TDT4265 - Computer Vision and Deep Learning
Assignment 3

Jakob Vahlin
Kristian Stensgård

Contents

1 Theory

1.1 Task 1a

This subtask will discuss the spatial convolution of an image, $\mathbf{P}$ with a kernel, $\mathbf{K}$ represented by the following $[3 \times 5]$ and $[3 \times 3]$ matrices.

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 2 & 3 & 1 \\ 3 & 2 & 0 & 7 & 0 \\ 0 & 6 & 1 & 1 & 4 \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{1}$$

It is desirable that the resulting convolved image, $\mathbf{C}$ has dimension $[3 \times 5]$. Using a stride length of 1, one *layer* of zero padding must be added to the original image $\mathbf{P}$. The zero padded image, $\mathbf{P_z}$ will be represented by the following $[7 \times 5]$ matrix.

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

Where the red zeros represent the zero padding. Thus the resulting operation will be to convolve the elements in $[3 \times 3]$ "sub matrcies" of $\mathbf{P_z}$ with the kernel matrix, $\mathbf{K}$. To clarify what is meant by the term "sub matrix" equation (3) show the first and last sub matrices highlighted in blue.

$$\mathbf{P_z} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 3 & 1 & 0 \\ 0 & 3 & 2 & 0 & 7 & 0 & 0 \\ 0 & 0 & 6 & 1 & 1 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{P_z} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 3 & 1 & 0 \\ 0 & 3 & 2 & 0 & 7 & 0 & 0 \\ 0 & 0 & 6 & 1 & 1 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

With a stride length of 1, this blue square is shifted one step to the left (and one row down when the rightmost column has been reached) for each convolution with the kernel. The j, kth element in the convoled matrix $\mathbf{C}$, $c_{j,k}$ is given by

$$c_{jk} = \sum_{l=0}^{3} \sum_{m=0}^{3} k_{l,m} a_{j+l,k+m} \tag{4}$$

where $k_{l,m}$ represent the l, mth element in the Kernel matrix, $\mathbf{K}$ and $a_{x,y}$ represent the x, yth element in the zero padded input image, $\mathbf{P_z}$.

Thus from (4), the element c_{11} in the $\mathbf{C}$ matrix becomes

$$c_{11} = 0(-1) + 0(0) + 0(1) + 0(-2) + 1(0) + 0(2) + 0(-1) + 3(0) + 2(1) = 2 \tag{5}$$

Computing the rest of the elements in the following way finally gives the resulting covoluted image

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

1.2 Task 1b

It is the convolution layer that gives a Convolutional Neural Network its translational invariance. This is a result of the fact that each convolutional layer uses the same *kernel* (weights and biases). This means that all neurons in the convolutional layer detect the same *feature* of the image that is fed into the network. Thus, if an image of i.e a cat were to be shifted some pixels to the left, the network would still be able to handle this, since the *features* of the cat remains intact, and it is these features that will cause the activation of the neurons in a convolutional layer. What would happen is that the features describing the cat would be "discovered" a couple of pixels to the left.

1.3 Task 1c

Given an image with height H_1 and width W_1, that is fed into a Convolutional Neural Network (CNN) with a stride of S, kernel size of $[F_H \times F_W]$, then height and with of the first convolutional layer is given by

$$H_2 = \frac{H_1 - F_H + 2P_H}{S} + 1 \tag{7}$$
$$W_2 = \frac{W_1 - F_W + 2P_W}{S} + 1 \tag{8}$$

where P_W and P_H are the amount of zero-padding of the original image in the vertical and horizontal directions respectively.

To determine what amount of zero-padding needed for the output shape of the convolutional layer to be equal to the input image for a CNN with stride, $S = 1$, Kernel size of $[F_H = 5 \times F_W = 5]$, consider equations (7) and (8) with the values inserted.

$$H_2 = H_2 - 5 + 2P_H + 1 \tag{9}$$
$$W_2 = W_2 - 5 + 2P_W + 1 \tag{10}$$

Note that $H_2 = H_1$ and $W_2 = W_1$, since the aim of the task is to achieve the same dimensions. Solving equations (9) and (10) w.r.t P_H and P_W respectively yields

$$\begin{aligned} 2P_H &= H_2 - H_2 + 5 - 1 \\ &= 5 - 1 \\ &= 4 \\ \implies P_H &= 2 \end{aligned} \qquad \begin{aligned} 2P_W &= W_2 - W_2 + 5 - 1 \\ &= 5 - 1 \\ &= 4 \\ \implies P_W &= 2 \end{aligned} \tag{11}$$

Thus, **two** layers of zero padding, both in the width direction and height direction, is needed to achieve equal input and output dimensions. Thus, a $[m \times n]$ input matrix will be expanded to the dimensions $[m + 4 \times n + 4]$, illustrated by the originally $[3 \times 3]$, zero padded matrix shown in (12)

$$\mathbf{X_I} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a & b & c & 0 & 0 \\ 0 & 0 & d & e & f & 0 & 0 \\ 0 & 0 & g & h & i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{12}$$

where red zeros indicate the zero-padding.

1.4 Task 1d

For this task, formula describing the dimensions of the layers in the network in equations (7) and (8) can again be used. For a network whose inputs are RGB colored images with dimensions $[H_1 = 514 \times W_1 = 514]$ without zero padding, and the feature maps in the first hidden convoluted layer are of dimensions $[H_2 = 504 \times W_2 = 504]$ with stride, $S = 1$, equations (7) and (8) becomes

$$504 = 512 - F_H + 0 + 1 \tag{13}$$
$$504 = 512 - F_W + 0 + 1 \tag{14}$$

Solving for F_H and F_W yields

$$\begin{aligned} & 504 = 512 - F_H + 0 + 1 & \quad & 504 = 512 - F_W + 0 + 1 \\ \implies & F_H = 9 & \implies & F_W = 9 \end{aligned} \tag{15}$$

Thus the Kernels are of dimensions $[9 \times 9]$.

1.5 Task 1e

Subsampling and pooling will produce feature maps of size $H \times W$ given by the same formulas for convolution, i.e Equation 7 and Equation 8. From task 1d the output of the first convolutional layer has dimensions 504×504. This is the input to the subsampling, and with a stride $S = 2$ with a neighbourhood(kernel?) $F = 2 \times 2$. The padding P is zero. Since the kernel is square and the input is square, the resulting feature map will also be square. Solving Equation 7 for the subsampling gives:

$$\begin{aligned} H_2 &= \frac{504 - 2 + (2 \cdot 0)}{2} + 1 \\ H_2 &= 252 \implies W_2 = 252 \end{aligned} \tag{16}$$

The spatial dimensions after the pooling is 252×252

1.6 Task 1f

For this task the same equations used in the three previous tasks will be used, namely Equation 7 and Equation 8. With subsampling between the two convolutional layers the input to the second convolutional layer has spatial dimensions 252×252. Using the same equations as previously, the output dimensions are:

$$\begin{aligned} H_2 &= \frac{252 - 3 + (2 \cdot 0)}{1} + 1 \\ H_2 &= 250 \implies W_2 = 250 \end{aligned} \tag{17}$$

The spatial dimensions after the second convolution are 250×250

1.7 Task 1g

The network specified in the assignment consists of 3 convolutional layers with a kernel with dimensions 5×5 and 32, 64 and 128 filters respectively. Lastly there is two fully connected layers with 64 and 10 hidden units.

The number of parameters in a convolutional layer is the total number of filters, times the number of parameters for each kernel associated with the filter. The number of parameters for a kernel is the kernel size plus a bias.

For the first layer in the network, there is a total of 32 filters. This means that there are 32 sets of unique weights and biases. Since the input image have 3 colour channels (RGB) and we have a square kernel of 5×5, the weights have $(5 \cdot 5 \cdot 3) + 1 = 76$ parameters, the $+1$ is the bias. This results in $76 \cdot 32 = 2432$ parameters for the first layer. Doing this for the rest of the network we get:

$$\begin{aligned} (5 \cdot 5 \cdot 32) + 1) \cdot 64 &= 51,264 \\ (5 \cdot 5 \cdot 64) + 1) \cdot 128 &= 204,928 \end{aligned} \tag{18}$$

For the fully connected layers we have one weight matrix per layer with dimensions defined by the input layer and the number of hidden units in the layer. The spatial dimensions in each feature map after the last convolutional layer is 4×4 this follows from calculations using the same approach as in the previous tasks. Additionally, there are biases in the input and hidden layer of the fully connected network. Thus the input dimension is increased by one. This gives

$$\begin{aligned} ((128 \cdot 4 \cdot 4) + 1) \cdot 64 &= 131,136 \\ (64 + 1) \cdot 10 &= 650 \end{aligned} \tag{19}$$

Thus the total number of parameters in the network is:

$$2432 + 51,264 + 204,928 + 131,136 + 650 = 390,410 \tag{20}$$

2 Task 2

In this task a Convolutional Neural Network has been implemented and trained on the CIFAR-10 dataset, with 50,000 training images and 10,000 test images. 10% of the training images has been used as a validation set and was not used for the training. The network is described in Table 1.

2.1 Task 2a & 2b: Training and validation loss and accuracy.

The network was trained on the CIFAR-10 dataset. A plot of the resulting training and validation loss as well as the validation accuracy is shown in Figure 1 and Figure 2.

The network had a minimum validation loss of 0.85 and a peak validation accuracy of 74.6%. After training the model a average loss of 0.798 and average accuracy of 73.8% was acheived on the test set using the same version of the network that got the best performance on the validation set. For the training set the average loss was 0.369 and the average accuracy was 87.7% when the trained network was used.

Table 1: Description of the CNN in task 2.

Layer	**Layer Type**	**Number of Hidden Units**	**Activation Function**
1	Conv2D	32	ReLU
1	MaxPool2D	-	-
2	Conv2D	64	ReLU
2	MaxPool2D	-	-
3	Conv2D	128	ReLU
3	MaxPool2d	-	-
	Flatten	-	-
4	Fully-Connected	64	ReLU
5	Fully-Connected	10	Softmax

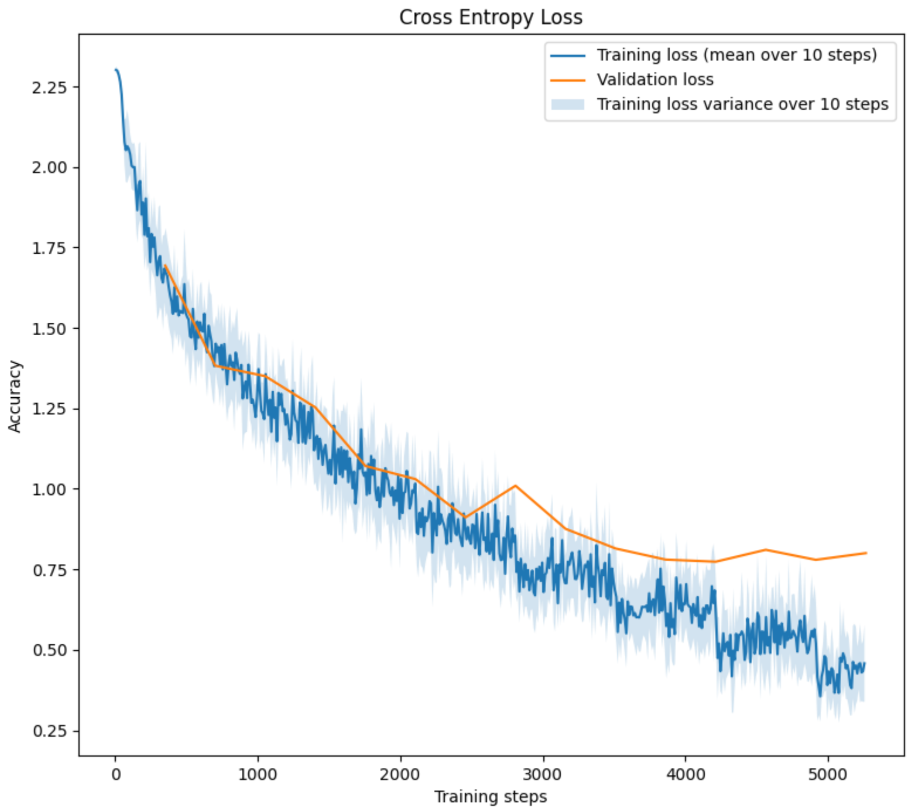

Figure 1: Training and validation loss for the network during training.

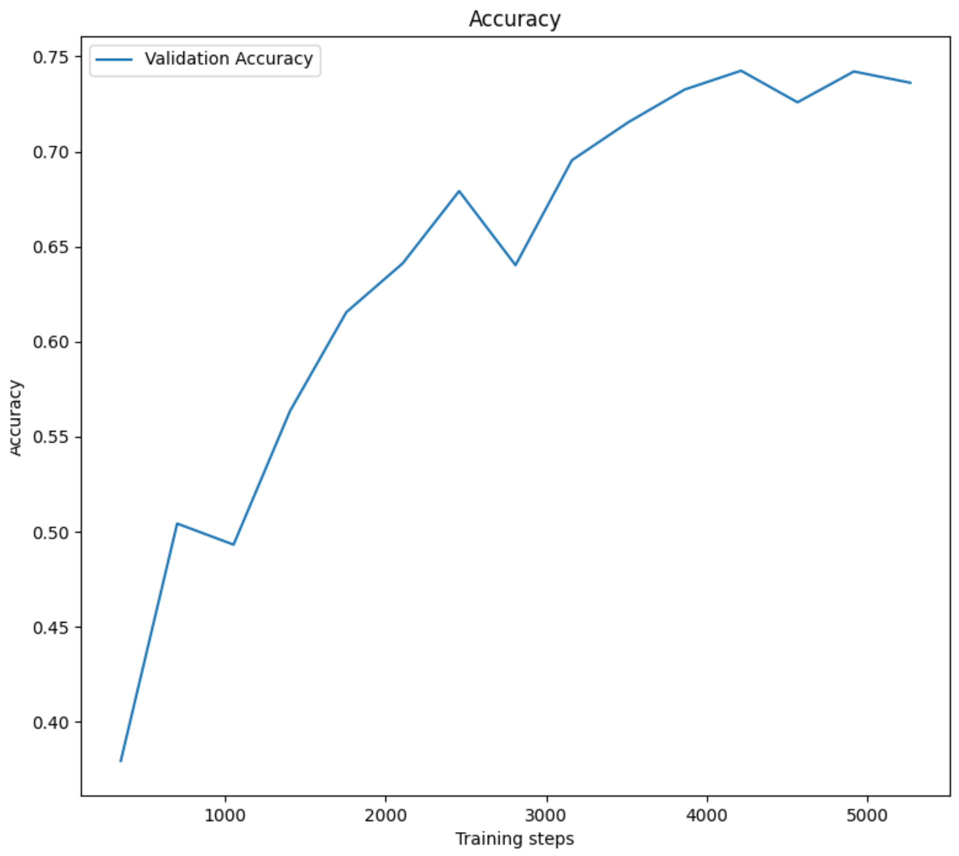

Figure 2: Validation accuracy for the network during training.

From the plot of the resulting cross entropy loss shown in Figure 1 one can observe that the validation loss seems to be on an upward trend as the training stops. This upward trend could be a sign of the model starting to overfit. Additionally the training loss tends towards zero, creating a "gap" between the validation and training loss, which supports the claim of overfitting.

3 Task 3

This task adresses the design of two different Convulotional Neural Networks using different architectures, hyperparameters, loss functions, and optimizers.

3.1 Task 3a: Description of the networks

A description of Network 1 is given in Table 2. Each convolutional layer has filter size of $[5 \times 5]$, with a padding of 2 and a stride of 1. The flatten layer takes an image with shape (Height) $\times$ (Width) $\times$ (Number of Feature Maps), and flattens it to a single vector with size (Height) $\cdot$ (Width) $\cdot$ (Number of Feature Maps). The MaxPool layers have filters with size $[2 \times 2]$, stride of 2 and no zero-padding.

The training details of Network 1 are outlined in Table 3. In addition to the details outline in Table 3, data augmentation on the input data was used. The input images were cropped randomly, meaning that a random selection of the image was chosen and padding added to keep the dimensions intact. "Random horizontal flip" was also added, meaning that there was a 50% chance of the image being mirrored.

Table 2: The network architecture of Network 1. (task3$_{n}etwork1.py$)

Layer	Layer Type	Number of Hidden Units	Activation Function
1	Conv2D	32	ReLU
1	BatchNorm2D	-	-
2	Conv2D	64	ReLU
2	MaxPool2D	-	-
3	Conv2D	128	ReLU
3	BatchNorm2D	-	-
4	Conv2D	128	ReLU
4	Maxpool2D	-	-
4	Dropout2D	-	-
	Flatten	-	-
5	Fully-Connected	64	ReLU
5	BatchNorm1D	-	-
5	Dropout	-	-
6	Fully-Connected	64	ReLU
6	BatchNorm1D	-	-
6	Dropout	-	-
7	Fully-Connected	64	ReLU
7	BatchNorm1D	-	-
7	Dropout	-	-
8	Fully-Connected	64	ReLU
8	BatchNorm1D	-	-
8	Dropout	-	-
8	Fully-Connected	10	SoftMax

Table 3: Training parameters used when training Network 1.

Optimizer	Regularization	Batch size	Learning rate	Weight Init	Epochs	Early stop count
SGD	Dropout, rate = 0.05	64	0.04	Uniform	10	4

A description of Network 2 is given in Table 4. Each convolutional layer has filter size of $[3 \times 3]$, with a padding of 2 and a stride of 1. The flatten layer takes an image with shape (Height) $\times$ (Width) $\times$ (Number of Feature Maps), and flattens it to a single vector with size (Height) $\cdot$ (Width) $\cdot$ (Number of Feature Maps). The MaxPool layers have filters with size $[2 \times 2]$, stride of 2 and no zero-padding.

The training details of Network 2 are outlined in Table 5. In addition to the details outline in Table 5, data augmentation on the input data was used. The input images were cropped randomly, meaning that a random selection of the image was chosen and padding added to keep the dimensions intact. "Random horizontal flip" was also added, meaning that there was a 50% chance of the image being mirrored.

Table 4: The network architecture of Network 2. (task3_3.py)

Layer	Layer Type	Number of Hidden Units	Activation Function
1	Conv2D	32	ReLU
2	Conv2D	64	ReLU
2	BatchNorm2D	-	-
2	MaxPool2D	-	-
3	Conv2D	128	ReLU
3	BatchNorm2D	-	-
3	MaxPool2D	-	-
3	Dropout	-	-
	Flatten	-	-
5	Fully-Connected	64	ReLU
5	BatchNorm1D	-	-
8	Fully-Connected	10	SoftMax

Table 5: Training parameters used when training Network 2.

Optimizer	Regularization	Batch size	Learning rate	Weight Init	Epochs	Early stop count
SGD	Dropout, rate = 0.1	64	0.05	Uniform	10	4

3.2 Task 3b: Loss and accuracy results

The resulting final train loss, final train accuracy, final validation accuracy and average test accuracy for the two networks are outlined in Table 6

Table 6: Final train loss, final train accuracy, final validation accuracy and average test accuracy for the two networks.

Model	Train loss	Train accuracy	Validation accuracy	Test accuracy
Network 1	0.605	79.5 %	79.1 %	78.6 %
Network 2	0.546	80.1 %	78.4 %	77.5 %

The validation accuracy and training and validation loss from the best model, Network 1, are shown in Figure 4 and Figure 3 respectively.

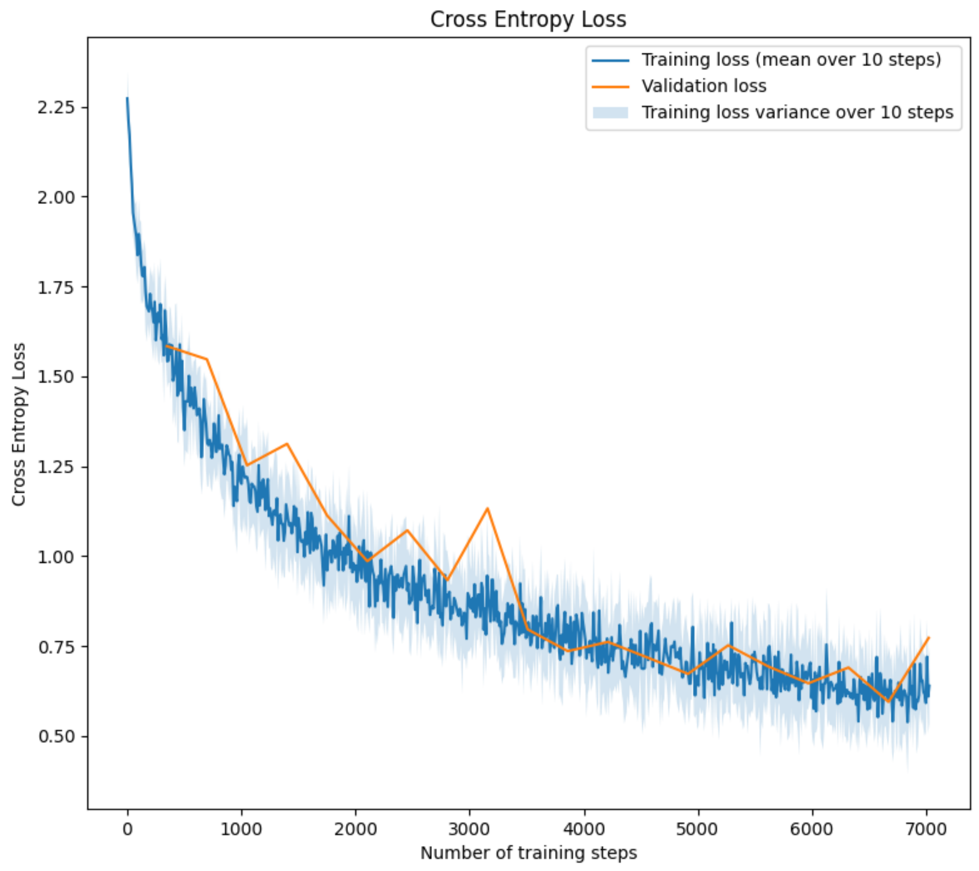

Figure 3: Training and validation loss for Network 1.

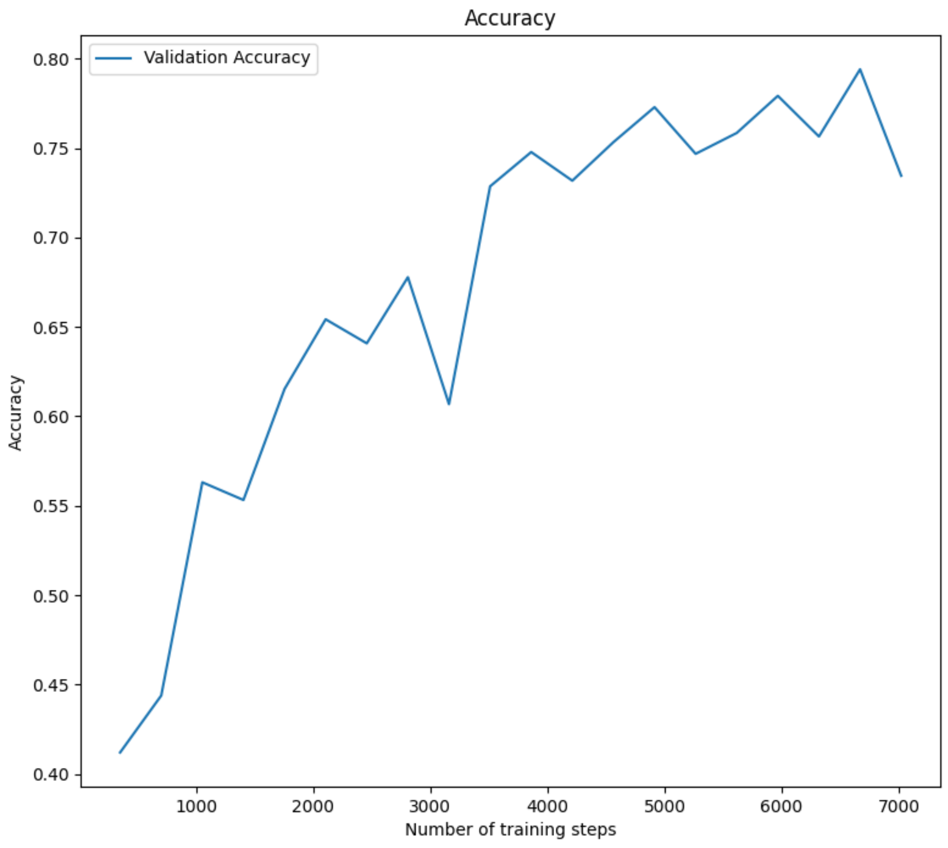

Figure 4: Validation accuracy for Network 1.

3.3 Task 3c: Useful methods

In an attempt to improve the networks performance, several different additions and alterations was done:

A lot of time was spent testing different network architectures. Both different number of filters, hidden units, and number of layers. After data augmentation, the network architecture was the most impactful. Overall, deeper networks, with more filters and more convolutional layers gave more successful results. However, if the networks became too deep, the results became worse. This is to be expected, as deeper netwtorks initially gives the model a better chance of improving on classifying the details, but if the network becomes too deep, the gradients will "vanish", and the model grows way too large for it to be possible to perform valuable training.

The most impactful increase in performance came after adding data augmentation. The performance increase can probably be explained by the fact that data augmentation slightly changes the input images. This creates a more diverse training set, resulting in a more robust model. However, augmenting the data too much gave worse results, probably due to the fact that changing the input *too* much creates a training set that is *too* diverse, and thus the model is unable to recognize patterns and draw worthy conclusions.

Different activation functions was tested including, tanh, sigmoid and leaky ReLU. However they did not improve the performance of the network. When testing these activation functions, different weight initializations was also tested to match the activation functions. The lack of success from the tanh and sigmoid activation functions could be a explained by their known issues with vanishing gradients.

Batch normalization seemed to have little to no effect on the performance of the network, altough the technique was not tested extensively. This might be a consequence of the "heavy" data augmentation used on the input data, meaning that batch normalizing will have little effect on the training of the network.

The Adam optimizer was tested in favour of SGD, but it didn't improve the performance of the network at the end of training, but it converged at approximately the same rate as SGD in the beginning. This is a well known fact for the Adam optimizer. In hindsight a weight decay could be added to the Adam optimizer and maybe improve the performance.

Experiments with dropout showed that it helps to reduce overfitting, but it should be used carefully as too much dropout resulted in very poor performance. This could be because if the model has a high drop out rate (meaning fewer connections between the layers, and fewer weights, resulting in a less comprehensive model), it missed out on some important characteristics that it might have learned about had it not had such a high drop out rate, leading to more generalization.

3.4 Task 3d: Illustrating the impact of most useful addition

The most impactful addition turned out to be the implementation of data augmentation. To illustrate the impact, the resulting train and validation loss before and after the implementation for Network 1, is shown in Figure 5

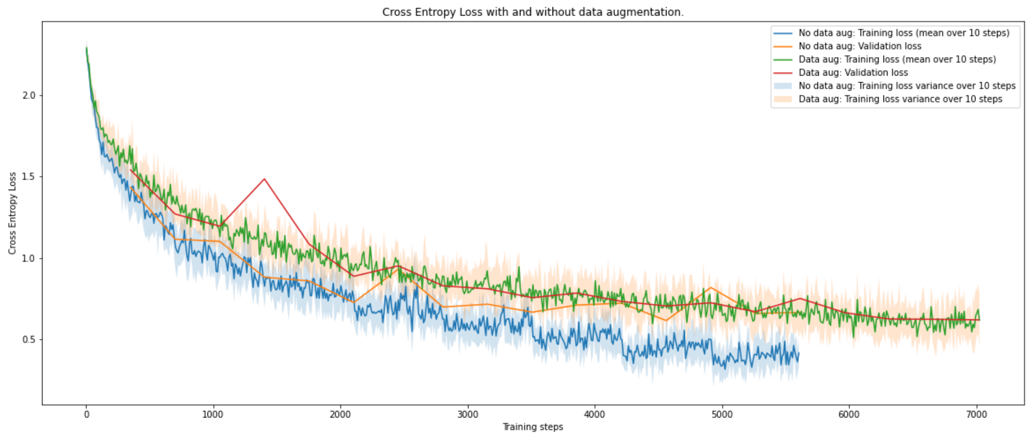

Figure 5: Training and validation loss with and without data augmentation.

3.5 Task 3e: Improving the best network

The best model from task 3a, Network 1, was further developed with the aim of reaching a test accuracy of more than 80%. In an effort to increase the test accuracy, the data augmentation was modified. The random cropping that was previously used, was removed and instead a random rotation of the image was introduced. Random rotation will rotate the image with a degree drawn uniformly from the range $[-10, 10]$.

In addition to the modified data augmentation, the learning rate was reduced from $4 \cdot 10^{-2}$ to $3.5 \cdot 10^{-2}$.

The network architecture remained mostly unchanged. However, dropout was removed from the fully connected network when performing classification.

A description of the improved network is given in Table 7. The modified training details are outlined in tab:netimprovedtrainparam.

With these changes implemented the network reached an average accuracy of80.5% on the test set. The performance metrics for the improved metrics are outlined in Table 9.

Table 7: The network architecture of the improved network. (task3.py)

Layer	Layer Type	Number of Hidden Units	Activation Function
1	Conv2D	32	ReLU
1	BatchNorm2D	-	-
2	Conv2D	64	ReLU
2	MaxPool2D	-	-
3	Conv2D	128	ReLU
3	BatchNorm2D	-	-
4	Conv2D	128	ReLU
4	Maxpool2D	-	-
4	Dropout2D	-	-
	Flatten	-	-
5	Fully-Connected	64	ReLU
5	BatchNorm1D	-	-
6	Fully-Connected	64	ReLU
6	BatchNorm1D	-	-
7	Fully-Connected	64	ReLU
7	BatchNorm1D	-	-
8	Fully-Connected	64	ReLU
8	BatchNorm1D	-	-
8	Fully-Connected	10	SoftMax

Table 8: Training parameters used when training the improved network.

Optimizer	Regularization	Batch size	Learning rate	Weight Init	Epochs	Early stop count
SGD	Dropout, rate = 0.05	64	0.035	Uniform	10	4

Table 9: Final train loss, final train accuracy, final validation accuracy and average test accuracy for the improved network

Model	Train loss	Train accuracy	Validation accuracy	Test accuracy
Improved	0.458	84.2 %	79.5	80.6 %

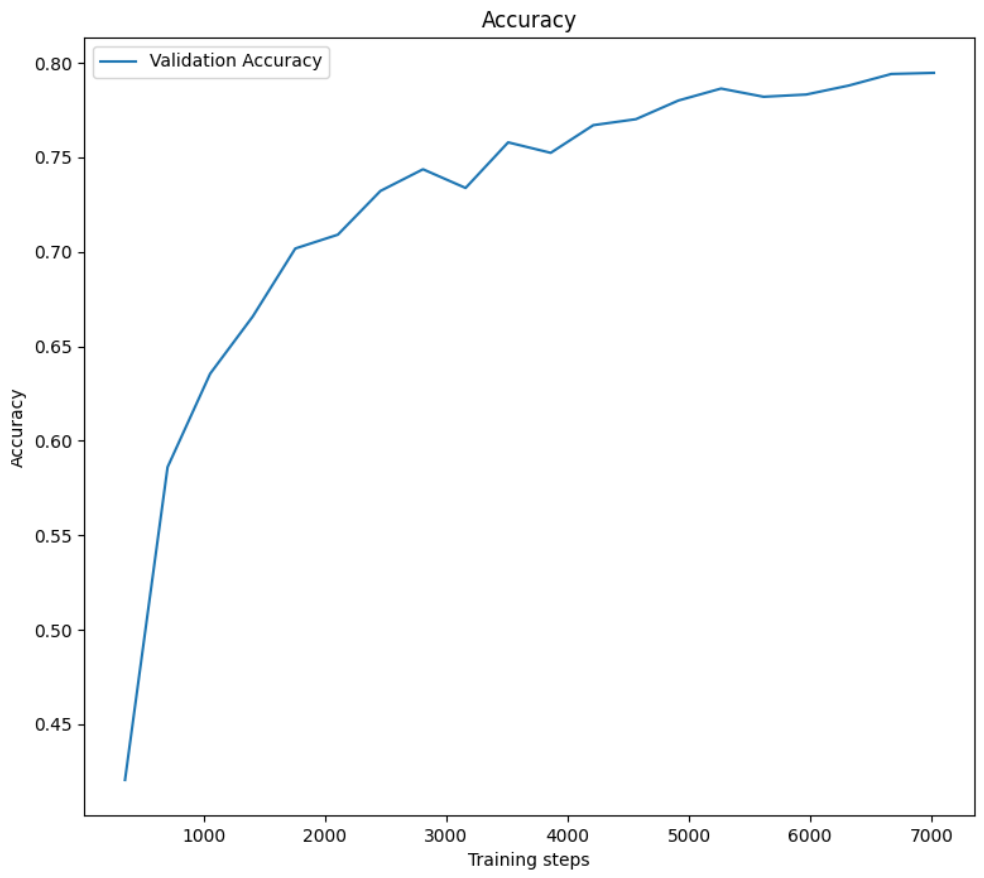

Figure 6: Validation accuracy for the improved model.

3.6 Task 3f: Signs of overfitting

From the plot of the resulting cross entropy loss shown in Figure 7 one can observe that the validation loss follows the loss for the training set. The validation loss is flattening a bit more than the training set. This not a direct sign of overfitting yet, as the "gap" between the two loss lines is not very significant.

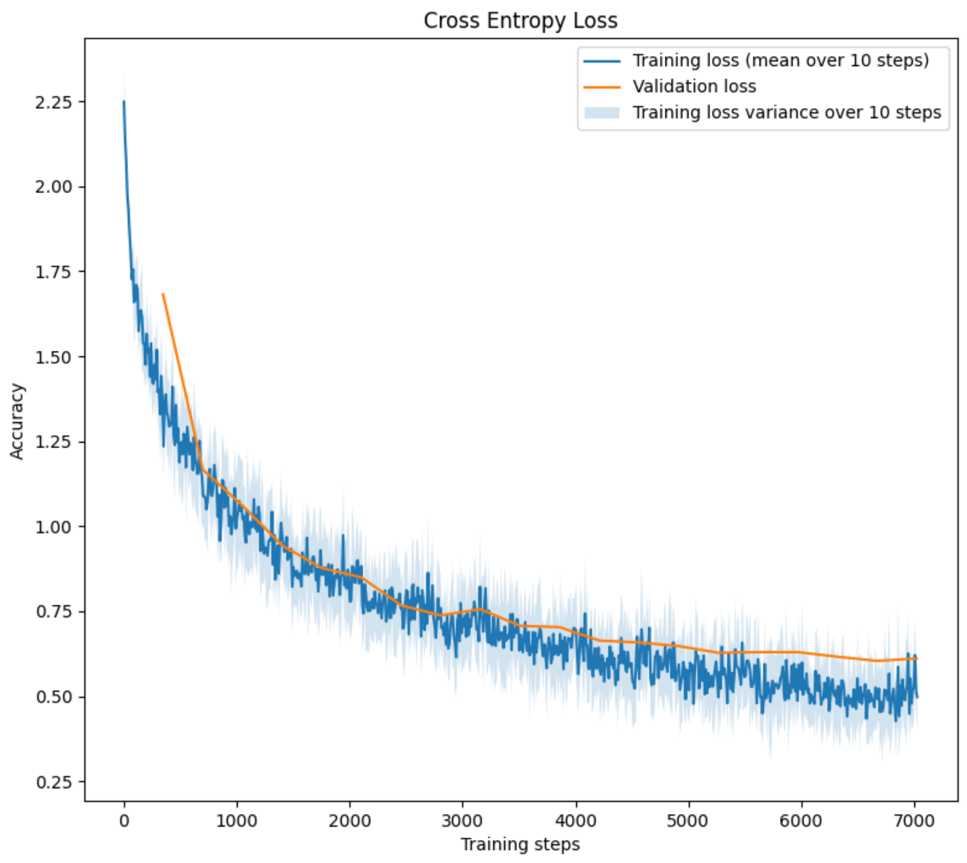

Figure 7: The training and validation loss for the improved network.

4 Task 4: : Transfer Learning with ResNet

4.1 Task 4a: Transfer learning with ResNet18

In this task a pre-trained version of ResNet called ResNet18 is used with the same training, validation and test sets as used in task 2 and task 3. The network is trained with the Adam optimizer with a learning rate of $5 \cdot 10^{-4}$ and a batch size of 32. The only data augmentation that was used was resizing the images to be correct size for the network and normalising. The network is trained for 4 epochs before early stopping kicks in.

The performance metrics of the network are shown in Table 10. Figure 8 show plots of the Cross Entropy Loss for both validation and training sets. Figure 9 show plot of the validation accuracy.

Table 10: Loss and accuracy for the training, validation and testing sets

Dataset	**Loss**	**Accuracy**
Training set	0.52	82.7 %
Validation set	0.48	84.3 %
Test set	0.55	82.0 %

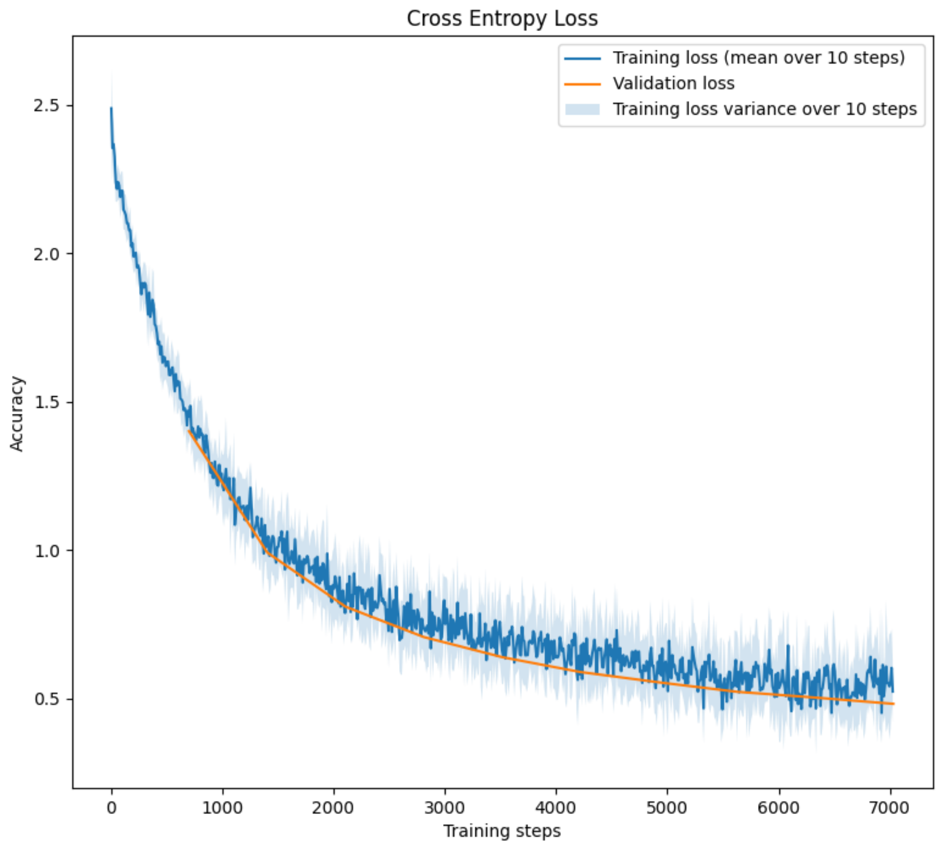

Figure 8: Training and validation loss for ResNet18.

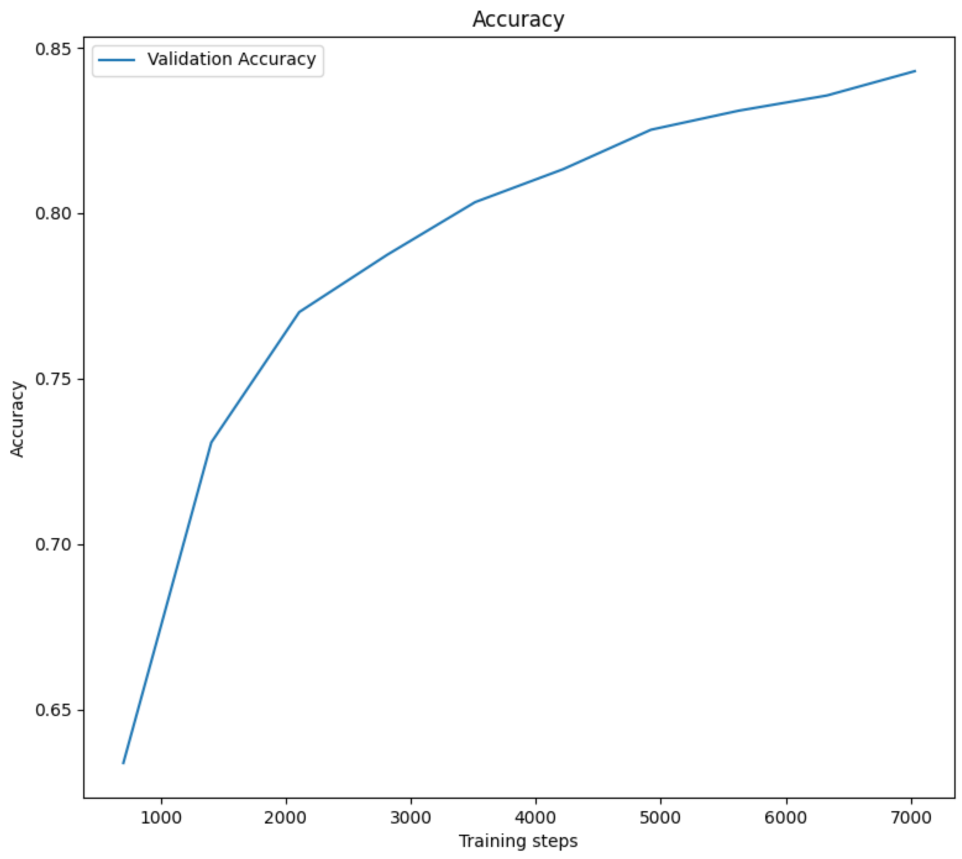

Figure 9: Validation accuracy for ResNet18

As we see from the results the pre-trained network performs way better than the networks in task 2 and task 3. Even with less training time on the CIFAR10 dataset.

4.2 Task 4b: Filter visualization

To visualize the feature extraction done by the filters, the weights and activation of the filters in the first convolutional layer of the trained model have been investigated. The image used as an example is an image of a zebra, shown in Figure 10. The feature extraction, beeing the weights and convolution outputs, from the filters in the first convolutional layer in the pretrained model is visualized in 11.

It is clear from Figure 11 that the two first weights detect vertical and horizontal lines using weights that resembles the Sobel kernel. The fourth kernel appears to detect diagonal lines, while the third and fifth kernel appears to be bell shapes meant to detect a specific colour.

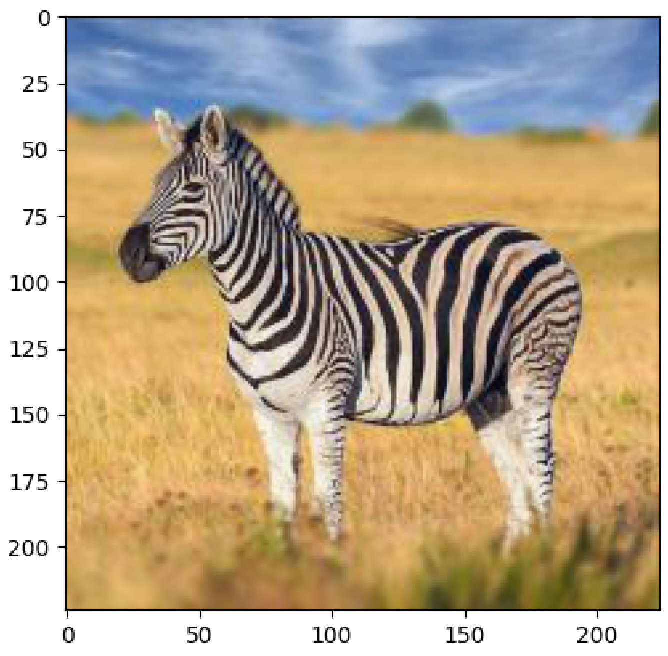

Figure 10: An image of a Zebra used when visualizing the feature extraction of the filters in the first convolutional layer.

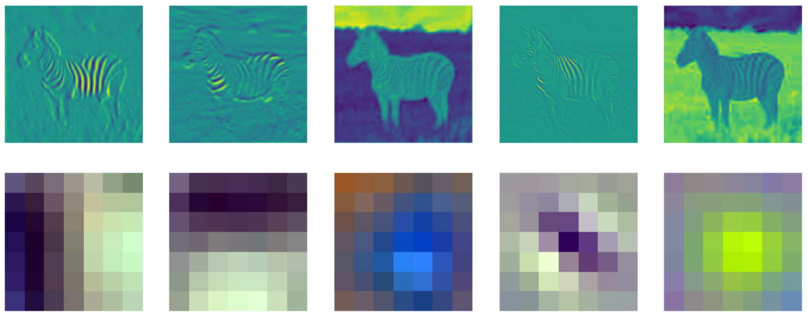

Figure 11: Feature extraction of from filters in the first convolutional layer. Top row: The convolution outputs. Bottom row: The corresponding weights that produced the output.

4.3 Task 4c

For this task the activations from the very last convolutional layer in ResNet18 is plotted in Figure 12. From looking at the activations themselves it is hard to find out what features the network has extracted. However by overlaying the input image as seen in Figure 13, we see that the network puts emphasis on the legs and on the head of the zebra.

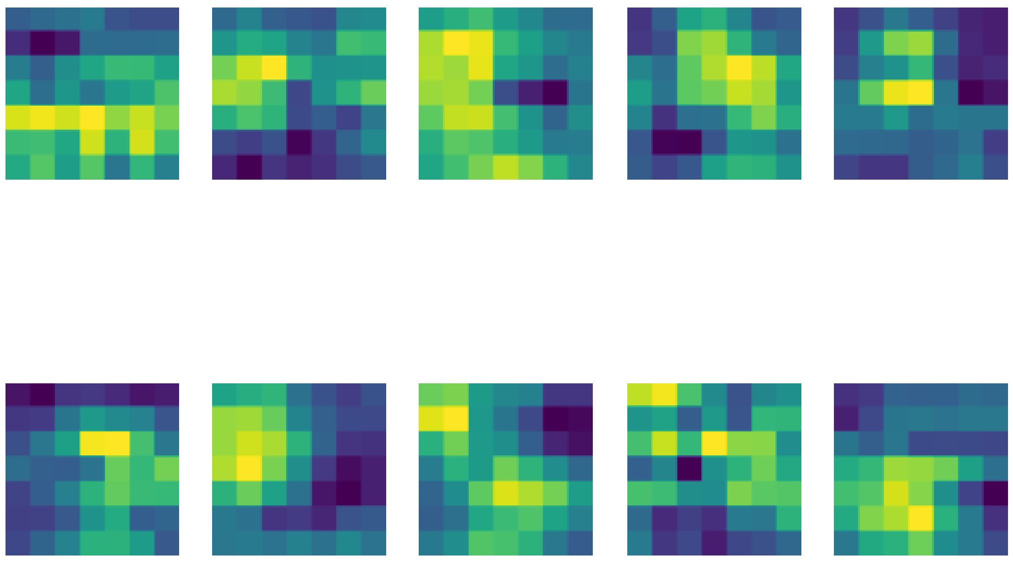
Figure 12: Activations from the last convolutional layer in ResNet18

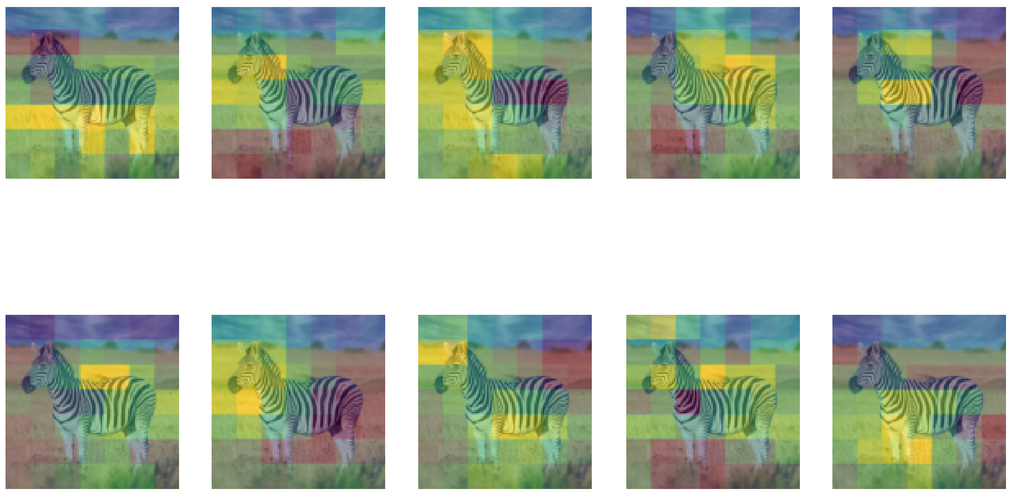
Figure 13: Activations from the last convolutional layer in ResNet18 oberlayed on the input image