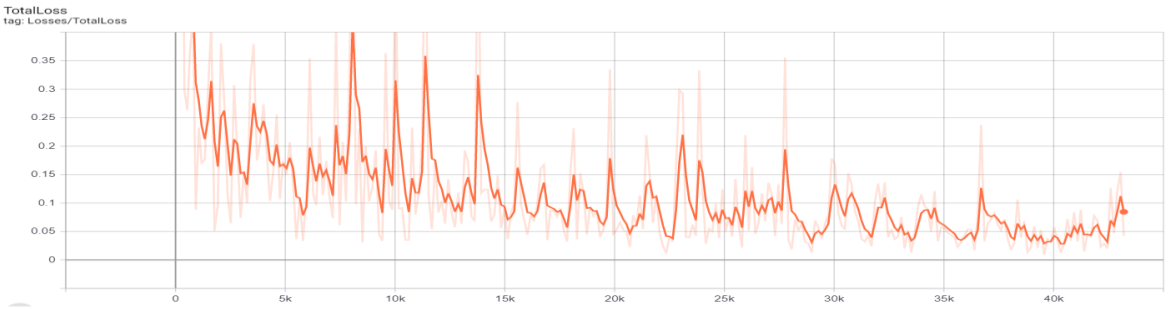

Figure 5 Total loss of model no. 2

Results

The model often detects bottles and/or cans with high accuracy and can also distinguish them most of the time. However, it still has some difficulties to properly detect objects, e.g. when the object is too small, or there are several disturbing things around the object.

Figure 6 example no. 4

Figure 7 example no. 6

Figure 8 example no. 5

Figure 9 example no. 6

As can be seen using the gpu allows for more effective training, since the processing rate is faster the amount of training images can be increased and a lower learning rate used. The overall result is also much better than can be achieved. The code submitted is for this trained object detector.

Question 2 – Manual Hyperparameter Tuning

For this task, we train a convolutional neural network to classify images of human cells that are either healthy or infected with malaria. In order to improve the model, we compare the performance of three variations of the size of the neural network and we train a network using three different learn rates. The validation results of the different variations will be compared, and the best contender will then be tested on the test set.

Model

The CNN consists of a convolutional stack, which contains several convolutional filters, batch normalization, relu-activations and max pooling. Batch normalization is always applied after the convolution operation, but before the activation. The convolutional stack is followed by a few fully connected layers, which utilize dropout during training. The output layer is a two-dimensional softmax output which shows the predictions of the CNN. Binary cross-entropy is used as loss function and the chosen optimizer is Adam. In order to avoid overfitting, early stopping is used, which means that after $T = 5$ epochs without improvement on the validation loss, the training will stop. Any other hyper parameters, including dropout rate, batch size and number of epochs will unless otherwise specified take the same values for each experiment, see table 1.

Experiments and Results

The malaria cell data set is split into a training set and a test set (80/20). During training 20% of the training set is used for validation to monitor the training process. The original images are all pasted on black backgrounds and cropped to square shape and then resized to colour images of size 64x64 pixels.

Include

- Training curves (training/validation loss and accuracy comparison)
- Training time – number of epochs and total time
- Test accuracy

2a)

A baseline model, see table 2, was first trained on the problem. The model was tweaked until a sufficient validation accuracy (> 95%) was reached. After that, two additional architectures were created, representing models with smaller and bigger capacity, see table 3 and 4. The capacity was regulated by increasing or decreasing the number of layers and the number of neurons per layer. The three models were than trained using the same hyper parameters. The training process and results can be seen in figure 10 and table 5.

Table 1: Hyper parameters used for training the neural network models.

Fixed hyper parameters	Value
Dropout	0.3
Max epochs	100
Batch size	32
Patience	5
Learn rate	1e-4
Decay	1e-5

Table 2: Neural network architecture "Baseline" version.

Layer	Size (number of units/channels)	Data shape
Input	-	(64,64,3)
Conv (3,3)	32	(64,64,32)
Max Pool (2,2)	-	(32,32,32)
Conv (3,3)	32	(32,32,32)
Max Pool (2,2)	-	(16,16,32)
Conv (3,3)	64	(16,16,64)
Max Pool (2,2)	-	(8,8,64)
Dense	64	(64)
Dropout	-	(64)
Dense	2	(2)

Table 3: Neural network architecture "Smaller" version.

Layer	Size (number of units/channels)	Data shape
Input	-	(64,64,3)
Conv (3,3)	16	(64,64,32)
Max Pool (2,2)	-	(32,32,32)
Conv (3,3)	32	(32,32,32)
Max Pool (2,2)	-	(16,16,32)
Dense	32	(32)
Dropout	-	(32)
Dense	2	(2)

Table 4: Neural network architecture "Bigger" version.

Layer	Size (number of units/channels)	Data shape
Input	-	(64,64,3)
Conv (3,3)	32	(64,64,32)
Conv (3,3)	64	(64,64,64)
Max Pool (2,2)	-	(32,32,64)
Conv (3,3)	128	(32,32,128)
Conv (3,3)	256	(32,32,256)
Max Pool (2,2)	-	(16,16,256)
Dense	724	(724)
Dropout	-	(724)
Dense	724	(724)
Dropout	-	(724)
Dense	2	(2)

Table 5: Results for the three neural network architectures.

Model	Validation accuracy	Training time (epochs)	Training time (minutes)	Test accuracy
Baseline	0.9608	17	3	0.9641
Smaller	0.9134	19	2.5	0.9249
Bigger	0.9633	27	40	0.9644

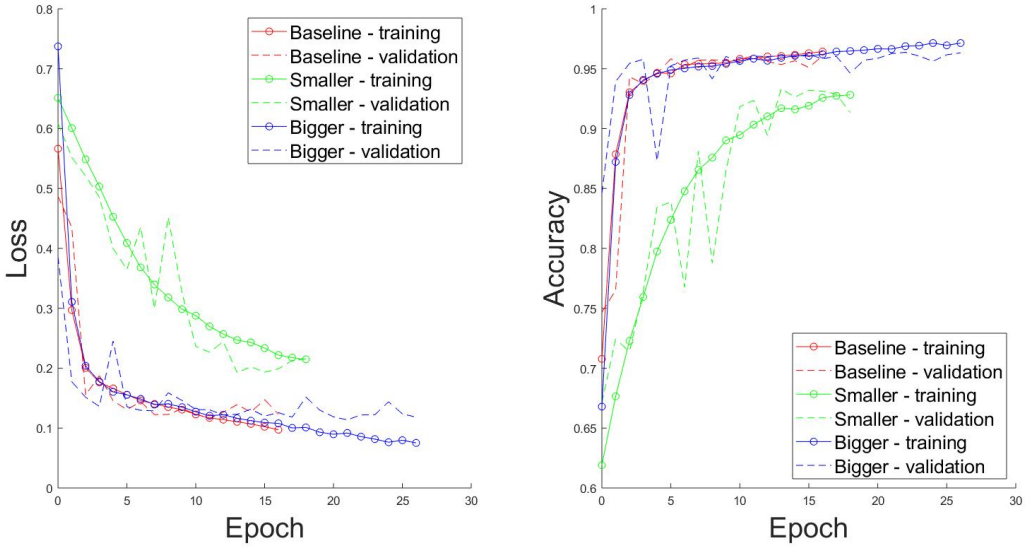

Figure 10. Training results for three different neural network architectures showing the loss (left) and accuracy (right) over the course of training.

Considering that the baseline model achieves a validation accuracy of 96% (see table 5), it is clear that it has a big enough capacity to learn the task. Increasing the capacity does not seem to increase the performance, considering that the bigger model only performs marginally better. On the other hand, the bigger model trains for 13 times longer time. When the capacity is decreased however, the performance is notably worse. The loss improvement for the smaller network is slower and stops at a higher loss level than for the other two models, see figure 10, and the final validation accuracy is approximately 5 percentages lower, see table 5.

It is worth mentioning that the training error still continues to improve for all models when the early stopping mechanism stops training, but as can be seen in figure 10 the validation loss is being stagnant. This means that letting the model train for longer, the generalization error will grow due to overfitting.

The results show that it is favorable to try a few different architectures when approaching a problem. The reason is that once sufficient capacity is reached, adding additional layers or neuron to the network only contributes to a prolonged training time. However, not having enough capacity will lead to an underperforming model.

2b)

For the second part the baseline model, which was also capable to achieve the best results, was trained with three different learning rates 0.00001 (small), 0.0001 (medium), 0.001 (large). In addition, a larger value with 0.01 (very large) is trained for an overall comparison. The learning rate is most commonly defined in the range between [0, 1] and is perhaps the most important hyperparameter when adjusting a deep neural network as it regulates the effective capacity of the model.

- The learning rate needs to be correct, not too high or too low for the model
- The learning rate typically has a U-shaped curve for training error

Baseline Model

Table 6: Neural network architecture "Baseline" version.

Layer	Size (number of units/channels)	Data shape
Input	-	(64,64,3)
Conv (3,3)	32	(64,64,32)
Max Pool (2,2)	-	(32,32,32)
Conv (3,3)	32	(32,32,32)
Max Pool (2,2)	-	(16,16,32)
Conv (3,3)	64	(16,16,64)
Max Pool (2,2)	-	(8,8,64)
Dense	64	(64)
Dropout	-	(64)
Dense	2	(2)

Table 7: Comparison of results between all "Baseline" models trained with different learning rates.

Model	Learning Rate	Training accuracy	Training time (epochs)	Training time (minutes)	Validation accuracy
M1 Baseline	0.00001	0.9738	16	5	0.9560
M2 Baseline	0.0001	0.9134	16	5	0.9531
M3 Baseline	0.001	0.9627	11	3.5	0.9619
M4 Baseline	0.01	0.4984	5	1.5	0.5154

M1 Small Learning Rate (0.00001) - The base model under M1 was trained with the smallest learning rate. Due to early stopping the model stopped after a rate of 16 epoch with a validation accuracy of 0.956 and a validation loss of 0.124.

M2 Medium Learning Rate (0.0001) - The base model under M2 had a similar solution as M1 and was trained with a comparably small learning rate. Due to early stopping the model also stopped after a rate of 16 epoch with a validation accuracy of 0.953 and a validation loss of 0.144.

M3 Large Learning Rate (0.001) - The base model under M3 had so far, the best results with a learning rate of 0.001. However, due to early stopping the model stopped after a rate of 11 epoch with a validation accuracy of 0.962 and a validation loss of 0.153.

M4 Very Large Learning Rate (0.01) - The base model under M4 achieved the weakest results. Due to early stopping and a constant training loss rate of 7.768 the model interrupted training after a rate of 5 epoch with a validation accuracy of 0.515 and a nearly constant validation loss of 7.77.

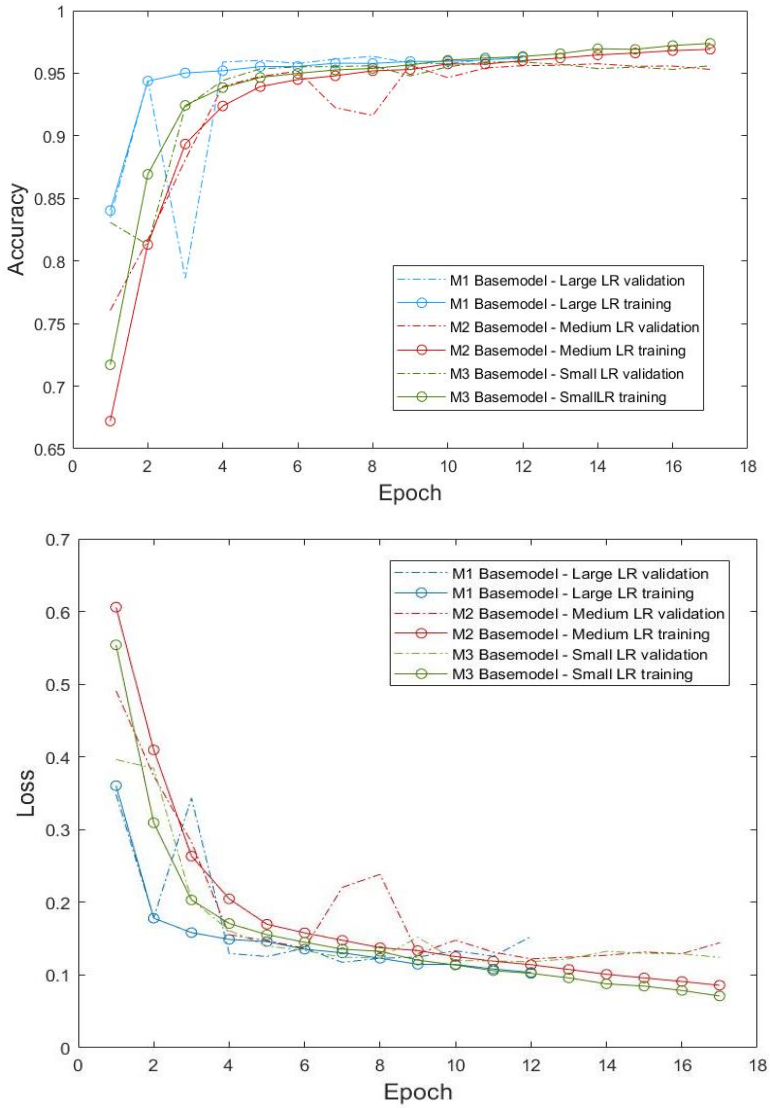

Figure 11. Training results for three different learning rates showing the loss (left) and accuracy (right) over the course of training between M1, M2 and M3.

The Adam optimization algorithm tweaks the parameters in order to minimize its cost function. A challenge is to find the optimal value for the learning rate hyper parameter in order to achieve the best loss without sacrificing training velocity. However, if the learning rate or step size is too small the algorithm requires more iterations which can on one hand make training more reliable but on the other hand also lead to an increased learning time. Therefore, compared to the overall results, training the base model with a learning rate of 0.00001 in M1 appears to be too low. Even that the algorithm did reach the solution with a validation accuracy of 0.956 it does take a larger amount of training time in comparison to M3, which was trained with a learning rate of 0.001 and was able to reach a validation accuracy of 0.962 in shorter time. Comparing the values of validation loss at M3 shows a faster decrease whilst both, M1 and M2 required, again due to their smaller step size, a larger amount of time to reach the same result.

Table 8: Results of M4 "Baseline" model trained with the largest learning rate of 0.01

Epoch	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss
0	0.498015	8.02986	0.51542	7.767941
1	0.498412	8.04057	0.51542	7.767941

2	0.498412	8.04057	0.51542	7.767941
3	0.498412	8.04057	0.51542	7.767941
4	0.498412	8.04057	0.51542	7.767941
5	0.498412	8.04057	0.51542	7.767941

On the other side, if the learning rate is selected too large the algorithm may overshoot the global minimum as visible in *Table 8*, which then could even cause the algorithm to diverge leading to increasing values. A failure to converge is clearly shown in the last example M4, where the baseline model was trained with a larger learning rate of 0.01. This finding is further visible in the nearly constant amount in training loss due to which early stopping interrupted the training after only 5 epochs.