Alaa Adel

241106

Project 2

The Fashion-MNIST dataset is a collection of grayscale images commonly used for benchmarking machine learning algorithms, particularly in the field of computer vision. It serves as an alternative to the original MNIST dataset, offering more challenging classification tasks.

1. Describe the data

1. **Format**: The dataset is split into two sets: a training set and a test set.
2. **Size**:
    - Training set: 60,000 images
    - Test set: 10,000 images
3. **Image Size**: Each image in the dataset is a grayscale image of size 28x28 pixels.
4. **Labels**: Each image in the dataset is assigned a label from one of the following 10 categories:
    - 0: T-shirt/top
    - 1: Trouser
    - 2: Pullover
    - 3: Dress
    - 4: Coat
    - 5: Sandal
    - 6: Shirt
    - 7: Sneaker
    - 8: Bag
    - 9: Ankle boot

4. Visualize the data using proper visualization methods.

5. Draw some of the images

We display sample images from each class to get an idea of the variety of images in the dataset.

Summary Statistics:
Number of classes: 10
Number of training samples: 60000
Number of test samples: 10000
Image shape: (28, 28)

Carry out any required preprocessing operations on the data

2. Clean the data

3. Check the data for missing values or duplicates and carry out proper correction methods

Iterate through each image in both the training and test sets and checks if there are any NaN values using np.isnan(). If any NaN values are found, it prints a message indicating that there are missing values; otherwise, it prints a message indicating that there are no missing values.
There are no missing values in the training set.
There are no missing values in the test set.

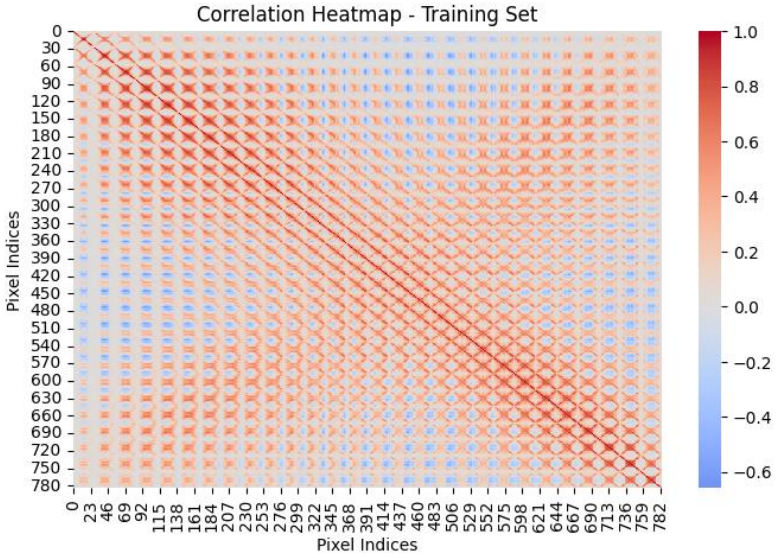
We use np.unique() along with return_counts=True to find unique rows and their counts in the flattened arrays. We extract duplicates from the unique arrays based on the counts greater than 1. Finally, we print whether duplicates were found in the training and test sets.

There are no missing values in the training set.
There are no missing values in the test set.

6. Carry out required correlation analysis

Heatmap:

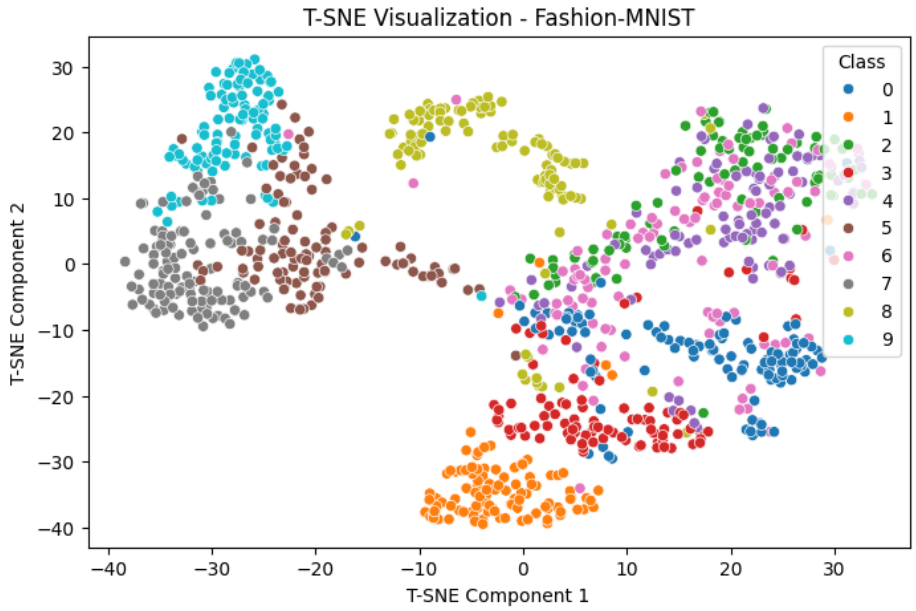
We compute the correlation matrix for the flattened training set (x_train_flat.T transposes the array so that each row represents a feature). We use seaborn.heatmap() to plot the heatmap of the correlation matrix. The cmap parameter sets the color map to use, center=0 centers the color map at 0, and annot=False disables the annotation of values in the heatmap. We set the title, x-axis label, and y-axis label for better visualization. This heatmap visualizes the correlation between pixel values in the training set images. A high positive correlation (closer to 1) between two pixels indicates that they tend to increase or decrease together, while a high negative correlation (closer to -1) indicates that they tend to vary inversely. A correlation close to 0 indicates no linear relationship between the pixels.



T-SNE:

The graph generated by T-SNE visualization depicts the distribution of the Fashion-MNIST dataset in a two-dimensional space, where each point represents an image sample from the dataset. Here's a description of the components of the graph:

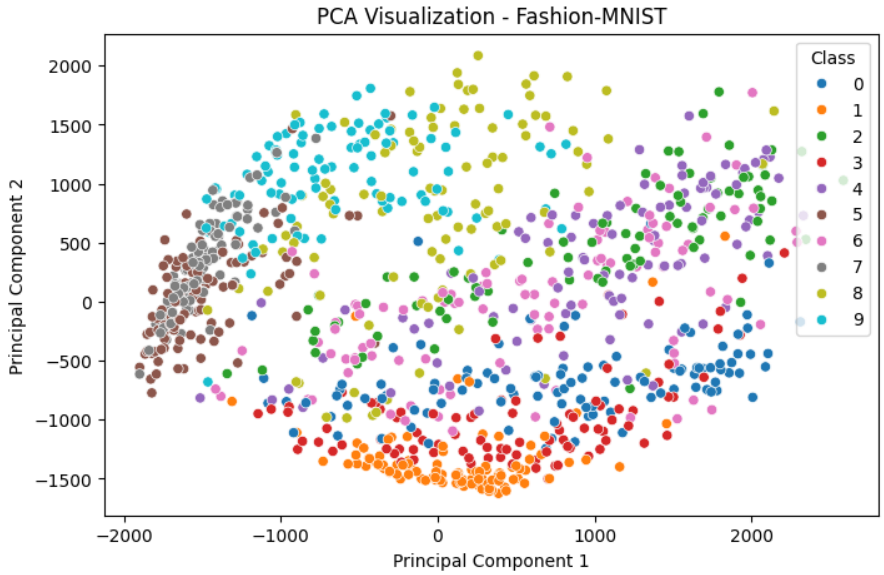
1. **Scatterplot**: The main visualization consists of a scatterplot, where each point represents an image sample. T-SNE has reduced the high-dimensional feature space of the dataset (which in this case consists of pixel values) to two dimensions while preserving the local structure of the data.
2. **Color Coding**: Points in the scatterplot are color-coded based on their class labels. Each class corresponds to a specific type of clothing item in the Fashion-MNIST dataset. Different colors represent different classes, allowing you to visually distinguish between them.
3. **Axes**: The two axes of the scatterplot represent the two components obtained by T-SNE dimensionality reduction. These components do not have any inherent meaning but capture the underlying structure of the data in a lower-dimensional space.
4. **Legend**: The legend on the plot provides information about the classes represented by each color. It helps in interpreting the colors assigned to different types of clothing items.



PCA:

The graph generated by PCA visualization depicts the distribution of the Fashion-MNIST dataset in a two-dimensional space, where each point represents an image sample from the dataset. Here's a description of the components of the graph:

1. **Scatterplot**: The main visualization consists of a scatterplot, where each point represents an image sample. PCA has reduced the high-dimensional feature space of the dataset (which in this case consists of pixel values) to two dimensions by identifying the principal components that capture the most variance in the data.
2. **Color Coding**: Points in the scatterplot are color-coded based on their class labels. Each class corresponds to a specific type of clothing item in the Fashion-MNIST dataset. Different colors represent different classes, allowing you to visually distinguish between them.
3. **Axes**: The two axes of the scatterplot represent the two principal components obtained by PCA. These components are linear combinations of the original features (pixel values) and capture the directions of maximum variance in the data.
4. **Legend**: The legend on the plot provides information about the classes represented by each color. It helps in interpreting the colors assigned to different types of clothing items.



Encode the labels

converting the target labels (class labels) into one-hot encoded vectors. The element in the row corresponding to the class label of the sample is set to 1, while all other elements are set to 0.

num_classes specifies the total number of classes in the dataset.

then converts the pixel values of the images from integers to floating-point numbers. Neural networks often work better with input data in the range of 0 to 1, so converting the pixel values to floats allows scaling them, scales the pixel values of the images to the range of 0 to 1 by dividing each pixel value by 255. Since pixel values typically range from 0 to 255 (for 8-bit grayscale images), dividing by 255 normalizes the pixel values to the range of 0 to 1. This normalization process helps in stabilizing and speeding up the training of neural networks.

implement a LeNet-5 network to recognize the FashionMNIST digits.

Defines the LeNet-5 model architecture using the Keras Sequential API.

Compiles the model with the Adam optimizer and categorical cross-entropy loss.

Prints the summary of the model.

Trains the model on the training data for 10 epochs with a batch size of 128 and 10% validation split.

Evaluates the trained model on the test data and prints the test accuracy.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 24, 24, 6)	156
max_pooling2d (MaxPooling2D)	(None, 12, 12, 6)	0
conv2d_1 (Conv2D)	(None, 8, 8, 16)	2416

max_pooling2d_1 (MaxPoolin g2D)	(None, 4, 4, 16)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 120)	30840
dense_1 (Dense)	(None, 84)	10164
dense_2 (Dense)	(None, 10)	850

===
Total params: 44426 (173.54 KB)
Trainable params: 44426 (173.54 KB)
Non-trainable params: 0 (0.00 Byte)
_____
Epoch 1/10
422/422 [==============================] - 20s 43ms/step - loss: 0.7087 - accuracy: 0.7454 - val_loss: 0.5089 - val_accuracy: 0.8155
Epoch 2/10
422/422 [==============================] - 18s 43ms/step - loss: 0.4604 - accuracy: 0.8334 - val_loss: 0.4338 - val_accuracy: 0.8405
Epoch 3/10
422/422 [==============================] - 18s 43ms/step - loss: 0.3983 - accuracy: 0.8553 - val_loss: 0.3916 - val_accuracy: 0.8538
Epoch 4/10
422/422 [==============================] - 18s 42ms/step - loss: 0.3638 - accuracy: 0.8669 - val_loss: 0.3727 - val_accuracy: 0.8648
Epoch 5/10
422/422 [==============================] - 19s 45ms/step - loss: 0.3371 - accuracy: 0.8769 - val_loss: 0.3424 - val_accuracy: 0.8750
Epoch 6/10
422/422 [==============================] - 19s 45ms/step - loss: 0.3154 - accuracy: 0.8839 - val_loss: 0.3365 - val_accuracy: 0.8758
Epoch 7/10
422/422 [==============================] - 19s 45ms/step - loss: 0.3008 - accuracy: 0.8889 - val_loss: 0.3267 - val_accuracy: 0.8790
Epoch 8/10
422/422 [==============================] - 18s 43ms/step - loss: 0.2893 - accuracy: 0.8934 - val_loss: 0.3190 - val_accuracy: 0.8825
Epoch 9/10
422/422 [==============================] - 24s 57ms/step - loss: 0.2772 - accuracy: 0.8977 - val_loss: 0.3122 - val_accuracy: 0.8865
Epoch 10/10
422/422 [==============================] - 18s 44ms/step - loss: 0.2682 - accuracy: 0.9006 - val_loss: 0.3192 - val_accuracy: 0.8827
313/313 [==============================] - 2s 5ms/step - loss: 0.3309 - accuracy: 0.8817

Test accuracy: 0.8816999793052673

Modify hyperparameters to get to the best performance you can achieve

We modifying the model by adding pool_size=(2, 2), strides=(2, 2) and adding dropout by 40 percent

Defining the Model: The LeNet-5 model architecture is defined using the Sequential API. It consists of the following layers:

- Two convolutional layers (Conv2D) with 6 and 16 filters, respectively, each followed by a ReLU activation function.
- Two max-pooling layers (MaxPooling2D) to reduce spatial dimensions.
- A Flatten layer to flatten the output from the previous layers into a 1D vector.
- Two fully connected (Dense) layers with 120 and 84 neurons, respectively, followed by ReLU activation functions.
- Two dropout layers (Dropout) with a dropout rate of 0.4 to prevent overfitting.
- Finally, an output layer with 10 neurons (equal to the number of classes) and a softmax activation function.

Compiling the Model: The model is compiled using the Adam optimizer, categorical cross-entropy loss function, and accuracy metric.

Model Summary: The summary of the model, including the number of parameters in each layer, is printed.

Training the Model: The model is trained using the training data (x_train and y_train) for 10 epochs with a batch size of 128. 10% of the training data is used as validation data for monitoring model performance during training.

Evaluating the Model: The trained model is evaluated on the test data (x_test and y_test) to calculate the test loss and accuracy.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 24, 24, 6)	156
max_pooling2d (MaxPooling2D)	(None, 12, 12, 6)	0
conv2d_1 (Conv2D)	(None, 8, 8, 16)	2416
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 16)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 120)	30840
dropout (Dropout)	(None, 120)	0
dense_1 (Dense)	(None, 84)	10164
dropout_1 (Dropout)	(None, 84)	0
dense_2 (Dense)	(None, 10)	850

Total params: 44426 (173.54 KB)
Trainable params: 44426 (173.54 KB)
Non-trainable params: 0 (0.00 Byte)

_____

Epoch 1/10
422/422 [==============================] - 21s 45ms/step - loss: 0.8814 - accuracy: 0.6741 - val_loss: 0.5124 - val_accuracy: 0.7983
Epoch 2/10
422/422 [==============================] - 20s 48ms/step - loss: 0.5552 - accuracy: 0.7946 - val_loss: 0.4342 - val_accuracy: 0.8358
Epoch 3/10
422/422 [==============================] - 19s 44ms/step - loss: 0.4762 - accuracy: 0.8281 - val_loss: 0.3858 - val_accuracy: 0.8528
Epoch 4/10
422/422 [==============================] - 34s 79ms/step - loss: 0.4303 - accuracy: 0.8456 - val_loss: 0.3749 - val_accuracy: 0.8575
Epoch 5/10
422/422 [==============================] - 18s 42ms/step - loss: 0.4014 - accuracy: 0.8569 - val_loss: 0.3511 - val_accuracy: 0.8660
Epoch 6/10
422/422 [==============================] - 19s 44ms/step - loss: 0.3802 - accuracy: 0.8637 - val_loss: 0.3260 - val_accuracy: 0.8773
Epoch 7/10
422/422 [==============================] - 18s 43ms/step - loss: 0.3623 - accuracy: 0.8718 - val_loss: 0.3243 - val_accuracy: 0.8753
Epoch 8/10
422/422 [==============================] - 26s 63ms/step - loss: 0.3484 - accuracy: 0.8757 - val_loss: 0.3076 - val_accuracy: 0.8842
Epoch 9/10
422/422 [==============================] - 21s 50ms/step - loss: 0.3370 - accuracy: 0.8798 - val_loss: 0.3247 - val_accuracy: 0.8752
Epoch 10/10
422/422 [==============================] - 22s 52ms/step - loss: 0.3275 - accuracy: 0.8831 - val_loss: 0.3080 - val_accuracy: 0.8868
313/313 [==============================] - 3s 8ms/step - loss: 0.3248 - accuracy: 0.8831

Test accuracy: 0.8830999732017517

Evaluate the model using 5-fold cross-validation.

Defining the Model: The LeNet-5 model architecture is defined within the create_model function using the Sequential API. It consists of:

- Two convolutional layers (Conv2D) with 6 and 16 filters, respectively, followed by ReLU activation functions.
- Two max-pooling layers (MaxPooling2D) to reduce spatial dimensions.
- Flattening the output to a 1D vector.
- Two fully connected (Dense) layers with 120 and 84 neurons, respectively, followed by ReLU activation functions.
- Two dropout layers (Dropout) with a dropout rate of 0.4 to prevent overfitting.
- Finally, an output layer with 10 neurons (equal to the number of classes) and a softmax activation function.

Compiling the Model: The model is compiled inside the create_model function using the Adam optimizer, categorical cross-entropy loss function, and accuracy metric.

Performing 5-fold Cross-Validation: The Fashion-MNIST dataset is split into 5 folds using KFold. For each fold:

- The model is created and trained on the training data for 10 epochs with a batch size of 128.
- The model's performance is evaluated on the test data, and the test accuracy is recorded.

Calculating Mean and Standard Deviation: The mean and standard deviation of the test accuracies obtained from cross-validation are calculated and printed.

Mean Test accuracy: 0.8771399974822998
Standard Deviation of Test accuracy: 0.003336225180424349

Try to use other two CNN models (using transfer learning) and compare the results with the full trained LeNet-5.

model 1: VGG16 model

1. **Data Preprocessing**:
    - The grayscale images in x_train and x_test are resized to meet the minimum input size requirement of VGG16 (32x32 pixels) using the resize function from skimage.transform.
    - The resized images are converted from grayscale to RGB format using the gray2rgb function from skimage.color.
2. **Model Architecture**:
    - The pre-trained VGG16 model is loaded with weights pre-trained on the ImageNet dataset, excluding the top classification layers (specified by include_top=False).
    - The layers of the pre-trained VGG16 model are frozen to prevent them from being updated during training.
    - Custom classification layers are added on top of the base VGG16 model. This includes a Flatten layer to convert the 3D feature maps to 1D, followed by a fully connected Dense layer with ReLU activation, a Dropout layer for regularization, and a final Dense layer with softmax activation for classification into 10 classes (assuming it's a multi-class classification task).
3. **Model Compilation**:
    - The model is compiled with the Adam optimizer, categorical cross-entropy loss function, and accuracy metric.
4. **Model Training**:
    - The model is trained on the preprocessed training data (x_train_rgb, y_train) for 10 epochs with a batch size of 128. Validation data is also specified for monitoring the model's performance during training.
5. **Model Evaluation**:
    - After training, the model is evaluated on the preprocessed test data (x_test_rgb, y_test) to assess its performance. The test accuracy is printed to the console.

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0
block5_conv1 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv2 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv3 (Conv2D)	(None, 2, 2, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
flatten_5 (Flatten)	(None, 512)	0
dense_15 (Dense)	(None, 256)	131328
dropout_10 (Dropout)	(None, 256)	0
dense_16 (Dense)	(None, 10)	2570

Total params: 14848586 (56.64 MB)
Trainable params: 133898 (523.04 KB)
Non-trainable params: 14714688 (56.13 MB)

Epoch 1/10
422/422 [==============================] - 626s 1s/step - loss: 0.7056 - accuracy: 0.7542 - val_loss: 0.4950 - val_accuracy: 0.8232

```
Epoch 2/10
422/422 [==============================] - 616s 1s/step - loss: 0.4905 - accuracy: 0.8233 - val_loss: 0.4366 -
val_accuracy: 0.8420
Epoch 3/10
422/422 [==============================] - 619s 1s/step - loss: 0.4445 - accuracy: 0.8384 - val_loss: 0.4159 -
val_accuracy: 0.8473
Epoch 4/10
422/422 [==============================] - 609s 1s/step - loss: 0.4204 - accuracy: 0.8473 - val_loss: 0.3978 -
val_accuracy: 0.8502
Epoch 5/10
422/422 [==============================] - 621s 1s/step - loss: 0.4047 - accuracy: 0.8530 - val_loss: 0.3870 -
val_accuracy: 0.8570
Epoch 6/10
422/422 [==============================] - 619s 1s/step - loss: 0.3924 - accuracy: 0.8559 - val_loss: 0.3818 -
val_accuracy: 0.8592
Epoch 7/10
422/422 [==============================] - 618s 1s/step - loss: 0.3811 - accuracy: 0.8608 - val_loss: 0.3732 -
val_accuracy: 0.8625
Epoch 8/10
422/422 [==============================] - 619s 1s/step - loss: 0.3725 - accuracy: 0.8631 - val_loss: 0.3658 -
val_accuracy: 0.8652
Epoch 9/10
422/422 [==============================] - 594s 1s/step - loss: 0.3632 - accuracy: 0.8662 - val_loss: 0.3642 -
val_accuracy: 0.8683
Epoch 10/10
422/422 [==============================] - 620s 1s/step - loss: 0.3568 - accuracy: 0.8675 - val_loss: 0.3629 -
val_accuracy: 0.8647
313/313 [==============================] - 105s 335ms/step - loss: 0.3678 - accuracy: 0.8681
```

Test accuracy: 0.8680999875068665

model 2: ResNet50 model

train a ResNet50 model on the Fashion-MNIST dataset.

1. **Data Preprocessing**:
    - The images in the training and test sets (`x_train` and `x_test`) are resized to meet the minimum input size requirement of ResNet50 (32x32 pixels) using the `resize` function from scikit-image (`skimage.transform.resize`).
    - As ResNet50 expects RGB images, the grayscale images are converted to RGB format by stacking the resized images along the third axis using NumPy's `np.stack`.
2. **Loading Pre-trained ResNet50 Model**:
    - The pre-trained ResNet50 model is loaded from the Keras Applications module (`tensorflow.keras.applications.ResNet50`). The `include_top` parameter is set to `False` to exclude the fully connected layers at the top of the network.
    - The input shape is specified as `(32, 32, 3)` to match the resized RGB images.
3. **Freezing Layers**:
    - All layers in the base model (ResNet50) are frozen to prevent their weights from being updated during training.
4. **Custom Classification Layers**:
    - Custom dense layers are added on top of the base ResNet50 model to perform classification. These layers include a `Flatten` layer to convert the 3D output of ResNet50

into 1D, followed by two Dense layers with ReLU activation and a dropout layer for regularization.
    - The output layer consists of 10 neurons with softmax activation, representing the probabilities for each class.
5. **Model Compilation**:
    - The model is compiled with the Adam optimizer, categorical cross-entropy loss function, and accuracy metric.
6. **Model Training**:
    - The model is trained using the fit method on the resized training data (x_train_resized and y_train) for 10 epochs with a batch size of 128. A validation split of 0.1 is used to monitor validation accuracy during training.
7. **Model Evaluation**:
    - After training, the model is evaluated on the resized test data (x_test_resized and y_test) to compute the test loss and accuracy.
8. **Summary**:
    - The summary of the model architecture is printed, showing the layers, output shapes, and the number of parameters.

Total params: 24114826 (91.99 MB)
Trainable params: 527114 (2.01 MB)
Non-trainable params: 23587712 (89.98 MB)

_____

Epoch 1/10
422/422 [==============================] - 203s 473ms/step - loss: 1.3131 - accuracy: 0.5208 - val_loss: 0.8669 - val_accuracy: 0.6952
Epoch 2/10
422/422 [==============================] - 224s 531ms/step - loss: 0.9327 - accuracy: 0.6561 - val_loss: 0.7571 - val_accuracy: 0.7278
Epoch 3/10
422/422 [==============================] - 197s 467ms/step - loss: 0.8555 - accuracy: 0.6834 - val_loss: 0.7038 - val_accuracy: 0.7447
Epoch 4/10
422/422 [==============================] - 197s 466ms/step - loss: 0.8237 - accuracy: 0.6924 - val_loss: 0.7133 - val_accuracy: 0.7348
Epoch 5/10
422/422 [==============================] - 197s 467ms/step - loss: 0.8008 - accuracy: 0.7019 - val_loss: 0.6708 - val_accuracy: 0.7475
Epoch 6/10
422/422 [==============================] - 196s 464ms/step - loss: 0.7834 - accuracy: 0.7076 - val_loss: 0.6679 - val_accuracy: 0.7545
Epoch 7/10
422/422 [==============================] - 197s 468ms/step - loss: 0.7795 - accuracy: 0.7074 - val_loss: 0.6712 - val_accuracy: 0.7480
Epoch 8/10
422/422 [==============================] - 197s 467ms/step - loss: 0.7838 - accuracy: 0.7060 - val_loss: 0.6668 - val_accuracy: 0.7617
Epoch 9/10
422/422 [==============================] - 195s 462ms/step - loss: 0.7719 - accuracy: 0.7109 - val_loss: 0.6573 - val_accuracy: 0.7515

```
Epoch 10/10
422/422 [==============================] - 196s 465ms/step - loss: 0.7638 - accuracy: 0.7140 -
val_loss: 0.6526 - val_accuracy: 0.7557
313/313 [==============================] - 40s 127ms/step - loss: 0.6831 - accuracy: 0.7436
```

Test accuracy: 0.7436000108718872

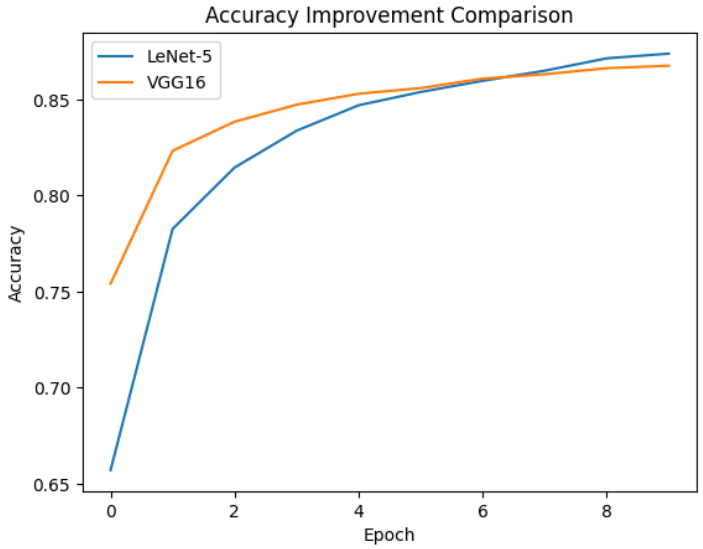
Compare the results with the full trained LeNet-5.

Plotting Accuracy Improvement: Two separate plots are created for each model's accuracy improvement over epochs.

- For the LeNet-5 model, the accuracy improvement data is extracted from the `history` object, which likely contains the training history of the LeNet-5 model obtained after fitting the data.
- For the VGG16 model, the accuracy improvement data is extracted from the `historyvgg16` object, which likely contains the training history of the VGG16 model obtained after fitting the data.
- Both plots show the accuracy improvement (y-axis) over epochs (x-axis) for their respective models.

Labels and Title: The x-axis is labeled as 'Epoch', the y-axis is labeled as 'Accuracy', and the title of the plot is set as 'Accuracy Improvement Comparison'.

Legend: A legend is added to the plot to differentiate between the two models. It indicates which line represents LeNet-5 and which represents VGG16.



Based on the comparison of test accuracies and the graph:

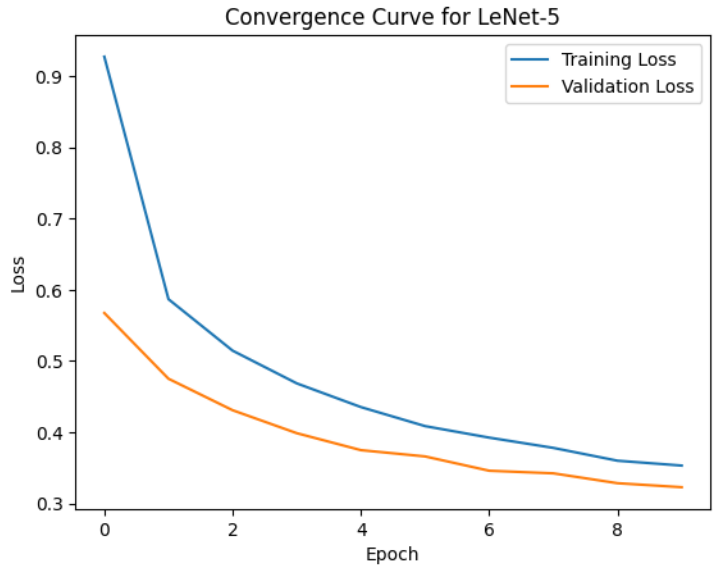
1. **Test Accuracy Comparison**:
    - VGG16 Model Test Accuracy: 0.8681
    - LeNet-5 Model using 5-fold cross-validation Mean Test Accuracy: 0.8771
    - Standard Deviation of Test Accuracy for LeNet-5: 0.0033
2. **Graph Description**:
    - The graph displays the accuracy improvement over epochs for both the LeNet-5 and VGG16 models.

- The x-axis represents the number of epochs, while the y-axis represents the accuracy achieved by the models during training.
- Each line in the graph corresponds to a different model: blue for LeNet-5 and orange for VGG16.
- The LeNet-5 line shows the accuracy improvement of the LeNet-5 model over epochs, while the VGG16 line represents the accuracy improvement of the VGG16 model.
- The plot allows for a visual comparison of how the accuracy of both models evolves over the training epochs.

3. **Comparison Interpretation**:
    - The test accuracy comparison shows that the LeNet-5 model using 5-fold cross-validation achieves a slightly higher mean test accuracy compared to the VGG16 model
    - From the graph, it seems that both models experience improvements in accuracy as the number of epochs increases, but the rate and extent of improvement may vary.

plot the convergence curve for LeNet-5

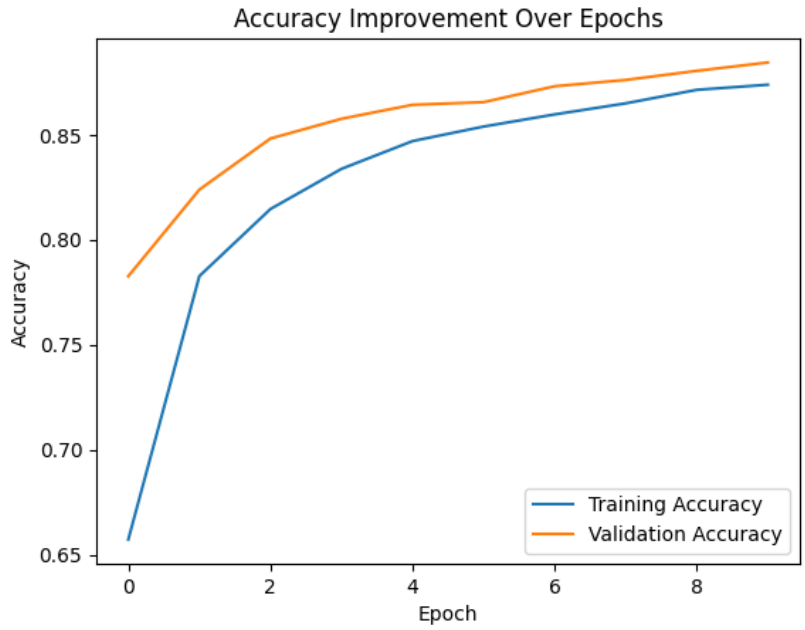
1. **Loss Curves**:
    - The training loss and validation loss values for each epoch of the LeNet-5 model training are retrieved from the `history` object, which stores the training history.
    - These loss values are then plotted against the number of epochs using `plt.plot`.
    - Two curves are plotted: one for the training loss (`'Training Loss'`) and one for the validation loss (`'Validation Loss'`).
2. **Axis Labels and Title**:
    - The x-axis represents the number of epochs, while the y-axis represents the loss value.
    - The plot is appropriately labeled and titled to indicate that it shows the convergence curve for the LeNet-5 model.
3. **Legend**:
    - A legend is added to the plot to differentiate between the training loss and validation loss curves.



This convergence curve helps visualize how the training and validation losses change over the course of training epochs for the LeNet-5 model. we can see the change is normal and to avoid overfitting we can utilize the capacity area at 8 epochs.

Plot of accuracy improvement using the previously mentioned techniques.

1. **Accuracy Curves**:
    - The training accuracy and validation accuracy values for each epoch of the LeNet-5 model training are retrieved from the history object, which stores the training history.
    - These accuracy values are then plotted against the number of epochs using plt.plot.
    - Two curves are plotted: one for the training accuracy ('Training Accuracy') and one for the validation accuracy ('Validation Accuracy').
2. **Axis Labels and Title**:
    - The x-axis represents the number of epochs, while the y-axis represents the accuracy value.
    - The plot is appropriately labeled and titled to indicate that it shows the accuracy improvement over epochs for the LeNet-5 model.
3. **Legend**:
    - A legend is added to the plot to differentiate between the training accuracy and validation accuracy curves.



Comment on why you think LeNet-5 further improves the accuracy if any at all. And if it doesn't, why not?

why LeNet-5 might not further improve accuracy:

1. **Complexity of the Dataset**: Fashion-MNIST is a complex dataset. The dataset contains images of clothing items belonging to 10 different classes, which may require deeper and more complex architectures to capture intricate patterns and features.
2. **Depth of the Network**: LeNet-5 is a relatively shallow network compared to more modern architectures like ResNet, Inception, or DenseNet. Deeper networks often have a greater capacity to learn complex patterns in the data, which may lead to better performance.
3. **Limited Capacity**: LeNet-5 has a limited number of parameters and capacity compared to more modern architectures. With a smaller number of parameters, the model may struggle to capture the variability present in the Fashion-MNIST dataset.
4. **Pooling Operations**: LeNet-5 uses max-pooling layers to downsample the feature maps, which may result in loss of spatial information. In more complex datasets, such as Fashion-MNIST, preserving spatial information may be crucial for achieving higher accuracy.

For the Fashion MNIST dataset the accuracies improves because:

1. **Model Architecture Suitability**: LeNet-5, with its relatively simple architecture comprising convolutional and pooling layers, well-suited for capturing the essential features in the Fashion MNIST dataset. Since the dataset consists of grayscale images of clothing items, LeNet-5's architecture, might generalize well to recognize different types of clothing items.
2. **Data Characteristics**: The Fashion MNIST dataset shares similarities with handwritten digit datasets like MNIST, which LeNet-5 was originally designed for. Both datasets contain grayscale images with similar resolution and structure. As a result, LeNet-5's architecture may naturally align with the characteristics of the Fashion MNIST dataset, leading to improved accuracy.
3. **Regularization Techniques**: The addition of dropout layers in LeNet-5 helps prevent overfitting by randomly dropping units during training. Given that the Fashion MNIST dataset contains relatively simple images, regularization techniques like dropout can prevent the model from memorizing noise in the training data, thus improving its generalization performance.
4. **Hyperparameter Tuning**: LeNet-5 might benefit from hyperparameter tuning specifically tailored to the Fashion MNIST dataset. For example, optimizing the learning rate, batch size, and other hyperparameters for LeNet-5 on Fashion MNIST could lead to faster convergence and better accuracy compared to other models.
5. **Data Preprocessing**: The preprocessing steps applied to the Fashion MNIST dataset, such as normalization and augmentation, might align well with the assumptions made by LeNet-5. For instance, normalizing pixel values to a range between 0 and 1 and augmenting the dataset with transformations like rotation and flipping could enhance LeNet-5's performance on Fashion MNIST.