Task 1

a) Adding padding to the image and flipping the kernel:

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

We then apply hadamard product between the kernel matrix and each of the matrix below, obtained by sliding the kernel with stride = 1:

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

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

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

We obtain:

$$-\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 9 \end{bmatrix} \quad -\begin{bmatrix} 0 & 0 & 0 \\ -4 & 0 & 4 \\ -3 & 0 & 1 \end{bmatrix} \quad -\begin{bmatrix} 0 & 0 & 0 \\ -2 & 0 & 6 \\ -9 & 0 & 1 \end{bmatrix} \quad -\begin{bmatrix} 0 & 0 & 0 \\ -4 & 0 & 2 \\ -1 & 0 & 4 \end{bmatrix} \quad -\begin{bmatrix} 0 & 0 & 0 \\ -6 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

$$-\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 18 \\ 0 & 0 & 5 \end{bmatrix} \quad -\begin{bmatrix} -2 & 0 & 2 \\ -6 & 0 & 2 \\ -4 & 0 & 0 \end{bmatrix} \quad -\begin{bmatrix} -1 & 0 & 3 \\ -18 & 0 & 2 \\ -5 & 0 & 7 \end{bmatrix} \quad -\begin{bmatrix} -2 & 0 & 1 \\ -2 & 0 & 8 \\ 0 & 0 & 0 \end{bmatrix} \quad -\begin{bmatrix} -3 & 0 & 0 \\ -2 & 0 & 0 \\ -7 & 0 & 0 \end{bmatrix}$$

$$-\begin{bmatrix} 0 & 0 & 9 \\ 0 & 0 & 10 \\ 0 & 0 & 0 \end{bmatrix} \quad -\begin{bmatrix} -3 & 0 & 1 \\ -8 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad -\begin{bmatrix} -9 & 0 & 1 \\ -10 & 0 & 14 \\ 0 & 0 & 0 \end{bmatrix} \quad -\begin{bmatrix} -1 & 0 & 4 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad -\begin{bmatrix} -1 & 0 & 0 \\ -14 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

the result is finally the 3×5 matrix where the ij^{th} component is the sum of the elements of the ij^{th} above:

$$-\begin{bmatrix} 11 & -2 & -4 & 1 & -7 \\ 24 & -8 & -12 & 5 & -12 \\ 19 & -10 & -4 & 3 & -15 \end{bmatrix}$$

b) The Convolutional layer (i) is the one reducing translational sensibility since a kernel for the convolution will detect a feature regardless of its position in the image

c) The output height H_O and width W_O are computed as follow, where H_I and W_I are the image height and width, S the stride, F the kernel (receptive **F**ield) side size and P the padding size:

$$H_O = \frac{H_I - F + 2P}{S} + 1 = H_I - 6 + 2P$$
$$W_O = \frac{W_I - F + 2P}{S} + 1 = W_I - 6 + 2P$$

Because we want $H_O = H_I$ and $W_O = W_I$, we need $2P = 6 \Rightarrow P = 3$

d) Using same equation as above, we have

$$F = H_I + 2P - S(H_O - 1) = 512 - 508 - 1 = 3$$

Kernel has dimensions 3×3

e) Again using same equation,

$$H_O = \frac{H_I - F + 2P}{S} + 1 = \frac{508 - 2}{2} + 1 = 254$$
$$W_O = \frac{W_I - F + 2P}{S} + 1 = \frac{508 - 2}{2} + 1 = 254$$

Pooled feature maps have dimensions 254×254

f) Similarly, feature maps of the second layer have dimension 252×252.

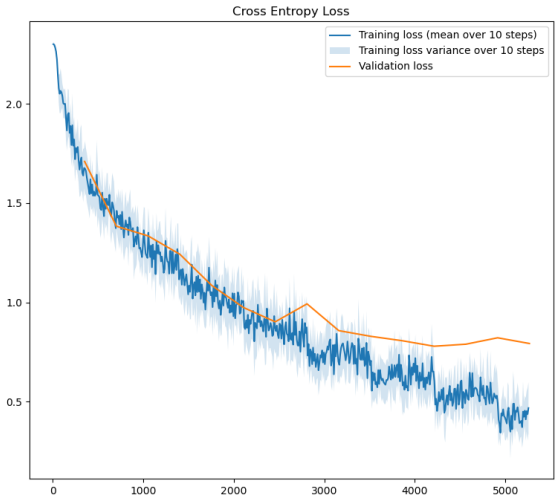
g) Each Conv layer filter has one weight kernel and one bias per input depth. Since the kernels have size 5×5, each conv layer has $D_I \cdot 26 \cdot D_O$ parameters. The Pooling layers and Flatten layer do not introduce parameters.
Each FC layers has a bias for each of its neuron and one weight per input unit for each of all its neurons, so each FC layer has $(I + 1) \cdot O$ parameters, where I is the number of input units of the layer, and O is the number of output units.
Each pooling layer divides width and height of input by 2 each, leaving depth untouched:

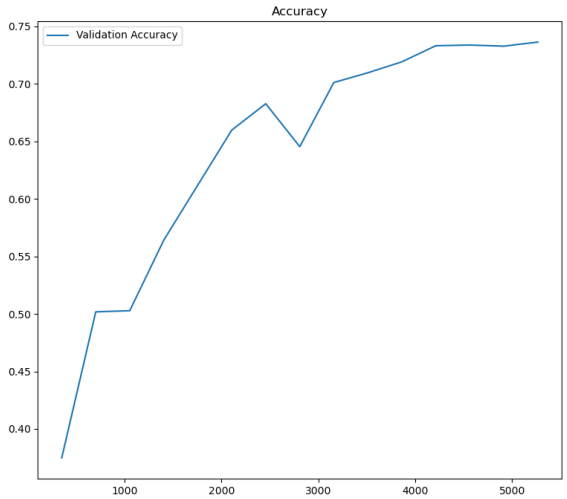
$$32 \times 32 \times 32 \underset{Pool1,Conv2}{\Rightarrow} 16 \times 16 \times 64 \underset{Pool2,Conv3}{\Rightarrow} 8 \times 8 \times 128 \underset{Pool3}{\Rightarrow} 4 \times 4 \times 128 \underset{Flatten}{\Rightarrow} 2048 \times 1$$

ConvLayer 1 has a 3-channel input and produce 32 feature maps:
$3 \cdot 26 \cdot 32 = 2\,496$ parameters.
ConvLayer 2 has a 32-channel input and produce 64 feature maps:
$32 \cdot 26 \cdot 64 = 53\,248$ parameters.
ConvLayer 3 has a 64-channel input and produce 128 feature maps:
$64 \cdot 26 \cdot 128 = 212\,992$ parameters.
FClayer1 has 2048 inputs and 64 output: $(2048 + 1) \cdot 64 = 131\,136$ parameters.
FClayer2 has 64 inputs and 10 output: $(64 + 1) \cdot 10 = 650$ parameters.
Taking the sum, we find that the model has $400\,522$ parameters.

Task 2

a)





b)

Table 1: Results

	Training	Validation	Test
Loss	0.39	0.79	0.81
Accuracy	0.87	0.73	0.73

Task 3

a) Trainer Details
Loss Function: Cross-entropy loss
Optimizer: SGD (LR: 5e-2)
Epochs: 10
Batch size: 32
Early Stopping: 4 Epochs

Model

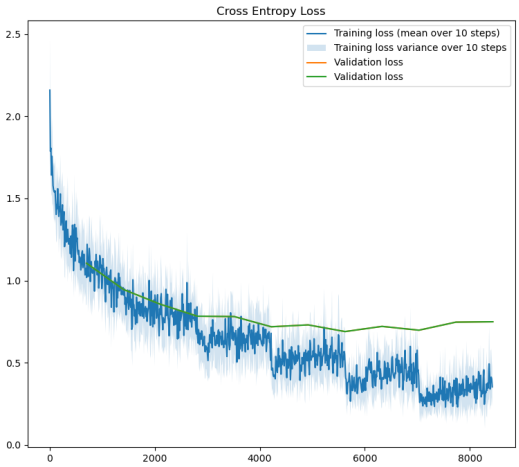
```
ImprovedModel(
 (feature_extractor): Sequential(
  (0): Sequential(
  (0): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): ReLU()
  (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_stats=True)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,ceil_mode=False)
  )
  (1): Sequential(
  (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): ReLU()
  (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_stats=True)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,ceil_mode=False)
  )
  (2): Sequential(
  (0): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): ReLU()
  (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_stats=True)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,ceil_mode=False)
  )
  (3): Flatten(start_dim=1, end_dim=-1)
 )
 (classifier): Sequential(
  (0): Dropout(p=0, inplace=False)
  (1): Linear(in_features=2048, out_features=64, bias=True)
  (2): ELU(alpha=1.0)
  (3): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_stats=True)
  (4): Linear(in_features=64, out_features=10, bias=True)
 )
)
```

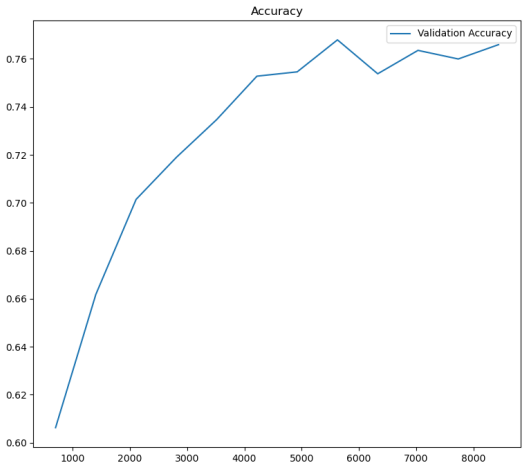
Note: See c) for further details.

b)

Table 2: Results

	Training	Validation	Test
Loss	0.23	0.80	0.83
Accuracy	0.92	0.76	0.75





c) (and a little a)) Values are the final validation accuracy. The baseline accuracy was 0.74

Changed optimizer to Adam: 0.715

Note: The results got worse, so we changed back to SGD. It is said, that SGD can generalize better and since our initial model already suffered from overfitting, Adam made it worse. However, once we added dropout later, Adam performed similar to SGD.

Change ReLU in classifier layer to ELU: 0.743

Note: ELU avoids the dead ReLU problem, but in our case the improvements were barely noticable. So dying ReLU was not an issue.

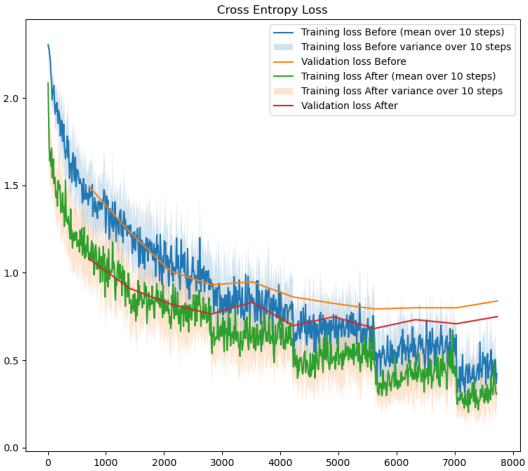
Batch normalization: 0.77

> Note: We used BatchNorm2d in convolutional layers and BatchNorm1d in the last layer. This caused the biggest jump in performance.

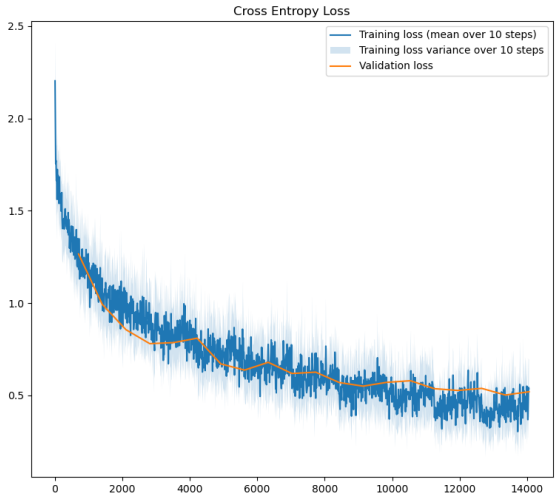
From here on the improvements count towards the 80% goal.
increased convolutional layer size from $[32, 64, 128]$ to $[64, 192, 512]$: 0.79
added dropout of 77%: 0.82

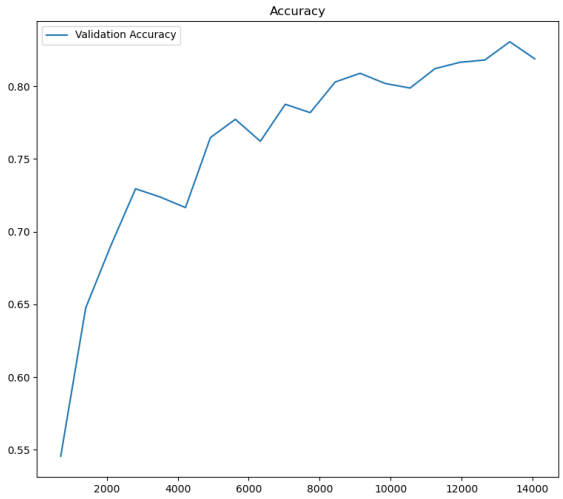
> Note: More layers lead to even more overfitting, but still increased accuracy. However, with dropout we finally tackled the overfitting and made a significant jump again.

d) We saw the biggest improvement towards the 75% goal using batch normalization as described in c)



e) As described in c), we applied dropout and increased the convolutional layer size.





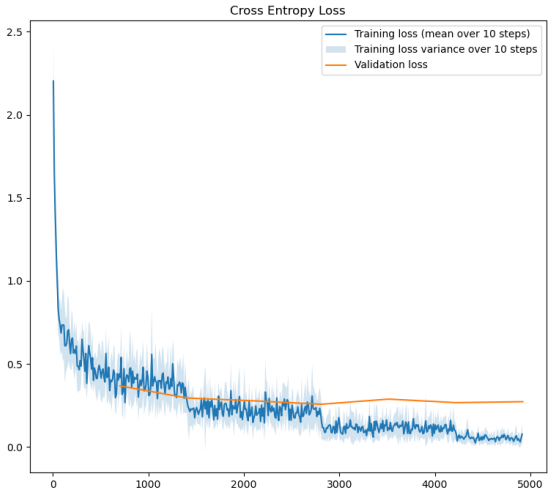
f) Our best model is still overfitting slightly, as we can see on the validation loss towards the end. However dropout massively reduced the overfitting issue compared to previous versions.

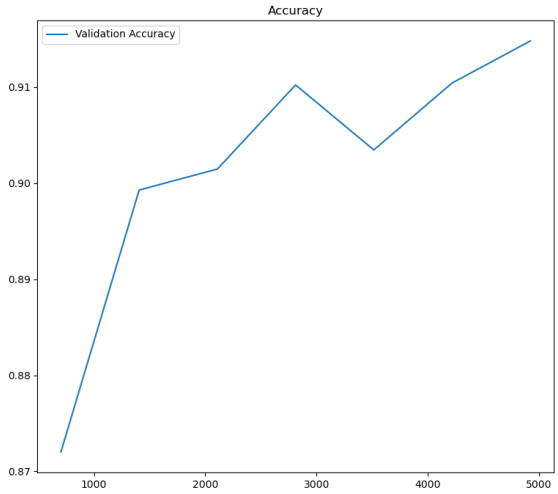
Task 4

Trainer Details
Loss Function: Cross-entropy loss
Optimizer: SGD (LR: 2e-2)
Epochs: 10
Batch size: 32
Early Stopping: 4 Epochs
Data Augmentation: transforms.Resize(224)

Table 3: Results

	Training	Validation	Test
Loss	0.04	0.33	0.35
Accuracy	0.99	0.90	0.90


Cross Entropy Loss
Training loss (mean over 10 steps)
Training loss variance over 10 steps
Validation loss
2.5
2.0
1.5
1.0
0.5
0.0
0
1000
2000
3000
4000
5000


Accuracy
Validation Accuracy
0.91
0.90
0.89
0.88
0.87
1000
2000
3000
4000
5000