INFOMCV Assignment 4
Alejandro Magarzo Gonzalo, Alimohamed Jaffer (Group number: 38)

Description and motivation of your baseline model and four variants

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             156
              ReLU-2            [-1, 6, 28, 28]               0
         AvgPool2d-3            [-1, 6, 14, 14]               0
            Conv2d-4           [-1, 16, 10, 10]           2,416
              ReLU-5           [-1, 16, 10, 10]               0
         AvgPool2d-6             [-1, 16, 5, 5]               0
           Flatten-7                  [-1, 400]               0
            Linear-8                  [-1, 120]          48,120
              ReLU-9                  [-1, 120]               0
           Linear-10                   [-1, 84]          10,164
             ReLU-11                   [-1, 84]               0
           Linear-12                   [-1, 10]             850
================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
----------------------------------------------------------------
```

Our baseline model is LeNet-5 architecture. We begin with an input layer where the model takes 1-channel greyscale images of size 28x28. Then we implement our first convolution layer where we apply 6 filters of size 5x5 to the input image to extract the features. We use multiple filters to ensure that network is able to learn various aspects of the image such as edges, corners, textures etc. We also apply a padding of 2 pixels because our images are 28x28 pixels and also we do this to ensure that the output feature maps are large enough to apply pooling later. We then apply a ReLU activation allowing the model to learn more complex patterns. Next a 2x2 average pooling is applied with a stride of 2 to down sample the feature maps by half, reducing the spatial dimensions to 14x14. We then apply a second convolutional layer where we now have 16 filters, each receiving input from all 6 feature maps created by the previous convolutional layer. We do not apply padding and the size of the filters reduce the dimensions to 10x10. We once again apply ReLU as previously done before and thereafter, apply a second layer of pooling which further reduces the size of the feature maps to 5x5. We then have three connected layers. For the first connected layer we have 400 in-features because when flattening the output of the last pooling layer we get a single vector of 16*5*5. We have 120 out features where the connected layer transforms the 400-dimensional vector into a 120-dimensional space. For the second connected layer we have 120 in features because is the output from the previous fully connected layer. We have 84 out features because we are taking the 120-dimensional vector from the previous layer and reducing it to 84 dimensions. And finally, for the third connecting layer we have 84 in features which is the output from the previous layer. We have 10 out features which corresponds to the number of classes in the classification task.

Model 1: Adding Dropout

We decided to add dropout layers with a probability of 0.5 in the classifier part of the network. We did this to introduce regularization and we basically do this by randomly zeroing half of the output units during training, which helps to prevent overfitting.

Model 2: Adding Batch Normalization
We decided to include batch normalization layers after both convolutional layers. What basically happens is that we normalize the output of the convolutional layers, which can speed up the training by allowing higher learning rates and thus can potentially improve generalization. The parameters 6 and 16 correspond to the number of output channels from the convolutional layers.

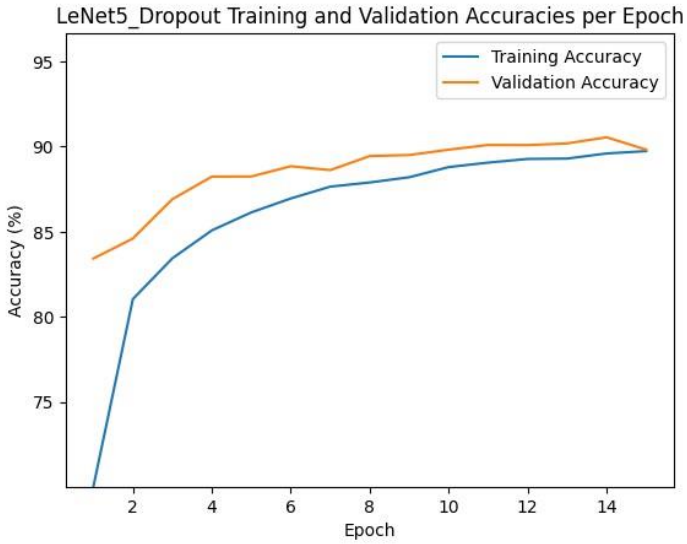
Model 3: Changing Activation Function to Leaky ReLU
We decided to change the ReLU activation function with the Leaky ReLU in the feature extraction and classifier sections. We wanted to try with Leaky ReLU because it allows for a small non-zero gradient when the unit is not active. This modification can help to alleviate the dying ReLU problem, where neurons can sometimes become inactive and never activate on any datapoint. It can potentially lead to more consistent training and better performance.

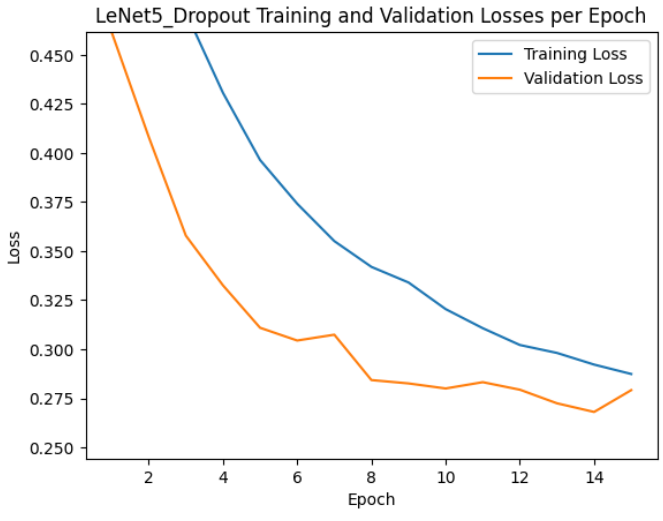
Model 4: Changing Kernel Size
We decided to change the kernel size from 5 to 7 which changes the output size of the convolutional layers and thus the in features of the first fully connected layer is updated to 256. Increasing the kernel size means that each filter in the convolutional layer looks at a larger area of the image in one glance. This allows the filter to pick up patterns that are more spread out, like a large curve or the background context around a digit, rather than just the small details.

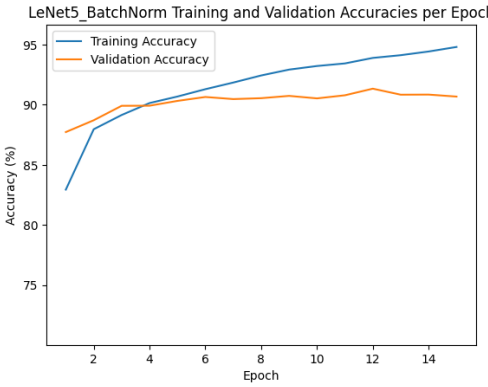
Model 0: Changing Kernel Size (Smaller)
The first convolutional layer's kernel size is reduced to 4 with padding adjusted (padding from 2 to 0) in the second convolution layer to maintain the output size. A smaller kernel size helps the network focus on more localized features and can be computationally more efficient with fewer parameters.

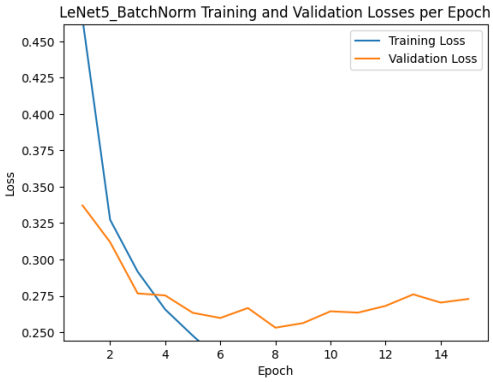
Model 5: Increasing Number of Filters
We decided to increase the number of filters in both convolutional layers. In the first convolutional layer the out channel is increased from 6 to 12 and in the second convolutional layer the out channel is increased from 16 to 32. Thus, the in features of the first fully connected layer is updated to reflect the increased number of features from the second convolutional layer in features=32*5*5 and this getting out features=120. Increasing the number of filters in the convolutional layers allows the network to learn a more diverse set of features at each layer. This could potentially improve the network's ability to distinguish between more complex patterns and lead to better performance on the classification tasks.

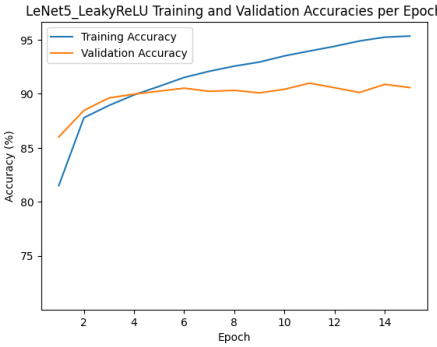
Training and validation loss for all models

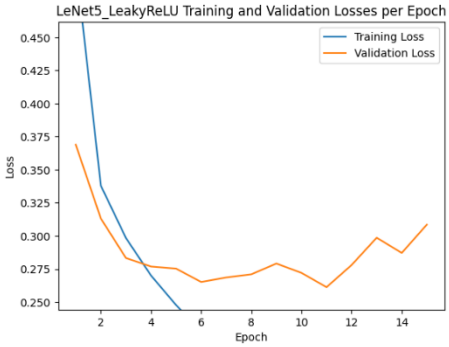


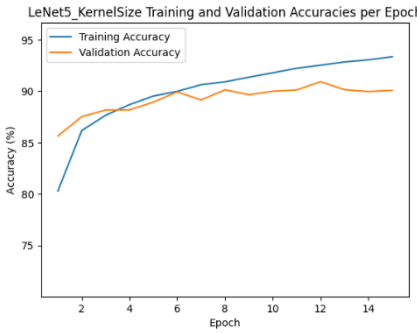


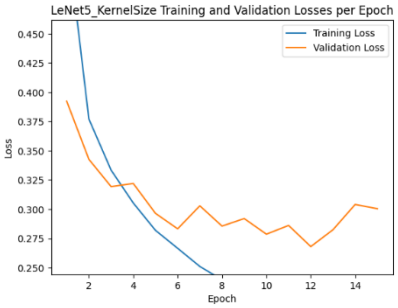

LeNet5_BatchNorm Training and Validation Accuracies per Epoch
Training Accuracy
Validation Accuracy
Accuracy (%)
Epoch

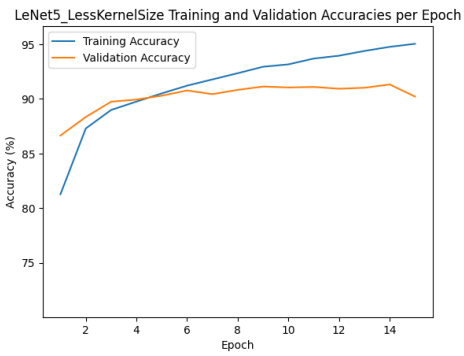

LeNet5_BatchNorm Training and Validation Losses per Epoch
Training Loss
Validation Loss
Loss
Epoch

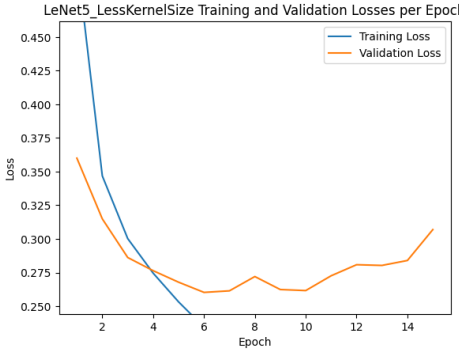

LeNet5_LeakyReLU Training and Validation Accuracies per Epoch
Training Accuracy
Validation Accuracy
Accuracy (%)
Epoch

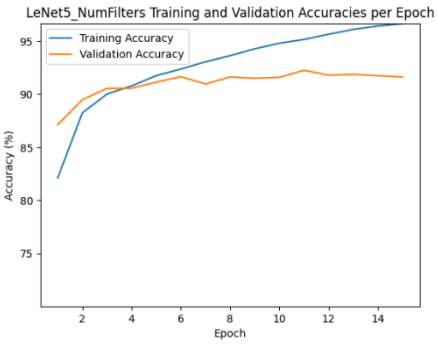

LeNet5_LeakyReLU Training and Validation Losses per Epoch
Training Loss
Validation Loss
Loss
Epoch

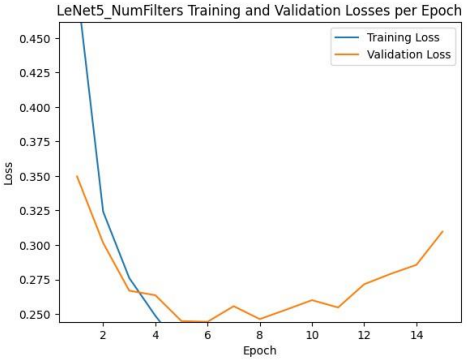

LeNet5_KernelSize Training and Validation Accuracies per Epoch
Training Accuracy
Validation Accuracy
Accuracy (%)
Epoch

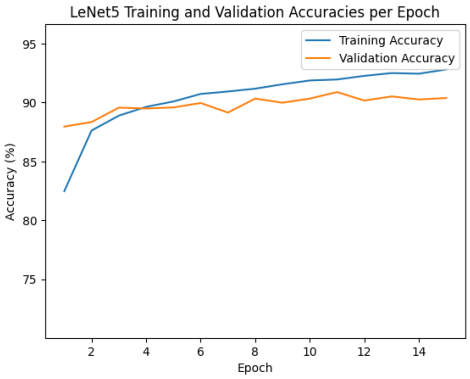

LeNet5_KernelSize Training and Validation Losses per Epoch
Training Loss
Validation Loss
Loss
Epoch

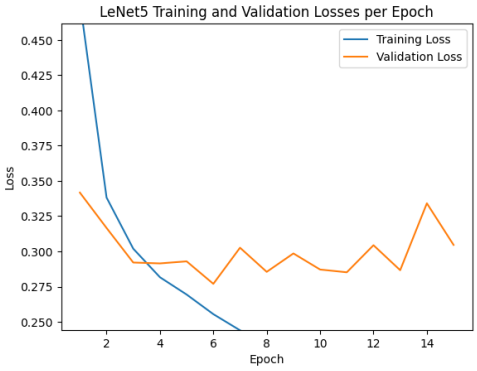

LeNet5_LessKernelSize Training and Validation Accuracies per Epoch
Training Accuracy
Validation Accuracy
Accuracy (%)
Epoch


LeNet5_LessKernelSize Training and Validation Losses per Epoch
Training Loss
Validation Loss
Loss
Epoch


LeNet5_NumFilters Training and Validation Accuracies per Epoch
Training Accuracy
Validation Accuracy
Accuracy (%)
Epoch


LeNet5_NumFilters Training and Validation Losses per Epoch
Training Loss
Validation Loss
Loss
Epoch





Link to your model weights

Link:

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

Model name	Training top-1 accuracy (%)	Validation top-1 accuracy (%)
Baseline Model	95.32%	90.98%
Model 1: Adding Dropout	89.73%	89.81%
Model 2: Adding Batch Normalization	94.8%	90.675%
Model 3: Leaky ReLU	95.34%	90.575%
Model 4: Changing Kernel Size (Large)	93.33%	90.075%
Model 0: Changing Kernel Size (Small)	95.035%	90.20%
Model 5: Increasing Number of Filters	96.64%	91.61%

Discuss your results in terms of your model

Baseline-Model1: The baseline model differs from the model 1 which has an added dropout layer aiming to reduce overfitting by randomly ignoring a subset of neurons during training. This approach can lead to a more generalized model but might slightly increase complexity due to the additional dropout operations which explains why there is a drop in accuracy in Model 1. Thus, there is a trade off between potential overfitting in the baseline and the attempt to mitigate it in Model 1 through dropout. Thus, this is at the expense of expense of immediate performance gains due to the regularization effect.

Model1-Model2: Model2 differs from Model1 because we add batch normalization layers after the convolutional layers but before activation. We aim to stabilize the learning process by normalizing the inputs of each layer, potentially leading to faster convergence and better overall performance and this can be seen by an increase in accuracy for Model2. The increase in parameters is minimal from 61,706 to 61,750 which has a negligible impact on model complexity and potentially offers significant other benefits in training efficiency and model stability.

Model2-Model3: Model3 varies from Model2 by changing the ReLU to a Leaky ReLU for all applicable layers. We aim to address the issue of dying ReLUs by allowing a small gradient when the unit is inactive and could potentially improve model performance on tasks where negative input values are significant. This can be seen by the increase in accuracy. The total parameters remain the same as Model 2 and this means that the primary difference lies in the non-linear processing of the inputs rather than structural complexity.

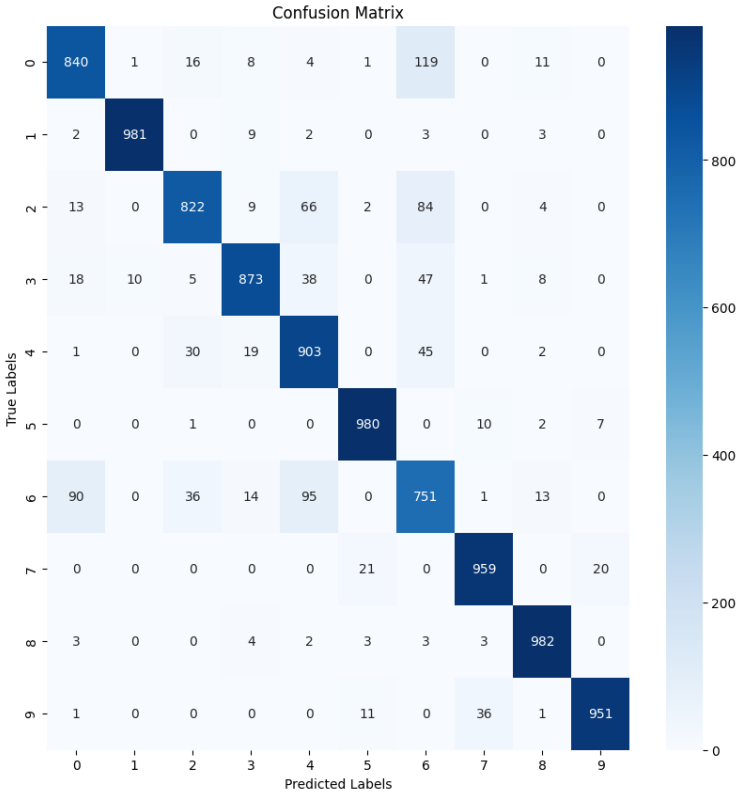
Model3-Model4: Model4 alters the kernel size in the convolutional layers, moving to a larger size, which affects the output shape and the receptive field of the network. This change can enhance the model's ability to capture larger patterns in the input data but may also increase the risk of

overfitting due to the model's increased sensitivity to broader input features. We can see that the accuracy dropped slightly in comparison to Model3.

Model4-Model0: Model0 alters the kernel size but decreases it in comparison to Model4 thus, leading to a focus on more localized features. This change could increase the depth and complexity of the model if additional layers were added to compensate for the reduced receptive field per layer but we can see that there is an increase in accuracy.

Model0-Model5: Model5 doubles the number of filters in the convolutional layers, significantly increasing the model's capacity to learn diverse features from the input data. This adjustment increases the total number of parameters to 117,078, nearly doubling the complexity of the baseline model. This increase in capacity can lead to better performance by capturing a more extensive array of patterns and this can be confirmed by the increase in the accuracy.

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



For both models we noticed that the training accuracy faired similar where 96.95% for model 1 (train) and 97.01% model 2(train+val). This indicates effective learning, with the model gradually refining its ability to generalize from the training data. However, for the test performance, there was a gap in comparison to the training performance. However, the models faired similarly to each other as well on the test where 90.27 % for model 1 (train) and 90.94 % model 2 (train+val). This is after training on the training set and then on both training and validation sets combined.

The most common explanation for the discrepancy between training and test performance is overfitting. Overfitting occurs when a model learns the training data too well, capturing noise and details that do not generalize well to unseen data. This is indicated by the model performing exceptionally well on the training data but less so on the test data. Secondly, the differences in performance could also stem from the complexity and diversity of the test set. If the test data contains examples that are significantly different or more challenging than those in the training set, the model may struggle to maintain the same level of accuracy.

Training on a combination of training and validation sets typically aims to increase the amount of data the model can learn from, potentially improving its ability to generalize. However, this also means the model is evaluated on a test set without having a separate validation set to tune hyperparameters without overfitting. The results indicate a slight improvement in test accuracy (from 90.27% to 90.94%) when the model is trained on a combined dataset. This suggests that the additional data may help the model generalize slightly better, though the effect is not large.

Choice task: 6

The dataset was obtained from Kaggle and included images of various fashion items. We transformed the images to 28x28 grayscale and normalized them for the neural network using a standard mean and standard deviation of 0.5. The dataset was split into an 80-20 ratio for training and testing purposes. The baseline model implemented was the LeNet-5 convolutional neural network, comprising two convolutional layers followed by average pooling layers, and three fully connected layers. The output layer has 10 neurons corresponding to the number of classes. We trained the model over 15 with a learning rate of 0.001. The Cross-Entropy Loss function was employed as it suits multi-class classification tasks and weights were initialized to ensure proper scaling.

The first step in our data filtering process involved the creation of a mapping, kaggle_to_fmnist_mapping, which sorted article types into a simplified classification scheme/dictionary mirroring that of the Fashion-MNIST dataset. For instance, different types of T-shirts, jeans, and jackets were grouped under single labels to match the Fashion-MNIST class structure. After creating the mapping, we proceeded to filter our dataset. This was done by selecting only those entries in the styles.csv DataFrame whose articleType matched one of the keys in our mapping. This step ensured that our dataset was composed solely of relevant article types that corresponded to our target classification labels. Following the filtering of article types, we split the data into training and testing sets, adhering to an 80-20 split ratio.

The final stage of our data preparation involved the verification of image file existence for each filtered entry. For each record, we checked if the corresponding image file was present in the designated directory. If the file existed, we added a tuple comprising the image path and its corresponding label (as per our mapping) to our dataset.

Our baseline model with a final training accuracy of 95.38% and an average loss of 0.1307. Upon evaluating the test set, the model achieved an accuracy of 91.76% with an average loss of 0.2479. The LeNet-5 baseline model demonstrated competent performance on the Kaggle fashion dataset. The high accuracy on the test set indicates that the model generalizes well to new data. The training and test loss metrics show that the model was not overfitting and had learned a general representation of the classes.