Assignment 3 Report

This is an outline for your report to ease the amount of work required to create your report. Jupyter notebook supports markdown, and I recommend you to check out this cheat sheet (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet). If you are not familiar with markdown.

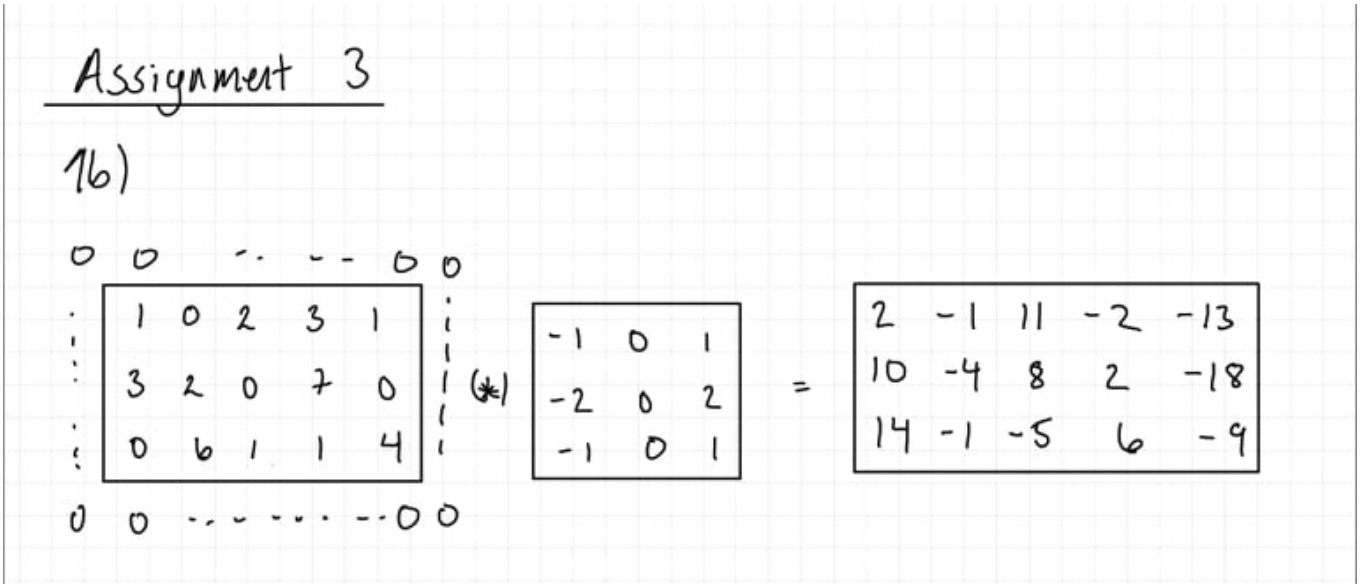
Before delivery, **remember to convert this file to PDF**. You can do it in two ways:

1. Print the webpage (ctrl+P or cmd+P)
2. Export with latex. This is somewhat more difficult, but you'll get somehwat of a "prettier" PDF. Go to File -> Download as -> PDF via LaTeX. You might have to install nbconvert and pandoc through conda; `conda install nbconvert pandoc` .

Task 1

task 1a)

Assuming that by convolution the task means the operation that happens in a convolutional layer in forward pass. Also to have the result be 3x5 (same as input image) we zero pad the image with one row/column.



Show work on the first column:

(0,0): $1 \cdot 0 + 0 \cdot 2 + 3 \cdot 0 + 2 \cdot 1 = 2$

(1,0): $1 \cdot 0 + 0 \cdot 1 + 3 \cdot 0 + 2 \cdot 2 + 0 \cdot 0 + 6 \cdot 1 = 10$

(2,0): $3 \cdot 0 + 2 \cdot 1 + 0 \cdot 0 + 6 \cdot 2 = 14$

task 1b)

The convolutional layer reduces sensitivity to translations in the image.

task 1c)

We need 2 rows and columns of zero padding on each side since this will make the center of the kernel be able to be placed at the edge of the image, but no further. This also depends on stride (which is 1 in this case).

task 1d)

The kernel is of size (9×9) since it reduces the spacial dimension by 8.

task 1e)

Pooling with neighborhood of (2×2) and stride of 2 will half the width and height. Therefore the dimension of the pooled feature maps (252×252).

task 1f)

Convolution with 3×3 kernel with stride of 1 and no padding will reduce the dimension by 2. Therefore the resulting spacial dimension is (250×250).

task 1g)

Input to first layer ($32 \times 32 \times 3$) There are 32 nodes that each has 3 5×5 kernels. This gives $32 \cdot 3 \cdot 5 \cdot 5 = 2400$ parameters in the first layer. The max pooling will half the width and height and the convolution will not affect the dimension.

Input to second layer ($16 \times 16 \times 32$) There are 64 nodes that each has 32 5×5 kernels. This gives $64 \cdot 32 \cdot 5 \cdot 5 = 51200$ parameters in the second layer. The max pooling will half the width and height and the convolution will not affect the dimension.

Input to third layer ($8 \times 8 \times 64$) There are 128 nodes that each has 64 5×5 kernels. This gives $128 \cdot 64 \cdot 5 \cdot 5 = 204800$ parameters in the third layer. The max pooling will half the width and height and the convolution will not affect the dimension.

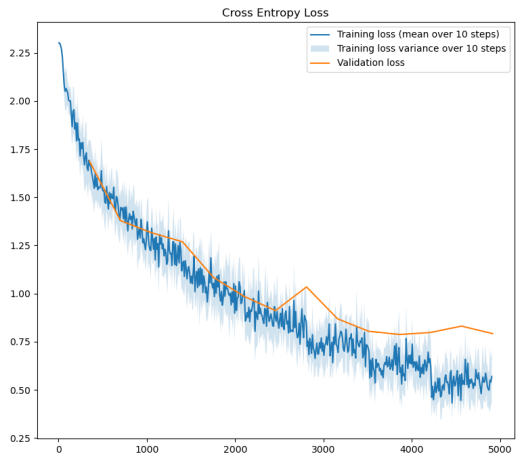
Before flattening the dimension is ($4 \times 4 \times 128$). This gives $4 \cdot 4 \cdot 128 = 2048$ input nodes in the fully-connected layers.

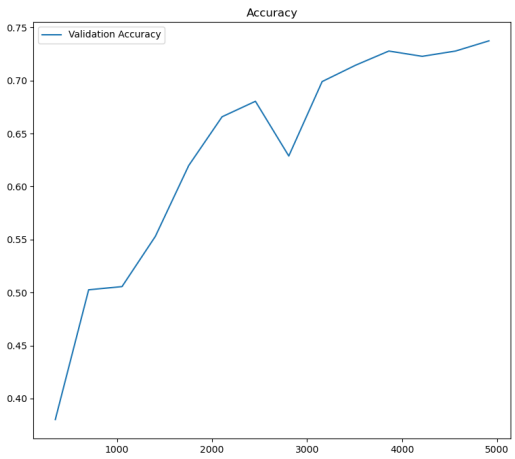
Fully connected number of weights and biases: $(2048 + 1) \cdot 64 + (64 + 1) \cdot 10 = 131786$

In total: $2400 + 51200 + 204800 + 131786 = 390186$

Task 2

Task 2a)





Task 2b)

Final validation loss: 1.79

Final validation accuracy: 0.737

Task 3

Task 3a)

For the first architecture we tried out batch normalization and a different kernel for the MaxPooling. For the optimizer, we went with Adam and used a learning rate of 7e-4 and weight decay of 2e-5. These numbers were found through searching on the internet and tuned by ourselves. This is how the network architecture looked:

Layer	Layer type	#Filters	Kernel details	Activation function
1	Conv2D	32	F=5, p=2, s=1	ReLU
1	MaxPool2D	-	F=4,s=2	-
	Batch normalization	-	-	-
2	Conv2D	64	F=5, p=2, s=1	ReLU
2	MaxPool2D	-	F=3,s=2	-
	Batch normalization	-	-	-
3	Conv2D	128	F=5, p=2, s=1	ReLU
3	MaxPool2D	-	F=2,s=2	-
	Batch normalization	-	-	-
		#Hidden units		
	Flatten	-	-	-
4	Fully-Connected	64	-	ReLU
5	Fully-Connected	10		Softmax

In our second architecture, we used drop out as a way of reducing overfitting.

Layer	Layer type	#Filters	Kernel details	Activation function
1	Conv2D	32	F=5, p=2, s=1	ReLU
1	MaxPool2D	-	F=2,s=2	-
	Batch normalization	-	-	-
2	Conv2D	64	F=5, p=2, s=1	ReLU
2	MaxPool2D	-	F=2,s=2	-
	Batch normalization	-	-	-
3	Conv2D	128	F=5, p=2, s=1	ReLU
3	MaxPool2D	-	F=2,s=2	-
	Batch normalization	-	-	-
		#Hidden units	Dropout details	
	Flatten	-	-	-
4	Fully-Connected	64	p=0.4	ReLU
5	Fully-Connected	10	p=0.4	Softmax

Task 3b)

Include final accuracy scores and plot for two models

First architecture:

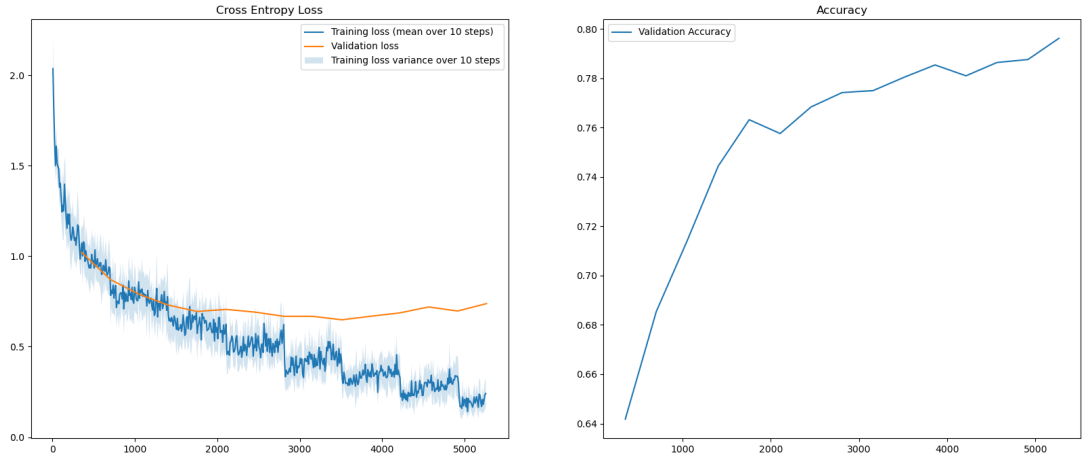
Final train loss	Training accuracy	Validation accuracy	Test accuracy
0.1665413975715637	0.9440566897392273	0.7961999773979187	0.7798999547958374

Second architecture:

Final train loss	Training accuracy	Validation accuracy	Test accuracy
0.29495593905448914	0.9061833024024963	0.7892000079154968	0.7788999676704407

The first architecture is slightly better in test and validation accuracy.

Plot of training and validation loss, and validation accuracy for the first architecture.



Task 3c)

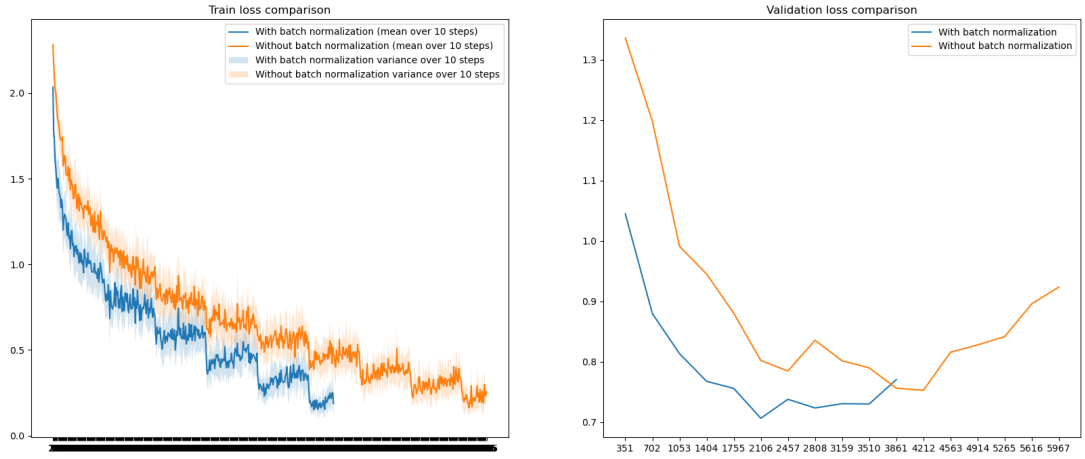
We found batch normalization useful for the architecture, which is why it is used in both of our architectures. By only applying batch normalization on the original architecture the accuracy improved a lot. This is because it becomes easier for the network to train, when we don't have huge differences in the weights and outputs.

Including dropout also helped a lot with overfitting for our model. It is also nice to have in terms of how long the training takes. It did take some time to find a good drop out probability, however we landed on 0.4 on the linear layer. With more filters on the last convolutional layer we saw better performance with dropout.

We tried using average pooling for a model, however this did not give very good results. It might be because it makes the model more complex and difficult to train, in comparison to max pooling.

Task 3d)

We saw the largest improvement with batch normalization and decided to compare with and without with this feature.



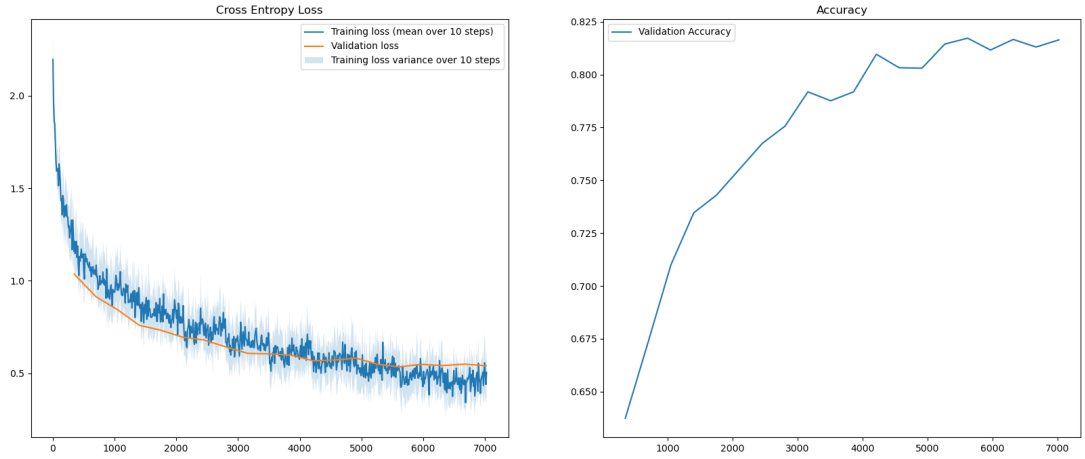
Task 3e)

We started by combining the two architectures from 3a using batch normalization, drop out with $p = 0.4$ and max pooling from architecture one. This improved to give test accuracy around 79. After trial and error we finally found that doubling the number of kernels in each convolutional layers ($[64, 128, 256]$) as well as using smaller kernels in the convolution $(f = 3, s = 1, p = 1)$ got an final test accuracy of 81.4%. In the end we used the same optimizer and hyperparameters as in task 3b.

Final architecture:

Layer	Layer type	#Filters	Kernel details	Activation function
1	Conv2D	64	F=3, p=1, s=1	ReLU
1	MaxPool2D	-	F=4,s=2	-
	Batch normalization	-	-	-
2	Conv2D	128	F=3, p=1, s=1	ReLU
2	MaxPool2D	-	F=3,s=2	-
	Batch normalization	-	-	-
3	Conv2D	256	F=3, p=1, s=1	ReLU
3	MaxPool2D	-	F=2,s=2	-
	Batch normalization	-	-	-
		#Hidden units	Dropout details	
	Flatten	-	-	-
4	Fully-Connected	64	p=0.4	ReLU
5	Fully-Connected	10	p=0.4	Softmax

Plot of test and validation loss and validation accuracy



Task 3f)

The model does not seem to be overfitting to the test set. The training accuracy was 91% while the test and validation accuracy was 81% and 82% respectively. This is not as large of a difference compared to other models where the test accuracy has been very high with lower validation and test accuracy.

The validation loss also seems to still decreasing at the end of the training which suggest that the model could have been trained even more and could improve further.

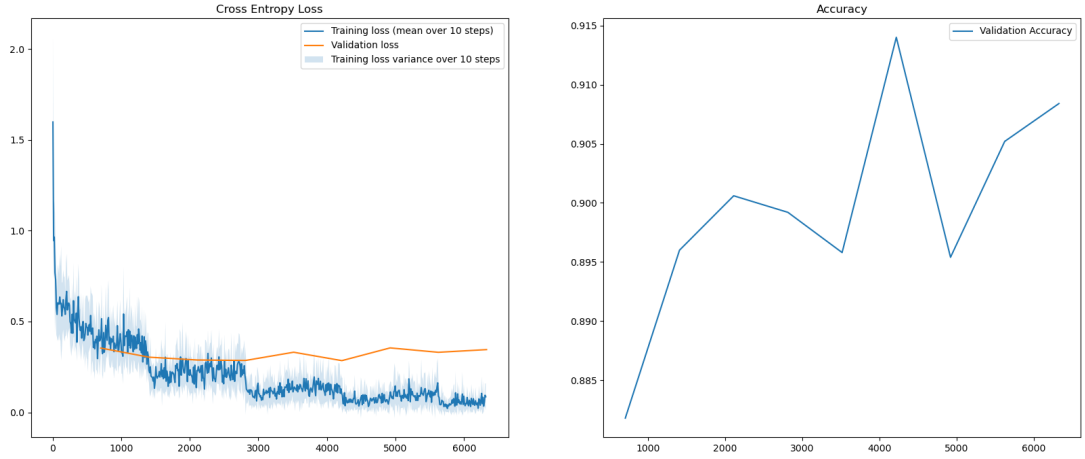
We have used used both weight regularization and drop out in our model to reduce over fitting.

Task 4

Task 4a)

We used everything as recommended in the hints. We used the Adam optimizer (which we also used in task 3) with batch size of 32, learning rate of $5 \cdot 10^{-4}$. We also resized the images to (224x224) and normalized the images with the mean and standard deviation given in the task4b.py starting code. We ran the training for 5 epochs with early stop count of 4. The training stopped early after 4,5 epochs.

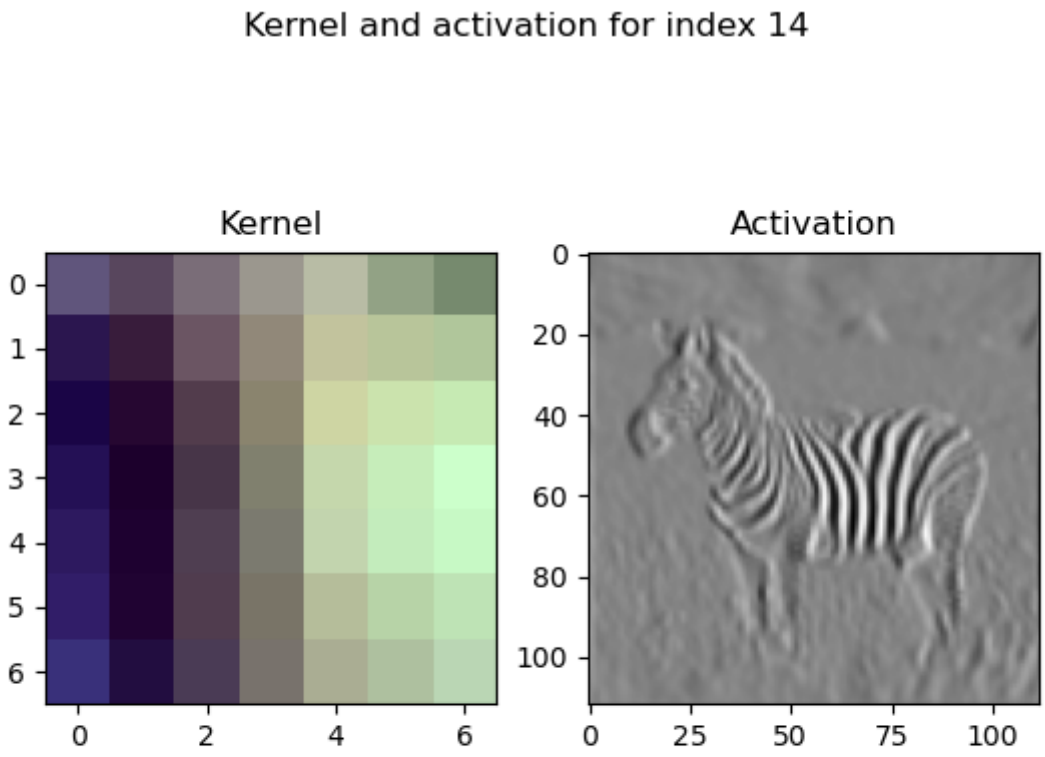
Plot of the training and validation loss and the validation accuracy.



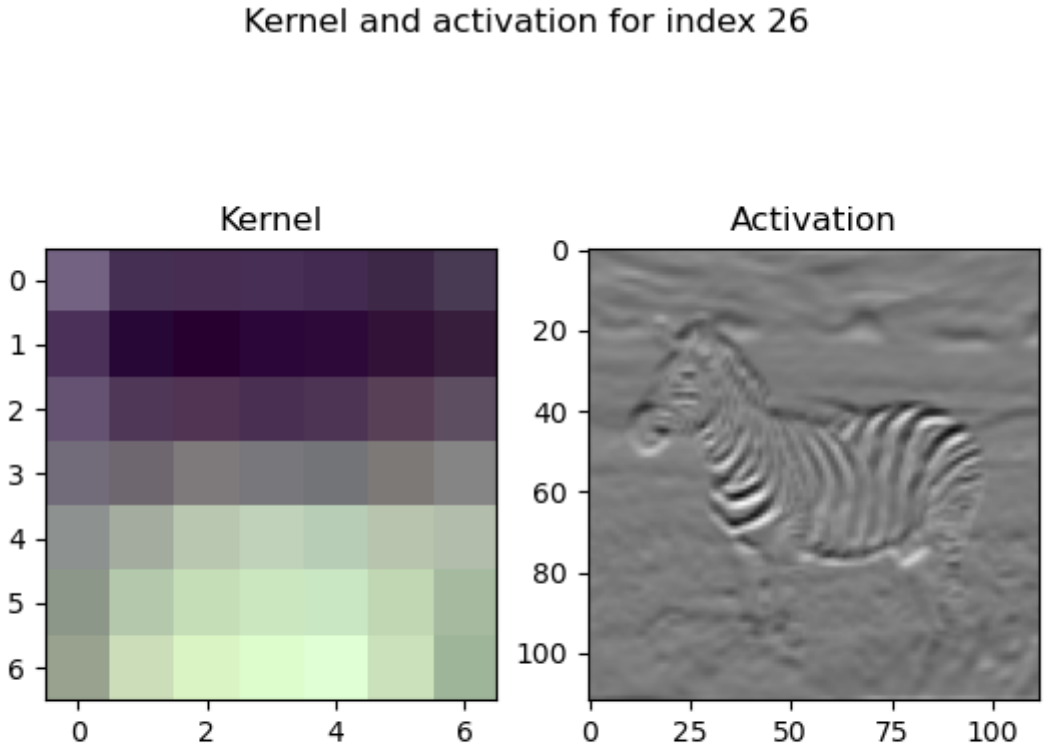
The final test and validation accuracy was 90%.

The model seems to be over trained since the training accuracy was 99%. Perhaps regularization of some kind is needed.

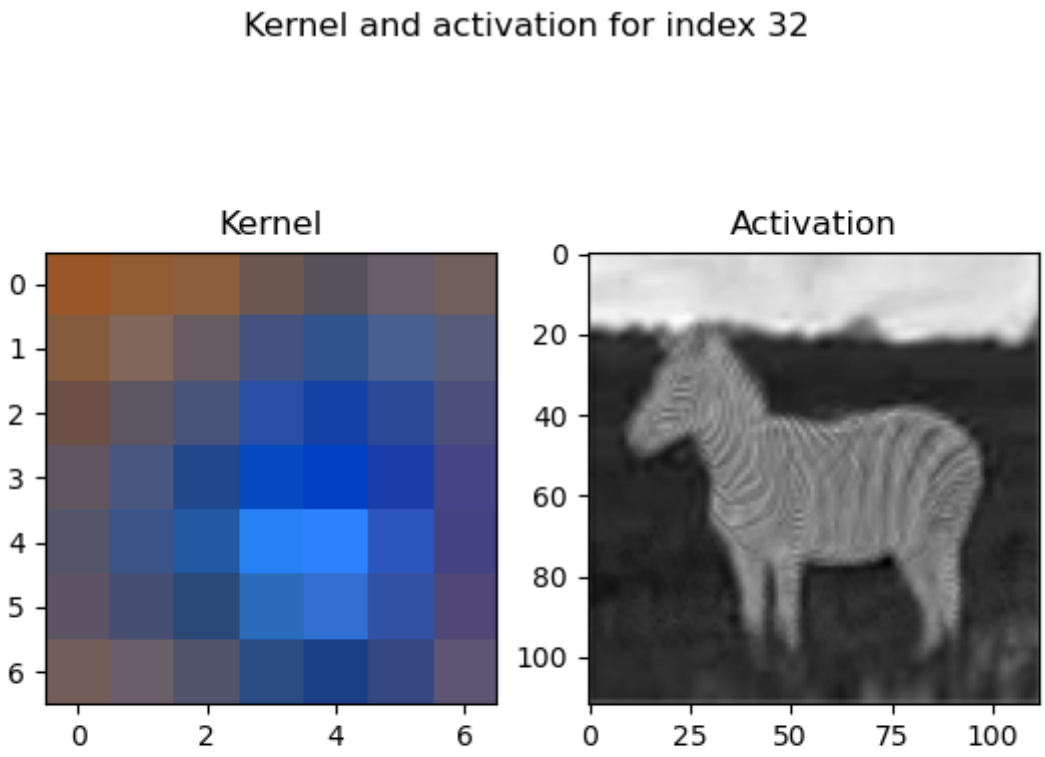
Task 4b)



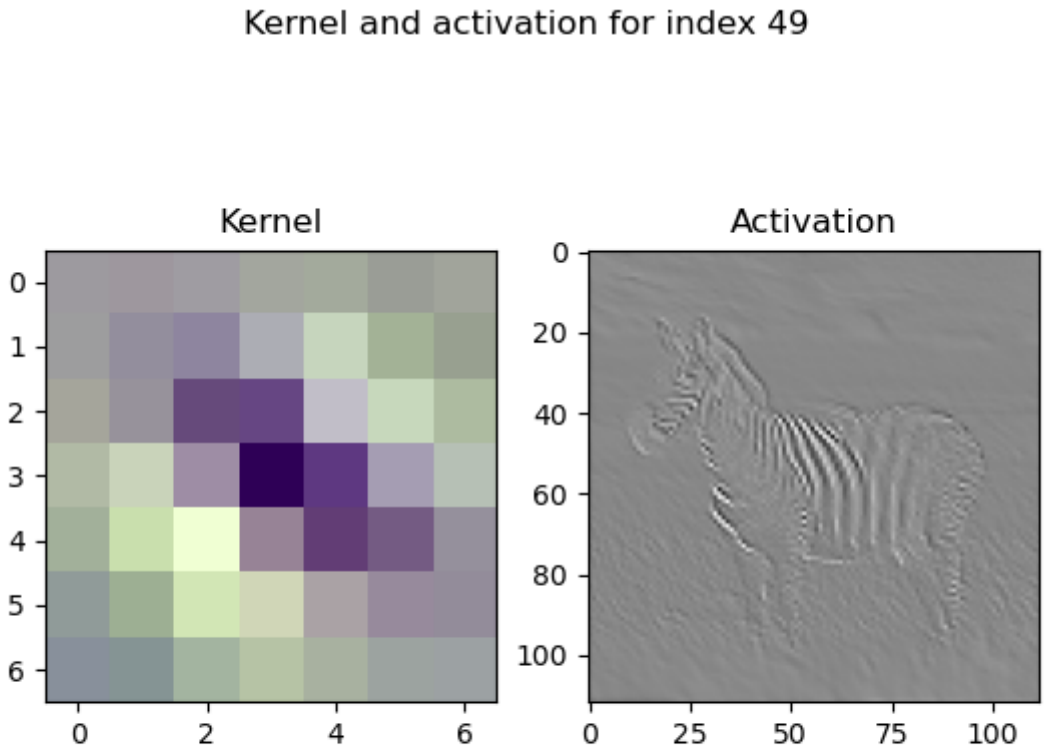
Kernel 14 seems be detecting vertical lines. This kan be seen both in the visualization of the kernel which has vertical stripes and in the activation where the stripes on the zebra is enhanced.



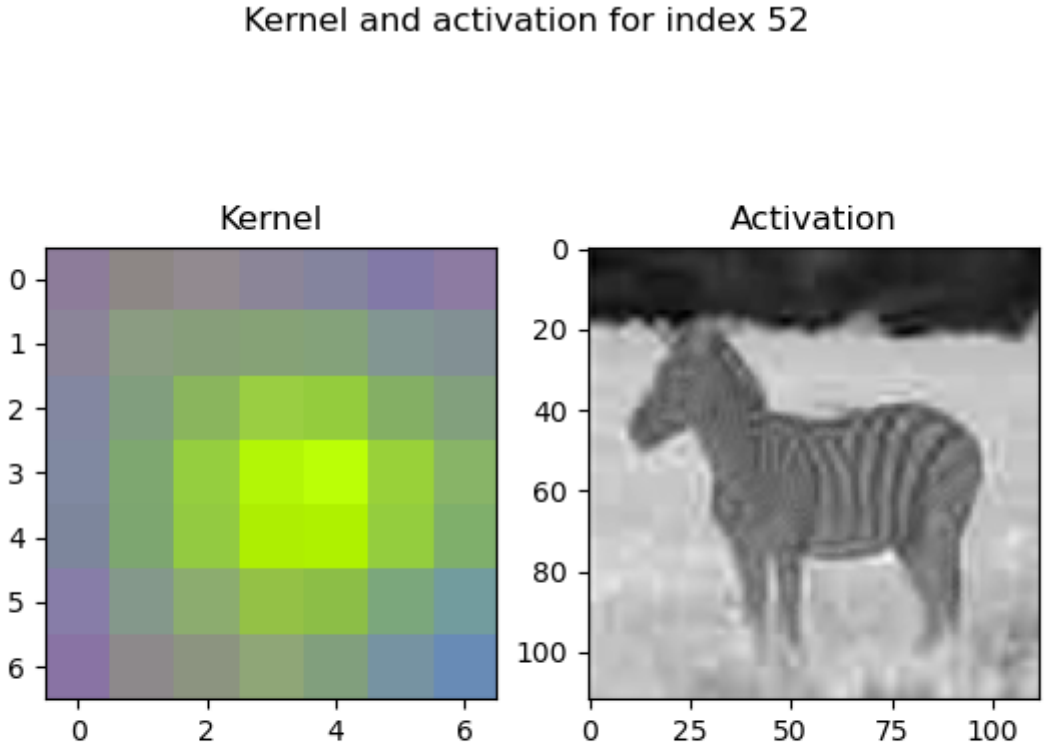
Similarly this kernel (26) seems to be detecting horizontal stripes.



This kernel seems to be detecting blue areas. The kernel has a blue blob in the middle with red on the sides making it activate blue areas especially if there is a contrast in color around the blue (e.g red). In the activation the sky and zebra is highly activated because the sky is blue and the zebra is white (containing blue).



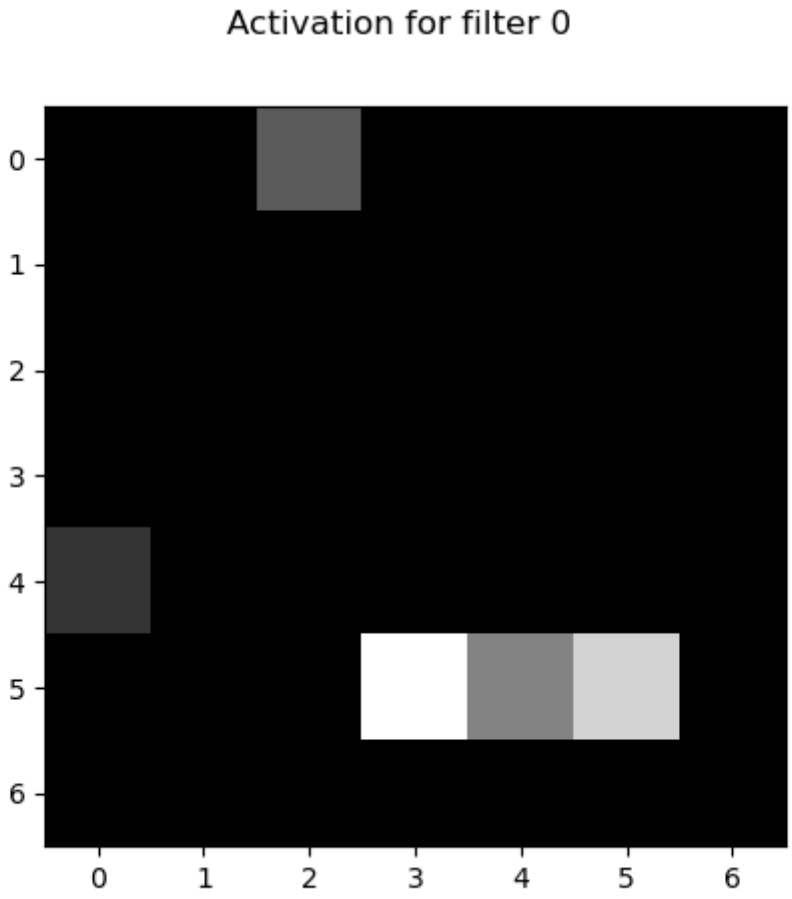
From the visualization of the kernel it seems like it maybe should detect diagonal lines. And this seems that in the activation the diagonal lines on the zebra is somewhat activated. The activation image is very gray which suggest that this kernal generally did not find what it is designed to detect.

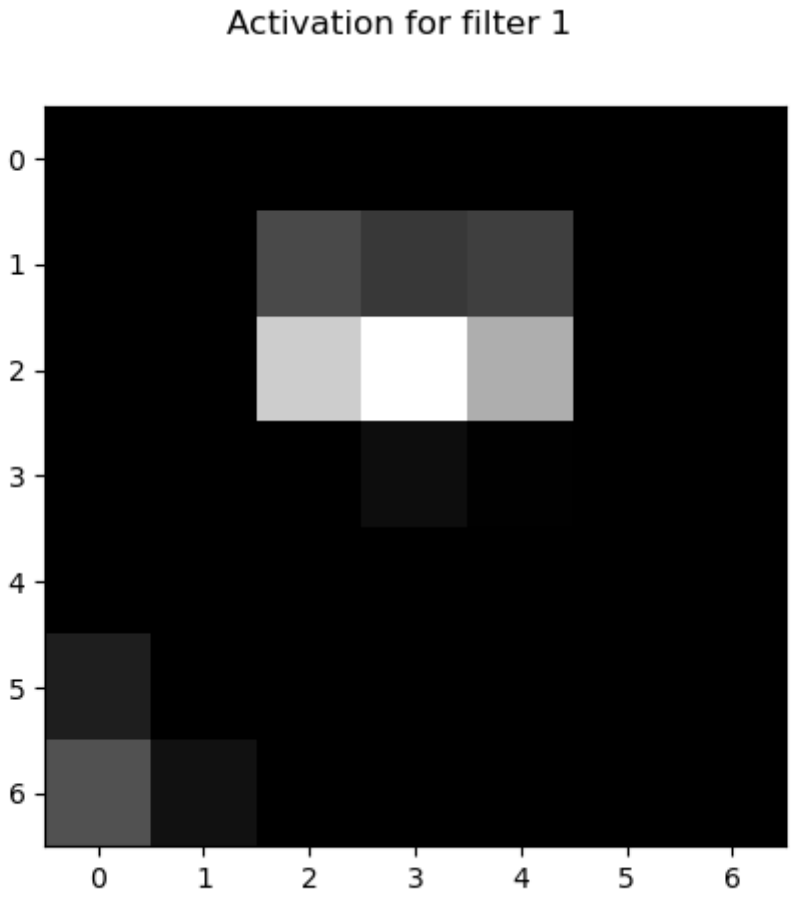


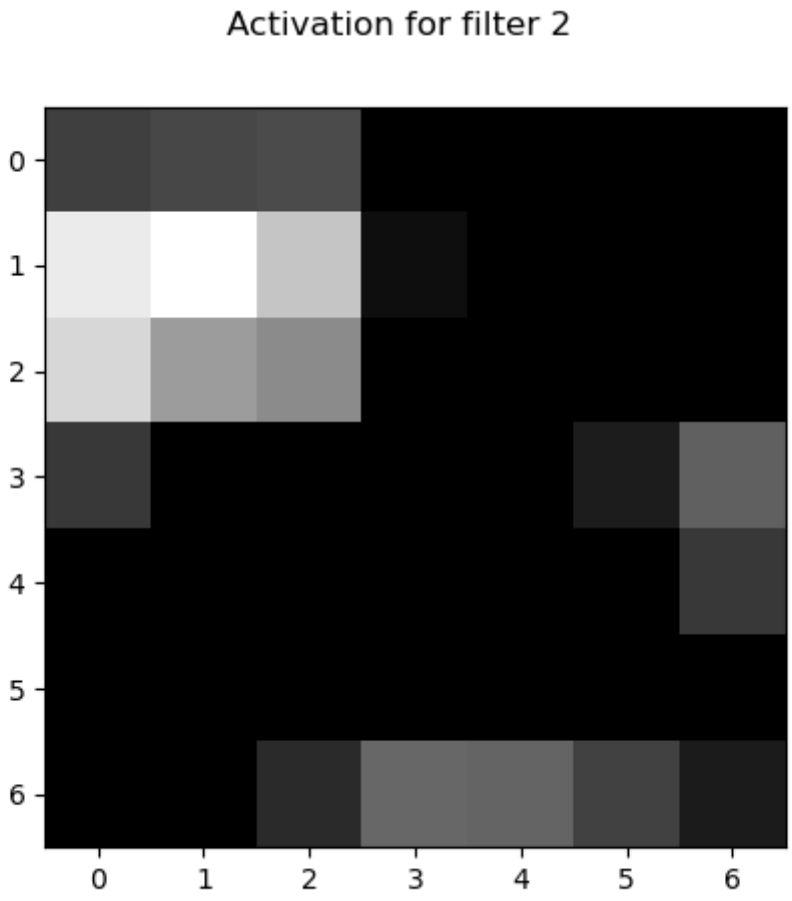
The last kernel (52) seems to be detecting green/yellow in the same way that kernel 32 detects blue. The yellow grass in the image has a high activation in the activated image.

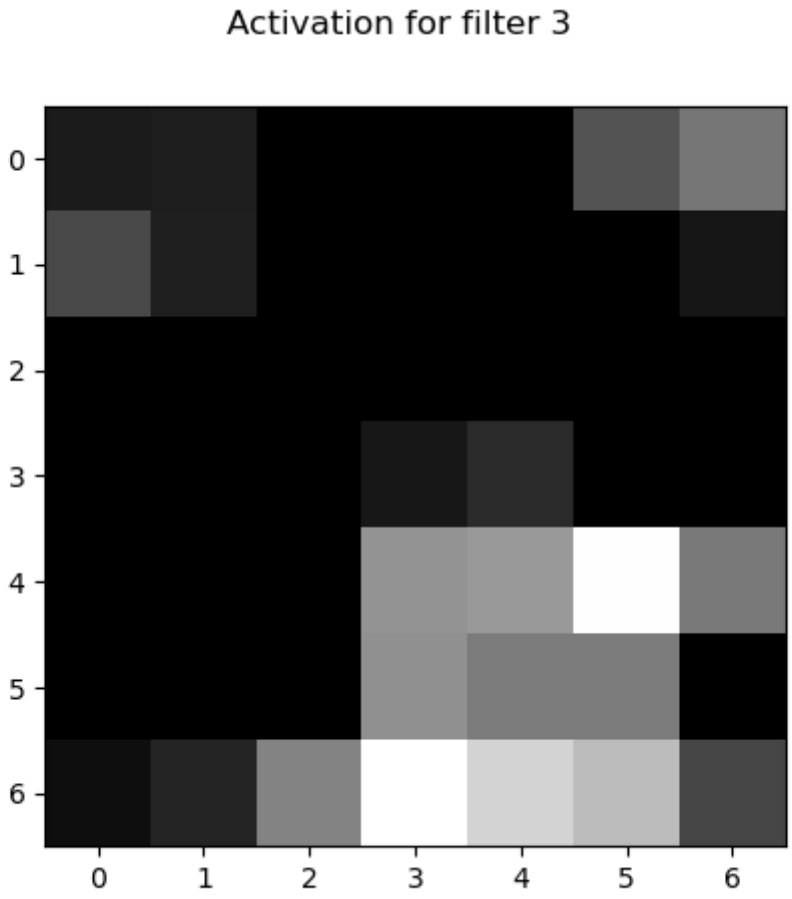
Task 4c)

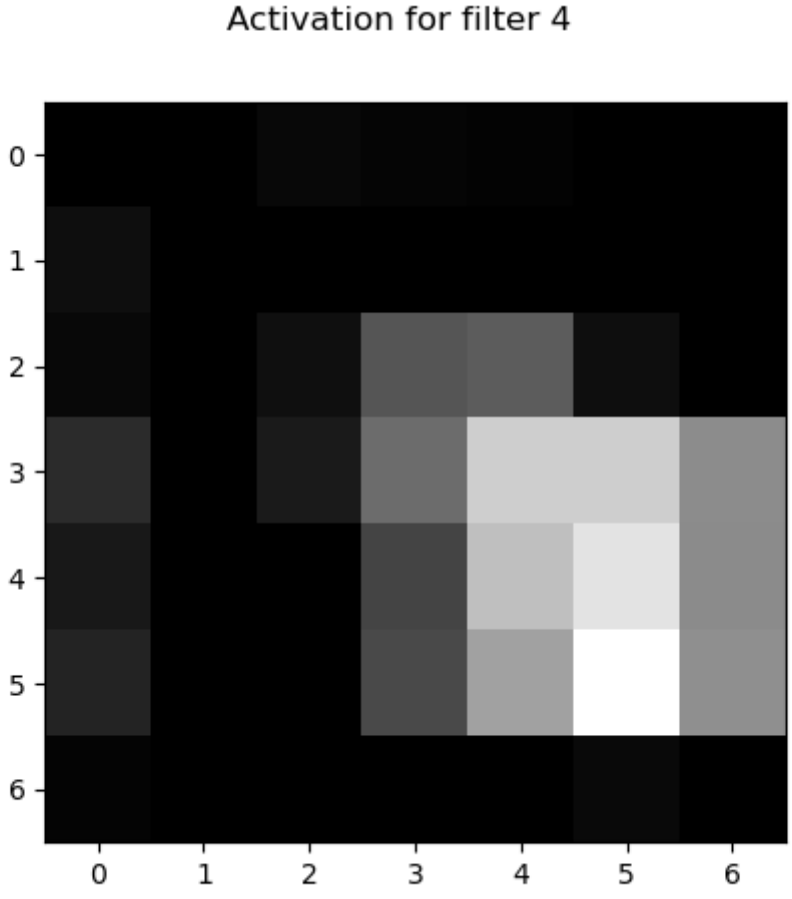
It is harder to see what these activations represent. They have been though many layers which possibly kan make the reason for activation very complex (this is some of the reason why one want to use deep NNs). Generally the activations seems to have high contrast (either close to white or black) and some of the activations have quite sparse activations (few white pixels).

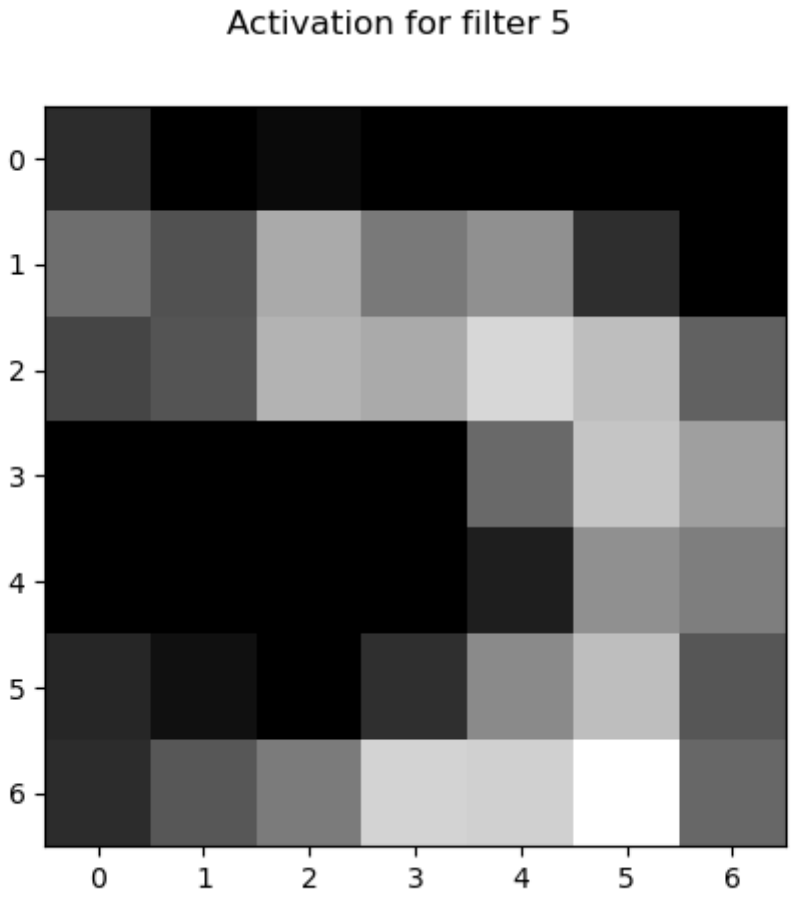


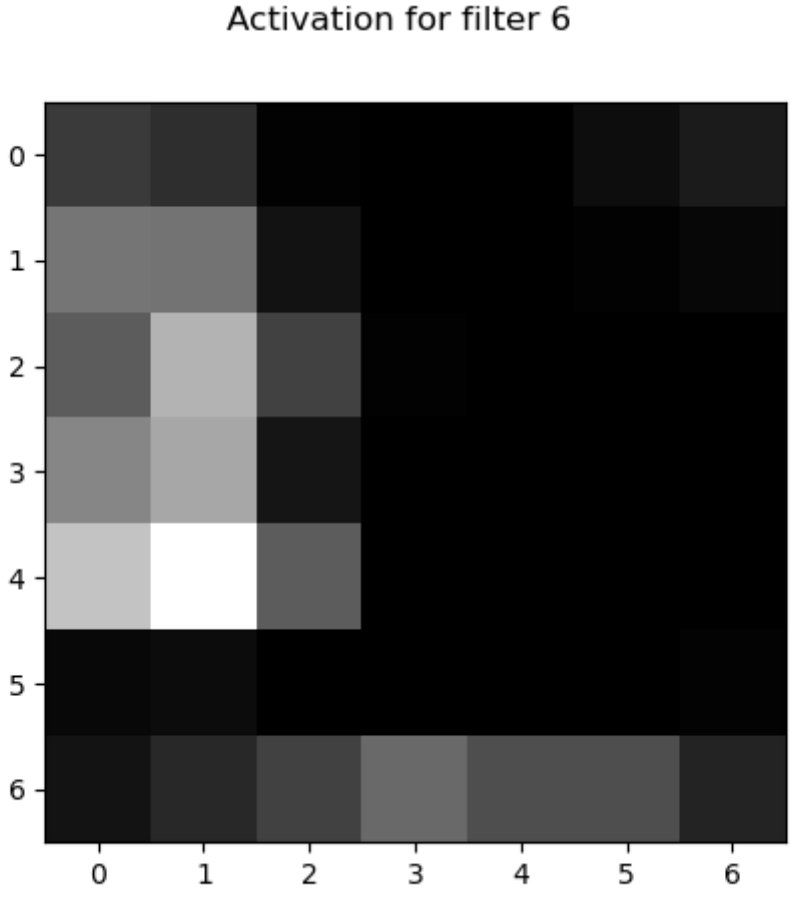

Activation for filter 1
0
1
2
3
4
5
6
0
1
2
3
4
5
6

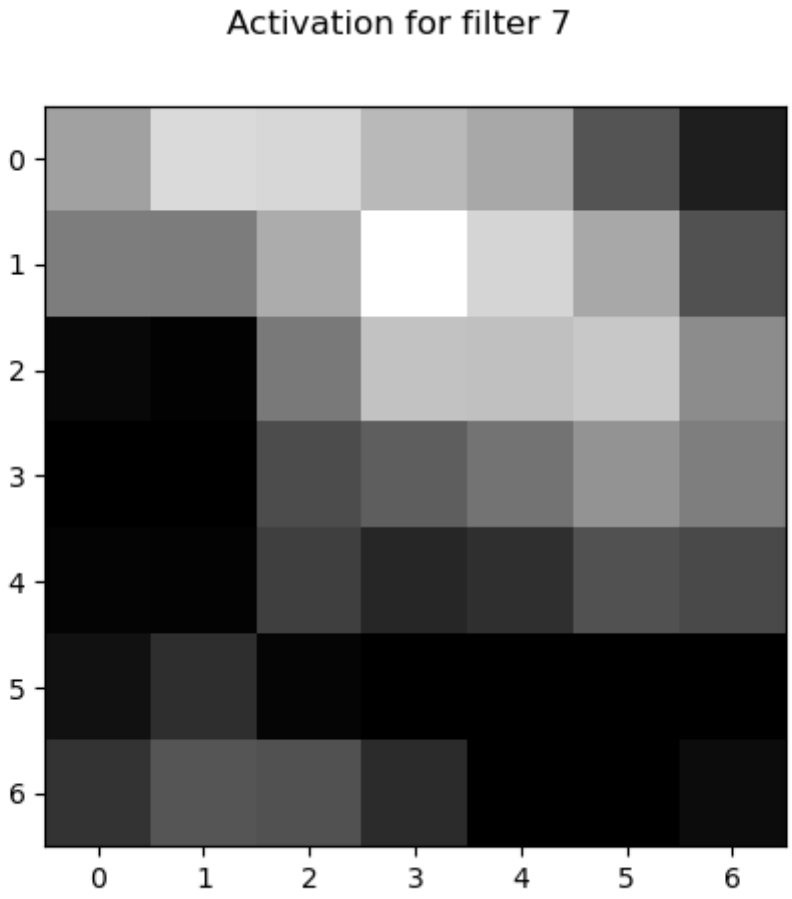

Activation for filter 2
0
1
2
3
4
5
6
0
1
2
3
4
5
6

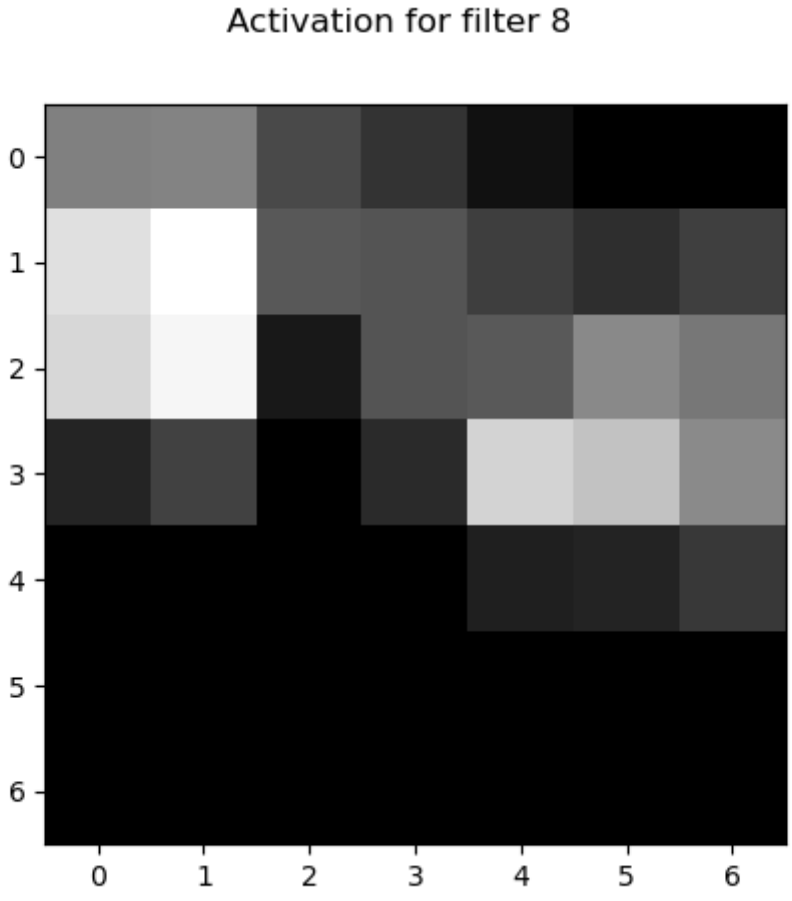

Activation for filter 3
0
1
2
3
4
5
6
0
1
2
3
4
5
6

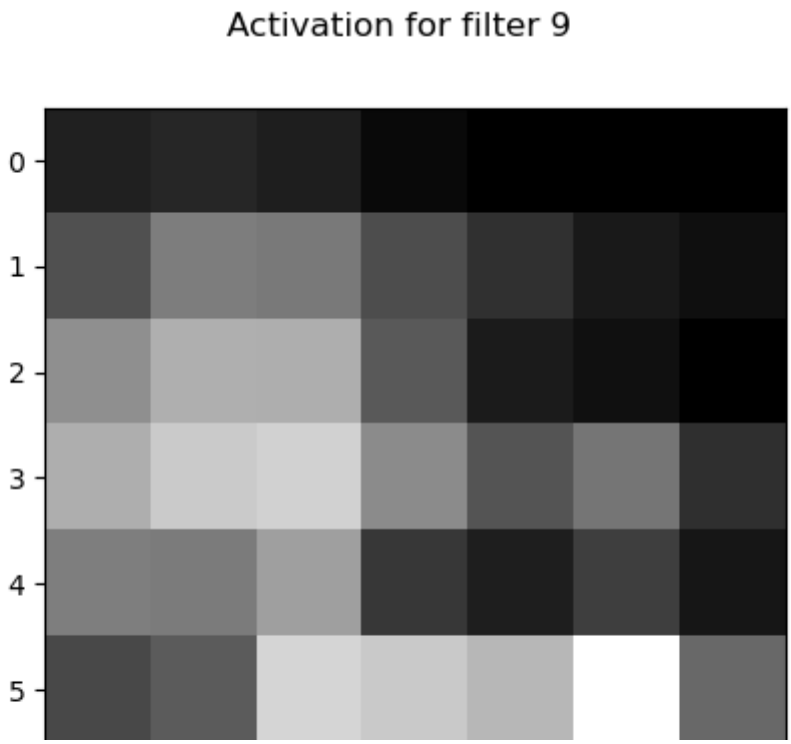

Activation for filter 4
0
1
2
3
4
5
6
0
1
2
3
4
5
6


Activation for filter 5
0
1
2
3
4
5
6
0
1
2
3
4
5
6


Activation for filter 6
0
1
2
3
4
5
6
0
1
2
3
4
5
6


Activation for filter 7
0
1
2
3
4
5
6
0
1
2
3
4
5
6


Activation for filter 8
0
1
2
3
4
5
6
0
1
2
3
4
5
6


Activation for filter 9
0
1
2
3
4
5