Assignment 3

March 5, 2024

Alfred Indrehus
Andreas Måkestad

1 Task 1

1.1 task 1a)

Task 1a)

Image with zero-padding (Flipped) Kernel

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 2 & 3 & 1 & 0 \\ 0 & 3 & 9 & 1 & 1 & 4 & 0 \\ 0 & 4 & 5 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Going cell by cell we get:

$$a_{11} = 0+0+0+0+0+(-2)\cdot 1+0+0+(-1)\cdot 9 = -11$$

$$a_{12} = 0+0+0+2\cdot 2+0-2\cdot 2+3+0-1\cdot 1 = 2$$

$$a_{13} = 0+0+0+2\cdot 1+0-2\cdot 3+9+0-1\cdot 1 = 4$$

$$a_{14} = 0+0+0+2\cdot 2+0-2\cdot 1+1\cdot 1+0-1\cdot 4 = -1$$

$$a_{15} = 0+0+0+2\cdot 3+0+0+1\cdot 1+0+0 = 7$$

$$a_{21} = 0+0-1\cdot 1+0+0-2\cdot 9+0+0-1\cdot 5 = -24$$

$$a_{22} = 2\cdot 1+0-1\cdot 2+3\cdot 2+0-2\cdot 1+4\cdot 1+0+0 = 8$$

$$a_{23} = 1\cdot 1+0-1\cdot 3+9\cdot 2+0-2\cdot 1+5\cdot 1+0-1\cdot 7 = 12$$

$$a_{24} = 2\cdot 1+0-1\cdot 1+2\cdot 1+0-2\cdot 4+0+0+0 = -5$$

$$a_{25} = 3\cdot 1+0+0+2\cdot 1+0+0+7\cdot 1+0+0 = 12$$

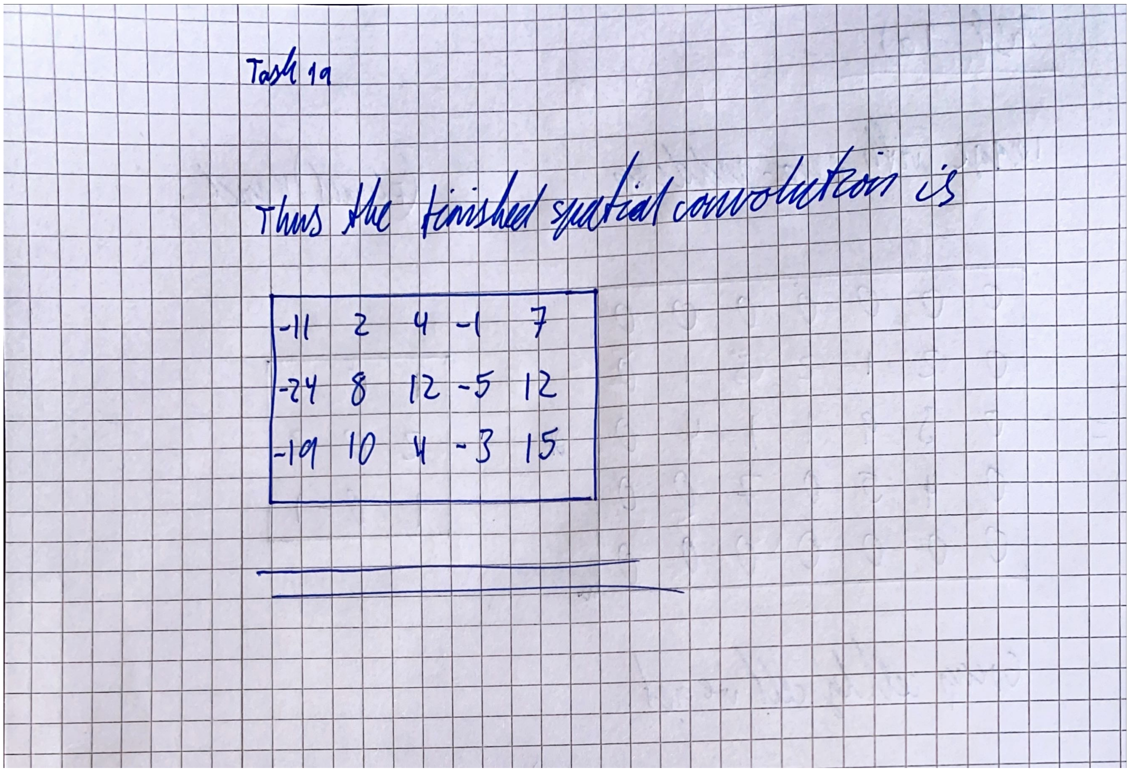
$$a_{31} = 0+0-1\cdot 9+0+0-2\cdot 5+0+0+0 = -19$$

$$a_{32} = 3\cdot 1+0-1\cdot 1+4\cdot 2+0+0+0+0+0 = 10$$

$$a_{33} = 9\cdot 1+0-1\cdot 1+5\cdot 2+0-2\cdot 7+0+0+0 = 4$$

$$a_{34} = 1\cdot 1+0-1\cdot 4+0+0+0+0+0+0 = -3$$

$$a_{35} = 1\cdot 1+0+0+7\cdot 2+0+0+0+0+0 = 15$$



Task 1a

Thus the finished spatial convolution is

-11	2	4	-1	7
-24	8	12	-5	12
-19	10	4	-3	15

1.2 task 1b)

The max pooling layer is the layer that reduces the sensitivity to translational variations in the input.

1.3 task 1c)

Formulas for output of convolutional layer (taken from appendix) :

- $W_2 = [(W_1 - F_W + 2P_W)/S_W] + 1$
- $H_2 = [(H_1 - F_H + 2P_H)/S_H] + 1$

Solving these for PW and PH respectively, with $H_1 = H_2, W_1 = W_2, S_H = S_W = 1$

$P_W = (F_W - 1)/2$

$P_H = (F_H - 1)/2$

With $F_H = F_W = 7$

$P_W = (7 - 1)/2 = 3 = P_H$

Thus, three layers of padding are needed

1.4 task 1d)

Spatial dimensions feature map layer 1 are given as 508 x 508. With $S_W = S_H = 1$ and $P_W = P_H = 0$, the formulas are then reduced to: - $W_2 = [(W_1 F_W)/1] + 1$ - $H_2 = [(H_1 F_H)/1] + 1$

With $W_1 = H_1 = 512$, and $W_2 = H_2 = 508$ and solving for F_W and F_H we get:

- $F_W = 512 - 508 + 1 = 5$
- $F_H = 512 - 508 + 1 = 5$

The spatial dimensions for the kernel are **5x5**.

1.5 task 1e)

Using $W_1 = H_1 = 508$ as input to the subsampling layer, with $F_W = F_H = 2$ and $S_W = S_H = 2$, we can solve for W_2 and H_1 and get:

- $W_2 = [(502 - 2 + 0)/2] + 1 = 254$
- $H_2 = [(502 - 2 + 0)/2] + 1 = 254$

Thus, the spatial dimensions are **254x254**.

1.6 task 1f)

Using the output from the previous layer as input, namely $W_1 = H1 = 254$ and kernel sizes $F_H = F_W = 3$, $S_W = S_H = 1$ and $P_W = P_H = 0$ we can use the same formulas as before and get:

- $W_2 = [(254 - 3 + 0)/1] + 1 = 252$
- $H_2 = [(254 - 3 + 0)/1] + 1 = 252$

The spatial dimensions of the feature maps in the second layer are 252 x 252.

1.7 task 1g)

The number of parameters is the number of weights + biases. Assuming the network takes an RGB image ($C_1 = 3$) with a width of 32.

We have a network with the following convolutional layers: - Conv2D: **32 filters**, 5x5 kernel size, padding = 2, stride = 1 - Conv2D: **64 filters**, 5x5 kernel size, padding = 2, stride = 1 - Conv2D layer: **128 filters**, 5x5 kernel size, padding 2, stride 1

We also have two fully connected layers: - Fully connected: **64 hidden units** - Fully connected: **10 hidden units**

For the first convolutional layer, each filter has F_H x F_W x C_1 = 5 x 5 x 3 = 75 weights, multiplied by the number of filters we get 75x32 =2400 weights.

The number of biases for each convolutional layer is the same as the number of output filters. The total number of parameters for the first layer is then 2400 + 32 = 2432 parameters.

Our second convolutional layer has F_H x F_W x C_1 = 5 x 5 x 32 = 800 weights, multiplied by each filter and adding the biases we get 800 x 64 + 64 = 51264 parameters.

The last convolutional layer has F_H x F_W x C_1 = 5 x 5 x 64 = 1600 weights, multiplied by each filter and adding the biases we get 1600 x 128 + 128 = 204928 parameters.

The parameters in the fullt connected layers are respectively

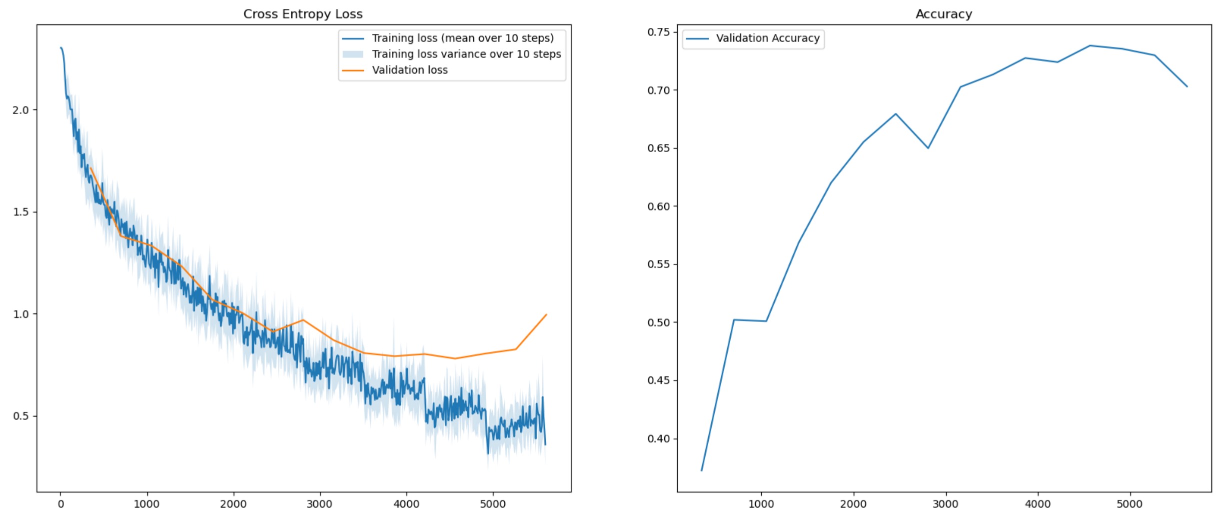
- First layer: 128 x 4 4 x 64 + 64 = 131136 parameters
- Second layer: 64 x 10 + 10 = 650 parameters

Thus, the total number of parameters is 2432 + 51264 + 204928 + 131136 + 650 = **390410** parameters

2 Task 2

2.0.1 Task 2a)

Plot showing validation loss and training loss. On the right the accuracy is also included:



2.0.2 Task 2b)

Training Loss: 0.3922, Training Accuracy: 0.8699
Validation Loss: 0.7848, Validation Accuracy: 0.7354
Test Loss: 0.8025, Test Accuracy: 0.7327

3 Task 3

3.0.1 Task 3a)

3.0.2 Model Architecture

Layer	Layer Type	Number of Hidden Units / Number of Filters	Activation Function
1	Conv2D BatchNorm2d	64	ELU
2	Conv2D BatchNorm2d MaxPool2d	64	ELU
3	Conv2D BatchNorm2d	128	ELU
4	Conv2D BatchNorm2d MaxPool2D	128	ELU
5	Conv2D BatchNorm2d	256	ELU

Layer	Layer Type	Number of Hidden Units / Number of Filters	Activation Function
6	Conv2D BatchNorm2d MaxPool2D	256	ELU
7	Conv2D BatchNorm2d Dropout	512	ELU
8	Conv2D BatchNorm2d MaxPool2D	512	ELU
-	Flatten		
9	Fully-Connected	128	ELU
10	Fully-Connected	10	SoftMax

3.0.3 Training Details

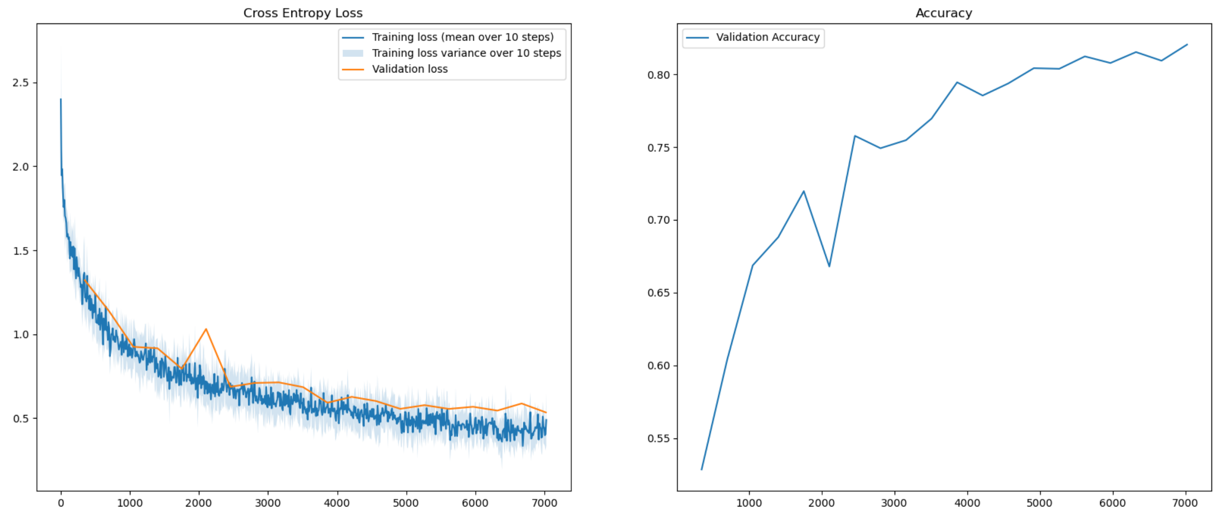
- **Optimizer**: Stochastic Gradient Descent (SGD) optimizer with weight decay (L2 regularization) of 1e-5
- **Learning Rate**: 5e-2
- **Batch Size**: 64
- **Epochs**: 10
- **Early Stopping**: Stopped after 4 consecutive epochs of no improvement in validation loss
- **Dropout parameter (p)**: We used p=0,4 in the dropout in layer 7.

3.0.4 Task 3b)

Final train loss, training accuracy, validation accuracy and test accuracy

Metric	Training	Validation	Test
Loss	0.4198	0.5530	0.5123
Accuracy	0.8505	0.8125	0.8279

Plots for task3 model



3.0.5 Task 3c)

L2 regularization We extend the optimizer in "trainer.py" like this "self.optimizer = torch.optim.SGD(self.model.parameters(), self.learning_rate,weight_decay=1e-5)". That is, we added weight_decay. This actually make the model performa slightly worse:
- Test Accuracy without L2: 0.8323
- Test Accuracy with L2: 0.8279

We find this result quite od, as we exptected L2 relularization to make the model performe better on test and validation data. L2 regularization encourages small weights, which is usefull for prevventing overfitting. Since it did not make our model performe better, we belive that the L2 regularization make the model overly simple.

Data Augmentation From data augmentation we added: - transforms.RandomHorizontalFlip()
- transforms.RandomRotation(10)

to the implementation of "transform_train" in the file "dataloaders.py". Here are the results with and without this extention:

	Test Loss	Test Accuracy
Without data augmentation	0.6089	0.7976
With data augmentatuion	0.4980	0.8323

This shows that data augmentation has been very usefull in improving our model. When we apply this type of data augmentation, we alter the data by flipping and rotationg the images. This makes the model better at capturing the underlaying patterns regardles of rotation, which in turn gives better performence on unseen data.

Batch normalization This methode made our model performe better. We applyed “BatchNorm2d” after each convolutional layer. This is a batch normalization technique usefull for normalizing multidimensional spation input (such as RBG-images) accross several dimentions (channels). This is usefull to prevent unintended covariance accros channels.
This technique proved to be very efficient in our model.
See task 3d for plot of performance with and without this.

Filter size We reduced the filtersize from 5 to 3. This mede the model performe slightly better.

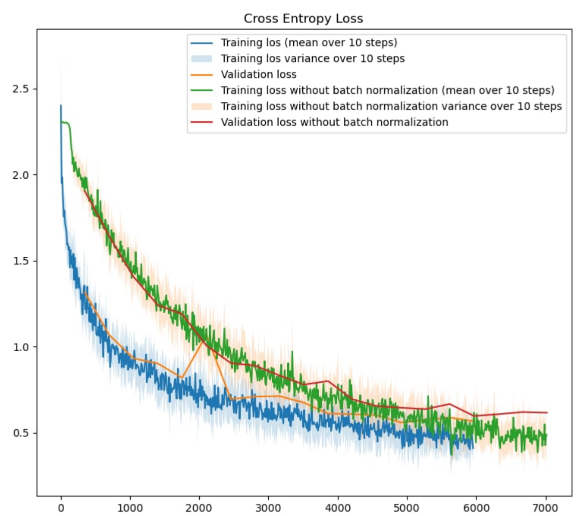
Number of filters Increase the number of filters from 32 to 64. This mede the model performe slightly better. Introduce more parameters, making the model more complex.

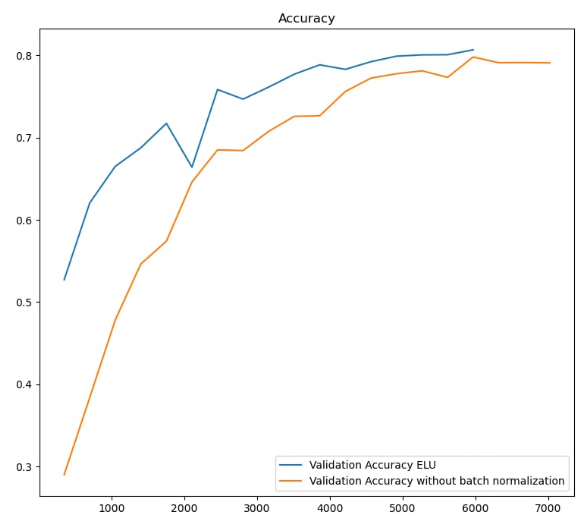
Network architecture The architecture was changed as shown in task 3a). Made the model able to capture more complex patterns.

Activation Functions We tried chainging the activation function from ReLU to ELU. This mede the model performe slightly better. Helped to address the “dying ReLu problem”.

3.0.6 Task 3d)

This task shows a plot of performance and loss before and after applying Batch normalization.

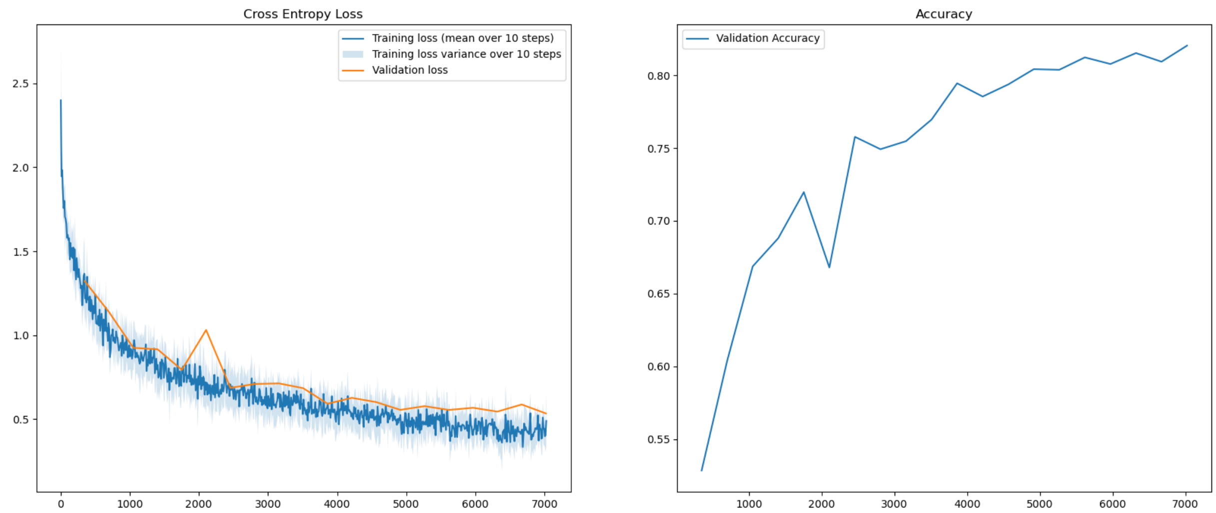




It shows significant improvements in both loss and accuracy.

3.0.7 Task 3e)

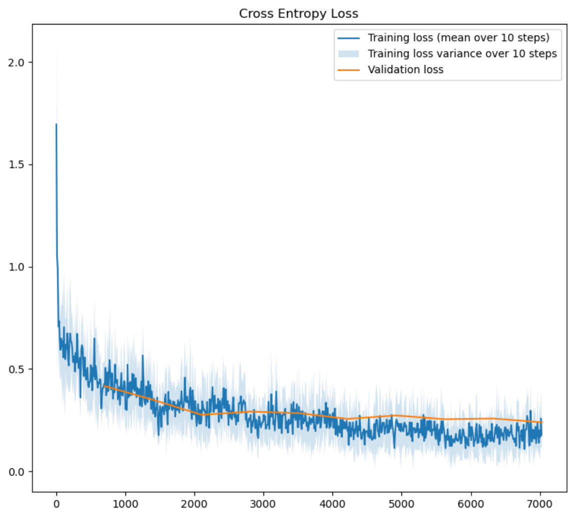
This is the same model as before, be reaced an acuracy of over 80% before reading task 3e, so we answered the previous task for this model.

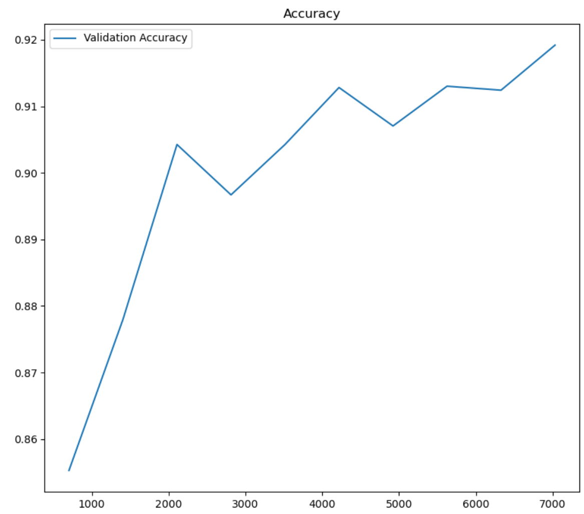


3.1 Task 3f)

From the plot above we observe that the validation loss is above the training loss, this it natural, but what we can also see is that the model gap between test and validation loss seems to increase towards the end of the graph. This can be a sign of the model starting to overfit towards the end of training. We belive that the model stoped training before this became a problem, as this is only happening towards the end.

4 Task 4





final test accuracy: 0.8901

Parameters:

- Omtomizer: Adam
- Batch size: 32
- Learning rate: $5 * 10^{-4}$
- Data augemntation: RandomHorizontalFlip() and RandomRotation(10)
- Number of epochs: 5