# AssignmentReport-Group110

March 5, 2024

## 1 Assignment 1 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. 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`.

## 2 Task 1

### 2.1 task 1a)

Task 1: Theory

a)

Applying zero padding to get correct output:

Image I:

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

Kernel k:

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

We flip the kernel both horizontally and vertically:

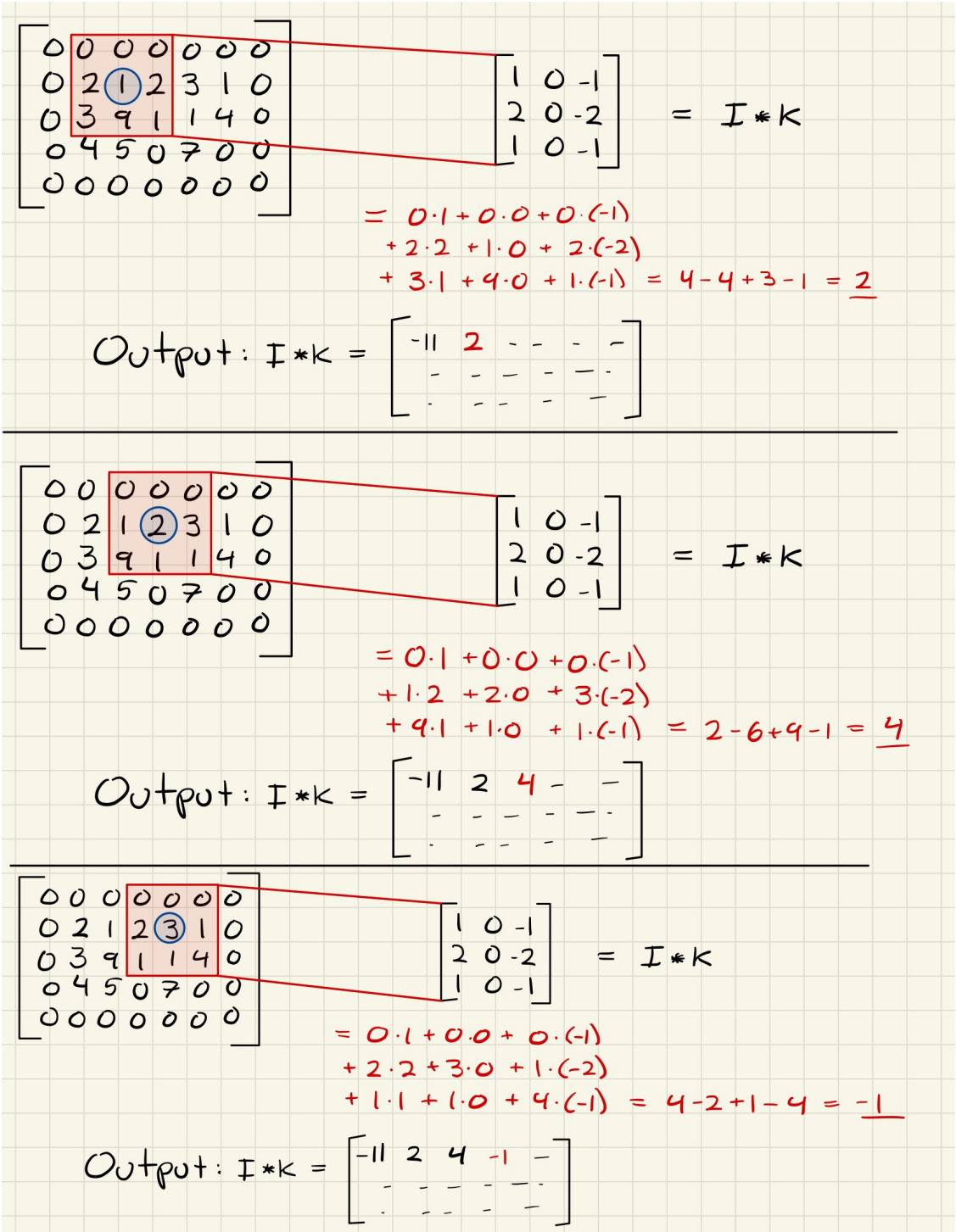
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

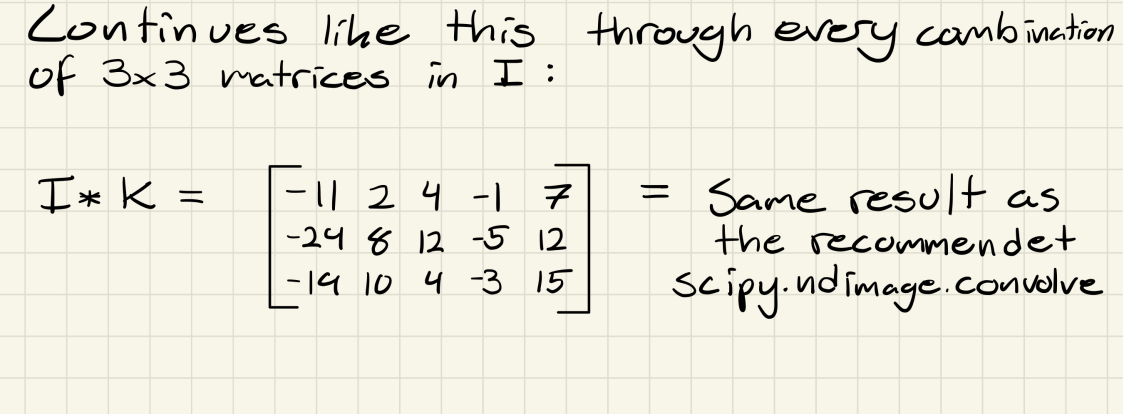
Performing Convolution:

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

$$= 0 \cdot 1 + 0 \cdot 0 + 0 \cdot (-1) + 0 \cdot 2 + 2 \cdot 0 + 1 \cdot (-2) + 0 \cdot 1 + 3 \cdot 0 + 9 \cdot (-1) = -2 - 9 = -11$$

Output: $I * k = \begin{bmatrix} -11 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$


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
1 0 -1
2 0 -2
1 0 -1
= I * K
= 0·1 + 0·0 + 0·(-1)
+ 2·2 + 1·0 + 2·(-2)
+ 3·1 + 9·0 + 1·(-1) = 4 - 4 + 3 - 1 = 2
Output: I*K = -11 2
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
1 0 -1
2 0 -2
1 0 -1
= I * K
= 0·1 + 0·0 + 0·(-1)
+ 1·2 + 2·0 + 3·(-2)
+ 9·1 + 1·0 + 1·(-1) = 2 - 6 + 9 - 1 = 4
Output: I*K = -11 2 4
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
1 0 -1
2 0 -2
1 0 -1
= I * K
= 0·1 + 0·0 + 0·(-1)
+ 2·2 + 3·0 + 1·(-2)
+ 1·1 + 1·0 + 9·(-1) = 4 - 2 + 1 - 9 = -1
Output: I*K = -11 2 4 -1




### 2.2 task 1b)

Answer is: (iii) Max Pooling reduces the sensitivity to translational variations in the input by choosing the maximum value of a small region, thereby making the output invariant to small shifts and distortions.

### 2.3 task 1c)

Task 1c)

We can calculate the output size of a convolution with:

$$\text{Output size } O = \frac{W - F + 2 \cdot P}{S} + 1$$

where $D$ = Output size ($O \times O$)
$W$ = width (could have been height aswell)
$F$ = Spatial extent (kernel size $F \times F$)
$P$ = Padding on all sides
$S$ = Stride

We want $O = W$ and to find $P$:

$$\Rightarrow W \cdot S = W - F + 2 \cdot P + 1 \cdot S$$

$$\Rightarrow P = \frac{W \cdot S - W + F - S}{2}$$

$$\Rightarrow P = \frac{W \cdot 1 - W + 7 - 1}{2} = \frac{7 - 1}{2} = \underline{\underline{3}}$$

$\Rightarrow$ Padding should be 3 on each side

### 2.4 task 1d)

Task 1d)

We have input size $W_1 \times H_1 = 512 \times 512$

Output of convolution layer has size $W_2 \times H_2 = 508 \times 508$

Stride $S = 1$ and Padding $P = 0$

Since all matrices are quadratic $\Rightarrow W = H$

Using formula $W_2 = \frac{W_1 - F_w + 2 \cdot P}{S} + 1$ to find $F \; (= F_w)$

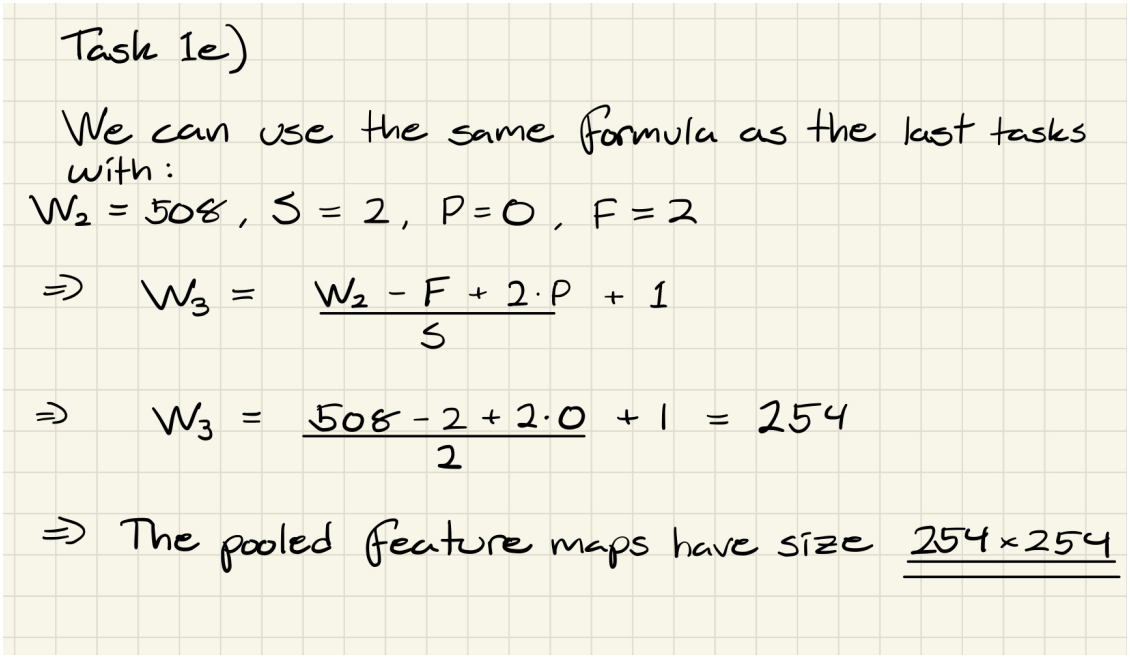
$W_2 \cdot S - W_1 - 2 \cdot P - 1 \cdot S = -F$

$F = -W_2 \cdot S + W_1 + 2 \cdot P + S = -508 \cdot 1 + 512 + 2 \cdot 0 + 1$

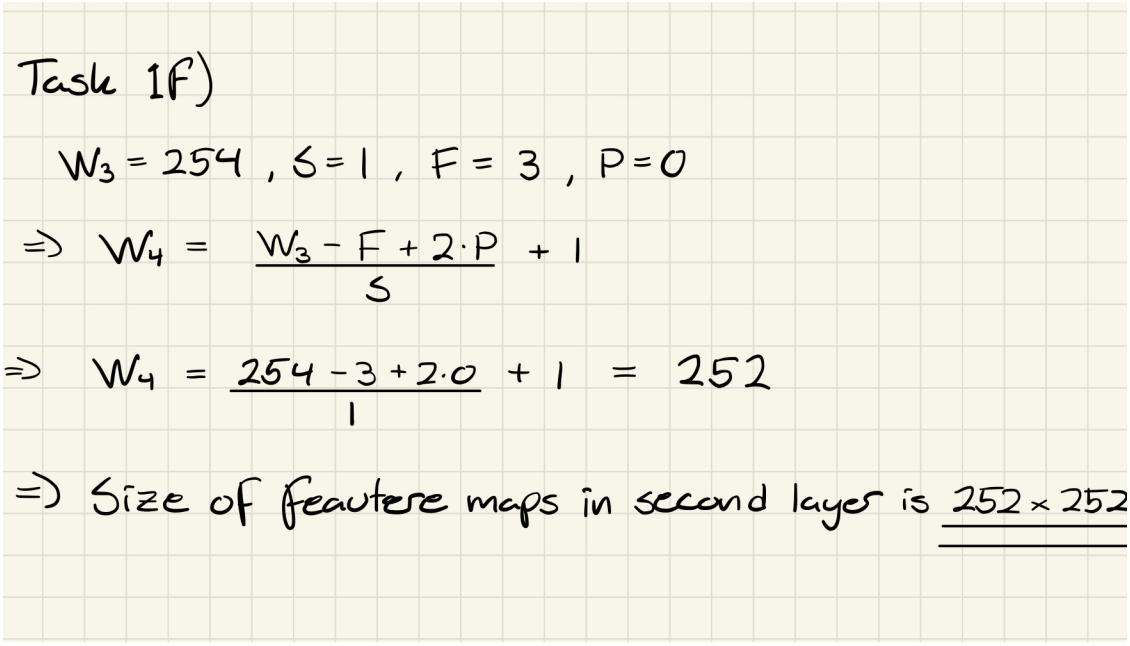
$F = 5$

The kernel size is $F \times F = \underline{\underline{5 \times 5}}$

### 2.5 task 1e)




## task 1f)




## task 1g)

Task 1g)

To find the total number of parameters in the network we need to add the number of weights and biases in every layer.

Each Convolution layer:

Number of weights: $F_H \cdot F_W \cdot C_1 \cdot C_2$

$F_H$ = Filter Height = 5

$F_W$ = Filter Width = 5

$C_1$ = Depth

$C_2$ = Number of filters

Number of Biases = $C_2$

**Layer 1 - Conv2D** Depth = 3 "RGB"

Params $P = 5 \cdot 5 \cdot 3 \cdot 32 + 32 = 2432$

**Layer 2 - Conv2D** Depth = 32

$P = 5 \cdot 5 \cdot 32 \cdot 64 + 64 = 51264$

**Layer 3 - Conv2D** Depth = 64

$P = 5 \cdot 5 \cdot 64 \cdot 128 + 128 = 204928$

Each Fully-Connected layer:

Number of weights = $I \cdot H$

I = Input size

H = Number of hidden units

Biases = H

We need to calculate the size after the

Flatten layer:

Size = Height · Width · Number of feature maps

= 4 · 4 · 128 = 2048 → Input for next layer

Back to parameters:

Layer 4 - FCC

$P = I \cdot H + \text{Bias} = 2048 \cdot 64 + 64 = 131136$

Layer 5 - FCC

$P = 64 \cdot 10 + 10 = 650$

↳ Number of units in previous layer

Total parameters = 2432 + 51264 + 204928 + 131136 + 650

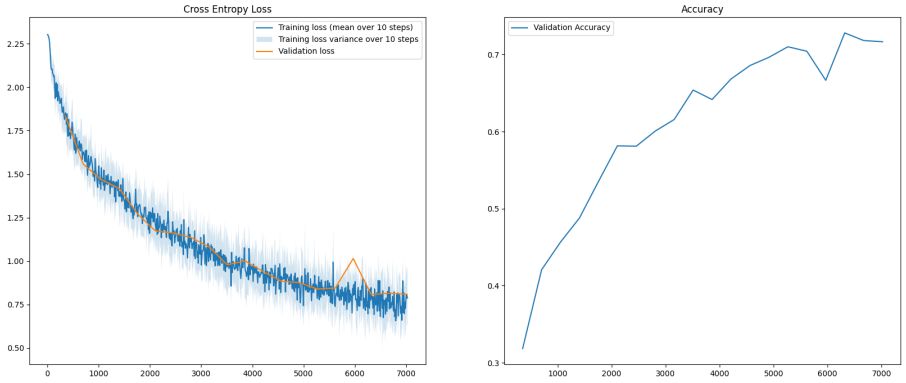
= 390 410

# 3 Task 2

### 3.0.1 Task 2a)

Epoch: 0, Batches per seconds: 278.71, Global step: 351, Validation Loss: 1.71, Validation Accuracy: 0.375

Epoch: 0, Batches per seconds: 258.48, Global step: 702, Validation Loss: 1.38, Validation Accuracy: 0.502
Epoch: 1, Batches per seconds: 254.40, Global step: 1053, Validation Loss: 1.33, Validation Accuracy: 0.503
Epoch: 1, Batches per seconds: 256.43, Global step: 1404, Validation Loss: 1.24, Validation Accuracy: 0.564
Epoch: 2, Batches per seconds: 253.93, Global step: 1755, Validation Loss: 1.09, Validation Accuracy: 0.612
Epoch: 2, Batches per seconds: 254.71, Global step: 2106, Validation Loss: 0.97, Validation Accuracy: 0.660
Epoch: 3, Batches per seconds: 254.61, Global step: 2457, Validation Loss: 0.90, Validation Accuracy: 0.683
Epoch: 3, Batches per seconds: 256.22, Global step: 2808, Validation Loss: 0.99, Validation Accuracy: 0.645
Epoch: 4, Batches per seconds: 257.73, Global step: 3159, Validation Loss: 0.86, Validation Accuracy: 0.701
Epoch: 4, Batches per seconds: 258.32, Global step: 3510, Validation Loss: 0.83, Validation Accuracy: 0.710
Epoch: 5, Batches per seconds: 257.61, Global step: 3861, Validation Loss: 0.81, Validation Accuracy: 0.719
Epoch: 5, Batches per seconds: 258.77, Global step: 4212, Validation Loss: 0.78, Validation Accuracy: 0.733
Epoch: 6, Batches per seconds: 257.56, Global step: 4563, Validation Loss: 0.79, Validation Accuracy: 0.734
Epoch: 6, Batches per seconds: 260.10, Global step: 4914, Validation Loss: 0.82, Validation Accuracy: 0.733
Epoch: 7, Batches per seconds: 260.24, Global step: 5265, Validation Loss: 0.79, Validation Accuracy: 0.736
Early stop criteria met
Early stopping.



#### 3.0.2 Task 2b)

Train accuracy: 0.757, Validation accuracy: 0.734, Test accuracy: 0.756
Train loss: 0.693, Validation loss: 0.761, Test loss: 0.713

## 4 Task 3

#### 4.0.1 Task 3a)

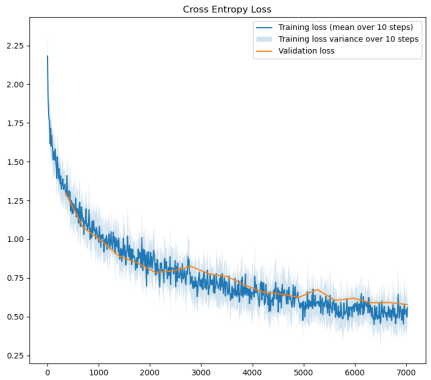
| Layer | Layer Type | Number of Hidden Units / Number of Filters | Activation Function |
|---|---|---|---|
| 1 | Conv2D | 128 | LeakyReLU |
| 1 | BatchNorm2d (128) | - | - |
| 1 | Dropout2d (p=0.05) | - | - |
| 1 | MaxPool2D | - | - |
| 2 | Conv2D | 256 | LeakyReLU |
| 2 | BatchNorm2d (256) | - | - |
| 2 | Dropout2d (p=0.05) | - | - |
| 2 | MaxPool2D | - | - |
| 3 | Conv2D | 512 | LeakyReLU |
| 3 | BatchNorm2d (512) | - | - |
| 3 | Dropout2d (p=0.05) | - | - |
| 3 | MaxPool2D | - | - |
| 4 | Conv2D | 1024 | LeakyReLU |
| 4 | BatchNorm2d (1024) | - | - |
| 4 | Dropout2d (p=0.05) | - | - |
| 4 | MaxPool2D | - | - |
| - | Flatten | - | - |
| 5 | Fully-Connected | 512 | LeakyReLU |
| 5 | BatchNorm1d (512) | - | - |
| 5 | Dropout (p=0.05) | - | - |
| 6 | Fully-Connected | 64 | LeakyReLU |
| 6 | BatchNorm1d (64) | - | - |
| 6 | Dropout (p=0.05) | - | - |
| 7 | Fully-Connected | 10 (num_classes) | |

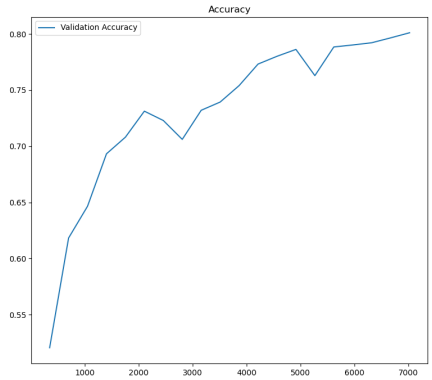
Optimizer: Stochastic Gradient Descent (SGD)
Learning rate: 5e-2
Batch size: 64
Regularization methods: - Dropout: Applied with a probability of 0.05 - Batch Normalization:

Applied after each convolutional and fully-connected layer. - Data Augmentation: Applied on training data in dataloaders.py, if "augment" is True.

#### 4.0.2 Task 3b)

| Metric | Train | Validation | Test |
|---|---|---|---|
| Accuracy | 0.835 | 0.774 | 0.811 |
| Loss | 0.475 | 0.644 | 0.562 |




#### 4.0.3 Task 3c)

**What improved the test accuracy:**

1. **Added Batch Normalization**: This enhances the stability of the network by normalizing the inputs to any activation layer. It helps in faster convergence and also has a regularization effect, reducing the chance of overfitting. By ensuring that the distribution of the inputs to layers deep in the network stays stable, it helps the network learn more effectivel, and thereby increasing the test accuracy.

2. **Decreased Kernel Size from 5x5 to 3x3**: Smaller kernels require fewer parameters, reducing the computational cost and the risk of overfitting. Furthermore, by using a 3x3 kernel instead of a 5x5, the network can retain more fine-grained details in the feature map. This improved the test accuracy.

3. **Changed ReLU to LeakyReLU**: While ReLU is an effective activation function for introducing non-linearity into the network, it has a drawback known as the "dying ReLU" problem, where neurons can sometimes permanently output zeros due to negative input values. LeakyReLU addresses this by allowing a small, positive gradient when the unit is not active and not strictly zero. This change can lead to better learning and prevent neurons from becoming inactive.

4. **Increased number of filters**: Increasing the number of output channels in a convolutional layer means that the layer can learn a larger variety of features from the input data. This can significantly enhance the network's ability to represent complex functions and distinguish between more advanced patterns in the data.

5. **Added One More Layer**: Adding more layers to a network increases its depth, potentially improving its ability to learn more complex features. However, this comes with the risk of overfitting and increased computational cost. Deeper networks can represent more complex functions but require more data and regularization techniques to train effectively.

6. **Added Data Augmentation**: This technique involves artificially expanding the training dataset by applying random transformations (such as rotation, scaling, and cropping). This can significantly improve the model's generalization capability by forcing it to learn invariant to these transformations, reducing the risk of overfitting on the training data. It's particularly beneficial when the available dataset is limited in size. It is also necessary considering we added an extra layer.

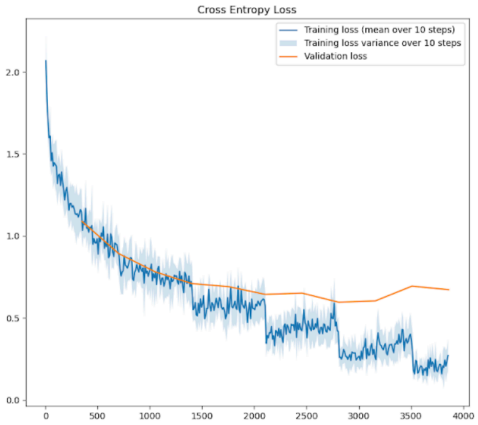
#### 4.0.4 What decreased test accuracy

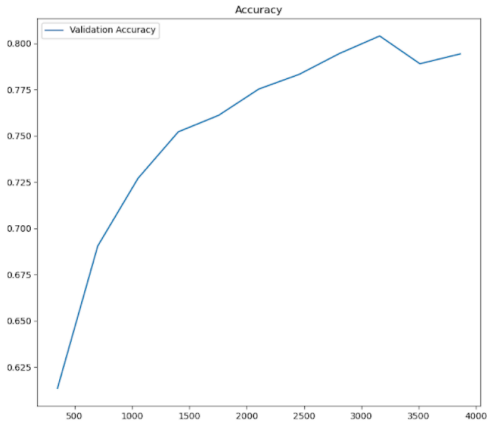
1. **Decreased number of filters**: When we tries decreasing the output channels in the convulational layers, the layer became too simple, and not able to represent the complex functions good enough. This strongly decreased the test accuracy.

2. **Decreased number of layers**: When we added a lauer to a network it decreased the depth, which worsened its ability to learn complex features. The risk of overfitting became smaller, and the computational cost was lower. The test accuracy got a lot worse from this.

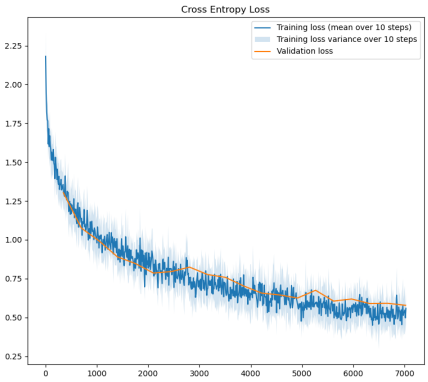
#### 4.0.5 Task 3d)

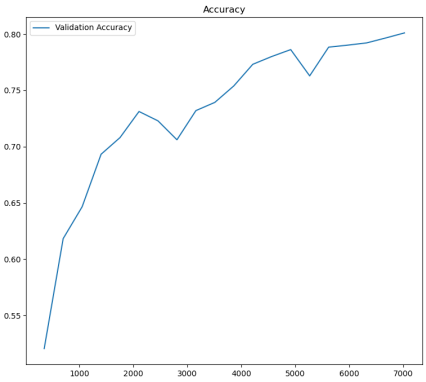
Below is the plots of loss and accuracy from before and after the improvements. Note that the plots of validation accuracy is seemingly lower in the after version. This is because of the data augmentation implementation. This increased the training dataset-size and made it more general than the previous training. It made it harder to validate during the training, but improved the overall test accuracy when the model was done training.




**Before improvements**:




**After improvements**:

**Test accuracy before**: 77.0%

**Test accuracy after**: 81.1%

#### 4.0.6 Task 3e)

We had already reached an accuracy of over 80% from the beginning of task 3. Therefor the plots below is the same as from task 3d) The final test accuracy is 81.1%.

#### 4.0.7 Task 3f)

In the last improvement we reached an accuracy of 81.1%, and from looking and the validation accuracy it seems like we were getting close to overfitting. The graph flattened out and the early stopping kicked in, which hindered overfitting.

### 4.1 Task 4a)

#### 4.1.1 Hyperparameters:

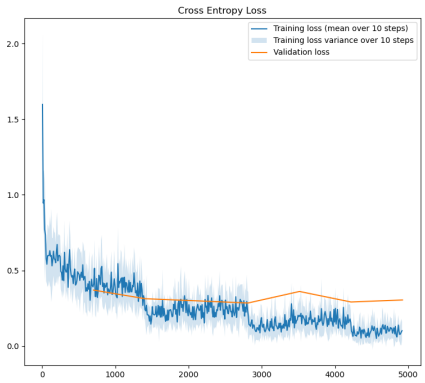
**Optimizer**: Adam

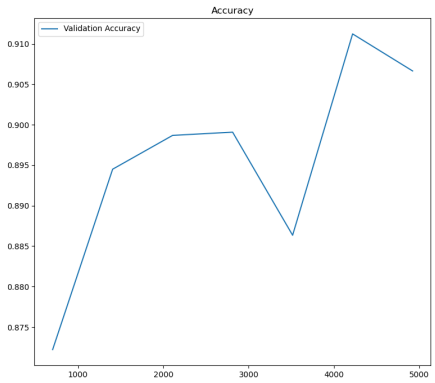
**Batch size**: 32

**Learning rate**: 5e-4

**Data augmentation used**: Only used resize to 224 x 224

### 4.1.2 Plot




**Accuracies and losses:**

| Metric | Train | Validation | Test |
|---|---|---|---|
| Accuracy | 0.969 | 0.891 | 0.889 |
| Loss | 0.089 | 0.360 | 0.381 |