

# Convolutional Neural Networks

Oskar Mampe: 201368087

March 1, 2020

## Contents

<b>1</b>	<b>Part I: Experimenting with Features</b>	<b>1</b>
1.1	Implementation Decisions . . . . .	1
1.2	Results . . . . .	2
1.3	Effect on Performance . . . . .	3
<b>2</b>	<b>Part II: Visualising Filters</b>	<b>4</b>
2.1	Implementation . . . . .	4
2.2	Results . . . . .	4
2.3	Comment on the evolution of the filters . . . . .	5
<b>3</b>	<b>Part III: Visualising Features</b>	<b>5</b>
3.1	Results . . . . .	5
3.2	Comment on the evolution of the features . . . . .	5
<b>4</b>	<b>Part IV: Experimenting with Network</b>	<b>6</b>
4.1	Change 1: Different Kernel Sizes . . . . .	6
4.1.1	Experiments Made . . . . .	6
4.2	Change 2: Different Output Features . . . . .	6
4.3	Change 3: Different optimizer . . . . .	7
4.4	Change 4: More Epochs . . . . .	7
4.5	Result . . . . .	7

## 1 Part I: Experimenting with Features

Firstly, for my experiments, I have tested various number of layers to see how it affects the performance of the network, changing it from the original 3 layers. Furthermore, I have decided to test the performance of different filter size's of the last convolutional layers. I have decided to choose different kernel size, as to see whether the change in kernel size will have a positive affect by capturing larger, more complex features.

### 1.1 Implementation Decisions

When implementing the 2-layer model, I have deleted the last convolutional layer that took in 24-dimensional input and outputted a 32-dimensional output, as that seems the most reasonable jump from 16-dimensional output from the first convolutional layer.

For the networks with more than 3 convolutional layers, in order to stay consistent, I have decided to follow the linear sequence of increasing the input layer by 8. Therefore, the third layer spits out a 32-dimensional tensor and the fourth layer will then output a 40-dimensional tensor. This also allows the model to train faster, as a larger difference between input and output will take significantly more time. Having to train 8 models takes significant amount of time without time-consuming features. There is an argument that doubling the input for the output features increases the accuracy, however in my experiment all 4 experiments follow the same sequence, so having a layer that does not follow the sequence will skew the results.

For the kernel sizes, I have decided to follow the sequence 3, 5, 7, 9. Generally speaking, I wanted to see what effect the larger kernel sizes would have on the network. I have decided to start at 3 as it seems to be the lowest one used in literature.

I have trained for 10 epochs, just as in the original model.

## 1.2 Results

2 Layer Model  
Epoch / Batch [10 / 39] - Loss: 0.449  
Epoch / Batch [10 / 78] - Loss: 0.417  
Epoch / Batch [10 / 117] - Loss: 0.459  
Epoch / Batch [10 / 156] - Loss: 0.476  
Epoch / Batch [10 / 195] - Loss: 0.507  
Epoch / Batch [10 / 234] - Loss: 0.607  
Epoch / Batch [10 / 273] - Loss: 0.516  
Epoch / Batch [10 / 312] - Loss: 0.472  
Epoch / Batch [10 / 351] - Loss: 0.578  
Epoch / Batch [10 / 390] - Loss: 0.581  
Epoch 10 - Loss: 0.507  
Accuracy of the epoch throughout training: 84 %  
Validation loss = 2.72  
Time Took to Train: 50.95

----- 2 Layer Model -----  
Confusion matrix, without normalization

Confusion matrix									
True label	0	1	2	3	4	5	6	7	8
0	73	9	15	35	7	9	7	10	4
1	12	75	8	16	3	14	8	14	14
2	19	11	58	15	21	12	6	16	10
3	31	19	13	64	3	14	8	11	10
4	19	18	18	25	34	18	14	23	9
5	9	18	5	19	12	73	18	19	15
6	19	14	7	22	20	23	34	17	12
7	20	14	8	30	10	16	7	32	16
8	23	10	14	19	9	15	20	17	47
9	5	49	8	3	3	3	4	9	6
Predicted label	0	1	2	3	4	5	6	7	8

Accuracy of the network on the test images: 33 %

(a) 2 Layer Network

4 Layer Model  
Epoch / Batch [10 / 39] - Loss: 1.822  
Epoch / Batch [10 / 78] - Loss: 1.843  
Epoch / Batch [10 / 117] - Loss: 1.810  
Epoch / Batch [10 / 156] - Loss: 1.760  
Epoch / Batch [10 / 195] - Loss: 1.803  
Epoch / Batch [10 / 234] - Loss: 1.799  
Epoch / Batch [10 / 273] - Loss: 1.815  
Epoch / Batch [10 / 312] - Loss: 1.801  
Epoch / Batch [10 / 351] - Loss: 1.835  
Epoch / Batch [10 / 390] - Loss: 1.850  
Epoch 10 - Loss: 1.813  
Accuracy of the epoch throughout training: 35 %  
Validation loss = 1.75  
Time Took to Train: 51.14

----- 4 Layer Model -----  
Confusion matrix, without normalization

Confusion matrix									
True label	0	1	2	3	4	5	6	7	8
0	124	3	9	27	3	0	0	4	1
1	18	77	5	17	3	1	1	7	3
2	15	6	85	14	29	0	0	13	2
3	68	9	6	67	7	0	0	13	5
4	21	5	33	23	65	7	7	14	4
5	15	12	12	24	12	72	17	11	11
6	40	12	4	22	23	11	41	5	13
7	29	15	9	30	23	3	4	31	10
8	32	11	21	27	17	2	9	23	24
9	5	34	1	2	0	2	0	4	4
Predicted label	0	1	2	3	4	5	6	7	8

Accuracy of the network on the test images: 41 %

(c) 4 Layer Network

3 Layer Model  
Epoch / Batch [10 / 39] - Loss: 1.394  
Epoch / Batch [10 / 78] - Loss: 1.455  
Epoch / Batch [10 / 117] - Loss: 1.402  
Epoch / Batch [10 / 156] - Loss: 1.458  
Epoch / Batch [10 / 195] - Loss: 1.370  
Epoch / Batch [10 / 234] - Loss: 1.428  
Epoch / Batch [10 / 273] - Loss: 1.485  
Epoch / Batch [10 / 312] - Loss: 1.398  
Epoch / Batch [10 / 351] - Loss: 1.482  
Epoch / Batch [10 / 390] - Loss: 1.439  
Epoch 10 - Loss: 1.429  
Accuracy of the epoch throughout training: 51 %  
Validation loss = 1.53  
Time Took to Train: 51.67

----- 3 Layer Model -----  
Confusion matrix, without normalization

Confusion matrix									
True label	0	1	2	3	4	5	6	7	8
0	115	2	11	32	2	0	4	4	1
1	5	117	4	13	4	1	12	6	10
2	8	11	95	14	12	1	12	12	4
3	32	6	10	102	1	4	11	8	1
4	3	5	19	2	86	12	38	7	9
5	4	10	8	8	17	81	40	11	12
6	15	8	6	17	19	13	71	12	11
7	11	14	18	29	11	3	9	51	10
8	7	13	8	15	18	5	36	20	49
9	4	46	3	3	4	1	4	3	4
Predicted label	0	1	2	3	4	5	6	7	8

Accuracy of the network on the test images: 50 %

(b) 3 Layer Network

5 Layer Model  
Epoch / Batch [10 / 39] - Loss: 1.906  
Epoch / Batch [10 / 78] - Loss: 1.881  
Epoch / Batch [10 / 117] - Loss: 1.881  
Epoch / Batch [10 / 156] - Loss: 1.928  
Epoch / Batch [10 / 195] - Loss: 1.904  
Epoch / Batch [10 / 234] - Loss: 1.914  
Epoch / Batch [10 / 273] - Loss: 1.889  
Epoch / Batch [10 / 312] - Loss: 1.936  
Epoch / Batch [10 / 351] - Loss: 1.906  
Epoch / Batch [10 / 390] - Loss: 1.877  
Epoch 10 - Loss: 1.901  
Accuracy of the epoch throughout training: 31 %  
Validation loss = 1.78  
Time Took to Train: 52.16

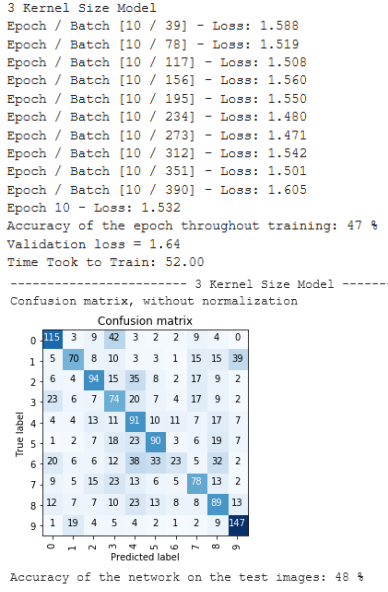
----- 5 Layer Model -----  
Confusion matrix, without normalization

Confusion matrix									
True label	0	1	2	3	4	5	6	7	8
0	109	0	11	31	1	0	2	14	1
1	20	19	13	16	0	3	6	9	7
2	18	3	97	21	2	1	6	12	5
3	70	1	15	67	1	1	3	14	1
4	38	3	40	26	10	7	20	31	3
5	21	6	16	16	3	77	27	12	7
6	42	7	11	23	2	11	57	14	3
7	37	3	26	23	2	4	7	43	6
8	37	4	22	21	2	3	21	33	12
9	5	9	8	5	0	4	0	5	4
Predicted label	0	1	2	3	4	5	6	7	8

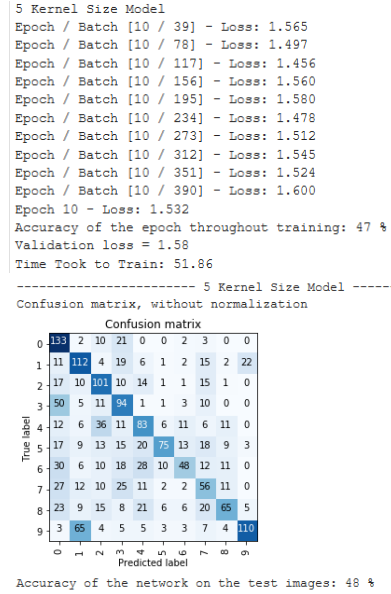
Accuracy of the network on the test images: 37 %

(d) 5 Layer Network

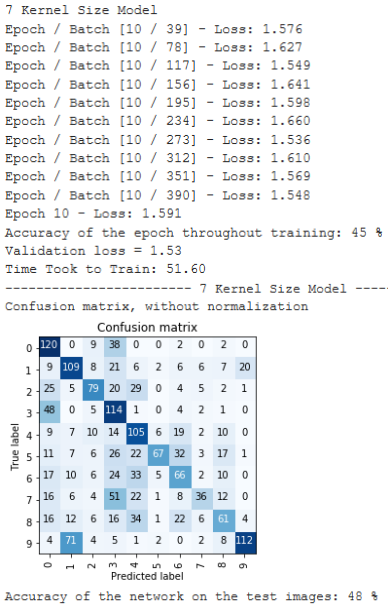
Figure 1: Confusion Matrices for the Different Layers



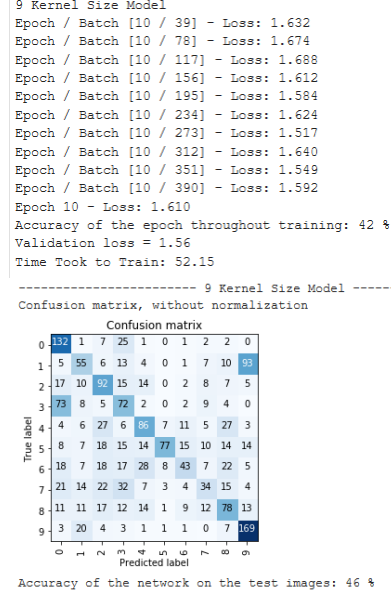
(a) 3 Filter Size Network



(b) 5 Filter Size Network



(c) 7 Filter Size Network



(d) 9 Filter Size Network

Figure 2: Confusion Matrices for the Different Layers

### 1.3 Effect on Performance

There were several differences, that occurred after tweaking the features, as seen above.

Firstly, the obvious effect can be seen in the 2 layer model where the training loss is very low, amassing a 80%+ accuracy. However, looking at the validation loss, and the confusion matrix, reveals that this high accuracy was only reached whilst training, leading to the conclusion that the model simply overfitted the data. The effect of that is that the model is not very good, since the model adjusted to the detail and noise of the data, therefore training weights that are optimised to this set of data. Hence, the 2 layer model could not take a random set of pictures and predict with an 80%+ accuracy.

Another issue is that saving the lower layers' weights takes more space in terms of hard disk space, with the 2-layer model weights taking 180MB+. This is an disadvantage, as a validation set is very useful metric to see whether the model is converging. Once an epoch, the model's weights could be saved. Then, when the training loss and validation loss are diverging, the model can be loaded from the previous save, where the validation loss was acceptable. Saving a 2-layer model can be quite memory intensive.

Furthermore, higher number of layers seem to have the opposite problem of the 2-layer model, where the training loss is poor, but the validation loss is better. This might mean that the model takes a very long time to converge, and possibly require more than 10 epochs before any noticeable change is seen.

As for the different kernel sizes, the difference in testing and training performance is minute. The worst performers were kernel size 7 and 9, where the data was seemingly overfitting, having a higher test accuracy than validation accuracy. However, they all seemed to hover at around about 48%, expect for kernel size 9, which is at 42%. This can be somewhat counter-intuitive as a higher number of kernel size should capture bigger and more rich features, but the model seems to preffer smaller features. This is probably due to some objects having small differences, for example, the difference between a cat and a dog are due to small features like ear shape etc.

The training time required for each model seemed to be around about 52 seconds per epoch, without any model deviating from that pattern.

The overall best performer was the original network with 3 layers and 4 kernel size. It seems to have the best balance in terms of not having too much layers and output sizes that it takes a very long time to train, and not overfitting, as it performed admirably in both validation and training.

## 2 Part II: Visualising Filters

### 2.1 Implementation

I have implemented the filter visualisations by taking the weights of the first convolutional layer and showed it using `plt.imshow()`. However, I have used `transforms.ToPILImage()`, as normalising with `pytorch.norm()` didn't have noticeable effect, since the values were still very low and any change in them was not noticeable.

### 2.2 Results



(a) Filters before training



(b) Filters during training



(c) Filters after training

Figure 3: Filters at different times

## 2.3 Comment on the evolution of the filters

Firstly, when the filters were initialised they were done so randomly, so there doesn't seem like there is a particular pattern. As the model trains, the filters don't change drastically, it instead does so slowly. Some filters haven't even changed at all from the initial point right until the end. This would point towards the model adapting the current filters and tweaking them, where any filters that already have a recognisable pattern stay the same. One of the filters in my example looks like an 'n' and it stays so throughout the experiment.

## 3 Part III: Visualising Features

### 3.1 Results

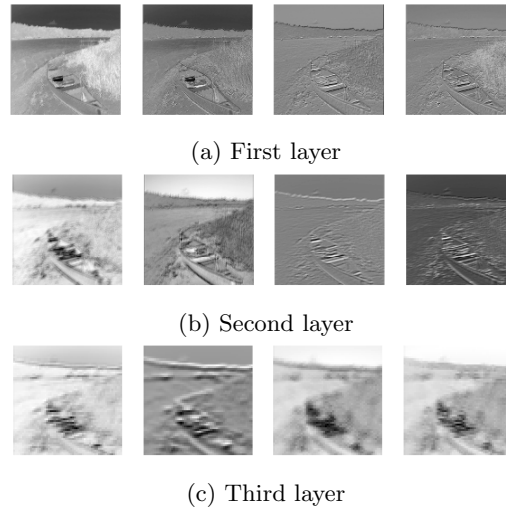


Figure 4: Features of the first image

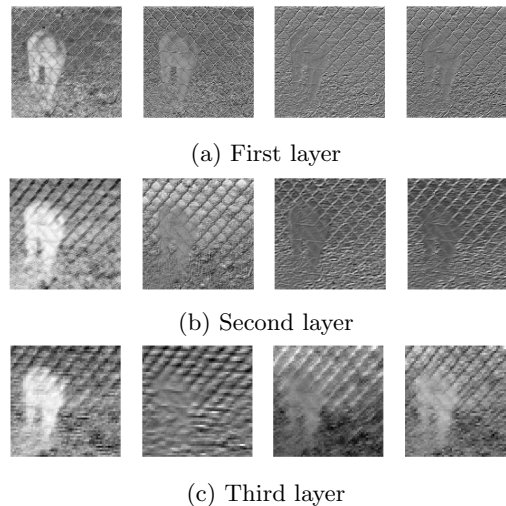


Figure 5: Features of the second image

### 3.2 Comment on the evolution of the features

Once the best model from the previous part has been trained, I have forwarded an image, to see how the image gets transformed over the layers. In the first layer, you can see that the model tries to separate the background from the object. In both figures first picture from the left, you can see that the boat/cat is clearly highlighted, where in the other pictures, the focus is more on the backgrounds, and how the different objects are separated. The first 2 of images on the left seem to have a lot of detail too, but as you go to the right, the detail is lost. This is especially true for the second image with the cat,

where 2 of the pictures look close to identical, and the foreground blends with the background.

As you go down the layers, the images get more blurry, which is expected since the images will get smaller and smaller. Alongside getting blurry, the images also increasingly lose detail, but they seem to lose detail in the background not in the foreground. This can be seen with the cat, as it looks like it is zooming in on the cat as you go down the layers.

However, the cat on by the third layer looks like a giant round blur. Although, the model is still being able to track where it is, which is tricky in this image since there is a metal net between the cat and camera. The boat on the other hand seems to be being coloured in black where the boat is. In the first layer, only a particularly dark part of the boat is in black, then by the third layer, almost half the boat is coloured in black.

## 4 Part IV: Experimenting with Network

### 4.1 Change 1: Different Kernel Sizes

#### 4.1.1 Experiments Made

The first change I have made do the model, is to tweak the kernel sizes both in the max pool layer as well as convolutional layer. The model seem to favour smaller kernel sizes for the convolutional layers, though it could be partially attributed to the fact that max pool operation can be replaced by a larger stride in the convolutional layers [1]. So I have decided to drop max pooling, but increase the stride by two. The effect of that wasn't as expected, and I did not receive results similar to the paper. Performance has dropped to 41% on unseen data as seen on the confusion matrix.

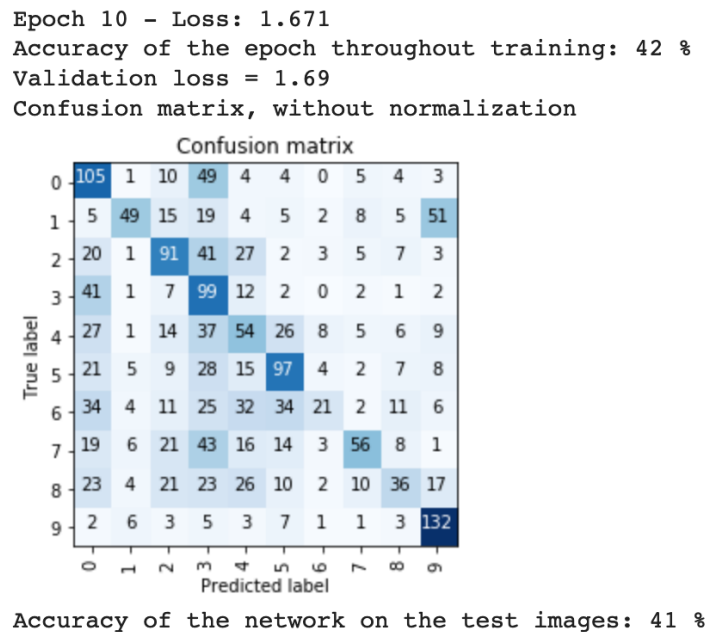


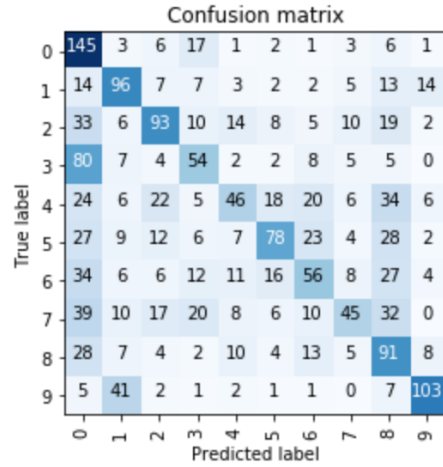
Figure 6: Removal of Max Pool

Next, seeing as the filters from the first layer have not been tested, I have decided to test different sizes of its kernel. I believed that would work as the features from the first layer seem to be extremely important and set the precedent for the following layers. Furthermore, the lower kernel sizes of the previous layers could benefit from having to extract smaller features from a bigger size of features. In conjunction with that I decided to also increase max pooling size and kernel size to 3 to balance it out, similarly to GoogleNet [2]. The experiment lead to 42% accuracy and overall worse model. As seen by the confusion matrix.

### 4.2 Change 2: Different Output Features

I have then decided that changing the size of the convolutional kernel or max pool on it's own won't be beneficiary. Therefore, I have decided to change the output features and input features of all the convolutional layers. I have chosen the following

Epoch 10 - Loss: 1.526  
 Accuracy of the epoch throughout training: 47 %  
 Validation loss = 1.57  
 Confusion matrix, without normalization

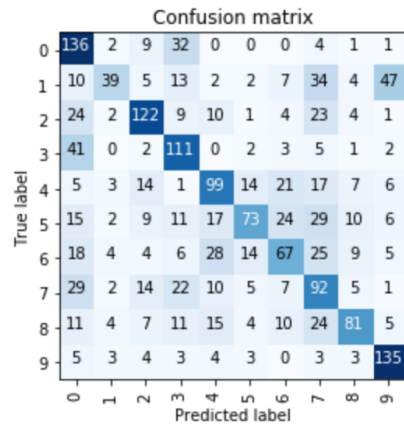


Accuracy of the network on the test images: 44 %

Figure 7: Increasing Kernel and Max Pool

parameters for each layer: first convolutional layer (input size = 3, output size = 32, kernel size = 3), second convolutional layer (input size = 32, output size = 64, kernel size = 5), first convolutional layer (input size = 64, output size = 128, kernel size = 5). I have also set all the max pools size of (3,3) and stride of 2. The increase in kernel size and max pool was due to the increased number of outputs. Once through the last layer, the flattened outputs reached dimensions of 100,000, which was huge. Hence, I used larger pools to reduce the size of these dimensions. The overall effect of that was much more positive rising the accuracy of the model to 53%.

Epoch 10 - Loss: 1.179  
 Accuracy of the epoch throughout training: 59 %  
 Validation loss = 1.34  
 Confusion matrix, without normalization



Accuracy of the network on the test images: 53 %

Figure 8: Increasing Output features

### 4.3 Change 3: Different optimizer

### 4.4 Change 4: More Epochs

### 4.5 Result

The final result of my network is as following

## References

- [1] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. 12 2014.
- [2] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.