TDT4195: IP Assignment 3

Embla Flatlandsmo

November 2021

1 Task 1: Theory

1.1 Task 1a

To perform spatial convolution, we must rotate the kernel by 180 degrees, then we can perform the easier-to-understand (to me, at least) spatial correlation. So, the kernel becomes

$$K = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{1}$$

Since the convolved image should be 3×5, I choose to zero-pad the image by 1 on both vertical and horizontal. This means that we perform our convolution on:

$$K = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 3 & 1 & 0 \\ 0 & 3 & 2 & 0 & 7 & 0 & 0 \\ 0 & 0 & 6 & 1 & 1 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

For a pixel in the original image, we then perform the correlation using the 3x3 pixels surrounding it.
To make it a bit more clear what I'm doing, here is how I compute the top-left value in the original image:

$$out = 1 * 0 + 0 * 0 + (-1) * 0 + 2 * 0 + 0 * 1 + (-2) * 0 + 1 * 0 + 0 * 3 + (-1) * 2 = -2$$

Another example: for the centermost pixel in the original image image *(it has a value of 0)*:

$$out = 1 * 0 + 0 * 2 + (-1) * 3 + 2 * 2 + 0 * 0 + (-2) * 7 + 1 * 6 + 0 * 1 + (-1) * 1 = -8$$

Performing this correlation on all the image values yields

$$K = \begin{bmatrix} -2 & 1 & -11 & 2 & 13 \\ -10 & 4 & -8 & -2 & 18 \\ -14 & 1 & 5 & -6 & 9 \end{bmatrix} \tag{3}$$

1.2 Task 1b

I think **max pooling** contributes to this, as it will output the same pixel intensity regardless of where the pixel is located. Of course, this only applies if the pixel with the max intensity is within the same region as it was before the shift.

1.3 Task 1c

Please note that I'm explaining this very thoroughly so I can revisit this if I ever forget.
The convolutional layer input size is $H_1 \times W_1 \times C_1$ and has output size $H_2 \times W_2 \times C_2$

$$W_2 = (W_1 - F_W + 2P_W)/S_W + 1 \tag{4}$$

$$H_2 = (H_1 - F_H + 2P_H)/S_H + 1 \tag{5}$$

Here, F_W and F_H is the width and height of the filter, P_W and P_H is the padding in the width and height dimension, S_W and S_H is the stride of the convolution operation in the width and height dimension.
In our case, we have $F_W = F_H = 5$, $S_W = S_H = 1$, $H_1 = H_2$ and $W_1 = W_2$. By rearranging (4) and (5) to isolate the padding width and height, we end up with (6) and (7). Inserting the numerical values into this yields the answers in (8)

$$P_W = ((W_2 - 1)S_W - W_1 + F_W)/2 \tag{6}$$

$$P_H = ((H_2 - 1)S_H - H_1 + F_H)/2 \tag{7}$$

$$P_W = P_H = 2 \tag{8}$$

Which means that the correct amount of padding is (**Height**) $\times$ (**Width**) = $\mathbf{2 \times 2}$

1.4 Task 1d

Again, rearranging (4) and (5) forms (9):

$$\begin{aligned} F_W &= W_1 + 2P_W - (W_2 - 1)S_W \\ F_H &= H_1 + P_H - (H_2 - 1)S_H \end{aligned} \tag{9}$$

Inserting the numerical values $W_1 = H_1 = 512$, $W_2 = H_2 = 504$, $S_W = S_H = 1$ and $P_W = P_H = 0$ yields $F_W = F_H = 9$, which is means that **the answer is $\mathbf{9 \times 9}$**

1.5 Task 1e

With a stride length of 2, the spatial dimension (both height and width) of the pooled feature maps is $504/2 = 252$, which means that **the answer is $\mathbf{252 \times 252}$**

1.6 Task 1f

Again using the equations (4) and (5), this time with $H_1 = W_1 = 252$, $S_W = S_H = 1$, $F_W = F_H = 3$, $P_W = P_H = 0$ yields $W_2 = H_2 = 250$ which means that **the answer is $\mathbf{250 \times 250}$**

1.7 Task 1g

Each filter in any of the convolutional layers has $F_H \times F_W \times C$ weights. The max pool layers do not have any weights. From the table in the assignment text, I extracted the relevant pieces of information and have as such calculated the number of weights in the convolutional layers, seen in rows layer 1-3 of Table 1.

For the fully connected layers, we know that the number of weights is $numInputs \times numOutputs$ and that the number of biases if equal to $numOutputs$. To calculate the number of inputs to the first fully connected layer, I use two observations:

1. The convolutional layers do not change the dimensions of the output image
2. The MaxPool2D layers halve the dimensions of the output image.

Since we have 3 MaxPool2D layers, we simply need to divide the input $H \times W$ dimensions by 8. This means that the image output in layer 3 is 4×4.
Since there are 128 filters, the flattening layer will transform the output into a vector of length $4 * 4 * 128$. As such, the number of inputs to the 4th layer is $4 * 4 * 128$, as seen in Layer 4 of Table 1. *numOutputs* for each of the fully connected layers is given by the entry "Number of Hidden Units/Filters" of the assignment table.

By summing all the number of weights and number of biases from Table 1, we end up with $2400 + 32 + 51200 + 64 + 204800 + 128 + 131072 + 64 + 640 + 10 =$ **390410 total parameters**.

Convolutional layers						
Layer	C_{in}	F_W	F_H	C_{out}	Number of weights ($C_{in} \times F_W \times F_H \times C_{out}$)	Number of biases (C_{out})
1	3 (RGB)	5	5	32	2400	32
2	32	5	5	64	51200	64
3	64	5	5	128	204800	128
Fully connected layers						
Layer	C_{in}	N/A	N/A	C_{out}	Number of weights ($C_{in} \times C_{out}$)	Number of biases (C_{out})
4	128*4*4	-	-	64	128*4*4*64=131072	64
5	64	-	-	10	64*10=640	10

Table 1: Task 1e: Number of weights and biases for the neural network

2 Task 2: Convolutional Neural Networks

2.1 Task 2a

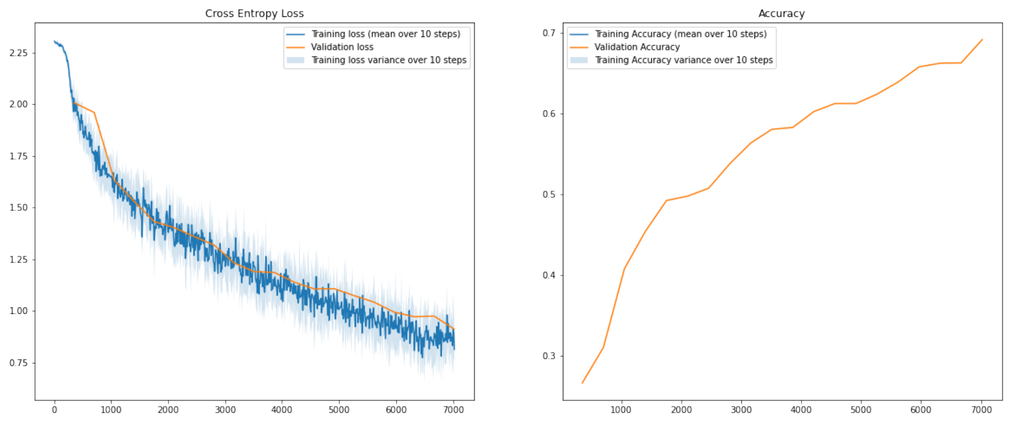

Figure 1: Task 2a: Loss and Accuracy for CNN

2.2 Task 2b

```
Train Accuracy:  0.7233508179231863
Test Accuracy:  0.6754
Validation Accuracy:  0.6796
```

3 Task 3: Deep Convolutional Network for Image Classification

3.1 Task 3a

I chose to name my two models Apple and Banana because they are both tasty fruits.
Table 2 shows some general information about the way that the networks were trained and initialized.

Model	**Apple**	**Banana**
Optimizer	SGD with learning rate 10^{-2}	Adam with learning rate=10^{-3}
Batch size	64	64
Weight initialization	PyTorch default	PyTorch default
Image transform	Normalization from Task 2	Normalize with means[0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]

Table 2: Task 3a: Description of various settings

For Table 3 (Apple), I tested out one of the other network architectures described in the assignment text. For Table 4, I tried to modify the network I implemented in Task 2 to get better performance. I implemented batch normalization to improve the learning rate. I also did dropout in some places to help prevent overfitting.

Layer	Layer Type	Number of Hidden Units / Number of Filters	Activation Function
1	Conv2D	32	ReLU
1	Conv2D	32	ReLU
1	Batch Normalization	-	-
1	MaxPool2D	-	-
2	Conv2D	64	ReLU
2	Conv2D	64	ReLU
2	Batch Normalization	-	-
2	MaxPool2D	-	-
3	Conv2D	128	ReLU
3	Conv2D	128	ReLU
3	Batch Normalization	-	-
3	MaxPool2D	-	-
	Flatten	-	-
4	Fully-Connected	64	ReLU
5	Fully-Connected	10	Softmax

Table 3: Task 3a: Architecture for Apple

Layer	Layer Type	Number of Hidden Units / Number of Filters	Activation Function
1	Conv2D	32	ReLU
1	Batch Normalization	-	-
1	MaxPool2D	-	-
2	Conv2D	64	ReLU
2	Dropout with p=0.1	-	-
2	Batch Normalization	-	-
2	MaxPool2D	-	-
3	Conv2D	128	ReLU
3	Batch Normalization	-	-
3	MaxPool2D	-	-
4	Conv2D	256	ReLU
4	Batch Normalization	-	-
4	MaxPool2D	-	-
	Flatten	-	-
5	Fully-Connected	64	ReLU
5	Dropout with p=0.1	-	-
5	Batch Normalization	-	-
6	Fully-Connected	10	Softmax

Table 4: Task 3a: Architecture for Banana

3.2 Task 3b

As we can see in Table 5, the first model (Apple) is the one who performed the best.

	Apple	Banana
Final Train Loss	0.16401002054257718	0.3243910984625009
Final Train Accuracy	0.9481241109530584	0.8897137268847796
Final Validation Accuracy	0.7814	0.7536
Final Test Accuracy	0.7702	0.753

Table 5: Task 3b: An overview of the two model metrics

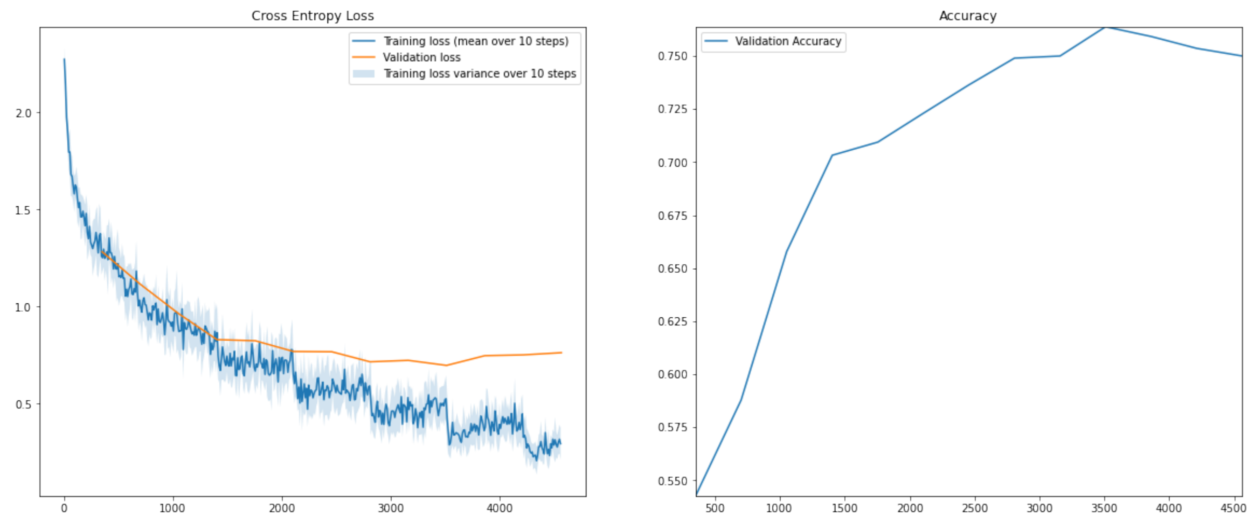

Figure 2: Task 3b: Plots from the model **Apple**

3.3 Task 3c

The most useful method was batch normalization. This really helped in making the neural network converge faster. I often had trouble with the network training for 10 epochs and stopping its training before it had started to converge. When implementing batch normalization, the network sometimes stopped training at epoch 5 which was really helpful! Since normalizing the input data to the network makes for better training, I can also see how normalizing the output for each layer (and thus the input to the subsequent layer) is helpful for training the network.
A method that did not work, was image augmentation. I learned that PyTorch has an AutoAugmentation feature for CIFAR10 which transforms the images in a way that makes sense for CIFAR10. However, this made the training take way longer to converge, so for the goal of getting 75% accuracy within 10 epochs, the CIFAR autoaugmentation was not helpful. I think it made the convergence take way longer because we are transforming the images which basically extends our dataset. We are able to constantly feed the network ”never-before-seen” images which in reality are just rotated/scaled/translated versions of earlier images.

3.4 Task 3d

As we can see in Figure 3, the loss and accuracy converges way faster after doing batch normalization.

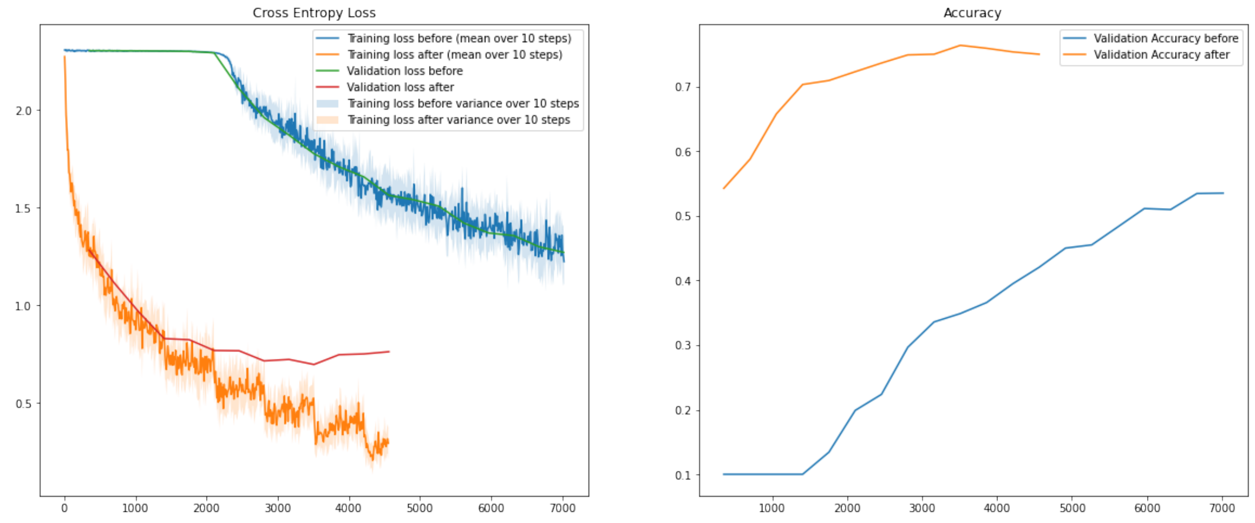

Figure 3: Task 3d: Comparison of the model **Apple** before and after batch normalization

3.5 Task 3e

I chose to extend the *Apple* model. First, I tried to use the same trainer as in *Banana*, that is, Adam optimizer with `lr = 1e-3`. This yielded somewhat improved results but not to 80%.
Then, I added dropout to some layers. Still, it was not able to reach 80% but it did stop early! Since I had some "training epochs to spare", I figured I could try to increase the network complexity a bit. So, I added a 4th convolutional layer with 256 filters. This brought the network over the 80% mark with the following metrics:

```
Train Accuracy:  0.9533028093883357
Test Accuracy:  0.8033
Validation Accuracy:  0.8104
Train Loss:  0.13700129737703934
Test Loss:  0.7065070639750001
Validation Loss:  0.6497920698757413
```

The final architecture is described in Table 6

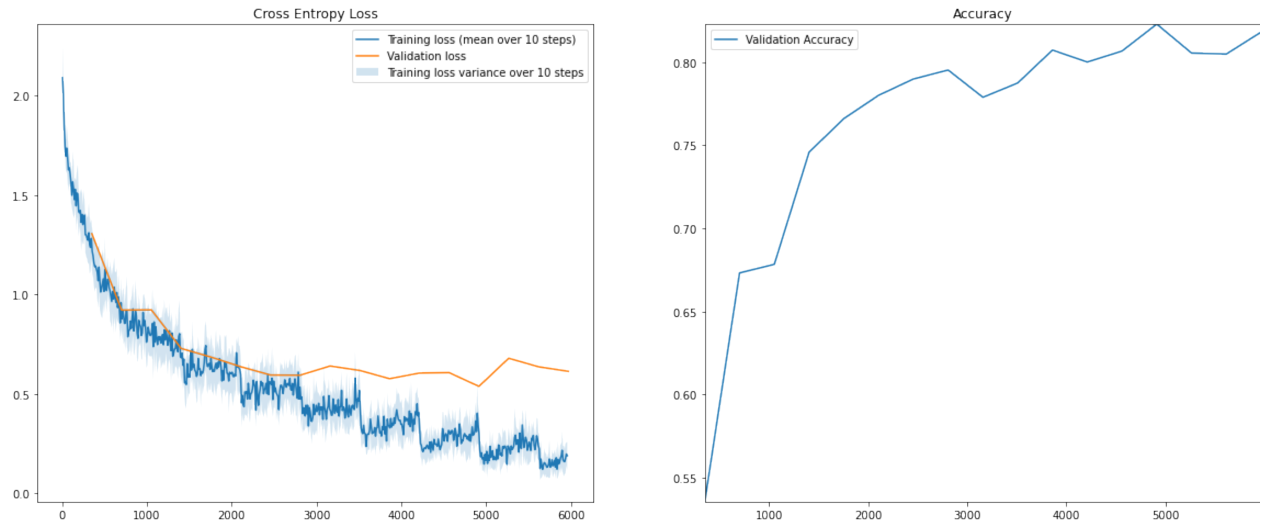

Figure 4: Task 3e: Loss and accuracy for final model

Layer	Layer Type	Number of Hidden Units / Number of Filters	Activation Function
1	Conv2D	32	ReLU
1	Conv2D	32	ReLU
1	Batch Normalization	-	-
1	MaxPool2D	-	-
2	Conv2D	64	ReLU
2	Conv2D	64	ReLU
2	Batch Normalization	-	-
2	MaxPool2D	-	-
3	Conv2D	128	ReLU
3	Conv2D	128	ReLU
3	Dropout with p=0.1	-	-
3	Batch Normalization	-	-
3	MaxPool2D	-	-
4	Conv2D	256	ReLU
4	Conv2D	256	ReLU
4	Batch Normalization	-	-
4	MaxPool2D	-	-
	Flatten	-	-
5	Fully-Connected	64	ReLU
5	Batch Normalization	-	-
6	Fully-Connected	10	Softmax

Table 6: Task 3e: Final network architecture

3.6 Task 3f

By looking at the loss and accuracy plots, I don't think we see any signs of overfitting. I think the fact that validation loss increases does not necessarily mean that it is overfitting, seeing as accuracy increases. However, if early stopping had not kicked in (at epoch 8 in my case), I think we would have seen overfitting.

4 Task 4

4.1 Task 4a

When implementing the neural network as described in task 4a, the final test accuracy was found to be `0.8882`.

4.2 Task 4b

Here, we seem to have horizontal and vertical direction-dependent edge finding (1st and 2nd image). There is also a diagonal direction-independent edge finding (4th image). The 3rd filter seems to look for colors with a lot of blue in them, and the 5th filter looks for color with a lot of green in them.

4.3 Task 4c

In the activations of Figure 7, it is hard to say what the network is ”seeing”. The only piece of information I can take from the figure is which general areas of the image it finds some features.

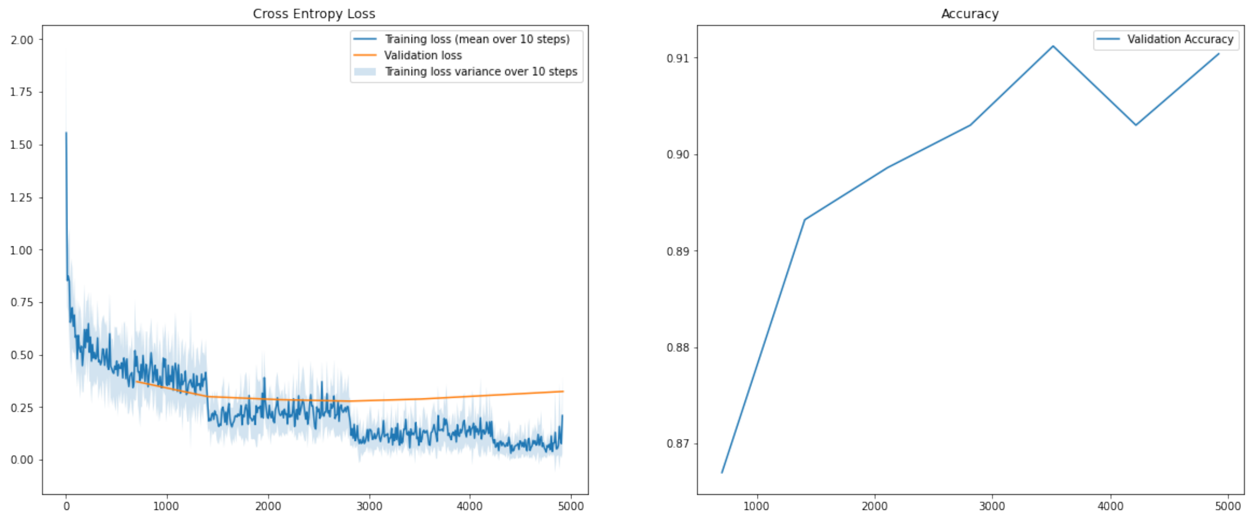

Figure 5: Task 4a: Loss and accuracy with ResNet18

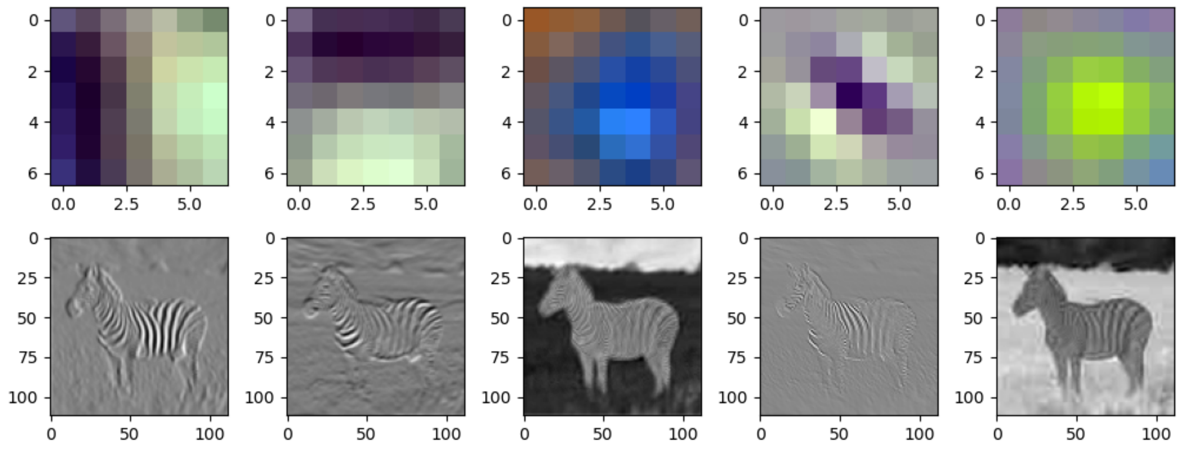

Figure 6: Task 4b: Weights of first convolutional layer in ResNet18 with indices [14, 26, 32, 49, 52]

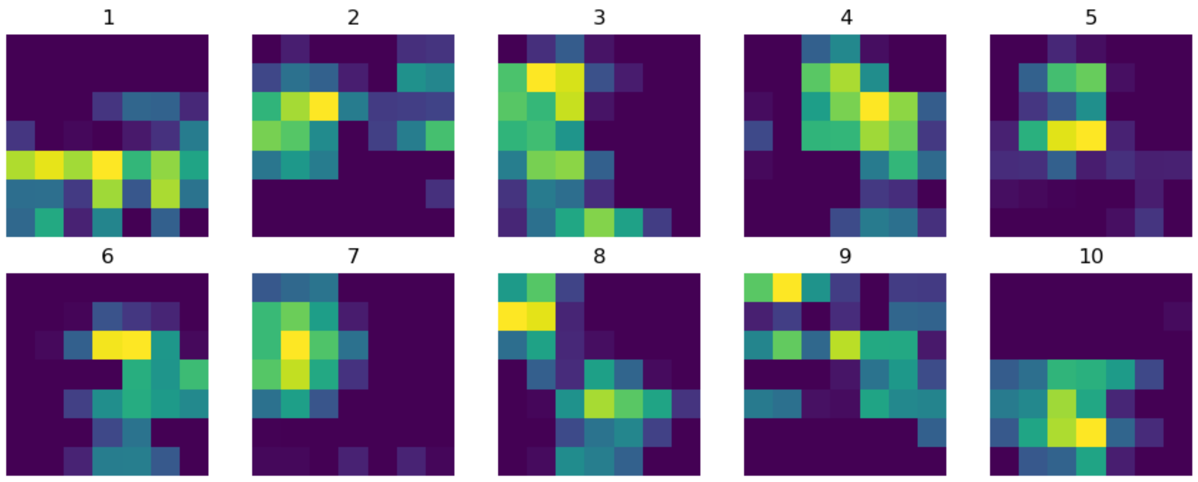

Figure 7: Task 4c: Visualization of first ten filter activations of the last convolutional layer in ResNet18