TDT4265 - Computer Vision and Deep Learning

Assignment 3

Martin Johannes Nilsen

4th March 2022

1 Theory

1.1 Task 1a

In the first task we are to spatially convolve the given image (Figure 1a) by hand.

1	0	2	3	1
3	2	0	7	0
0	6	1	1	4

(a) A 3 × 5 image

-1	0	1
-2	0	2
-1	0	1

(b) A 3 × 3 sobel kernel

1	0	-1
2	0	-2
1	0	-1

(c) Flipped sobel kernel

Figure 1: Image and kernels

First of all, we know that as we want the same dimensions for the resulting image, we have to pad the image while convolving. This is called "same"-padding in popular machine learning libraries. Based on the stride and image- and kernel dimensions above we should use 1 level of zero-padding. For convolving by hand, we can simply flip (rotate 180°) the kernel and apply correlation, which is an easier and more intuitive operation to perform by hand.

We can proceed to perform the correlation operation using the following two matrices, F for filter and I for image:

$$F = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 3 & 1 & 0 \\ 0 & 3 & 2 & 0 & 7 & 0 & 0 \\ 0 & 0 & 6 & 1 & 1 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The first step on the upper left corner would yield the following value:

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

and we can perform the full operation with stride = 1, which gives us the following image:

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

Which I have verified using the conv2d-function in pyTorch.

1.2 Task 1b

I think the convolution layer is the one reducing the sensitivity to translational variations as it moves/slides a kernel over the image. The activation functions are connected to certain input nodes, hence not reducing the sensitivity to translational variations, and Max Pooling is mainly used to reduce the data complexity without removing cruical information. Pooling can also have some effect in my opinion if the variations is within the range of pixel locations being pooled together, but this may be a naive assumption.

1.3 Task 1c

Based on the kernel being 5×5, and the use of stride $= 1$, we know that we should be using 2 layers of zero-padding. This can be confirmed using the following equation:

$$P_H = P_W = \frac{S_W\left(W_2 - 1\right) - W_1 + F_W)}{2} = \frac{(5-1) - 5 + 5}{2} = 2$$

1.4 Task 1d

We know the input images have the size 512×512. Further on we have no padding, and the difference from the input to the first layer is $512 - 504 = 8$. Based on this, we know that the kernel has $8/2 = 4$ entries out from center in both directions, giving us the length of both rows and columns to be $4 * 2 + 1 = 9$. The kernel size is therefore 9×9.

This can be checked in reverse, where a 9×9 kernel with stride $= 1$ and no padding will chop off 4 pixels in all directions, ending up with an image of dimensions 504×504.

1.5 Task 1e

Using neighborhoods of size 2×2 and a stride of 2 will halve the dimensions from 504×504 to 252×252. The spatial dimensions of the pooled feature maps in the first pooling layer is therefore 252×252.

1.6 Task 1f

Assuming the spatial dimensions of the convolution kernels in the second layer to be 3×3, no padding and a stride of 1, we can use the same check we did in task 1d for the final dimensions. The convolution will result in cutting off $\frac{3-1}{2} = 1$ pixel in each direction, ending up with the a dimension of $(252 - 2) \times (252 - 2) = 250 \times 250$.

1.7 Task 1g

How many parameters we have for each layer depend of the type of layer. In a **pooling layer** we have no learnable parameters. In a **convolution layer** we can use the following formula $((w * h * f_p) + 1) * f_c)$, where we have the width w, height h, filters in the previous layer f_p and in the current layer f_c. The 1 is to account for the bias. In a **fully connected layer** we can use the formula $((n_p * n_c) + 1 * n_c)$, where we have the current layer neurons n_c and the previous layer neurons n_p.

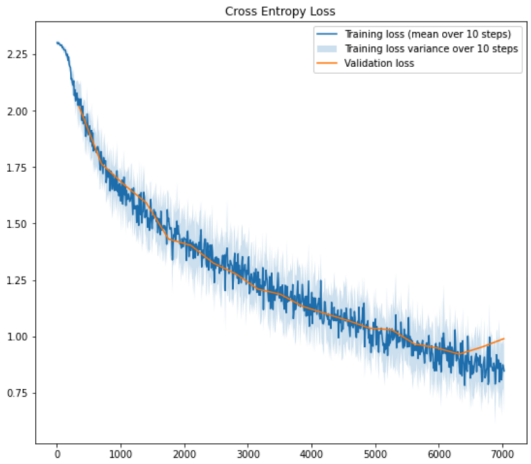
Based on these formulaes we can calculate the amount of learnable parameters in each layer in the model, yielding the following table:

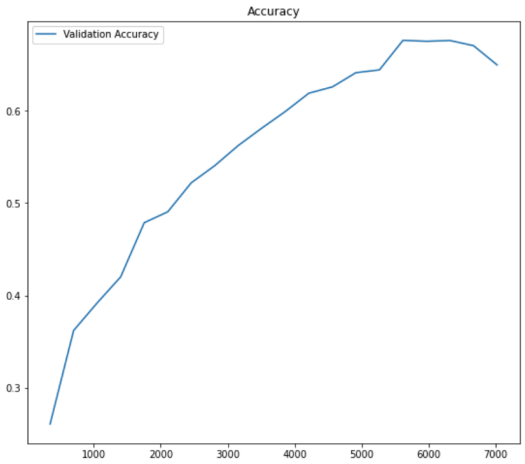
Layer	Layer type	Formula	Learnable parameters
1	Conv2D	(5*5*3+1)*32	2432
1	MaxPool2D	-	-
2	Conv2D	(5*5*32+1)*64	51264
2	MaxPool2D	-	-
3	Conv2D	(5*5*64+1)*128	204928
3	MaxPool2D	-	-
-	Flatten	-	-
4	Fully-Connected	(4*4*128)*64+64	131136
5	Fully-Connected	64*10 + 10	650
			390410

Note that the first convolution layer has 3 as the former filters from the input, as we have a RGB image. For the layer 4 we need to calculate the width and height of the third layer output for finding the neurons of the previous layer. Because of the pooling the height and width of 32 is halved 3 times, yielding 4 as the width and height.

2 Convolutional Neural Networks

2.1 Task 2a - Plot





2.2 Task 2a - Final accuracy

Train accuracy	Val accuracy	Test accuracy
0.721	0.679	0.684

3 Deep CNN for Image Classification

3.1 Task 3a - Models

3.1.1 Model S

For the first model, the model from task 2 were used as a base. I extended it with one layer, and experimented with both kernel size 5 and 3 for the convolution layers. The final architecture for the first model looked like:

Layer	Layer Type	Number of hidden units / Filters	Activation
1	Conv2D	32	ReLU
1	MaxPool2D	-	-
2	Conv2D	64	ReLU
2	MaxPool2D	-	-
3	Conv2D	128	ReLU
3	MaxPool2D	-	-
4	Conv2D	256	ReLU
4	MaxPool2D	-	-
	Flatten	-	-
5	Fully-Connected	64	ReLU
6	Fully-Connected	10	Softmax

Data augmentation is the main key here. From earlier subjects I have had good effect of expanding the training set, and I used this tactic in this model. Pytorch has a list of transforms available, and I chose the following transforms. **RandomCrop()** crops an image at a random location. **RandomHorizontalFlip(p)** flips an image horizontally with a probability of p. **ColorJitter()** randomly changes brightness, saturatuion and contrast of an image. **RandomRotation(rot)** rotates the images randomly from $-rot$ to rot degrees. When applying these I utilized the **RandomApply(p)** function, which applies the tranasformations with a probability of p. Lastly, I experimented with and ended up with a learning rate of 0.15. It could be mentioned that the difference between the kernel size being 3×3 and 5×5 is almost negligable.

3.1.2 Model E

For the second model, I continued with Model S and added batch normalization and the Adam optimizer. The data augmentation was kept from the last model, but the learning rate was decreased to 0.0015.

Layer	Layer Type	Number of hidden units / Filters	Activation
1	Conv2D	32	ReLU
1	BatchNorm2D	-	-
1	MaxPool2D	-	-
2	Conv2D	64	ReLU
2	BatchNorm2D	-	-
2	MaxPool2D	-	-
3	Conv2D	128	ReLU
3	BatchNorm2D	-	-
3	MaxPool2D	-	-
4	Conv2D	256	ReLU
4	BatchNorm2D	-	-
4	MaxPool2D	-	-
	Flatten	-	-
5	Fully-Connected	64	ReLU
5	BatchNorm1D	-	-
6	Fully-Connected	10	Softmax

3.2 Task 3b - Loss and accuracy

Here I compare the two models. The main differences is that Model E have taken the good features of Model S, and included batch normalization and the Adam optimizer. For the sake of comparing, I will use the statistics from the runs with the same kernel size as I have tested with both 5×5 and 3×3. The learning rate of the last one is significantly smaller, but was what yielded the best result for this model.

Model	Train loss	Train accuracy	Validation accuracy	Test accuracy
Model S	0.238	0.948	0.891	0.761
Model E	0.124	0.972	0.937	0.812

3.3 Task 3c - Brief discussion

The thre most effective changes I did was using **data augmentation**, change the **optimizer** and implement **batch normalization**.

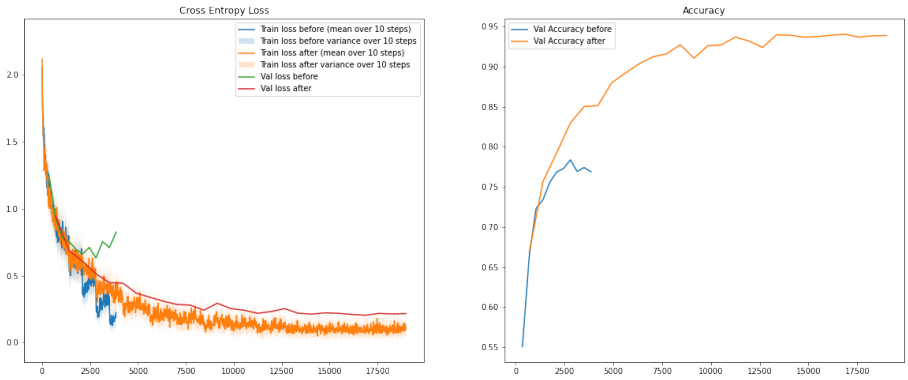
Augmenting the data yields a larger data set to train on, crteating a better base for finding the best generalized patterns, not overfitting on certain features of specific images. Said in another way, the network has a better chance of finding the good

features/patterns, and avoid overfitting, as it has more photos to generalize upon. The adam optimizer is a well-known optimizer, which I have used in earlier projects, and was therefore seen as a good alternative to the Stochastic Gradient Descent (SGD). Finally, batch normalization was introduced for normalizing the data along the way - giving a better chance of generalization.

On the other side, I tried multiple techniques which did not yield noticable improvements in the performance. When changing the **number of filters** in the convolutional part of the network, I only got worse results. Neither did changing the **filter size** from 5×5 to 3×3 yield much of a noticeable impact on the training. The other thing I tried was **dropout**, but I have always seen the placement of dropout as a challenge as some networks seem to benefit from placing it in between all conv-layers, and some only in the classifier part. It is also important to set the dropout-rate correct here, and based on earlier experience I am sure that I could have utilized this in a better way by experimenting more with it. One factor one also should keep in mind is that as the model approaches and surpasses 80% accuracy, new techniques does not necesarily have a large impact.

3.4 Task 3d - Most improving technique

A comparison between the best model with and without data augmentation is shown in the figure below. Every other factor is the same. It could also have been interesting to see the differences with and without the Adam optimizer and batch normalization besides each other, but thats a task for the future :)



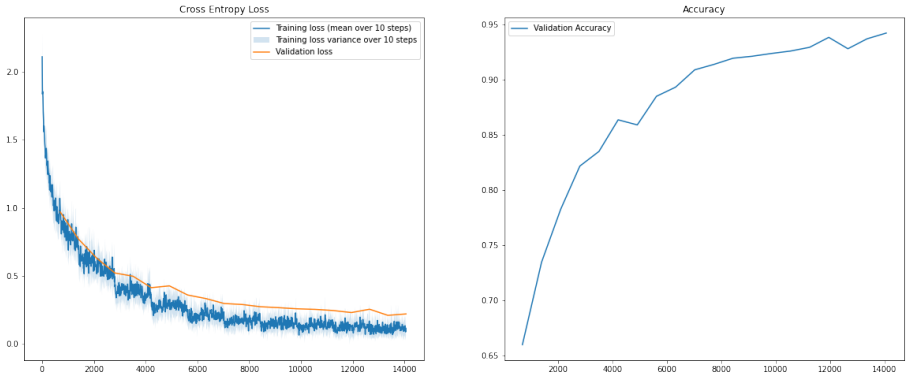
As we can see in the figure above, the training without data augmentation seem to converge much earlier on a lower accuracy/higher loss.

3.5 Task 3e - Minimum 80% test accuracy

I combined this task with the implementation of models in task 3a, where I got a model performing better than 80% on the test set. It should be mentioned that due to random initialization, the runs are not deterministic and therefore I had the

problem where some of the runs where over 80% and some below, with the same code.

Model E	Train	Validation	Test
Accuracy	0.972	0.937	0.812
Loss	0.097	0.246	0.759



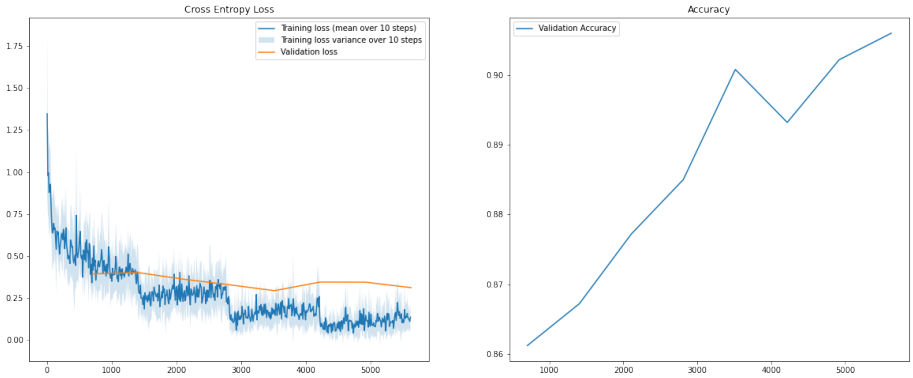
3.6 Task 3f - Overfitting vs underfitting?

As we can see in the two figures above, there might be some overfitting going on here. This can be seen due to the fact that the loss of train and validation happens to be much lower than the one of test. The validation loss is also higher than the one of training, indicating the same thing.

4 Transfer learning with ResNet

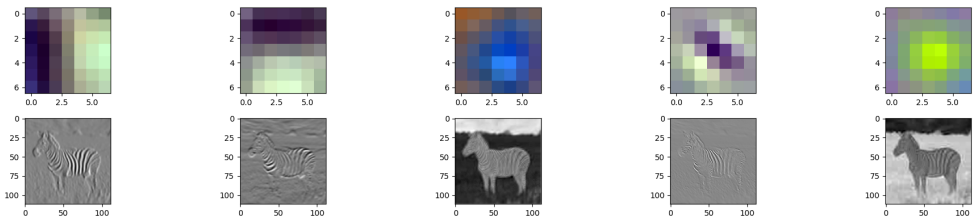
4.1 Task 4a - Implementation

I used a learning rate of 0.0005, batch size of 32 and the Adam optimizer, as suggested in the assignment. A new function for loading the data for the ResNet-model have been included, as the mean and standard deviation is different, in addition to the need of resizing the images to size 224×224. My implementation have a final test accuracy of 0.89%. A plot of training and validation loss is shown in a figure below.



4.2 Task 4b - Filter activation of first conv layer

The first filter to the left seems like a vertical edge-detection filter. We see the vertical stripes clearly, while the horizontal lines in the image is blurred out. The weight look like a vertical Sobel-filter, which confirms this thought. The second filter looks like a horizontal edge-detection filter, as the sky-grass line and the zebra's horizontal lines are visible and in focus. Based on the blue color in the next filter, it seems like this filter is there for detecting the blue color. The sky being in focus confirms this. The fourth filter looks like a filter detecting edges in diagonal, which can be a correct guess when taking a look at the activation image of the zebra. The last filter seems like a color detection filter, focusing on the color green. This can be confirmed as the grass behind the zebra is what lights up the image.



4.3 Task 4c - Filter activation of last conv layer

Something which is quite interesting in my eyes, is the fact that a neural network tries to generalize patterns and features, looking for the cruical information for separating entities, and therefore end up relying on weights/activations looking like the ones below. For a human being trying to guess the contents of these images, it is wild guesses at this point. In the earlier layers of a CNN we have low-level features such as edge detection and color detection, which makes it possible for us to still gather the same information in the images - we can still see a zebra in the first layers. As we go deeper down, these features are abstracted away more and more as it tries to fit the important information into the classifier. The activation of the ten first filters in the last convolutional layer of the trained model can be found below.