**INFOMCV Assignment 4**

**Authors (Group number)**

**Description and motivation of our baseline model and four variants**

**Baseline**

As baseline we build an adaptation of the LeNet model with the same structure that is described in LeCun’s paper with ReLU activation functions and MaxPooling2d layers after each convolution. It consists of two main parts: feature extraction (“first\_wave” and “second\_wave“ in our code) and classification.

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. This keeps the output size the same as the input (28x28 pixels), while increasing the depth to 6. 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, followed by a ReLU. The next linear layer reduces the size to 84, again followed by a ReLU. Finally, the last linear layer transforms the output to 10 units (10 classes).

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.

**Variant 1**

For the first variant we applied the adaptive learning rate (CHOICE 1) that decays the learning rate of the optimizer every 5 epochs. To model 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 LR by 2) and applied it to the optimizer. This helps in fine-tuning the model by taking smaller steps in the optimization process as training progresses.

**Variant 2**

In the second 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 Fashion MNIST 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.

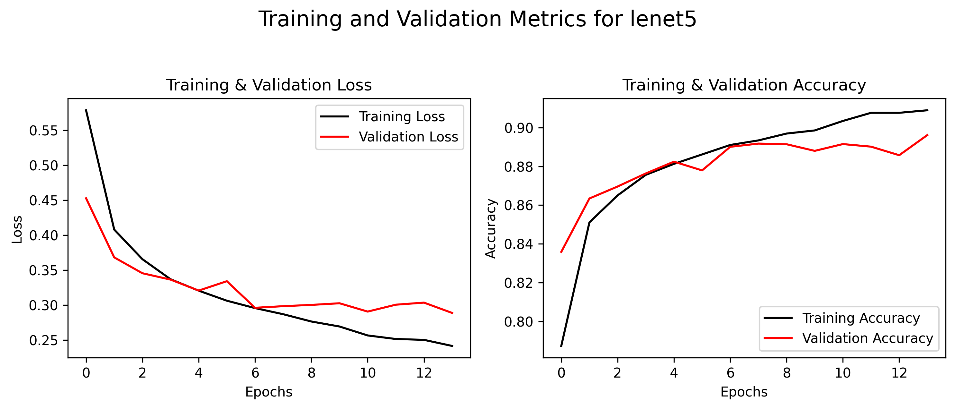
**Variant 3**

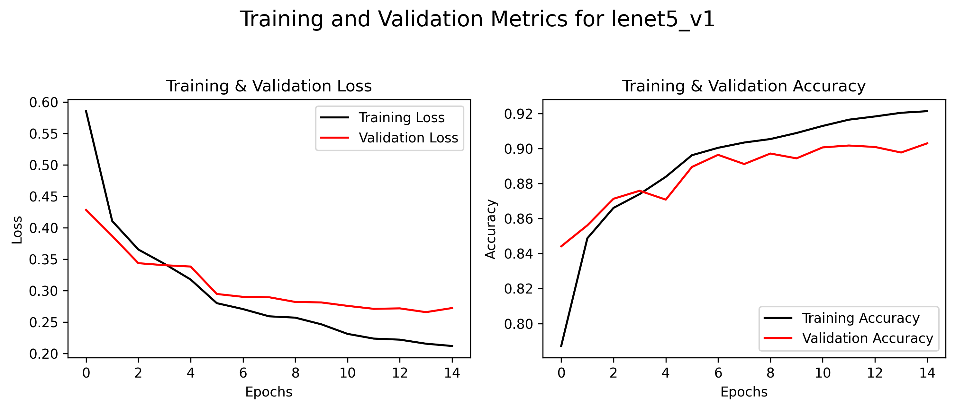
In the third variant, we increased the number of neurons in the fully connected layers from 120 and 84 to 200 and 140, respectively, while maintaining a 0.7 ratio. This enhancement was designed to boost the model's ability to process and integrate the more complex features extracted by the enlarged convolutional layers. 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.

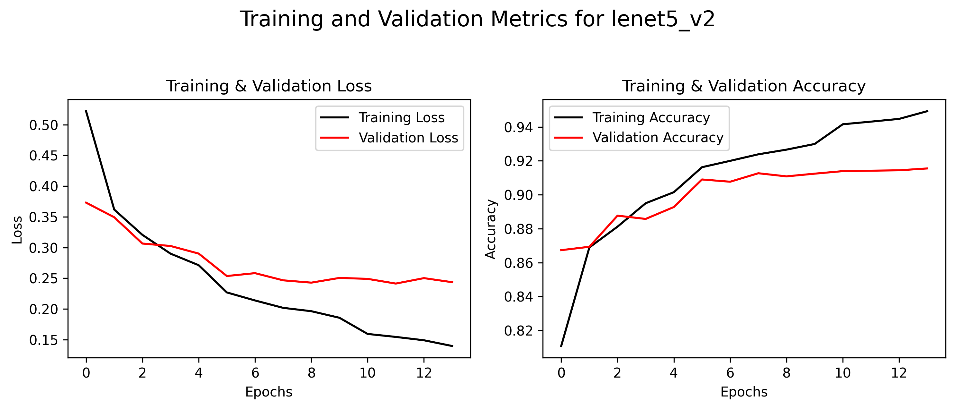
**Variant 4**

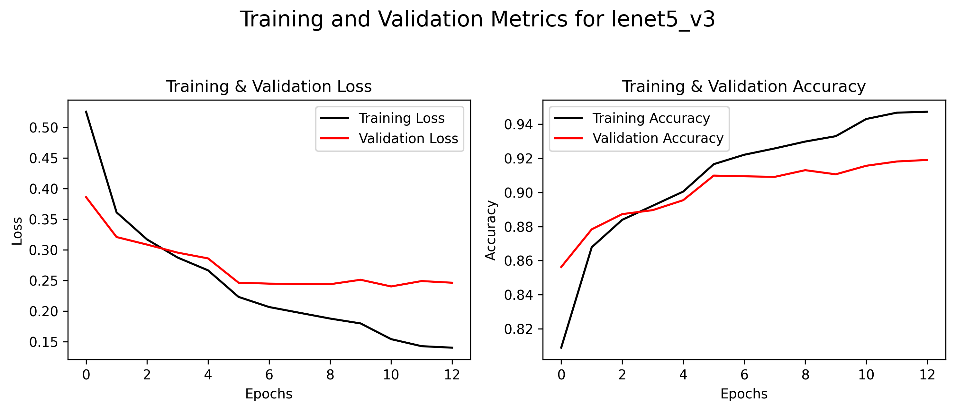
In the fourth variant, we integrated batch normalization into both the convolutional and fully connected layers of our LeNet model. Batch normalization was used to standardize the inputs to a layer for each mini-batch. This process stabilizes the learning process and reduces the number of training epochs required to train deep networks. By applying batch normalization, we aimed to achieve faster convergence during training, improve the model's performance by reducing internal covariate shift, and make the model less sensitive to the initial learning rate and initialization.

**Training and validation loss for all five models**



****



****

**A graph of a line and a line

Description automatically generated with medium confidence**

**Link to our model weights**

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

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

|  |  |  |
| --- | --- | --- |
| **Model name** | **Training top-1 accuracy (%)** | **Validation top-1 accuracy (%)** |
| lenet\_5 | 90.1% | 89.62% |
| lenet\_5\_v1 | 92.13% | 90.29% |
| lenet\_5\_v2 | 94.92% | 91.55% |
| lenet\_5\_v3 | 94.72% | 91.9% |
| lenet\_5\_v4 | 95.71% | 92.56% |

**Discuss your results in terms of your model**

(Discuss the results in terms of complexity, type of layers, overfitting measures, etc. Make pair-wise comparisons between the four variants and the baseline model. Approx. 0.5 page.)

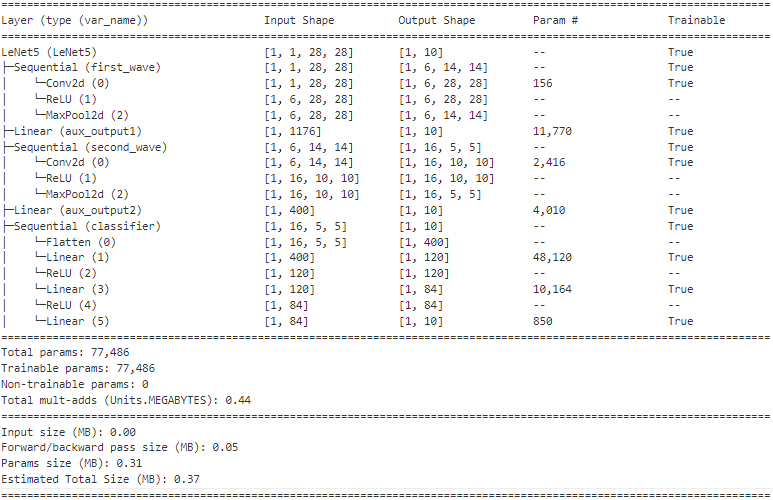
**Discuss the differences between the two models evaluated on the test set**

(Discuss the potential causes for (lack of) differences in terms of architecture. Also compare the results of each model to the training performance. Approx. 0.5 page.)

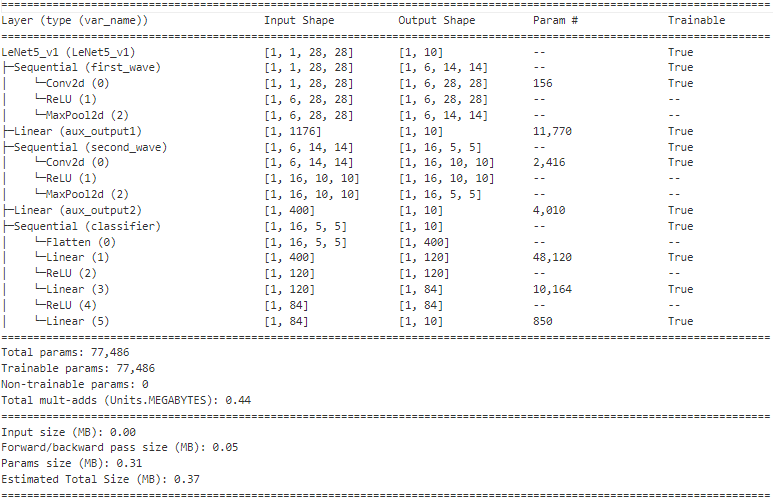
**Choice tasks**

(Indicate which ones you did, and how you did them; Approx. half a page.)

**Lenet5 baseline summary**



**Lenet5 v1 summary**



**Lenet5 v2 summary**

A screenshot of a computer

Description automatically generated

**Lenet5 v3 summary**

A screenshot of a computer

Description automatically generated

**Lenet5 v4 summary**

A screenshot of a computer

Description automatically generated