INFOMCV Assignment 4

Marios Iacovou (1168533), Christos Papageorgiou (9114343) (Group 74)

Description and motivation of our baseline model and four variants

Baseline:

As a baseline model we build an adaptation of the LeNet model with the same structure that is described in LeCun's paper but with ReLU activation functions and max-pooling layers after each convolution instead of Tanh activation functions and average-pooling layers that more closely resemble the original architecture, in order to leverage more recent findings and advancements in deep learning. The model consists of two main parts: feature extraction ("first_wave" and "second_wave" in our code) and classification. Each of the convolution waves is followed by an auxiliary output which is explained later in the choices section.

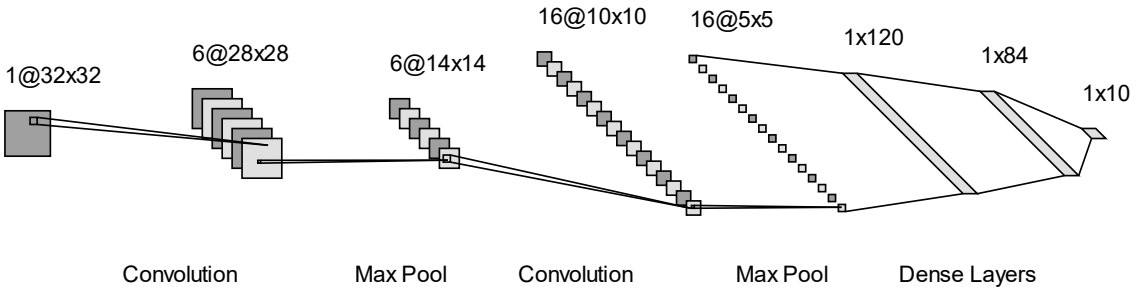
The "first_wave" starts with a convolutional layer with 1 input channel (suitable for grayscale images), 6 output channels, a 5x5 kernel size, a stride of 1, and padding of 2. The padding is used to increase the input size from 28x28 to 32x32 which was the original input size in LeCun's paper. The convolution has an output size of 28x28 as the original image size while increasing the depth to 6 channels. It is followed by a ReLU activation function and a max-pooling layer that reduces the spatial dimensions by half to 14x14 pixels. The "second_wave" contains another convolutional layer, with 6 input channels and 16 output channels, with a 5x5 kernel and a stride of 1. This results in reducing the spatial dimensions from 14x14 to 10x10 pixels. It's followed by another ReLU activation and a max-pooling layer that further reduces the size to 5x5 pixels.

The classifier part flattens the output from the convolutional layers and then passes it through a series of fully connected (Linear) layers. The first linear layer transforms the flattened features to a size of 120 and is followed by a ReLU. The next linear layer reduces the size to 84 and is again followed by a ReLU. Finally, the last linear layer transforms the output to 10 units (10 classes) and uses a SoftMax function (implemented as cross-entropy loss in PyTorch) for the final classification.

The model uses Kaiming uniform initialization for its weights, which helps in maintaining a controlled flow of gradients, which in turn stabilizes the learning process.

The summary and schematics of our baseline model are as follows:

```
==========================================================================================================================
Layer (type (var_name))                  Input Shape          Output Shape         Param #              Trainable
==========================================================================================================================
LeNet5 (LeNet5)                          [1, 1, 28, 28]       [1, 10]              --                   True
├─Sequential (first_wave)                [1, 1, 28, 28]       [1, 6, 14, 14]       --                   True
│    └─Conv2d (0)                        [1, 1, 28, 28]       [1, 6, 28, 28]       156                  True
│    └─ReLU (1)                          [1, 6, 28, 28]       [1, 6, 28, 28]       --                   --
│    └─MaxPool2d (2)                     [1, 6, 28, 28]       [1, 6, 14, 14]       --                   --
├─Linear (aux_output1)                   [1, 1176]            [1, 10]              11,770               True
├─Sequential (second_wave)               [1, 6, 14, 14]       [1, 16, 5, 5]        --                   True
│    └─Conv2d (0)                        [1, 6, 14, 14]       [1, 16, 10, 10]      2,416                True
│    └─ReLU (1)                          [1, 16, 10, 10]      [1, 16, 10, 10]      --                   --
│    └─MaxPool2d (2)                     [1, 16, 10, 10]      [1, 16, 5, 5]        --                   --
├─Linear (aux_output2)                   [1, 400]             [1, 10]              4,010                True
├─Sequential (classifier)                [1, 16, 5, 5]        [1, 10]              --                   True
│    └─Flatten (0)                       [1, 16, 5, 5]        [1, 400]             --                   --
│    └─Linear (1)                        [1, 400]             [1, 120]             48,120               True
│    └─ReLU (2)                          [1, 120]             [1, 120]             --                   --
│    └─Linear (3)                        [1, 120]             [1, 84]              10,164               True
│    └─ReLU (4)                          [1, 84]              [1, 84]              --                   --
│    └─Linear (5)                        [1, 84]              [1, 10]              850                  True
==========================================================================================================================
Total params: 77,486
Trainable params: 77,486
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.44
==========================================================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.05
Params size (MB): 0.31
Estimated Total Size (MB): 0.37
==========================================================================================================================
```

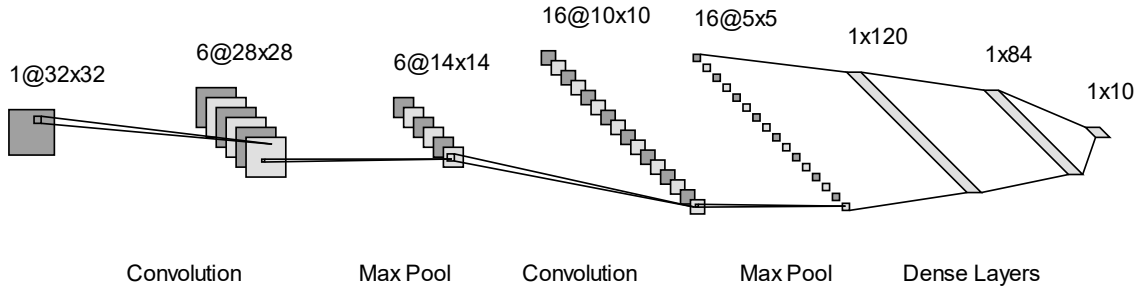


Variant 1:

For the first model variant we changed the learning rate of the optimizer to be adaptive by halving it every 5 epochs from the initial 0.001 value (some more details about the implementation can be found in the choice task section). This change should help in fine-tuning the model and converging to a better minimum of the loss function by taking smaller steps in the optimization process as training progresses.

The summary and schematics of our baseline model are as follows:

```
==========================================================================================================================
Layer (type (var_name))                  Input Shape          Output Shape         Param #              Trainable
==========================================================================================================================
LeNet5_v1 (LeNet5_v1)                    [1, 1, 28, 28]       [1, 10]              --                   True
├─Sequential (first_wave)                [1, 1, 28, 28]       [1, 6, 14, 14]       --                   True
|    └─Conv2d (0)                        [1, 1, 28, 28]       [1, 6, 28, 28]       156                  True
|    └─ReLU (1)                          [1, 6, 28, 28]       [1, 6, 28, 28]       --                   --
|    └─MaxPool2d (2)                     [1, 6, 28, 28]       [1, 6, 14, 14]       --                   --
├─Linear (aux_output1)                   [1, 1176]            [1, 10]              11,770               True
├─Sequential (second_wave)               [1, 6, 14, 14]       [1, 16, 5, 5]        --                   True
|    └─Conv2d (0)                        [1, 6, 14, 14]       [1, 16, 10, 10]      2,416                True
|    └─ReLU (1)                          [1, 16, 10, 10]      [1, 16, 10, 10]      --                   --
|    └─MaxPool2d (2)                     [1, 16, 10, 10]      [1, 16, 5, 5]        --                   --
├─Linear (aux_output2)                   [1, 400]             [1, 10]              4,010                True
├─Sequential (classifier)                [1, 16, 5, 5]        [1, 10]              --                   True
|    └─Flatten (0)                       [1, 16, 5, 5]        [1, 400]             --                   --
|    └─Linear (1)                        [1, 400]             [1, 120]             48,120               True
|    └─ReLU (2)                          [1, 120]             [1, 120]             --                   --
|    └─Linear (3)                        [1, 120]             [1, 84]              10,164               True
|    └─ReLU (4)                          [1, 84]              [1, 84]              --                   --
|    └─Linear (5)                        [1, 84]              [1, 10]              850                  True
==========================================================================================================================
Total params: 77,486
Trainable params: 77,486
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.44
==========================================================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.05
Params size (MB): 0.31
Estimated Total Size (MB): 0.37
==========================================================================================================================
```



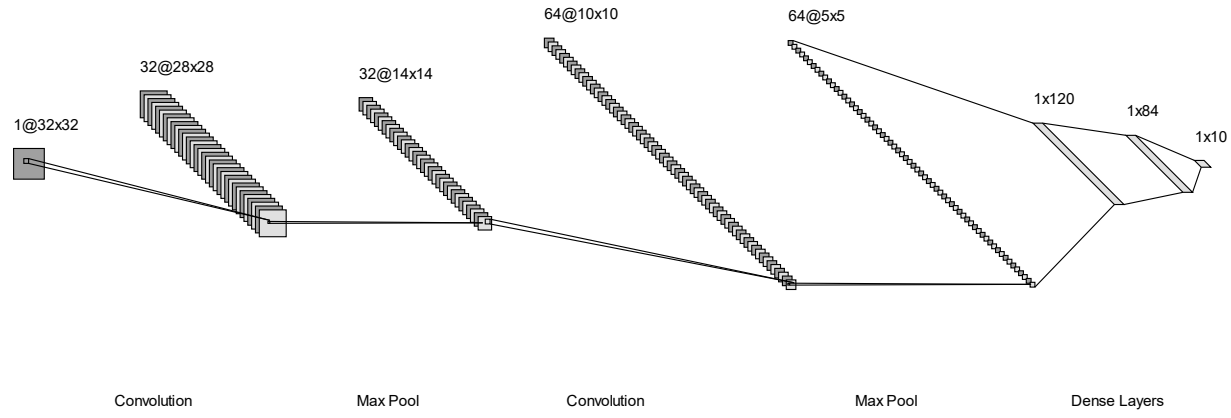
(They remain the same as the baseline model, just learning rate is decayed)

Variant 2:

For the second model variant we increased the number of filters in the convolution layers from 6 and 16 to 32 and 64 respectively. This modification was aimed at enhancing the model's capacity to learn more complex features from the dataset. A higher number of filters allows the model to capture a wider variety of patterns and details in the input images, potentially leading to improved recognition and classification performance.

The summary and schematics of our baseline model are as follows:

```
==========================================================================================================================
Layer (type (var_name))                  Input Shape          Output Shape         Param #              Trainable
==========================================================================================================================
LeNet5_v2 (LeNet5_v2)                    [1, 1, 28, 28]       [1, 10]              --                   True
├─Sequential (first_wave)                [1, 1, 28, 28]       [1, 32, 14, 14]      --                   True
│    └─Conv2d (0)                        [1, 1, 28, 28]       [1, 32, 28, 28]      832                  True
│    └─ReLU (1)                          [1, 32, 28, 28]      [1, 32, 28, 28]      --                   --
│    └─MaxPool2d (2)                     [1, 32, 28, 28]      [1, 32, 14, 14]      --                   --
├─Linear (aux_output1)                   [1, 6272]            [1, 10]              62,730               True
├─Sequential (second_wave)               [1, 32, 14, 14]      [1, 64, 5, 5]        --                   True
│    └─Conv2d (0)                        [1, 32, 14, 14]      [1, 64, 10, 10]      51,264               True
│    └─ReLU (1)                          [1, 64, 10, 10]      [1, 64, 10, 10]      --                   --
│    └─MaxPool2d (2)                     [1, 64, 10, 10]      [1, 64, 5, 5]        --                   --
├─Linear (aux_output2)                   [1, 1600]            [1, 10]              16,010               True
├─Sequential (classifier)                [1, 64, 5, 5]        [1, 10]              --                   True
│    └─Flatten (0)                       [1, 64, 5, 5]        [1, 1600]            --                   --
│    └─Linear (1)                        [1, 1600]            [1, 120]             192,120              True
│    └─ReLU (2)                          [1, 120]             [1, 120]             --                   --
│    └─Linear (3)                        [1, 120]             [1, 84]              10,164               True
│    └─ReLU (4)                          [1, 84]              [1, 84]              --                   --
│    └─Linear (5)                        [1, 84]              [1, 10]              850                  True
==========================================================================================================================
Total params: 333,970
Trainable params: 333,970
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 6.06
==========================================================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.25
Params size (MB): 1.34
Estimated Total Size (MB): 1.59
==========================================================================================================================
```

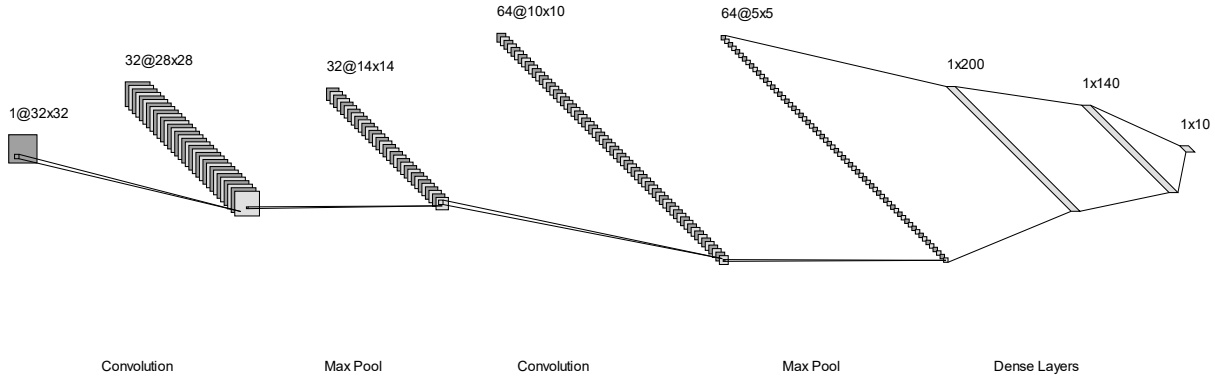


Variant 3:

For the third model variant we increased the number of neurons in the fully connected layers from 120 and 84 to 200 and 140 respectively while maintaining the original 0.7 ratio. This enhancement was designed to boost the model's ability to process and integrate the more complex features extracted by the enlarged convolutional layers from the previous model variant. By expanding the capacity of the fully connected layers, the model gains a greater potential for learning and representing patterns in the data, which is crucial for distinguishing subtle differences between various clothing categories.

The summary and schematics of our baseline model are as follows:

```
==========================================================================================================================================
Layer (type (var_name))                  Input Shape          Output Shape         Param #              Trainable
==========================================================================================================================================
LeNet5_v3 (LeNet5_v3)                    [1, 1, 28, 28]       [1, 10]              --                   True
├─Sequential (first_wave)                [1, 1, 28, 28]       [1, 32, 14, 14]      --                   True
|    └─Conv2d (0)                        [1, 1, 28, 28]       [1, 32, 28, 28]      832                  True
|    └─ReLU (1)                          [1, 32, 28, 28]      [1, 32, 28, 28]      --                   --
|    └─MaxPool2d (2)                     [1, 32, 28, 28]      [1, 32, 14, 14]      --                   --
├─Linear (aux_output1)                   [1, 6272]            [1, 10]              62,730               True
├─Sequential (second_wave)               [1, 32, 14, 14]      [1, 64, 5, 5]        --                   True
|    └─Conv2d (0)                        [1, 32, 14, 14]      [1, 64, 10, 10]      51,264               True
|    └─ReLU (1)                          [1, 64, 10, 10]      [1, 64, 10, 10]      --                   --
|    └─MaxPool2d (2)                     [1, 64, 10, 10]      [1, 64, 5, 5]        --                   --
├─Linear (aux_output2)                   [1, 1600]            [1, 10]              16,010               True
├─Sequential (classifier)                [1, 64, 5, 5]        [1, 10]              --                   True
|    └─Flatten (0)                       [1, 64, 5, 5]        [1, 1600]            --                   --
|    └─Linear (1)                        [1, 1600]            [1, 200]             320,200              True
|    └─ReLU (2)                          [1, 200]             [1, 200]             --                   --
|    └─Linear (3)                        [1, 200]             [1, 140]             28,140               True
|    └─ReLU (4)                          [1, 140]             [1, 140]             --                   --
|    └─Linear (5)                        [1, 140]             [1, 10]              1,410                True
==========================================================================================================================================
Total params: 480,586
Trainable params: 480,586
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 6.21
==========================================================================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.25
Params size (MB): 1.92
Estimated Total Size (MB): 2.18
==========================================================================================================================================
```

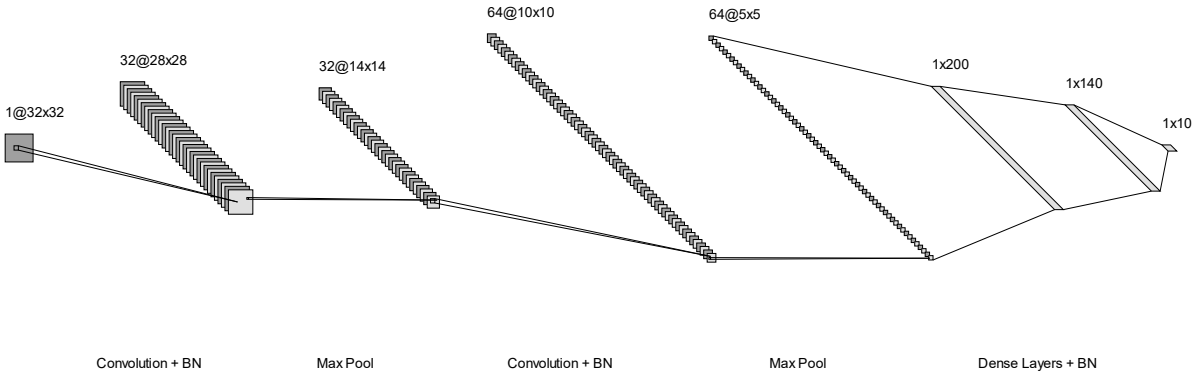


Variant 4:

For the fourth model variant we integrated batch normalization into both the convolutional and fully connected layers of our LeNet model. Batch normalization was used to standardize the inputs to a layer for each mini-batch. The re-centering and re-scaling of the inputs stabilizes the learning process by reducing internal covariate shift. The model also becomes less sensitive to the initial learning rate and initialization because the dependency of the gradients on the scale of the parameters is reduced. The batch statitistics should also introduce some noise to the network which acts as regularization and can alleviate any potential overfitting that may be introduced with the increased kernels and neurons in the previous model variants.

The summary and schematics of our baseline model are as follows:

```
==========================================================================================================================
Layer (type (var_name))                  Input Shape          Output Shape         Param #              Trainable
==========================================================================================================================
LeNet5_v4 (LeNet5_v4)                    [1, 1, 28, 28]       [1, 10]              --                   True
├─Sequential (first_wave)                [1, 1, 28, 28]       [1, 32, 14, 14]      --                   True
│    └─Conv2d (0)                        [1, 1, 28, 28]       [1, 32, 28, 28]      832                  True
│    └─BatchNorm2d (1)                   [1, 32, 28, 28]      [1, 32, 28, 28]      64                   True
│    └─ReLU (2)                          [1, 32, 28, 28]      [1, 32, 28, 28]      --                   --
│    └─MaxPool2d (3)                     [1, 32, 28, 28]      [1, 32, 14, 14]      --                   --
├─Linear (aux_output1)                   [1, 6272]            [1, 10]              62,730               True
├─Sequential (second_wave)               [1, 32, 14, 14]      [1, 64, 5, 5]        --                   True
│    └─Conv2d (0)                        [1, 32, 14, 14]      [1, 64, 10, 10]      51,264               True
│    └─BatchNorm2d (1)                   [1, 64, 10, 10]      [1, 64, 10, 10]      128                  True
│    └─ReLU (2)                          [1, 64, 10, 10]      [1, 64, 10, 10]      --                   --
│    └─MaxPool2d (3)                     [1, 64, 10, 10]      [1, 64, 5, 5]        --                   --
├─Linear (aux_output2)                   [1, 1600]            [1, 10]              16,010               True
├─Sequential (classifier)                [1, 64, 5, 5]        [1, 10]              --                   True
│    └─Flatten (0)                       [1, 64, 5, 5]        [1, 1600]            --                   --
│    └─Linear (1)                        [1, 1600]            [1, 200]             320,200              True
│    └─BatchNorm1d (2)                   [1, 200]             [1, 200]             400                  True
│    └─ReLU (3)                          [1, 200]             [1, 200]             --                   --
│    └─Linear (4)                        [1, 200]             [1, 140]             28,140               True
│    └─BatchNorm1d (5)                   [1, 140]             [1, 140]             280                  True
│    └─ReLU (6)                          [1, 140]             [1, 140]             --                   --
│    └─Linear (7)                        [1, 140]             [1, 10]              1,410                True
==========================================================================================================================
Total params: 481,458
Trainable params: 481,458
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 6.21
==========================================================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.51
Params size (MB): 1.93
Estimated Total Size (MB): 2.44
==========================================================================================================================
```

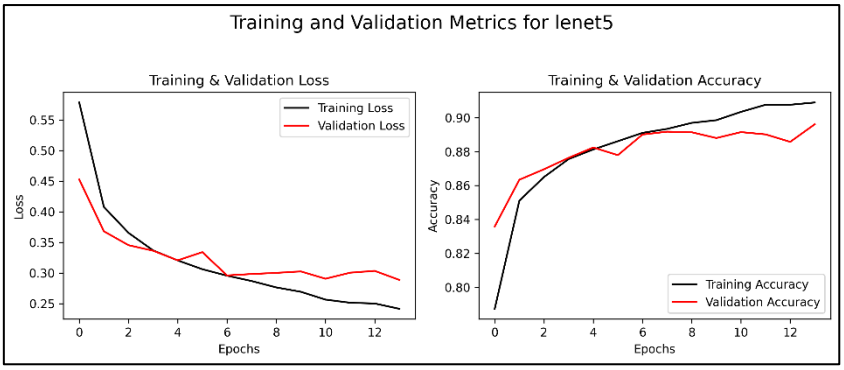


Training and validation loss for all five models and discussion of results in terms of models

The Fashion-MNIST dataset data is split into training and test sets of 60000 and 10000 examples with a batch size of 32. We calculate the mean and standard deviation of the data in the training set and normalize both the training set and testing set to ensure a consistent scale across the dataset for more effective learning dynamics. We also perform data augmentations to the training set which are explained in choice tasks. For the training of the models, we introduce a split of the training set into 80% training and 20% validation data. The training takes place in 15 epochs and at the end the model is reverted to its state in the epoch that had the highest validation accuracy.

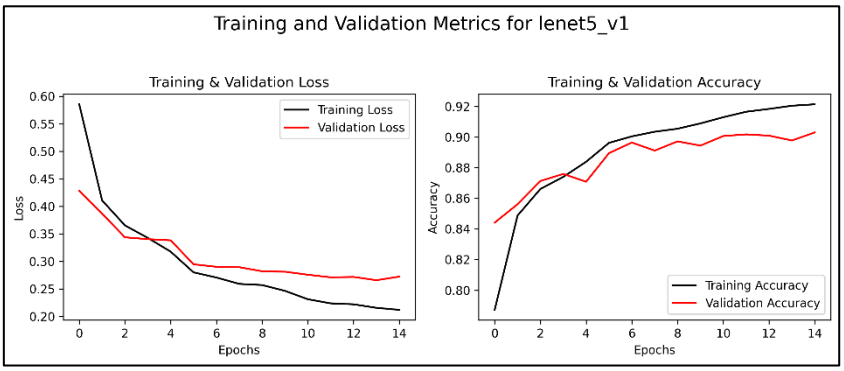
Baseline:

Training and validating using the baseline model gives us a final epoch training accuracy of 91.2% with loss of 0.237 and validation accuracy of 89% with a loss of 0.305. The best epoch results are at epoch 14 with a training accuracy of 90.9% with loss of 0.242 and validation accuracy of 89.6% and loss of 0.289. We can see that this initial architecture gives us relatively good results on the validation.



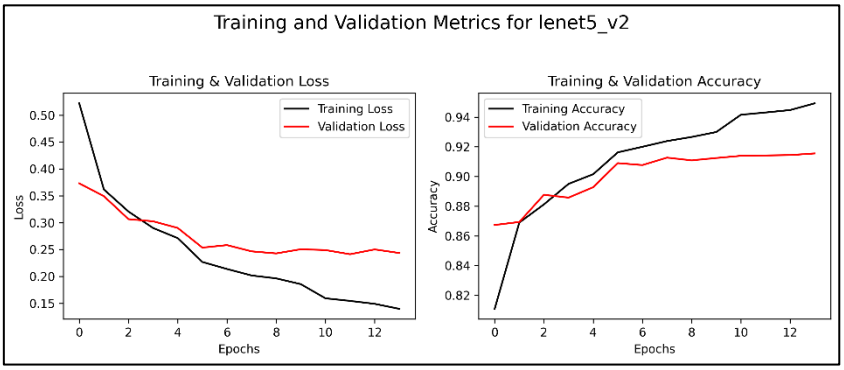
Variant 1:

Training and validating using the first variant model gives us a final epoch training accuracy of 92.1% with loss of 0.212 and validation accuracy of 90.3% with a loss of 0.272. The best epoch results are at the same epoch. We can see that the adaptive learning rate of this model has slightly improved on the results compared to the baseline model. The loss curves converge better due to the smaller steps as the training progresses.

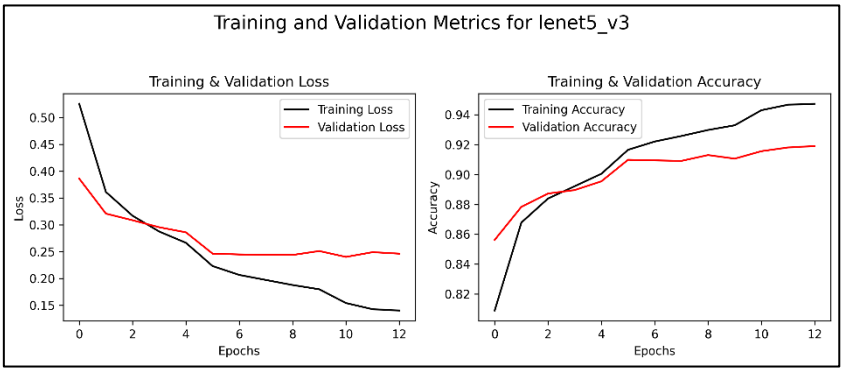


Variant 2:

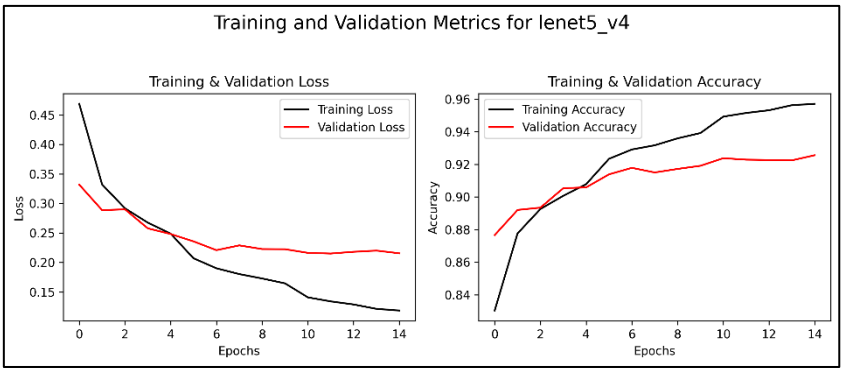
Training and validating using the second variant model gives us a final epoch training accuracy of 95.1% with loss of 0.137 and validation accuracy of 91.5% with a loss of 0.252. The best epoch results are at epoch 14 with a training accuracy of 94.9% with loss of 0.136 and validation accuracy of 91.6% and loss of 0.244. This more complex model gives a big increase in training accuracy compared to the previous variant due to the increased number of features that can be captured with the larger number of kernels introduced in the convolution layers.



Variant 3:
Training and validating using the third variant model gives us a final epoch training accuracy of 95.5% with loss of 0.124 and validation accuracy of 91.7% with a loss of 0.251. The best epoch results are at epoch 13 with a training accuracy of 94.7% with loss of 0.14 and validation accuracy of 91.9% and loss of 0.246. We can see that the performance of this model is similar to the performance of the previous variant with a slightly higher validation accuracy but slightly lower training accuracy. This tells us that the enlarged convolutional layers from the previous variant in this case do not necessarily require more neurons in the fully connected layers to be integrated well.



Variant 4:
Training and validating using the fourth variant model gives us a final epoch training accuracy of 95.7% with loss of 0.118 and validation accuracy of 92.6% with a loss of 0.215. The best epoch results are at the same epoch. We can see that this model performs the best compared to the previous model and all the other models as it has the highest validation accuracy and training accuracy. This validates our expectations of the benefits of batch normalization from the description of the model. Overall, the expanded architecture with more kernels and neurons coupled with batch normalization and the adaptive learning rate gives us a well-rounded model for this specific task that avoids overfitting while converging to good minimums of the loss functions with stable learning. This model is selected as our best model for testing on the test set.



Link to our model weights

Our models are exported to TorchScript and saved to .pt files. Our model directory includes the baseline model with its 4 variants, as well as the best model (variant 4) trained at different cross-validation folds for a choice task described later in the report.

https://github.com/ChristosP1/Convolutional-neural-networks/tree/main/models

Table with training and validation top-1 accuracy for all five models

As mentioned before, the fourth model variant had the best performance overall out of all the models. We see an improvement in the validation accuracy between the model iterations, with only model variants 2 and 3 being relatively close in performance in iterative paired comparisons. Validation accuracy for all models is close to the training accuracy so we expect our models to generalize well as they don't overfit on training data.

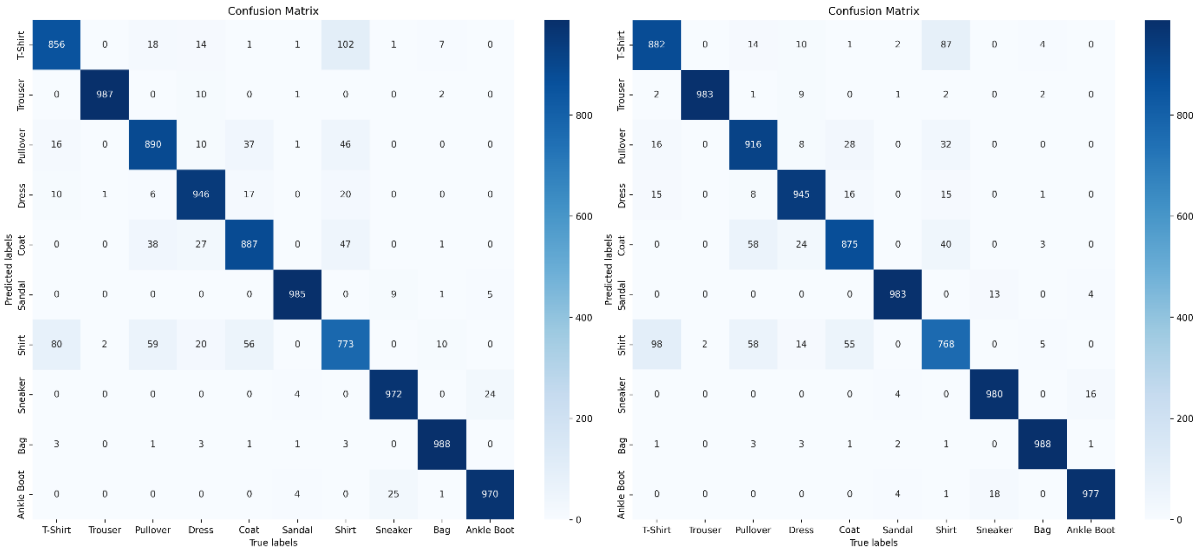
Model name (@ best epoch)	Training top-1 accuracy (%)	Validation top-1 accuracy (%)
Baseline (@ epoch 14)	90.9%	89.6%
Variant 1 (@ epoch 15)	92.1%	90.3%
Variant 2 (@ epoch 14)	94.9%	91.6%
Variant 3 (@ epoch 13)	94.7%	91.9%
Variant 4 (@ epoch 15)	95.7%	92.6%

Discussion of results of the best model evaluated on the test set

Variant 4 was selected as the best model to be tested on the test set data. First, we evaluate the model that has already been trained on the training data on the test set. This results in a testing accuracy of 92.5% and a loss of 0.231 which shows us that our model can generalize well to unseen data. We can also see that this accuracy is close to the validation accuracy from training of 92.6% which tells us that our performance estimation during training was reliable.

We then retrain the best model on training and validation data and evaluate it on the test set. This results in a testing accuracy of 92.9% and a loss of 0.214. Compared to the model only trained on training data, we can see that the testing accuracy now is a bit higher, so the model uses the additional data from the validation set to slightly improve on its predictions. Most of the accuracy however is derived from its ability to generalize well, which was already present when training on just the training data.

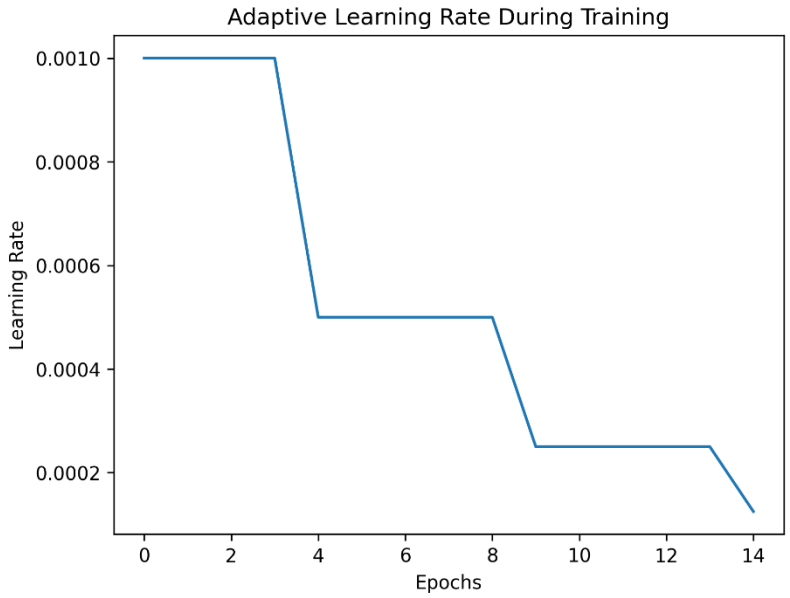
When looking at the confusion matrices of actual versus predicted labels resulting from each of the evaluations, we can see that the differences lie in a few difficult classes. The model trained on training and validation data can better classify shirts for example and not mistake them as much to t-shirts. This is important because when visualizing the classes using t-SNE we can see that these classes lie close together in 2D space. These difficult classes are discussed more and visualized in the choices section. The confusion matrices of the first (left) and second (right) evaluation can be seen below:



Choice tasks

CHOICE 1: «*Create and apply a function to decrease the learning rate at a 1/2 of the value every 5 epochs»*

For the first model variant onwards we changed the learning rate of the optimizer to be adaptive by halving it every 5 epochs from the initial 0.001 value. To achieve this, we employed a scheduler using the StepLR function from torch.optim.lr_scheduler with step_size=5 (to update every 5 epochs) and gamma=0.5 (to divide the learning rate by 2) and applied it to the optimizer. The following is a graph showing the adaptive learning rate over 15 epochs of training:



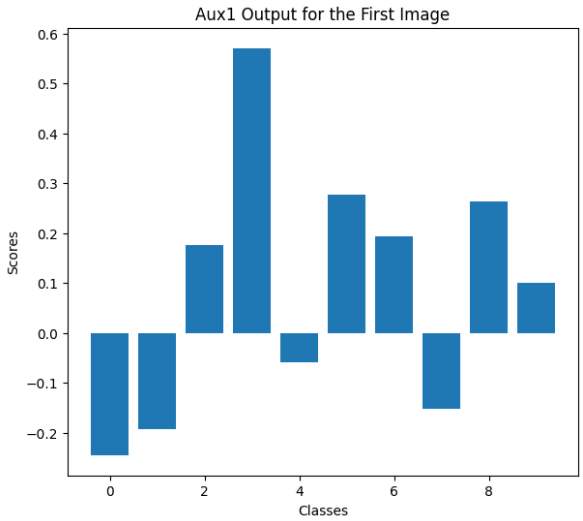
CHOICE 2: «*Instead of having a fixed validation set, implement k-fold cross-validation»*

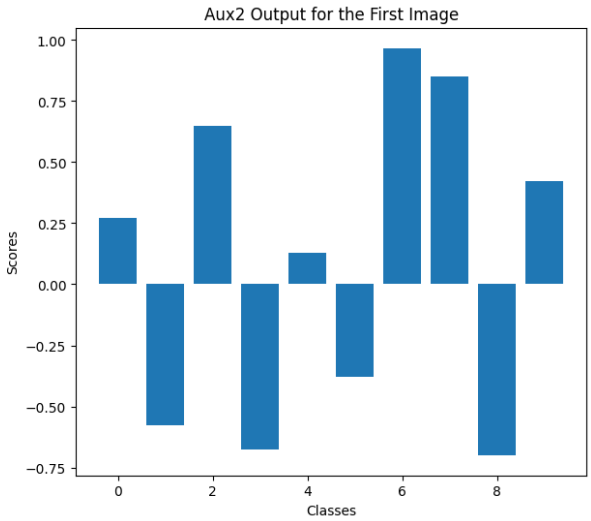
To see the differences in training validation results, we retrained the best model (variant 4) with k-fold cross-validation. For our experiment we used 5 folds. On the fixed validation set, as shown earlier in the report, on the best epoch we had a training accuracy of 95.7% with loss of 0.118 and validation accuracy of 92.6% with a loss of 0.215.

Comparatively, when averaging the results of the best epochs over the 5 folds during cross validation we get a training accuracy of 95.4% with a loss of 0.127 and validation accuracy of 92.3% with a loss of 0.228. Looking at the results of every fold individually, we can see that each fold has similar results without any deviations (plots can be found in our graphs folder). The results are almost the same as those using the fixed validation set so we can say that the fixed validation set provides a representative sample of the data. The model's performance is consistent across different subsets of the data which gives us an indication of its good generalization.

CHOICE 3: «*Create output layers at different parts of the network for additional feedback. Show and explain some outputs of a fully trained network»*

After each set of convolution and pooling layers, we implemented an auxiliary output layer that gives us an idea of what the network can predict in that stage. We get a test image from a batch of the test set to see what each of these auxiliary output layers classify this image as. As can be seen from the plots below, the predictions of each of these auxiliary output layers differ, with the first auxiliary output layer giving the highest chance of the image being a dress (class 3) and the second one giving the highest chance of the image being a shirt (class 6).



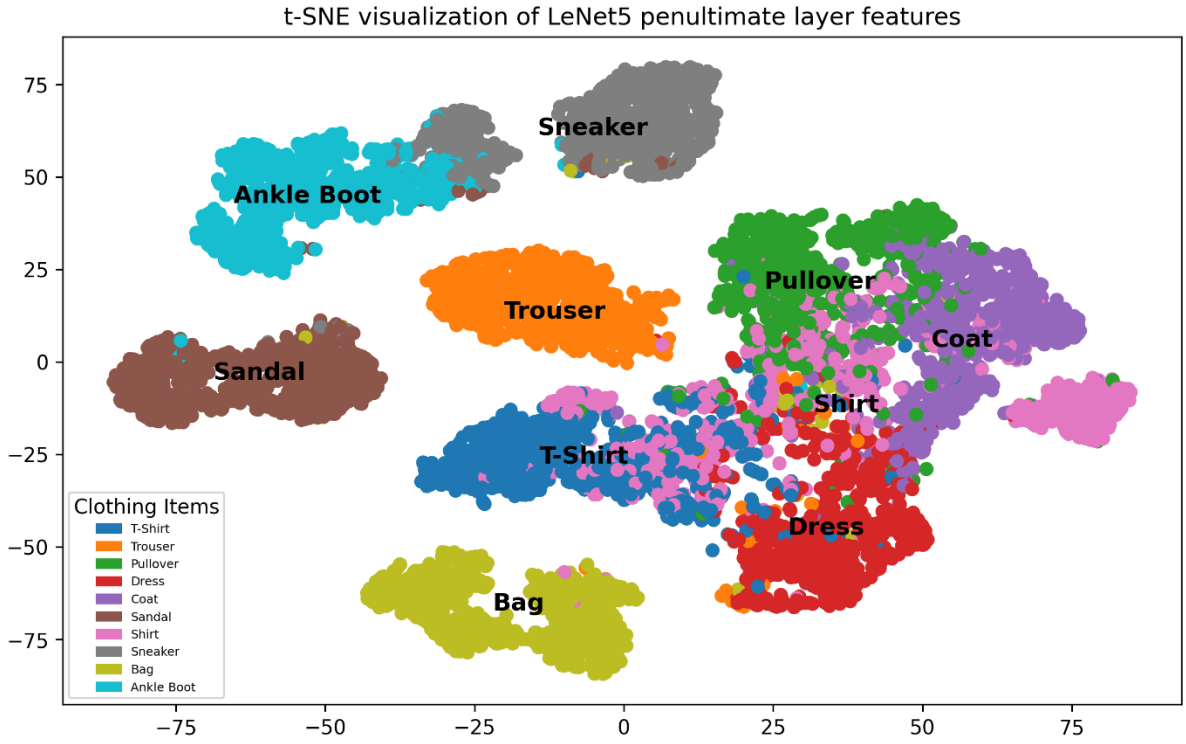


Additionally, we can look at how these highest scores match over multiple images. This is run over several iterations and the results are averaged to provide a better calculation. The result is that the percentage of matching highest scores is only about 14%. These findings tell us that the higher-level features from the second convolution change the understanding of the image by the network compared to low-level features. This is expected as lower-level features tend to be more generic while higher-level features are more specialized and class specific.

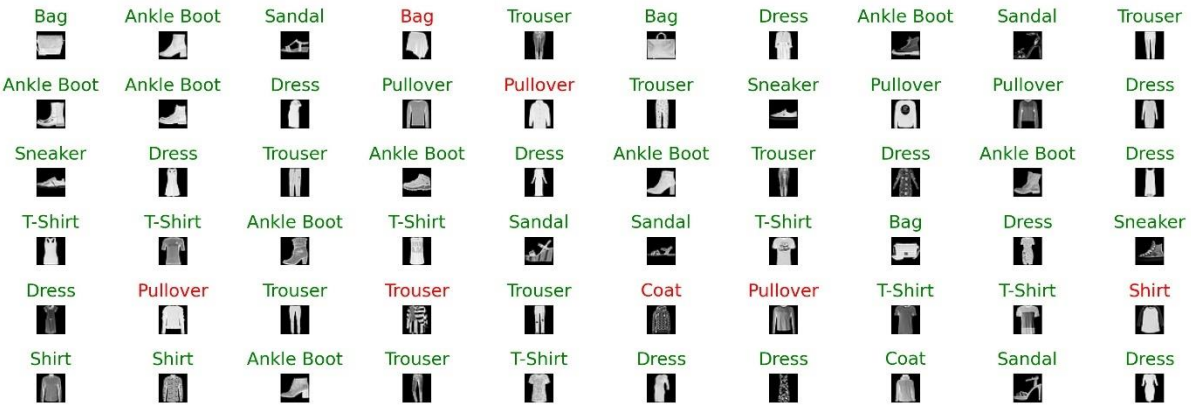
CHOICE 4: «*Perform data augmentation techniques*»
We implemented the following augmentation techniques to our training data to improve the performance of our models:
1. RandomHorizontalFlip (p=0.25): Randomly flips the image horizontally with a probability of 25%. For our dataset, such flipping is sensible as it mirrors images, representing how clothing can appear in different orientations. This helps the model generalize better to varying orientations of clothing in real-world scenarios.
2. ColorJitter (brightness=0.2, contrast=0.2): Randomly adjusts the brightness and contrast of the image. Although our dataset is a grayscale dataset, these slight adjustments can simulate variations in lighting and fabric texture. This makes the model robust to changes in lighting conditions and material qualities.
3. RandomAutocontrast (p=0.2): Improves the contrast in an image by scaling its pixel values. It can help emphasize features and shapes in clothing items, aiding the model to distinguish between different clothing items more effectively.
4. RandomSolarize (p=0.2, threshold=15): Inverts all pixel values above a specific threshold, introducing a form of color inversion effect. By setting a threshold of 15 we effectively change the pixel values of clothing items while keeping the background, which is black, unchanged. With this approach we introduce a variety of grey shades in the clothing images while maintaining a consistent background.

CHOICE 5: «*Provide t-SNE visualization of the fully connected layer before your output layer*»
When plotting a t-SNE visualization we can see the classes that lie closer together in 2D space to anticipate possible confusions in the network. From our plot we can see that classes such as sneakers and ankle boots have some areas of overlap as they are both footwear and look somewhat similar. The biggest confusions we can anticipate by looking at the 2D space are those relating to upper-body clothing. Classes like shirt, coat, t-shirt, dress, and pullover have a big area of overlap. We would expect shirts, being very sparse in that space of classes, to be one of the difficult classes

to classify. Looking at the confusion matrices after our training and evaluation reported in previous sections, we can see that this is the case, with shirts being confused for other classes like T-shirts.



Drawing random items from the dataset and looking at their classifications by the baseline model, we can see that most of the confusion comes from upper-body classes.



CHOICE Bonus: TensorBoard tracking and visualization of live model performance
We came up with an additional choice task of visualizing and comparing the live performance of our models using the TensorBoard toolkit. TensorBoard provides an interactive dashboard for visualizing various metrics, such as training and validation loss and accuracy, in real-time. This visualization is achieved using the SummaryWriter class from torch.utils.tensorboard. SummaryWriter facilitates the logging of these metrics during the model's training process. It creates logs that TensorBoard reads and displays in a graph format on a browser interface. These logs are generated each epoch, recording important statistics like accuracy and loss, and are stored in a designated directory. By tracking these metrics over time, TensorBoard allows us to monitor the model's learning progression and adjust strategies if needed, directly from the browser. For each of our five models, we used the same SummaryWriter instance, allowing us to overlay and compare these key metrics across all models on a unified set of graphs. We can update the graphs during the training process by clicking

the update arrow at the top right corner of Tensorboard's interface. Some visualizations of this interface during training and after training can be seen below: