Image Classification on Fashion MNIST Dataset

1. Introduction

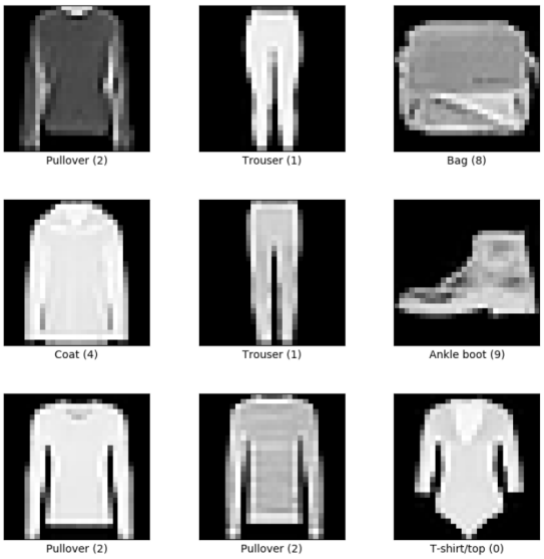
Image classification is the process of assigning a label or category to an image based on its visual content. The input is an image, and the output is a class label that represents the content or category of the image. The primary objective of image classification is to enable computers to recognize and interpret images in a manner similar to humans. By accurately classifying images, machines can perform tasks such as object detection, scene understanding and content-based image retrieval. The objective of this project is to classify the images present in Fashion MNIST dataset into their respective categories. Given an input image, the model should predict the correct clothing category. In this report, we delve into the methodologies employed, results obtained, and insights gained during the classification process.

2. Data Analysis

2.1 Dataset

The Fashion MNIST dataset contains 70,000 images. Each image is grayscale with dimensions of 28x28 pixels, resulting in 784 pixels per image. The dataset is divided into two main subsets: a training set and a test set. The training set consists of 60,000 images, while the test set contains 10,000 images.
Fashion MNIST is composed of 10 different classes, each representing a specific type of fashion item or clothing category. The classes/categories in the Fashion MNIST dataset are as follows: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot. Each class contains roughly an equal number of images, resulting in a balanced dataset suitable for classification tasks.
It is included as a built-in dataset in popular libraries such as TensorFlow. The dataset is loaded in the form of two tuples. The first tuple comprises train_images and train_labels, while the second tuple includes test_images and test_labels. This structured arrangement facilitates the organization and access of both training and testing data sets throughout the project pipeline.

Histogram is used to visualize class distribution. It shows that the dataset is balanced.



3. Pre-Processing

3.1 Normalization

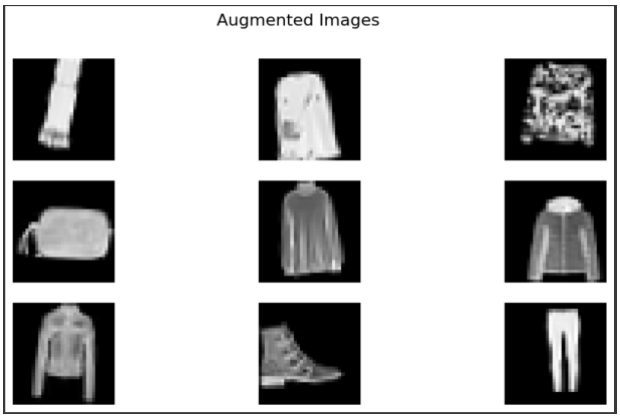
Normalization is a preprocessing technique used to scale numerical data to a range between 0 and 1. In the context of image data, pixel values typically range from 0 to 255 for grayscale images (0 representing black and 255 representing white). By dividing each pixel value by 255.0, we scale the values to the range [0, 1], making the data suitable for training machine learning models, particularly those sensitive to the scale of input features.

3.2 Data Augmentation

Data augmentation is a technique commonly used in computer vision tasks, to artificially increase the size of the training dataset by applying various transformations to the original images. This helps improve the generalization and robustness of machine learning models by exposing them to a wider range of variations in the input data.

The following are the parameters considered for Data Augmentation :

i. Rotation: Images are randomly rotated by angles within the range [-20, 20] degrees.
ii. Horizontal Shift: Images are horizontally shifted by a fraction of the total width, with a maximum shift of 10% of the total width.
iii. Vertical Shift: Images are vertically shifted by a fraction of the total height, with a maximum shift of 10% of the total height.
iv. Zoom: Random zooming into images by a factor of up to 10%.
v. Horizontal Flip: Images are horizontally flipped with a probability of 50%.
vi. Vertical Flip: Vertical flipping of images is disabled.
vii. Fill Mode: The strategy for filling in newly created pixels after rotation or shifting is set to 'nearest', where the nearest pixel value is used to fill the empty areas.



After configuring these augmentation parameters, the fit method of the ImageDataGenerator object is invoked with the training data (train_images) as input. This process ensures that the model is trained on diverse and augmented data, thus improving its generalization capabilities.

4. Model Structure

4.1 Random Forest Model

4.1.1 Introduction

A Random Forest model is a popular ensemble learning method used for both classification and regression tasks. At the core of a Random Forest are decision trees. Decision trees are a type of supervised learning algorithm used for both classification and regression tasks. They make decisions by splitting the data based on feature values. Instead of relying on a single decision tree, it creates multiple decision trees and combines their predictions to make a final prediction. This technique is called ensemble learning.

Benefits of using Random Forest model:

i. Random Forests are less prone to overfitting compared to individual decision trees, especially when dealing with high-dimensional datasets.
ii. They typically perform well on a wide range of datasets without much tuning of hyperparameters.
iii. They provide estimates of feature importance, which can be useful for feature selection and understanding the dataset.

4.1.2 Initial Model

Load Fashion MNIST Dataset: The model starts by loading the Fashion MNIST dataset, comprising grayscale images of clothing items.

Data Preprocessing:

i. Reshaping Images: It transforms the images into 1D arrays, facilitating data preparation for training and testing.

ii. Train-Validation Split: The training data is divided into a training set and a validation set using the train_test_split function from sklearn.

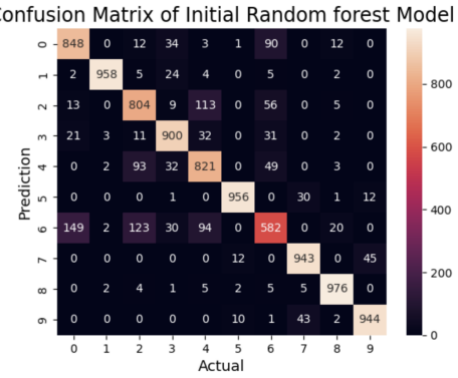
Model Training and Evaluation:

i. Random Forest Initialization: The model initializes a Random Forest classifier and proceeds to train it using the training data.
ii. Validation Set Prediction: Utilizing the trained model, predictions are made on the validation set, and the accuracy of these predictions is calculated.
iii. Test Set Evaluation: The model's performance is assessed on the test set, and the accuracy of its predictions is computed.

A classification report is generated and printed, providing precision, recall, and F1-score metrics for each class within the test set.

Visualization:

Finally, the model displays a 3x3 grid of random images extracted from the test set, accompanied by their true labels and the corresponding predicted labels. This visual presentation offers insight into the model's performance through sample predictions.



4.1.3 Hyper parameter tuning

Loading and Preprocessing Fashion MNIST Data:

i. Dataset Loading: The code loads the Fashion MNIST dataset.
ii. Reshaping Images: Following dataset loading, it reshapes the image data into 1D arrays, preparing the data for further processing.
iii. Normalization: Pixel values of the images are normalized to a range of [0, 1] using Min-Max scaling to enhance model performance.

Data Partitioning:

Train-Validation Split: Scaled training data is partitioned into training and validation sets using the train_test_split function from sklearn to facilitate model evaluation.

Random Forest Model Training and Evaluation:

i. Model Initialization: A Random Forest classifier is initialized with specified hyperparameters, including the number of estimators, max depth, and minimum samples for splitting and leaf nodes.
ii. Model Training: The initialized model is trained on the scaled training data.
iii. Validation Set Prediction: Subsequently, predictions are made on the validation set to assess model performance, and the validation accuracy is printed.
iv. Test Set Evaluation: The trained model is further evaluated on the test set, and the accuracy of its predictions on the test set is printed.

Classification Report Generation:

Report Generation: A classification report is generated and printed for the test set, providing precision, recall, and F1-score metrics for each class.

Visualizing Predictions:
Visualization: Finally, a 3x3 grid of random images from the test set is displayed, accompanied by their true labels and predicted labels, offering a visual representation of the model's performance through sample predictions.

4.1.4 Dimensionality Reduction using PCA

Loading and Preprocessing Fashion MNIST Data:

i. Dataset Loading: The code begins by loading the Fashion MNIST dataset.
ii. Reshaping Images: Subsequently, it reshapes the image data into 1D arrays, preparing the data for further processing.
iii. Normalization: Pixel values of the images are normalized to a range of [0, 1] using Min-Max scaling, enhancing model performance.

Dimensionality Reduction with PCA:

i. PCA Application: Principal Component Analysis (PCA) is applied for dimensionality reduction, preserving 95% of the variance in the data.
ii. Data Transformation: The scaled training and test data are transformed using PCA, reducing the dimensionality while retaining essential information.

Data Partitioning:
Train-Validation Split: The PCA-transformed training data is partitioned into training and validation sets using the train_test_split function from sklearn, facilitating model evaluation.
Random Forest Model Training and Evaluation:

i. Model Initialization: A Random Forest classifier is initialized with tuned hyperparameters such as the number of estimators, max depth, and minimum samples for splitting and leaf nodes.
ii. Training Process: The classifier is trained on the PCA-transformed training data.
iii. Validation Set Prediction: Predictions are made on the validation set, and the accuracy is calculated and printed to assess model performance.
iv. Test Set Evaluation: The trained model is evaluated on the test set, and the accuracy of its predictions is computed and printed.

Classification Report Generation:
Report Creation: A classification report is generated and printed for the test set, offering precision, recall, and F1-score metrics for each class.
Visualizing Predictions:
Visualization: Finally, a 3x3 grid of random images from the test set is displayed, accompanied by their true labels and predicted labels, providing a visual representation of the model's performance through sample predictions.
In summary, the code executes multiple steps, including data loading, preprocessing, dimensionality reduction with PCA, model training and evaluation with a Random Forest classifier, classification report generation, and visualization of predictions, to classify images in the Fashion MNIST dataset.

4.1.5 Random Forest Classifier trained on reduced dataset

Dataset Size Reduction:

It reduces the size of the dataset to a specified sample size of 10000, creating a smaller subset by selecting the first sample_size instances from the scaled training data and their corresponding labels.
Random Forest Model Training and Evaluation:

i. Initializes a Random Forest classifier with predefined hyperparameters and proceeds to train it on the reduced dataset.
ii. Performs predictions on the original test set using the trained model.

iii. Calculates the accuracy of the predictions made on the original test set using the model trained on the smaller dataset and prints the test accuracy.

Classification Report Generation:

Generates and prints a classification report for the predictions made with the Random Forest model trained on the sampled data, providing precision, recall, and F1-score for each class in the original test set.

Overall, the code reduces the size of the dataset, trains a Random Forest classifier on the smaller dataset, evaluates its performance, and provides a classification report for the predictions made on the original test set using the model trained on the sampled data.

4.1.6 Random Forest training and feature selection

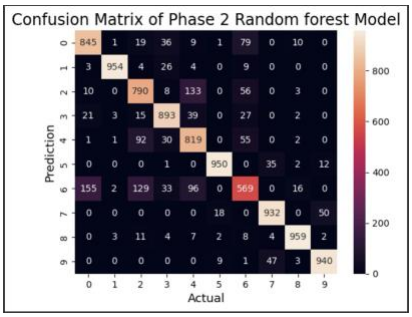
Random Forest Model Training and Feature Selection:

i. Initializes a Random Forest classifier with a specified number of estimators and a random state and fits the model to the scaled training data.
ii. Retrieves the feature importances from the trained Random Forest model and sorts them in descending order.
iii. Selects the top features based on their importance, for example, the top 200 features, from both the scaled training and test data using the indices of the selected top features.
iv. Trains a new Random Forest model using the selected features from the training data.

Model Evaluation and Reporting:

i. Makes predictions on the test set using the Random Forest model trained on the selected features.
ii. Calculates the accuracy of the predictions made on the test set using the model with selected features and prints the test accuracy.
iii. Generates and prints a classification report for the predictions made on the test set using the Random Forest model trained on the selected features, providing precision, recall, and F1-score for each class.

In summary, the code trains a Random Forest model, selects the top features based on their importance, trains a new Random Forest model using the selected features, evaluates its performance, and provides a classification report for the predictions made with the model trained on the selected features.



4.2 Support Vector Machine Classifier(SVM)

4.2.1 Introduction

Support Vector Machines (SVMs) are a type of supervised machine learning model used primarily for classification tasks, but they can also be applied to regression problems. The idea behind SVMs is to find the best boundary, or hyperplane, that maximizes the margin between different classes of data points. This

boundary is determined by the data points that are closest to the hyperplane on either side, known as support vectors.
SVMs are effective in high-dimensional spaces and are versatile in that they can handle both linear and non-linear separations. For non-linear classification, SVMs use kernel functions (like the linear, polynomial, radial basis function, and sigmoid kernels) to transform the data into a higher-dimensional space where a linear hyperplane can be used to separate the classes. They can be used for tasks like Image Classification in which predictive accuracy is crucial .
Benefits of using SVM: robustness, ability to manage overfitting by using regularization parameters.

4.2.2 Initial Model

Load Fashion MNIST Dataset: The model starts by loading the Fashion MNIST dataset, comprising grayscale images of clothing items.
Data Preprocessing:

i. Reshaping Images: It transforms the images into 1D arrays, facilitating data preparation for training and testing.
ii. Normalization: Pixel values of the images are normalized to a range of [0, 1] using Standard scaling to enhance model performance.

Model Initialization: LinearSVC(random_state=42) initializes a linear SVM classifier with a specific random state to ensure reproducibility of results.

Model Training: The model is trained using the standardized training images and their corresponding labels.

Model Evaluation: The trained SVM classifier to used predict the labels for the standardized test images. This step evaluates how well the classifier performs on new, unseen data.

Classification Report Generation:
Report Creation: A classification report is generated and printed for the test set, offering precision, recall, and F1-score metrics for each class.

Visualizing Predictions:
Visualization: Finally, a 3x3 grid of random images from the test set is displayed, accompanied by their true labels and predicted labels, providing a visual representation of the model's performance through sample predictions.

4.2.2 Applying Dimensionality Reduction using PCA

Dimensionality Reduction with PCA:

i. PCA Application: Principal Component Analysis (PCA) is applied for dimensionality reduction, preserving 95% of the variance in the data.
ii. Data Transformation: The scaled training and test data are transformed using PCA, reducing the dimensionality while retaining essential information.

Now, the initial model is trained with the data obtained after data transformation using PCA.

Model Evaluation: The trained SVM classifier to used predict the labels for the standardized test images. This step evaluates how well the classifier performs on new, unseen data.

Classification Report Generation:
Report Creation: A classification report is generated and printed for the test set, offering precision, recall, and F1-score metrics for each class.

Visualizing Predictions:
Visualization: Finally, a 3x3 grid of random images from the test set is displayed, accompanied by their true labels and predicted labels, providing a visual representation of the model's performance through sample predictions.

4.3.3 Model using Polynomial Kernel

Model Initialization: Create an SVM classifier that uses a polynomial kernel with a degree of 3. This type of kernel allows the SVM to fit a non-linear boundary between classes by mapping the input features into a higher-dimensional space using a polynomial function.

Model Training: Train the SVM on standardized training images and their corresponding labels.

Model Evaluation: The trained SVM classifier to used predict the labels for the standardized test images. This step evaluates how well the classifier performs on new, unseen data.

Classification Report Generation:
Report Creation: A classification report is generated and printed for the test set, offering precision, recall, and F1-score metrics for each class.

Visualizing Predictions:
Visualization: Finally, a 3x3 grid of random images from the test set is displayed, accompanied by their true labels and predicted labels, providing a visual representation of the model's performance through sample predictions.

4.3.4 Model using RBF Kernel

Model Training: Create an SVM classifier using an RBF kernel. The RBF kernel is a popular choice for SVMs because it can handle cases where the relationship between class labels and attributes is non-linear. The parameter C=1.0 controls the trade-off between achieving a low error on the training data and minimizing the model complexity for better generalization. A higher C value can lead to low bias but high variance (overfitting), and a lower C value can lead to higher bias but lower variance (underfitting).

Model Evaluation: The trained SVM classifier to used predict the labels for the standardized test images. This step evaluates how well the classifier performs on new, unseen data.

Classification Report Generation:
Report Creation: A classification report is generated and printed for the test set, offering precision, recall, and F1-score metrics for each class.

Visualizing Predictions:
Visualization: Finally, a 3x3 grid of random images from the test set is displayed, accompanied by their true labels and predicted labels, providing a visual representation of the model's performance through sample predictions.

4.3.4 Fine Tuning RBF Model

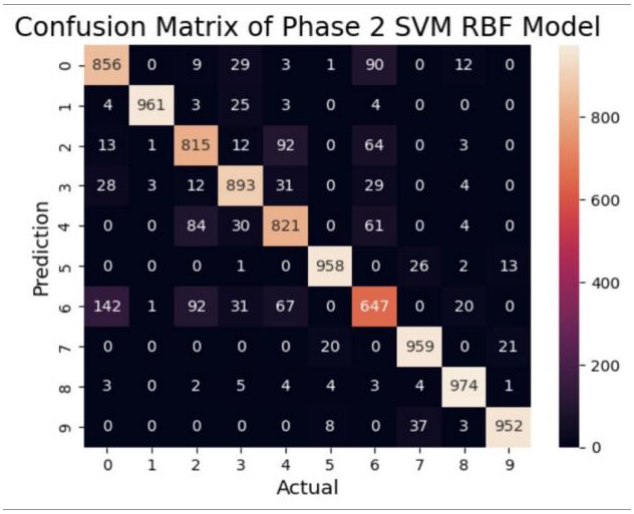
Model Training: configure an SVM with the following parameters

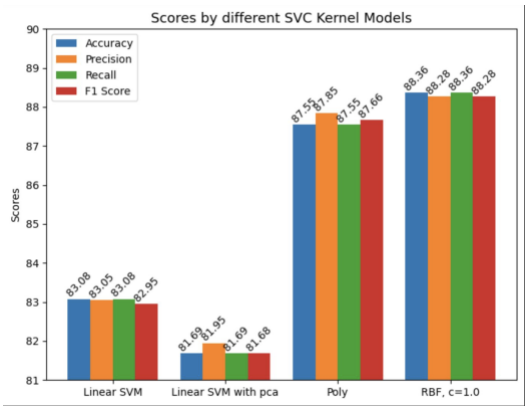
i. kernel = 'rbf': Specifies the use of the RBF kernel, suitable for non-linear classification.
ii. C = 0.1: Sets the regularization parameter to 0.1, which determines the trade-off between achieving a low error on the training data and minimizing the model complexity to avoid overfitting. A smaller C encourages a larger margin but allows more misclassifications.
iii. degree = 5: Although this parameter is typically used with polynomial kernels, it's specified here but does not affect the RBF kernel.
iv. gamma = 'auto': This sets the gamma parameter to 1/n_features, which influences the curvature of the decision boundary.
v. shrinking = True: Enables shrinking heuristic, which can speed up the training process.
vi. max_iter = -1: Allows unlimited iterations until convergence.

Model Evaluation: The trained SVM classifier to used predict the labels for the standardized test images. This step evaluates how well the classifier performs on new, unseen data.

Classification Report Generation:
Report Creation: A classification report is generated and printed for the test set, offering precision, recall, and F1-score metrics for each class.

Visualizing Predictions:
Visualization: Finally, a 3x3 grid of random images from the test set is displayed, accompanied by their true labels and predicted labels, providing a visual representation of the model's performance through sample predictions.





4.3 Convolutional Neural Network

4.3.1 Introduction

Convolutional Neural Networks (CNNs) are a class of deep neural networks designed specifically for processing and analyzing visual data, such as images. They consist of multiple layers, including convolutional layers that extract features from input images and pooling layers that reduce spatial dimensions. CNNs use learnable filters to capture patterns and structures within images, allowing them to automatically learn hierarchical representations. Through the process of training with labeled data, CNNs adjust their parameters to minimize the difference between predicted and actual outputs, enabling tasks such as image classification, object detection, and segmentation. Their effectiveness in handling complex visual tasks has made CNNs the cornerstone of many computer vision applications, from facial recognition systems to medical image analysis.

4.3.2 Initial Model

The Model consists of the following layers.

Convolutional layer - The model starts with a Conv2D layer with 32 filters/kernels of size 3x3, using the ReLU activation function. This layer is responsible for extracting low-level features from the input images, such as edges and textures.

MaxPoolingLayer2D - After each convolutional layer, a MaxPooling2D layer with a pool size of 2x2 is applied to downsample the feature maps, reducing spatial dimensions and retaining important information. This helps in making the model more robust to small variations in input images.

Additional Layers - Two more sets of Conv2D and MaxPooling2D layers are added to capture higher-level features and patterns from the input data. The second Conv2D layer has 64 filters, further increasing the depth and complexity of feature extraction.

Flattening Layer - After the final convolutional layer, a Flatten layer is used to flatten the 3D feature maps into a 1D vector, preparing them for input into the fully connected layers. This step is necessary for transitioning from the convolutional layers to the dense layers.

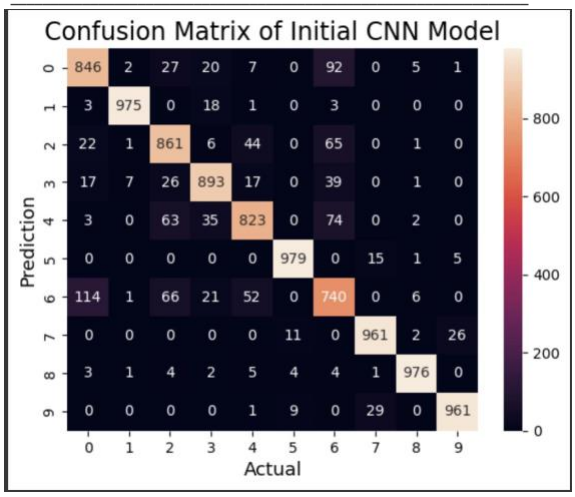
Dense Layers - Following the flattening layer, two fully connected (Dense) layers are added. The first dense layer consists of 64 neurons with the ReLU activation function, serving as intermediate feature

representations. The final dense layer has 10 neurons with the softmax activation function, producing probability scores for each of the 10 classes in the Fashion MNIST dataset.

Compiling the Model - Compile the model with the Adam optimizer to minimize the sparse categorical cross-entropy loss, while monitoring accuracy as the performance metric.

Training the Model - Train the model using train_images along with the images obtained after data augmentation with number of epochs as 25 and using 20% of training data for validation. Also use a callback for learning_rate to dynamically adjust the learning rate during training based on the epoch number.

```
Creating model
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 64)                36928

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 93322 (364.54 KB)
Trainable params: 93322 (364.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



4.3.3 Learning Rate Scheduler

Learning Rate Scheduler Function (lr_schedule):

i. The lr_schedule function takes two arguments: epoch and initial_lr.
ii. It calculates the learning rate for a given epoch using an exponential decay formula. This function decreases the learning rate exponentially as the number of epochs increases, which is a common strategy to fine-tune the learning rate during training.

Initial Learning Rate:
initial_lr = 0.001 sets the initial learning rate to 0.001. This value is commonly chosen based on empirical experimentation and domain knowledge.

Optimizer (Adam):
An Adam optimizer instance is created with the specified initial learning rate (initial_lr). Adam is an adaptive learning rate optimization algorithm commonly used for training neural networks.

4.3.4 Fine Tuning By adding Dropouts

The Model architecture is as follows. Dropout layers are added after each MaxPooling layer to prevent overfitting.

Convolutional layers - two Conv2D layers with 32 and 64 filters of size 3x3, respectively, are added sequentially. The first convolutional layer specifies the input shape as (28, 28, 1), indicating grayscale images of size 28x28 pixels.

MaxPoolingLayers - After the first convolutional layer, a MaxPooling2D layer with a pool size of 2x2 is applied to downsample the feature maps, reducing spatial dimensions and retaining important information.

Dropout layers - Dropout layers with a dropout rate of 0.5 are added after each max-pooling layer.
Dropout is a regularization technique used to prevent overfitting by randomly dropping a fraction of neurons (in this case, 50%) during training, forcing the network to learn more robust features.

Additional Layers - Another set go Convolutional, Maxpooling and dropout layers are added respectively.

Flattening layer - A flattening layer is added after the last dropout layer.

Dense layer - Two fully connected (Dense) layers are added. The first dense layer consists of 128 neurons with the ReLU activation function, serving as intermediate feature representations. The final dense layer has 10 neurons with the softmax activation function, producing probability scores for each of the 10 classes in the Fashion MNIST dataset.

Compiling the Model - Compile the model with the Adam optimizer to minimize the sparse categorical cross-entropy loss, while monitoring accuracy as the performance metric.

Training the Model - Train the model using train_images along with the images obtained after data augmentation with number of epochs as 30 and using 10% of training data for validation.

4.3.5 Fine Tuning Using ResNet Block

ResNet Block

The function resnet_block defines a basic building block for a ResNet architecture. This function is used to create residual blocks, which help in training deeper neural networks while mitigating the vanishing gradient problem.

Input Parameters

i. input: This parameter represents the input tensor or feature map to the ResNet block.
ii. filters: The number of filters or channels in the convolutional layer.
iii. kernel_size: The size of the convolutional kernel/filter. By default, it's set to 3x3.

iv. stride: The stride for the convolution operation. By default, it's set to 1.

Convolutional Layer

i. The function starts by applying a 2D convolution operation (Conv2D) with the specified number of filters, kernel size, and stride.
ii. It uses the 'same' padding to ensure that the spatial dimensions of the input and output feature maps remain the same.

Batch Normalization
Batch normalization (BatchNormalization) is applied after the convolution operation to stabilize and accelerate the training process by normalizing the activations of the previous layer.

Activation Function
ReLU activation function (Activation) is applied after batch normalization to introduce non-linearity into the network.

Output
The function returns the output tensor (x) which represents the feature map after passing through the convolutional layer, batch normalization, and activation function.

Defining Model Architecture using ResNet Block

i. inputs = Input(shape=(32, 32, 3)): Defines the input layer for the model with a shape of (32, 32, 3), indicating input images of size 32x32 pixels with 3 color channels (RGB).
ii. x = resnet_block(inputs, 32): Applies the first ResNet block to the input layer, extracting features using 32 filters.
iii. x = MaxPooling2D(pool_size=2)(x): Applies max pooling with a pool size of 2x2, reducing the spatial dimensions of the feature maps.
iv. x = resnet_block(x, 64, stride=2): Applies another ResNet block with 64 filters and a stride of 2, which downsamples the feature maps.
v. x = resnet_block(x, 128, stride=2): Applies a third ResNet block with 128 filters and a stride of 2, further downsampling the feature maps.
vi. x = GlobalAveragePooling2D()(x): Applies global average pooling to further reduce the spatial dimensions, resulting in a vector of features with global spatial information.
vii. x = Dense(128, activation='relu')(x): Adds a fully connected (dense) layer with 128 neurons and ReLU activation function. This layer processes the flattened feature vector obtained from pooling layers.
viii. outputs = Dense(10, activation='softmax')(x): Adds the final dense layer with 10 neurons (equal to the number of classes in the Fashion MNIST dataset) and softmax activation function. This layer produces the probability distribution over the output classes.

Model Creation

i. model1_resnet = Model(inputs, outputs): Creates the Keras model by specifying the input and output layers.
ii. model1_resnet.summary(): Prints a summary of the model architecture, including information about the number of parameters in each layer and the total number of trainable parameters.

Compiling the Model - Compile the model with the Adam optimizer with learning_rate 0.001 to minimize the sparse categorical cross-entropy loss, while monitoring accuracy as the performance metric.

Training the Model - Train the model using train_images with number of epochs as 20 and using 20% of training data for validation.

4.3.6 Adding regularizer to previous model

Add a kernel_regularizer l2 in the dense layer of the previous model.

Compiling the Model - Compile the model with the Adam optimizer with learning_rate 0.001 to minimize the sparse categorical cross-entropy loss, while monitoring accuracy as the performance metric.

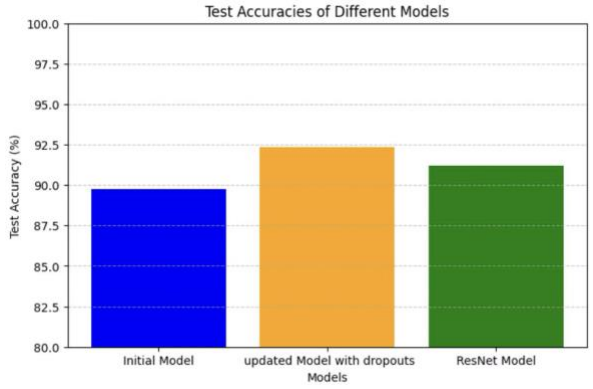
Training the Model - Train the model using train_images with number of epochs as 30 and using 20% of training data for validation. Also use a callback for learning_rate to dynamically adjust the learning rate during training based on the epoch number.

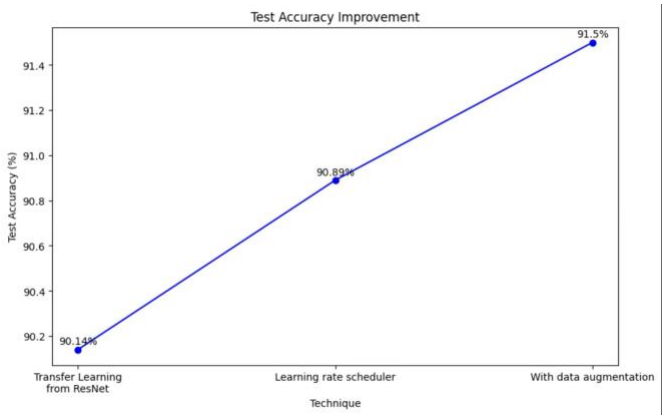
4.3.7 Training the model with Augmented Data

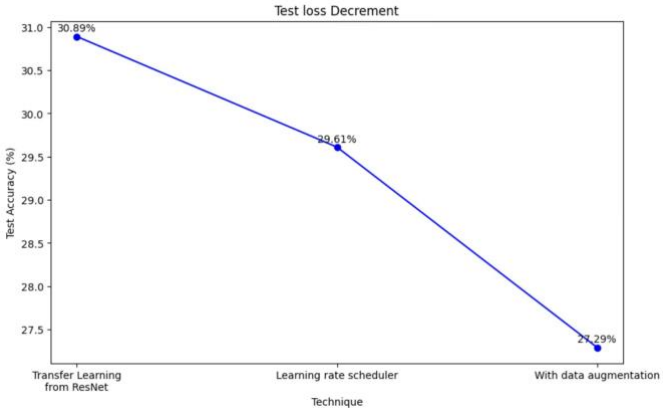
Training the Model - Train the model using train_images along with the images obtained after data augmentation with number of epochs as 30 and using 20% of training data for validation. Also use a callback for learning_rate to dynamically adjust the learning rate during training based on the epoch number.

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 26, 26, 32)	320
conv2d_10 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 64)	0
dropout (Dropout)	(None, 12, 12, 64)	0
conv2d_11 (Conv2D)	(None, 10, 10, 128)	73856
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 128)	0
dropout_1 (Dropout)	(None, 5, 5, 128)	0
flatten (Flatten)	(None, 3200)	0
dense_6 (Dense)	(None, 128)	409728
dense_7 (Dense)	(None, 10)	1290

Total params: 503690 (1.92 MB)
Trainable params: 503690 (1.92 MB)
Non-trainable params: 0 (0.00 Byte)


Test Accuracies of Different Models
Test Accuracy (%)
100.0
97.5
95.0
92.5
90.0
87.5
85.0
82.5
80.0
Initial Model
updated Model with dropouts
ResNet Model
Models


Test Accuracy Improvement
Test Accuracy (%)
91.4
91.2
91.0
90.8
90.6
90.4
90.2
91.5%
90.89%
90.14%
Transfer Learning from ResNet
Learning rate scheduler
With data augmentation
Technique


Test loss Decrement
Test Accuracy (%)
31.0
30.5
30.0
29.5
29.0
28.5
28.0
27.5
30.89%
29.61%
27.29%
Transfer Learning from ResNet
Learning rate scheduler
With data augmentation
Technique

5. Results

5.1 Random Forest Model Results

Initial Model

i. Validation Accuracy: 0.8814, which indicates that the model achieved an accuracy of approximately 88.14% on the validation set.
ii. Test Accuracy: 0.874, implying that the model attained an accuracy of around 87.4% on the test set.
iii. Classification Report:
    i. The classification report provides precision, recall, and F1-score for each class (0 to 9) in the test set, along with their respective support (number of occurrences).
    ii. Overall, the precision, recall, and F1-score for each class are reasonably high, with the model performing well across multiple classes.

Model with Dimensionality Reduction using PCA

i. Validation Accuracy: 0.8628, showing that the model achieved an accuracy of approximately 86.28% on the validation set.
ii. Test Accuracy: 0.8562, indicating that the model attained an accuracy of around 85.62% on the test set.
iii. Classification Report:
    - The precision, recall, and F1-score for each class (0 to 9) in the test set are provided, along with the support (number of occurrences) for each class.
    - Overall, the model exhibits reasonably high precision, recall, and F1-scores for each class, with the model performing well across different classes.

Model with Hyper-parameter training

i. Test Accuracy: 0.8759, indicating that the model achieved an accuracy of approximately 87.59% on the test set.
ii. Classification Report:
    - Precision, recall, and F1-score are provided for each class (0 to 9) in the test set, along with the support (number of occurrences) for each class.
    - The precision, recall, and F1-scores for each class are consistent with previous assessments, with the model consistently exhibiting high performance across different classes.

Model with reduced sample size

The Random Forest model trained on the sampled data achieved a test accuracy of 0.8509, signifying an accuracy of approximately 85.09% on the test set. The classification report provides precision, recall, and F1-score for each class (0 to 9) in the test set, along with their respective support (number of occurrences). The model exhibits relatively good performance across different classes, with strong precision, recall, and F1-scores.

Model with feature selection

The Random Forest model trained on the selected features yielded a test accuracy of 0.8651, indicating an accuracy of approximately 86.51% on the test set. The classification report provides precision, recall, and F1-score for each class (0 to 9) in the test set, along with their respective support (number of occurrences). The model exhibits strong performance across different classes, with high precision, recall, and F1-scores.

5.2 SVM Model Results

Initial Model

i. Test Accuracy: 0.8308, implying that the model attained an accuracy of around 83.08% on the test set.
ii. Classification Report:
    a. The classification report provides precision, recall, and F1-score for each class (0 to 9) in the test set, along with their respective support (number of occurrences).
    b. Overall, the precision, recall, and F1-score for each class are reasonably high, with the model performing well across multiple classes.

Initial Model with PCA Data

i. Test Accuracy: 0.8169, implying that the model attained an accuracy of around 81.69% on the test set.
ii. Classification Report:
    a. The classification report provides precision, recall, and F1-score for each class (0 to 9) in the test set, along with their respective support (number of occurrences).
    b. Overall, the precision, recall, and F1-score for each class are reasonably high, with the model performing well across multiple classes.

Polynomial Kernel

i. Test Accuracy: 0.8755, implying that the model attained an accuracy of around 87.55% on the test set.
ii. Classification Report:
    a. The classification report provides precision, recall, and F1-score for each class (0 to 9) in the test set, along with their respective support (number of occurrences).
    b. Overall, the precision, recall, and F1-score for each class are reasonably high, with the model performing well across multiple classes.

RBF Kernel

i. Test Accuracy: 0.8836, implying that the model attained an accuracy of around 88.36% on the test set.
ii. Classification Report:
    a. The classification report provides precision, recall, and F1-score for each class (0 to 9) in the test set, along with their respective support (number of occurrences).
    b. Overall, the precision, recall, and F1-score for each class are reasonably high, with the model performing well across multiple classes.

RBF Kernel with parameter tuning

i. Test Accuracy: 0.8475, implying that the model attained an accuracy of around 84.75% on the test set.
ii. Classification Report:
    a. The classification report provides precision, recall, and F1-score for each class (0 to 9) in the test set, along with their respective support (number of occurrences).
    b. Overall, the precision, recall, and F1-score for each class are reasonably high, with the model performing well across multiple classes.

5.3 CNN Model Results

Use the test dataset to evaluate the model performance on unseen data. This gives the following results for different CNN Models.

Initial Model

i. The initial model achieved high training accuracy (96.01%), indicating that it learned to classify the training data well. However, the validation and test accuracies were substantially lower (84.35% and 91.46%, respectively), suggesting that the model might have overfit the training data.
ii. The relatively high training loss (10.96) and significantly higher validation and test losses (45.44 and 37.04, respectively) also support the notion of overfitting. This indicates that the model's performance deteriorated significantly when presented with unseen data.

Fine-tuned Model with Dropouts

i. Introducing dropout layers helped mitigate overfitting, as evidenced by the improvement in validation and test accuracies compared to the initial model. Although the training accuracy decreased (88.44%), the model achieved better generalization, with validation and test accuracies reaching 85.72% and 92.37%, respectively.
ii. The decrease in training accuracy was accompanied by an increase in training loss (30.95), suggesting that dropout regularization introduced more uncertainty during training.

Model with Learning Rate Scheduler

i. Incorporating a learning rate scheduler during training further improved both training and validation accuracies. This adaptive adjustment of the learning rate likely helped the model converge more effectively, resulting in higher accuracies.
ii. The training loss decreased (21.66) compared to the fine-tuned model, indicating improved convergence and potentially smoother optimization trajectories.

Model Using ResNet Block

i. Utilizing ResNet blocks enhanced the model's ability to capture intricate features and patterns, resulting in higher training and validation accuracies compared to the initial model. However, there was a slight decrease in test accuracy, suggesting a potential trade-off between model complexity and generalization.
ii. The discrepancy between training and test accuracies may indicate some degree of overfitting, although the model's performance on unseen data remained relatively high.

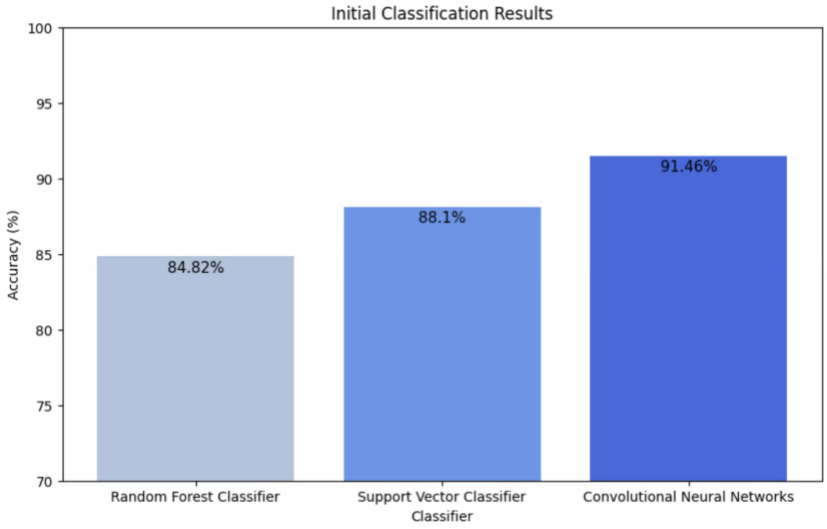
Model with Augmented Data

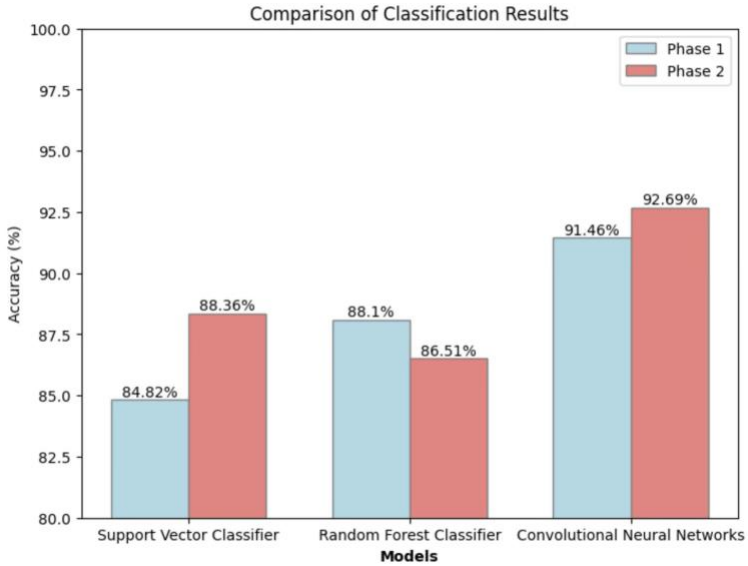
i. Augmenting the data helped the model generalize better to unseen data, as evidenced by the improvement in validation accuracy. However, the test accuracy slightly decreased compared to the model without data augmentation, suggesting that the augmented data might have introduced some noise or uncertainty.

ii. The training accuracy remained high (92.67%), indicating that the model effectively learned from the augmented data. However, the discrepancy between training and test accuracies suggests that further regularization or model adjustments may be beneficial.

5.4 CNN Model Results :

The plotted graph illustrates a comparison of the accuracy levels achieved by various models. The visual depiction showcases how different models perform in terms of accuracy, providing a concise and effective means of understanding their relative strengths in capturing the correct outcomes.





6. Conclusion

Mitigation of Overfitting: The initial model showed signs of overfitting with a high training accuracy but relatively lower validation and test accuracies. The introduction of dropout layers in the fine-tuned model helped in mitigating overfitting by improving generalization, as reflected in the increased validation and test accuracies. Adaptive Learning Rate Scheduler: The model with a learning rate scheduler demonstrated enhanced training and validation accuracies. The adaptive adjustment of the learning rate aided in better convergence during training, leading to improved accuracies. Utilization of ResNet Blocks: The model incorporating ResNet blocks was beneficial in capturing intricate features and patterns, resulting in higher training and validation accuracies compared to the initial model. Although there was a slight decrease in test accuracy, this approach showcased the model's improved ability to understand complex data patterns. Data Augmentation: The model using augmented data showed better generalization to unseen data with an improvement in validation accuracy. Despite a slight decrease in test accuracy, this approach indicated enhanced model robustness to diverse data scenarios. In summary, the CNN model outperformed others due to its ability to address overfitting through techniques like dropout layers, adaptive learning rate scheduling, utilization of ResNet blocks for intricate feature capture, and leveraging augmented data for improved generalization. These strategies collectively enhanced the model's performance on unseen data, showcasing its effectiveness in handling complex data sets.

Engaging in this project has been an enriching and insightful experience, as it provided abundant learning opportunities. One of the most fascinating aspects was exploring the correlation between hyperparameter tuning and performance statistics. This endeavor deepened our understanding of how the fine-tuning of hyperparameters can significantly impact the performance of machine learning models. It uncovered the intricate interplay between model architecture, hyperparameters, and overall model performance, shedding light on the need for meticulous exploration and optimization of various model parameters.

Furthermore, the project also highlighted the importance of comprehensive data preprocessing and augmentation techniques in preparing the dataset for training. The utilization of normalization and data augmentation strategies positively influenced the model's ability to generalize and make accurate predictions on unseen data, underlining the pivotal role of data quality and diversity in model training.

In addition, the project journey underscored the significance of employing Random Forest models, as well as the exploration of feature selection techniques. These strategies not only enhanced the model's performance but also provided valuable insights into the underlying dataset, emphasizing the multidimensional nature of machine learning model development.

Moreover, the visual representations utilized throughout the project, such as bar graphs and sample image grids, not only facilitated a comprehensive understanding of model

performance but also emphasized the importance of effective visualization in conveying complex technical concepts in a clear and accessible manner.

Overall, this project has been a valuable experience, delivering profound insights into the intricate nuances of machine learning model development. It has provided a platform for holistic learning, encompassing diverse aspects of model optimization, data preprocessing, feature selection, and the visualization of model performance.

The top learnings from the project can be summarized as follows:

1. Data Preprocessing and Augmentation: The project underscored the importance of data preprocessing techniques such as normalization and data augmentation. Normalization allowed for scaling the input features to a range suitable for machine learning models, while data augmentation enhanced model generalization and robustness by generating diverse training samples.
2. Model Complexity and Overfitting Mitigation: Various techniques, such as dropout layers, adaptive learning rate scheduling, utilization of ResNet blocks, and employment of augmented data, were found to be effective in addressing overfitting and enhancing model generalization. These strategies helped mitigate overfitting and improved the model's ability to capture intricate features and patterns, resulting in higher training and validation accuracies compared to the initial models.
3. Ensemble Learning and Feature Selection: The project demonstrated the benefits of ensemble learning using Random Forest models, which proved to be less prone to overfitting compared to individual decision trees. Additionally, feature selection using top feature importance enhanced the model's performance and provided insights into the dataset.
4. Hyperparameter Tuning and Model Fine-Tuning: The process of hyperparameter tuning and the exploration of different model architectures and regularization techniques resulted in improved model performance across different classification tasks. These efforts underscored the value of fine-tuning models to enhance their performance on specific datasets.
5. Visual Representation: The use of visual representations, such as bar graphs and sample image grids, provided concise and effective means of understanding model performance and prediction outcomes, facilitating insights into the relative strengths of different models in capturing correct classifications.

Overall, the project highlighted the significance of an iterative and comprehensive approach to model development, including data preprocessing, model architecture selection, regularization techniques, and effective evaluation and visualization of model performance, contributing to a deeper understanding of machine learning best practices and techniques in the context of image classification tasks.