Deep Learning Model on the Fashion MNIST Dataset

Submitted by:
Arun Rimal - N264S646

Submitted To:
Shruti Kshirsagar
CS770 Machine Learning

11/27/2024

Contents

List of Tables

List of Figures

1. Introduction

Neural Networks are models that mimic the complex functions of the human brain. They consist of interconnected nodes or neurons that process and learn from data, enabling tasks such as pattern recognition and decision-making in machine learning. In this project Neural network models were used to perform supervised learning tasks. The model used is the shallow neural network with one dense layer, simple convolutional neural networks and deep convolutional neural networks according to the need.

The dataset used is the Fashion-MNIST dataset which contains 60,000 training images (and 10,000 test images) of fashion and clothing items, taken from 10 classes. Each image is a standardized 28x28 size in grayscale (784 total pixels). Fashion-MNIST was created by Zalando as a compatible replacement for the original MNIST dataset of handwritten digits.

The objective of this work is to design and evaluate neural network models. For instance, shallow neural networks, simple convolutional neural networks, and deep convolutional neural networks, to classify images from the Fashion-MNIST dataset. This involves leveraging supervised learning techniques to accurately recognize fashion and clothing items across ten distinct categories.

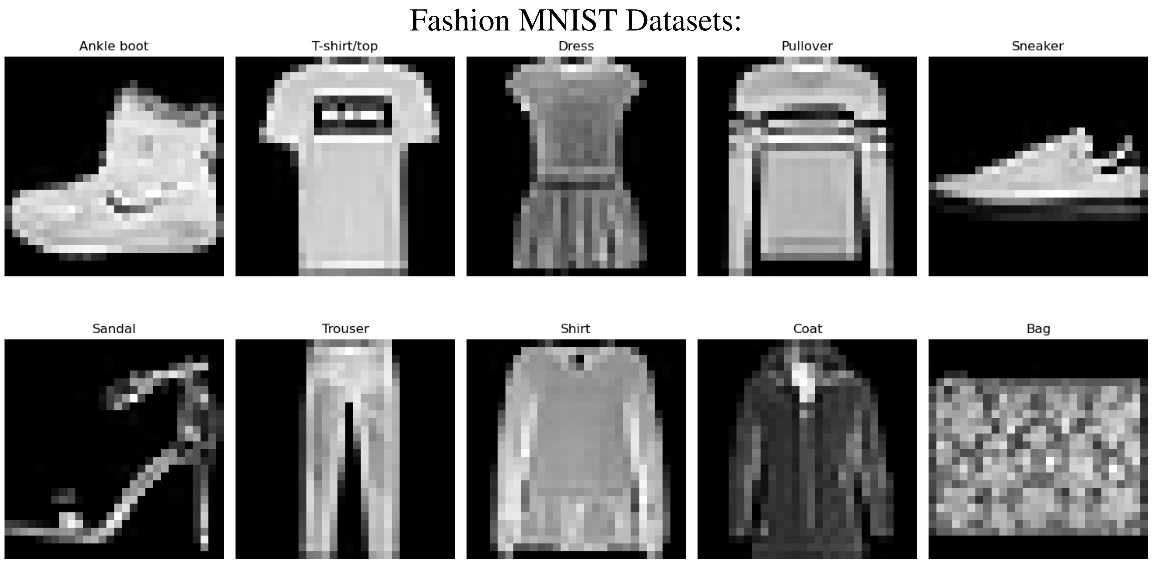


Figure 1.1: Fashion-MNIST Dataset

2. Methodology

2.1. Model Architecture

For supervised learning various models were chosen to determine the optimal performing model. Models are as follows.

The model structure for shallow neural network is shown below:

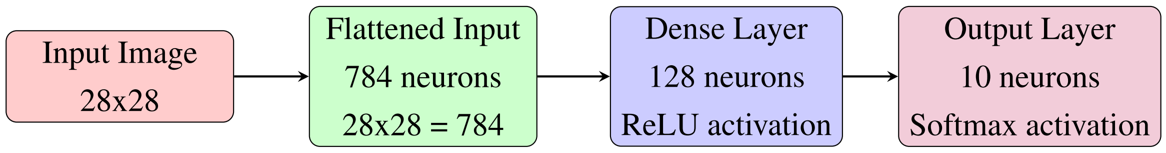


Figure 2.1: Model Structure of Shallow Neural Network

The diagram represents a shallow neural network used for image classification. The input layer gets 28x28 pixels of image as input into the network. This 2D image is then flattened into a 1D vector of 784 values. The flattened input is passed to a hidden dense layer of 128 neurons. The hidden layer consists of a ReLU as an activation function to introduce non-linearity. Finally, the output layer consists of 10 neurons. Each neuron represents one target class among all the target classes in the dataset. A softmax activation is applied to convert the raw outputs into probabilities. And the neuron with the highest probability is chosen as the network's prediction. This architecture is designed to classify images based on the features learned in the hidden layer.

Model Structure for Simple CNN is shown below:

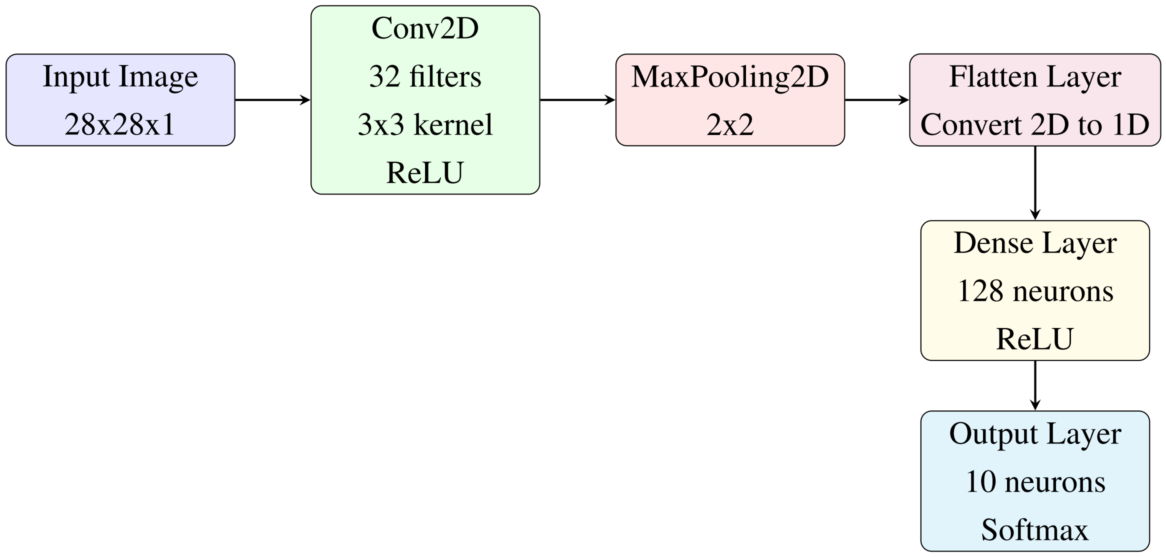


Figure 2.2: Model Structure of Simple CNN

The diagram above is the architecture of the simple Convolutional Neural Network (CNN) of image classification. It states that the input size of the image 28x28x1. The next step in the process consists of a convolution layer having 32 filters of size 3x3 with ReLU activation function to extract features like edges and textures. This is followed by a MaxPooling Layer with 2x2 filter. Max pooling downsamples the feature maps to reduce computational complexity and highlight dominant features. After max pooling the 2D output is flattened into a 1D vector to prepare the data for fully connected layers. A Dense Layer is present to learn high-level representations. Finally, the Output Layer, with 10 neurons and a Softmax activation function, predicts class probabilities for the given image.

Model Structure for Deep CNN is shown below:

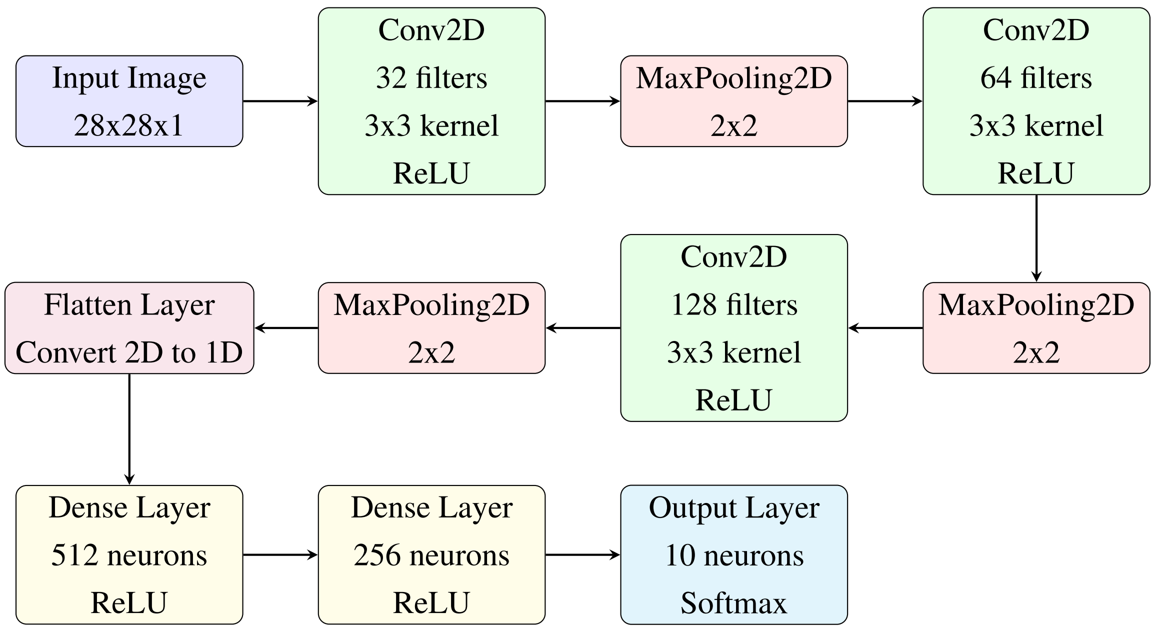


Figure 2.3: Deep Convolutional Neural Network Architecture

The above architecture is a deep Convolutional Neural Network (CNN) for image classification. The input is 28x28x1 consisting of a grayscale image. The input is fed to convolutional layer consisting of 32 filter of 3x3 size and a ReLU activation to extract basic features. After that, it is max pooled with 2x2 size and extract dominant features and downsampling the input. This process of convolution and maxpooling is repeated two times with 64 and 128 filter size respectively. After the convolution and max pooling the 2D data is flattened into 1D using a Flatten layer. It is changed into 1D to feed the data into Dense layer having 512 neurons and 256 neurons respectively with ReLU activation. Finally, an Output Layer with 10 neurons and a Softmax activation function predicts probabilities for the 10 output classes. This design ensures hierarchical feature extraction and efficient classification.

CNN with Batch Normalization and Dropout is shown below:

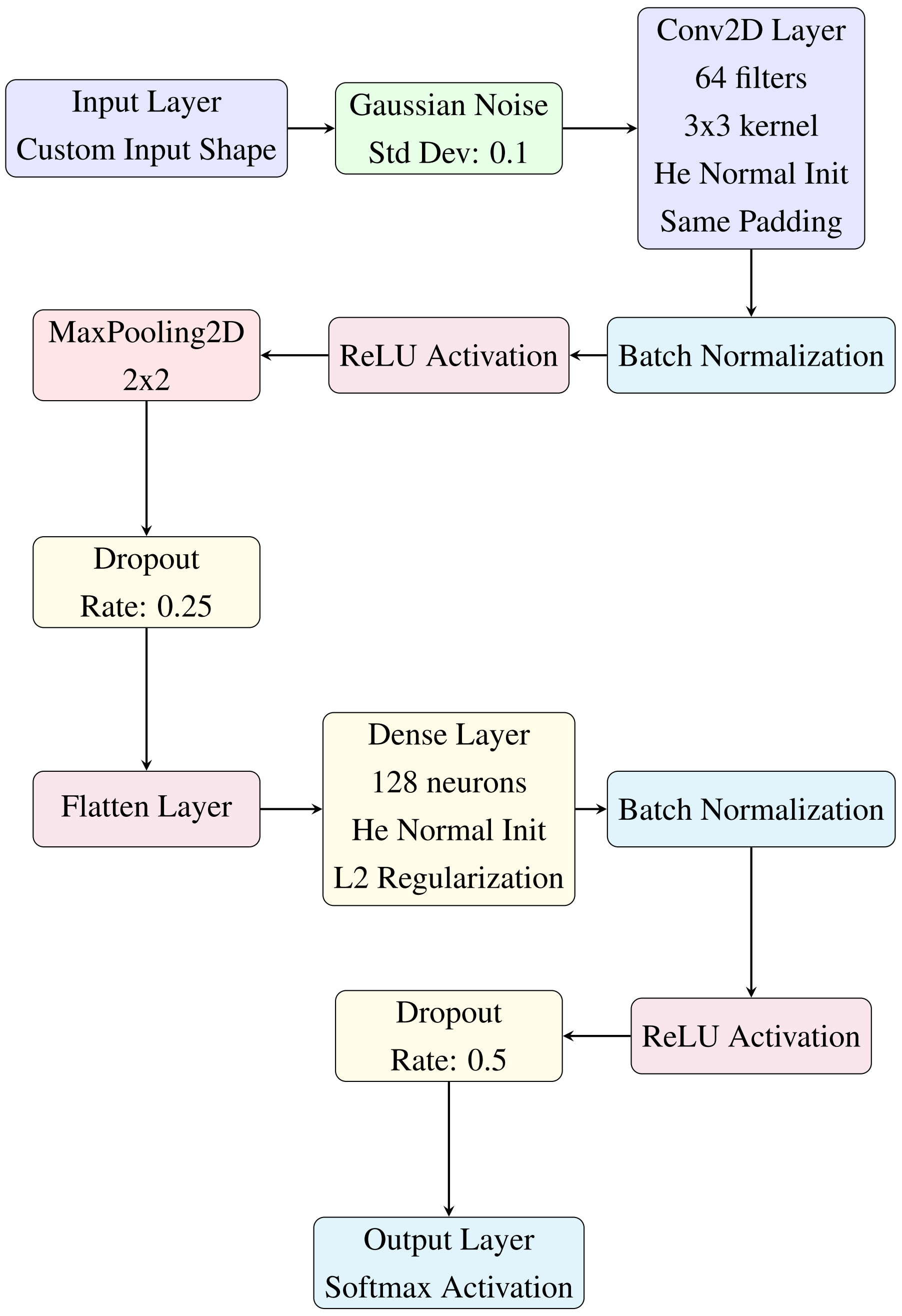


Figure 2.4: Custom Neural Network Architecture

The above neural consists of a robust connection of layers for image classification task. It consists of input of size 28x28x1 which is followed by addition of gausian Noise. Noise is added for regularization and feed the model with real practical data. Than, it is passed through Conv2D layer having 64 filters of size 3x3. After that, Batch normalization and ReLU activation is performed as a stabilization technique which ensures efficient learning.

MaxPooline is performed for downsampling feature maps to retain key information. After retaining key information Dropout layers (25% and 50%) and L2 regularization is performed to prevent overfitting. Dense layer with 128 neurons is used to connect extracted features with each other together to learn the hidden non-linear relation between the input and output for which RelU and softmax help to introduce non-linearity in the model.
This architecture is suitable for various dataset because it has balances complexity and regularization both.

Neural Network with skip connection (resembling ResNet) is shown below:

This model resembles with residual neural network that adds input to the layer's output. Where as this network concat the input with other layer's transformed output. So it not fully a residual network but only resembles like a ResNet. It consists of grayscale input of size 28x28 which is flattened and provided to the 784 Neurons Dense layer. After that, it is batch normalized and ReLU activated for stabilization and efficient learning. The model is introduced with Dropout which is used to reduce overfitting. Similarly, another block of dense layer also goes through the same process and the output form the layer is concatenated with direct input. This helps to preserve information and enables more complex feature intersection. Also, allows network to learn relationship between original input and processed features. Following step is to flatten the output into 1D so that Dense layer having softmax activation will categorize the into into 10 output classes.

This architecture is well suited for image classification due to its balanced combination of dense layer for feature extraction and skip connection for preserving original image features.

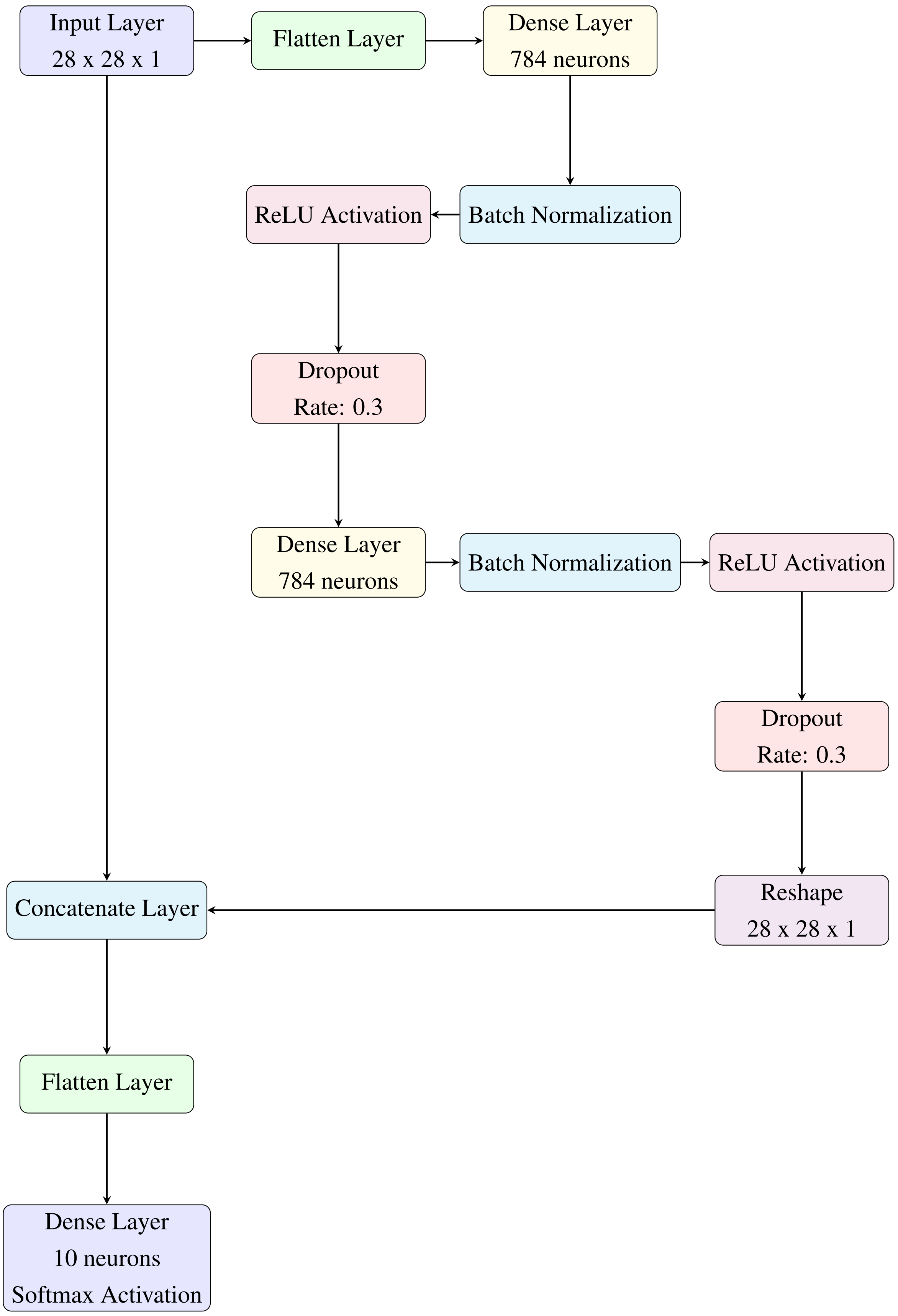


Figure 2.5: Neural Network with skip Connection

2.2. Preparation of Data

The first step involves the preparation of data suitable for the model building. This dataset requires data normalization, data formating etc.

- The Loaded data from the keras dataset

```
# Import Fashion MNIST Dataset from keras dataset
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) =
    fashion_mnist.load_data()
```

- Normalizing the pixel values between [0,1]

```
# Normalize the pixel values of the images between 0 and 1
train_images = train_images / 255.0
test_images = test_images / 255.0
```

- Perform data formating one-hot-encoding

```
from sklearn.preprocessing import OneHotEncoder
ecoder = OneHotEncoder(sparse_output=False)
train_labels_ohe = ecoder.fit_transform(train_labels.reshape
    (-1,1))
test_labels_ohe = ecoder.transform(test_labels.reshape(-1,1))
```

2.3. Model building

There are several methods to handle classification problems. Each can be chosen based on data size, complexity, and class distribution. After data preparation model building is performed using various methods. During model building, Hyperparameter tuning is performed using the GridSearchCV library.

Shallow Neural Network is a model used to predict the probability of a multiclass category target variable based on input image.

```
# Build the model
shallowModel_sgd = Sequential([
    Flatten(input_shape=(28, 28)), # Transforms the format of the
       images from a 2D array to a 1D array
    Dense(128, activation='relu'), # First Dense layer with 128
       nodes (neurons)
    Dense(10, activation='softmax') # Output Dense layer with 10
       nodes for 10 class labels
])

# Model Summary
shallowModel_sgd.summary()

# Compile the model

# Optimizer: 'sgd' is generally simpler and uses a initial learning
     rate(0.1)
# used 'momentum' as 0.9 to accelerate SGD in the relevent
    direction.
# 'nesterov' (also called Nesterov Accelerated Gradient (NAG)) is
    an advancement in the SDG

initial_learning_rate = 0.01
optimizer = SGD(
    learning_rate=initial_learning_rate,
    momentum=0.9,
    nesterov=True
)

# used CategoricalCrossentropy for loss calculation
shallowModel_sgd.compile(optimizer,
             loss= tf.keras.losses.CategoricalCrossentropy,
             metrics=['accuracy'])

# Train the model
shallowModelHistory_sgd = shallowModel_sgd.fit(
           train_images,
           train_labels_ohe,
           batch_size=32,
           epochs=100,
           validation_split=0.2
       )
```

Simple Convolutional Neural Network is a model used to predict the probability of a multiclass category target variable based on input image. This model consists of convolutional layer unlike shallow neural network that contains input and dense layer only.

```
# Build the Sequential Model
basicCNNModel_sgd = Sequential([
    # Convolutional layers
    layers.Input(shape=(28, 28, 1)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # Dense layers
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

initial_learning_rate = 0.01
optimizer = SGD(
    learning_rate=initial_learning_rate,
    momentum=0.9,
    nesterov=True
)

# Model Summary
basicCNNModel_sgd.summary()

# Compile the Model
basicCNNModel_sgd.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# 9. Train the Model
basicCNNModelHistory_sgd = basicCNNModel_sgd.fit(
    train_images,
    train_labels_ohe,
    batch_size=32,
    epochs=100,
    validation_split=0.2,
    verbose=1
)
```

Deep Convolutional Neural Network is a model used to predict the probability of a multi-class category target variable based on input image. This model consists of more number of convolutional layers and dense layer.

```
# Build the convolutional neural network model
deepCNNModel_sgd = models.Sequential([
    # Input
    layers.Input(shape=(28,28,1)),

    # Convolutional layers
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    # Dense layers
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(256, activation='relu'),
    layers.Dense(10, activation='softmax')
])

initial_learning_rate = 0.1
optimizer = SGD(
    learning_rate=initial_learning_rate,
    momentum=0.9,
    nesterov=True
)


# Compile the model
deepCNNModel_sgd.compile(optimizer='adam',
             loss='categorical_crossentropy',
             metrics=['accuracy'])

# Model Summary
deepCNNModel_sgd.summary()

# Train the model with the training data
deepCNNModelHistory_sgd = deepCNNModel_sgd.fit(
        train_images,
        train_labels_ohe,
        epochs=100,
        batch_size=32,
        validation_split=0.2,
        verbose=1
        )
```

2.4. Model Evaluation

Model evaluation is performed using evaluation matrices. Models are compared between the same model structure models and with the model with different model structure. i.e. horizontally as well as vertical comparison were performed. Model are categorized under two committee name ’SGD Committee’ and ’Adam Committee’. The performance were compared among these two committee. Also an individual level comparison is also performed among the models.

```
# Evaluate the model
test_loss, test_acc = shallowModel_adam.evaluate(test_images,
   test_labels_ohe, verbose=3)
print('\nTest accuracy:', test_acc, '\nTest loss:', test_loss)

# Make predictions on the test dataset
predictions = shallowModel_adam.predict(test_images)
predicted = np.argmax(predictions, axis=1)

# Calculate matrices
confusionMatrix = confusion_matrix(y_true=test_labels, y_pred=
   predicted)

classificationReport = classification_report(y_true=test_labels,
   y_pred=predicted)
print(f"Classification Report : \n {classificationReport}")

disp = ConfusionMatrixDisplay(confusion_matrix=confusionMatrix,
   display_labels=class_names)
disp.plot(cmap='viridis', xticks_rotation=90)
plt.tight_layout()
plt.show()
```

3. Result and Discussion

3.1. Performance Evaluation Using Evaluation Matrices

When solving classification problems, it is important to understand a model's strengths and weaknesses and identify areas for improvement. Among several evaluation matrices are Accuracy, Precision, Recall, and F1 Score. These metrics, which provide insights into a model's ability to correctly classify instances, are as follows.

Accuracy is the ratio of correctly predicted instances to the total number of instances. It is a straightforward measure of overall performance. It is easy to interpret and provides a general overview of model performance. It is suitable for a balanced dataset and does not provide a good overview for imbalanced datasets. In an imbalanced dataset, accuracy is misleading because it generally provides the accuracy of the highest repeated class or the class which are in the majority.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It measures the model's accuracy by identifying true positive instances.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall, or sensitivity, is the ratio of correctly predicted positive observations to all actual positives. It measures the model's ability to capture all true positive instances.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1 Score is the harmonic mean of Precision and Recall. It balances the two metrics, providing a single metric that accounts for both false positives and false negatives.

$$\text{Recall} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{False Negatives} \times \text{False Negatives}}$$

3.2. Comparision between same model Structure

All models in this study were trained using the following common hyperparameters:

- **Learning Rate:** 0.01
- **Loss Function:** `categorical_crossentropy`
- **Evaluation Metrics:** `accuracy`
- **Batch Size:** 32
- **Validation Split:** 0.2
- **Epoch:** 100
- **Optimizer Details:**
  - **Stochastic Gradient Descent (SGD):**
    - **Learning Rate:** 0.01 (initial)
    - **Momentum:** 0.9
    - **Nesterov Accelerated Gradient (NAG):** Enabled
  - **Adam Optimizer:**
    - **Learning Rate Schedule:** Exponential Decay
    - **Initial Learning Rate:** 0.01
    - **Decay Steps:** 100
    - **Decay Rate:** 0.9
    - **Staircase Decay:** True

3.2.1. Shallow Neural Network

There are two models based on shallow neural network which is structurally same but the optimizer and learning approach of the model is different. One using SGD and another using Adam as optimizer with respective learning rates and approach.
The total training parameters in both the model is shown below. :

Layer (type)	**Output Shape**	**Param #**
flatten_21 (Flatten)	(None, 784)	0
dense_45 (Dense)	(None, 128)	100,480
dense_46 (Dense)	(None, 10)	1,290

Table 3.1: Shallow Model Parameter Summary

Accuracy of the shallow models is shown below:

Model	**Test Accuracy**	**Test Loss**
shallowModel_sgd	0.857	0.855
shallowModel_adam	0.865	0.369

Table 3.2: Shallow Model Accuracy

The table compares the performance of the model one using SGD and another using Adam optimizer. The model using Adam optimizer performs (86.5%) slightly better and significantly lower test loss (0.369) compared to SGD, which has 85.7% accuracy and 0.855 loss.

The below graph shows the accuracy and loss plot of training and validation data for SGD and Adam optimized shallow models:

Line Graph Loss and Accuracy SGD vs Adam :

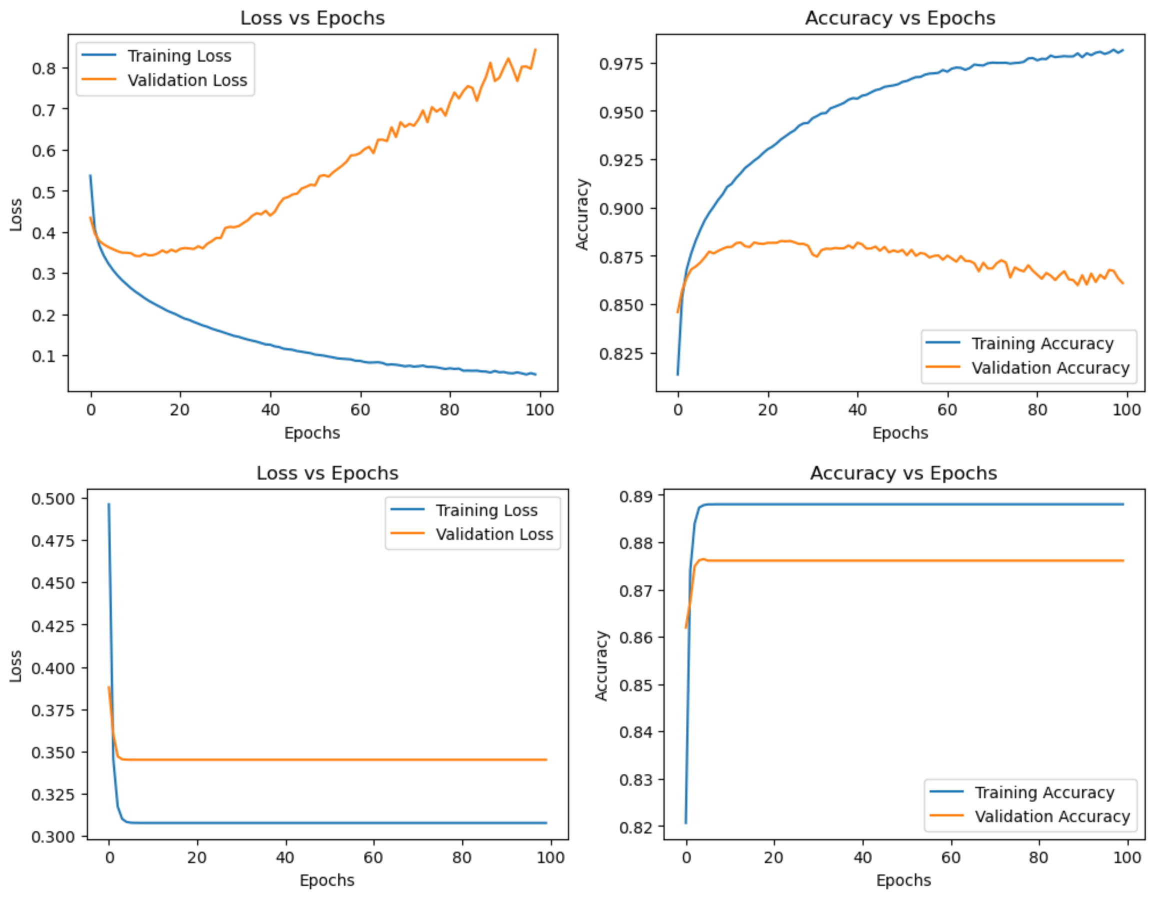


Figure 3.1: Shallow Model Accuracy and Loss Graph

The first graph shows the loss and accuracy plot for the model using the SGD optimizer, while the second graph shows the loss and accuracy plot for the model using the Adam optimizer. The loss plot for SGD increases in the validation dataset while decreasing in the training dataset. The loss plot for the Adam optimizer model shows a positive correlation, where both training and validation losses decrease together. After a few epochs, the losses remain steady at their respective values.

For the SGD optimizer model, accuracy increases for the training data and remains steady for the validation dataset. For the Adam optimizer model, accuracy increases up to a certain point and then remains stable throughout the epoch cycle.

Precision, Recall and F1 Score for Shallow Models is shown below:

Metric	SGD Optimizer			Adam Optimizer		
Metric Type	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
Accuracy			0.86			0.87
Macro Avg	0.87	0.86	0.86	0.86	0.87	0.86
Weighted Avg	0.87	0.86	0.86	0.86	0.87	0.86

Table 3.3: Shallow models classification Reports

All the performance metrics for SGD and Adam optimizer model is around 0.86 and 0.87 respectively with few exceptions.

Confusion Matrix for shallow models is shown below:

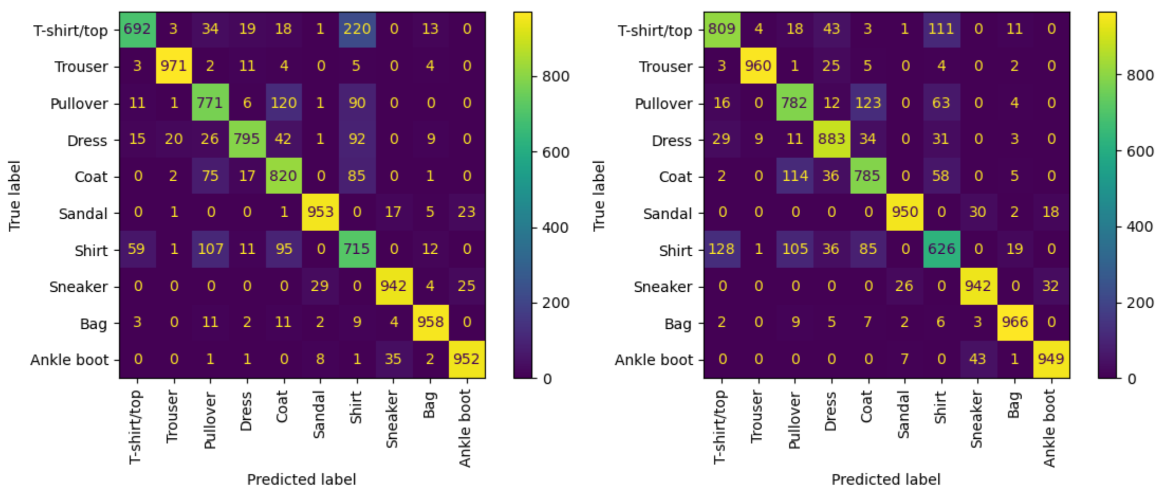


Figure 3.2: Shallow Model Confusion Matrix

From the above confusion matrix, class 'Shirt' seems to be confused quite often with other classes like 'T-shirt/top', 'Pullover' and 'Coat'. Both the model shows the similar behaviour. But model with adam optimizer is more confused with the class 'Shirt' and 'T-shirt/top'. Similarly, class 'Coat' with 'Pullover'.

3.2.2. Basic Convolution Neural Network

There are two models based on basic Convolutional neural network which is structurally same but the optimizer and learning approach of the model is different. One using SGD and another using Adam as optimizer with respective learning rates and approach.
The total training parameters in both the model is shown below. :

Layer (type)	Output Shape	Param #
conv2d_23 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_23 (MaxPooling2D)	(None, 13, 13, 32)	0
flatten_23 (Flatten)	(None, 5408)	0
dense_49 (Dense)	(None, 128)	692,352
dense_50 (Dense)	(None, 10)	1,290

Table 3.4: Basic CNN Parameter Summary

Accuracy for Shallow models is shown below:

Model	Test Accuracy	Test Loss
basicCNNModel_sgd	0.911	0.920
basicCNNModel_adam	0.906	0.262

Table 3.5: Basic CNN Model Accuracy

The table compares the performance of the model one using SGD and another using Adam optimizer. The model using SGD optimizer performs (91.4%) slightly better and significantly higher test loss (0.818) compared to Adam, which has 90.4% accuracy and 0.266 loss. But considering both Adam optimizer model is better than sgd optimizer model.

The below graph shows the accuracy and loss plot of training and validation data for SGD and Adam optimized models:

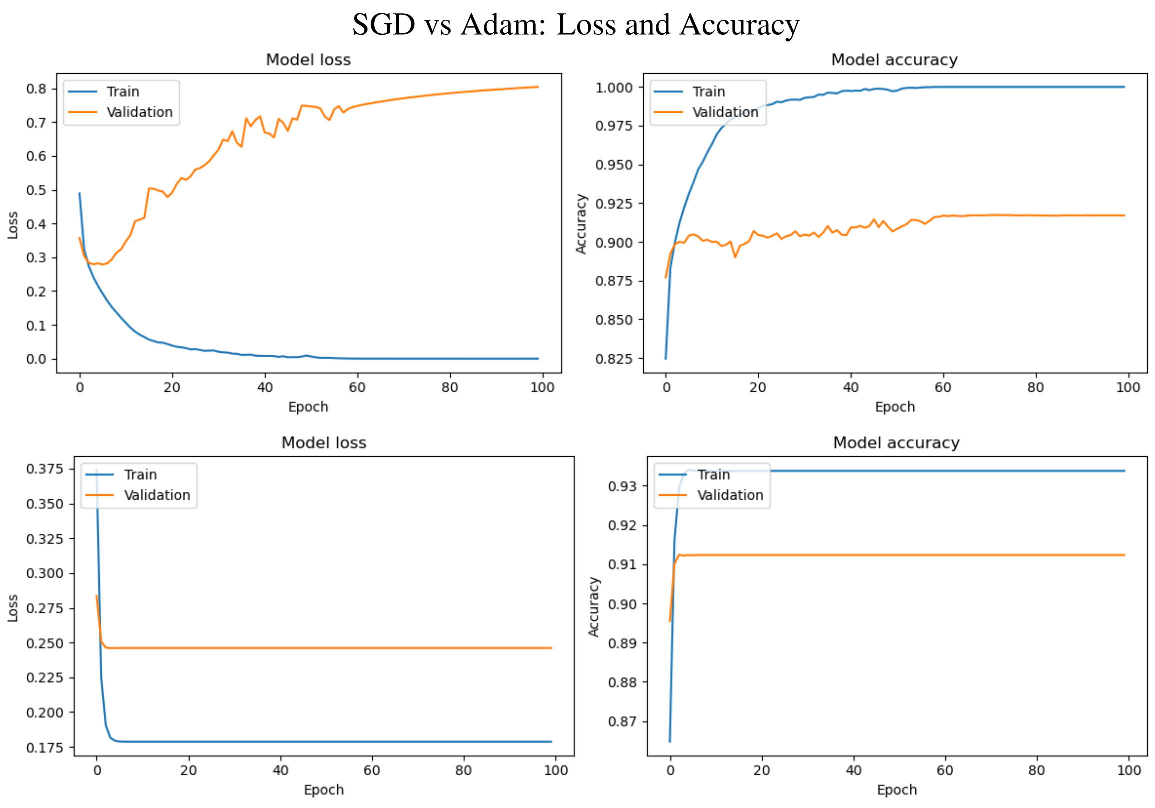


Figure 3.3: Basic CNN Model Accuracy and Loss Graph

The first graph shows the loss and accuracy plot for the model using the SGD optimizer, while the second graph shows the loss and accuracy plot for the model using the Adam optimizer. The loss plot for SGD increases in the validation dataset while decreasing in the training dataset. The loss plot for the Adam optimizer model shows a positive correlation, where both training and validation losses decrease together. After a few epochs, the losses remain steady at their respective values. For the SGD optimizer model, accuracy increases for the training data and remains steady for the validation dataset. For the Adam optimizer model, accuracy increases up to a certain point and then remains stable throughout the epoch cycle.

Precision, Recall and F1 Score for Basic CNN model is shown below:

Metric	SGD Optimizer			Adam Optimizer		
Metric Type	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
Accuracy			0.91			0.90
Macro Avg	0.91	0.91	0.91	0.90	0.90	0.90
Weighted Avg	0.91	0.91	0.91	0.90	0.90	0.90

Table 3.6: Classification Reports for Basic CNN models

All the performance metrics for SGD and Adam optimizer model is around 0.91 and 0.90 respectively.

Confusion Matrix for Basic CNN Model is shown below:

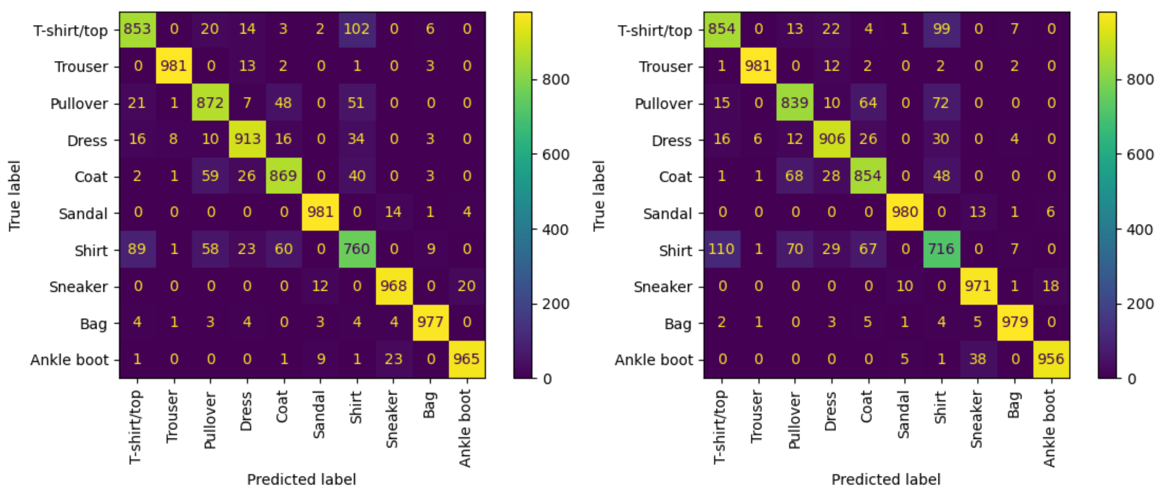


Figure 3.4: Basic CNN Model Confusion Matrix

From the above confusion matrix, class ’Shirt’ seems to be confused quite often with other classes like ’T-shirt/top’, ’Pullover’ and ’Coat’. Similarly, class ’T-shirt/top’ with class ’Shirt’.Both the model shows the similar behaviour. But model with adam optimizer is more confused with the class ’Shirt’ and ’T-shirt/top’. Similarly, class ’Coat’ with ’Pullover’.

3.2.3. Deep Convolution Neural Network

There are two models based on deep Convolutional neural network which is structurally same but the optimizer and learning approach of the model is different. One using SGD and another using Adam as optimizer with respective learning rates and approach. The total training parame- ters are same in both the model. Below is the parameter for both the models:

Layer (type)	**Output Shape**	**Param #**
conv2d_25 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_25 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_26 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_26 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_27 (Conv2D)	(None, 3, 3, 128)	73,856
max_pooling2d_27 (MaxPooling2D)	(None, 1, 1, 128)	0
flatten_25 (Flatten)	(None, 128)	0
dense_53 (Dense)	(None, 512)	66,048
dense_54 (Dense)	(None, 256)	131,328
dense_55 (Dense)	(None, 10)	2,570

Table 3.7: Deep CNN Model Parameter Summary

Accuracy for Deep CNN models is shown below:

Model	**Test Accuracy**	**Test Loss**
deepCNNModel_sgd	0.869	1.108
deepCNNModel_adam	0.847	0.416

Table 3.8: Deep CNN Model Accuracy

The table compares the performance of the model one using SGD and another using Adam optimizer. The model using SGD optimizer performs (86.9%) better and signifi- cantly higher test loss (1.108) compared to Adam, which has 84.7% accuracy and 0.416 loss. But considering both Adam optimizer model is better than sgd optimizer model.

The below graph shows the accuracy and loss plot of training and validation data for SGD and Adam optimized models:

SGD vs Adam: Accuracy and Loss:

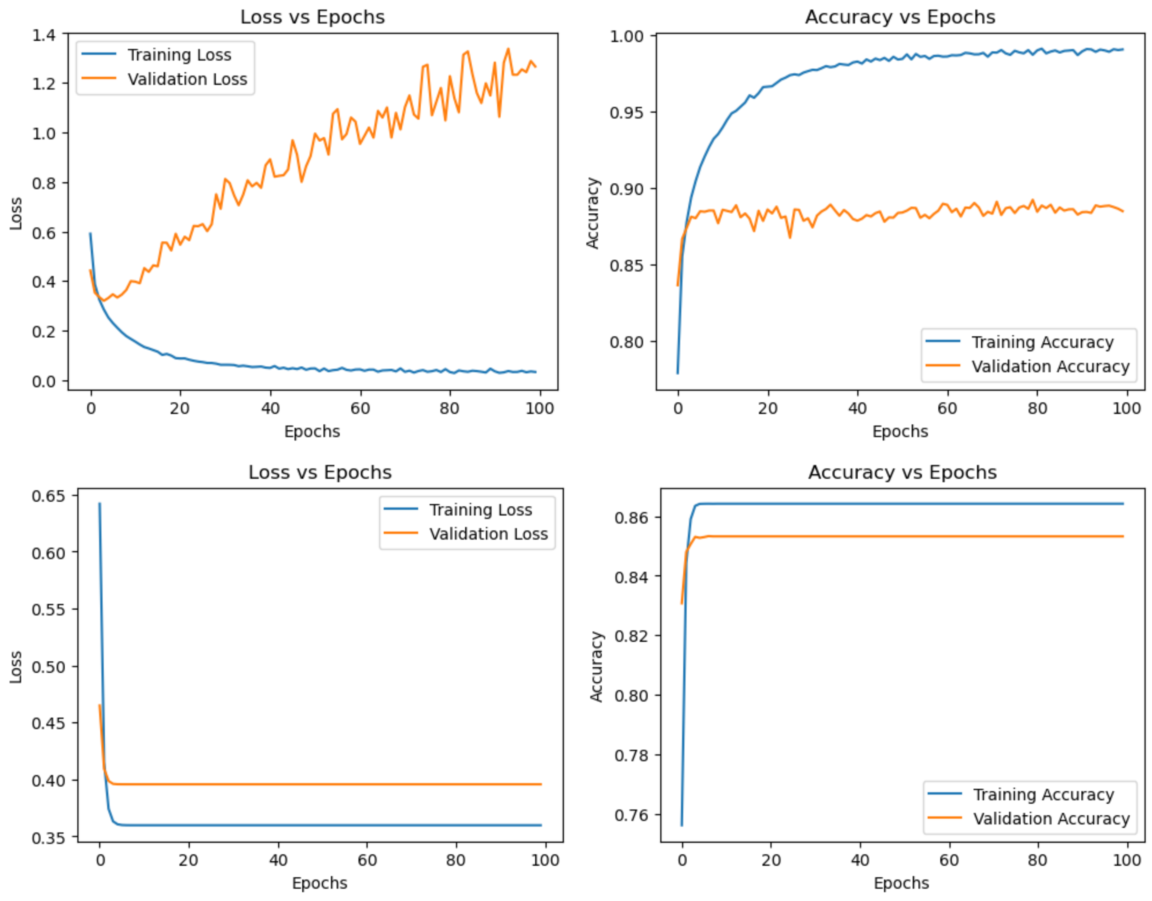


Figure 3.5: Deep CNN Model using SGD vs Adam

The first graph shows the loss and accuracy plot for the model using the SGD optimizer, while the second graph shows the loss and accuracy plot for the model using the Adam optimizer. The loss plot for SGD increases in the validation dataset while decreasing in the training dataset. The loss plot for the Adam optimizer model shows a positive correlation, where both training and validation datasets. Initially, both dataset losses decreases on initial epoch and remains steady afterwards.For the SGD optimizer model, accuracy increases for the training data and remains steady for the validation dataset. For the Adam optimizer model, accuracy increases to a certain point and remains stable throughout the epoch cycle.

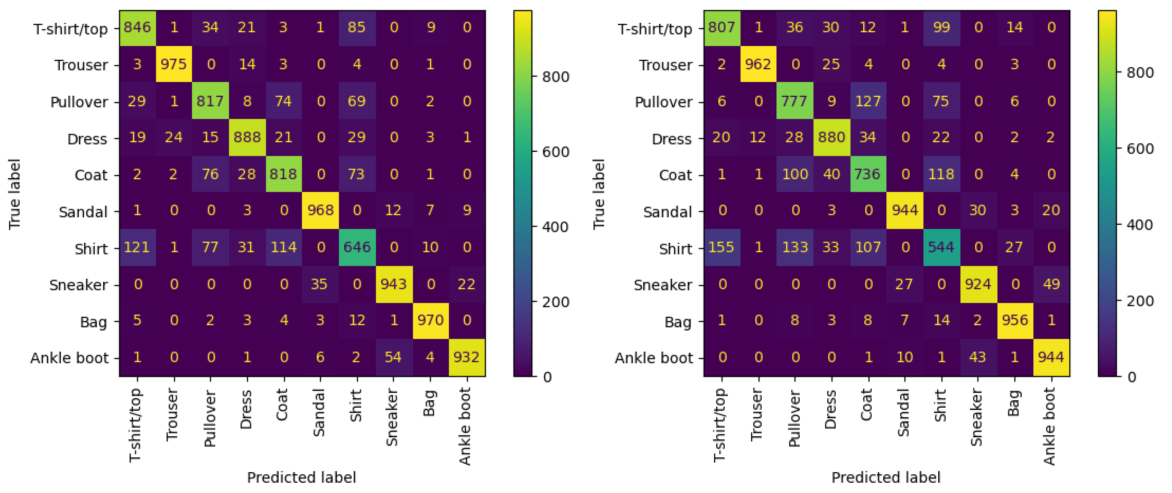
Precision, Recall and F1 Score for Deep CNN model is shown below:

Metric	SGD Optimizer			Adam Optimizer		
Metric Type	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
Accuracy			0.87			0.85
Macro Avg	0.87	0.87	0.87	0.85	0.85	0.85
Weighted Avg	0.87	0.87	0.87	0.85	0.85	0.85

Table 3.9: Classification Reports for Deep CNN

All the performance metrics for SGD and Adam optimizer model is around 0.87 and 0.85 respectively.

Confusion Matrix for Deep CNN Model is shown below:



True label \ Predicted label	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	846	1	34	21	3	1	85	0	9	0
Trouser	3	975	0	14	3	0	4	0	1	0
Pullover	29	1	817	8	74	0	69	0	2	0
Dress	19	24	15	888	21	0	29	0	3	1
Coat	2	2	76	28	818	0	73	0	1	0
Sandal	1	0	0	3	0	968	0	12	7	9
Shirt	121	1	77	31	114	0	646	0	10	0
Sneaker	0	0	0	0	0	35	0	943	0	22
Bag	5	0	2	3	4	3	12	1	970	0
Ankle boot	1	0	0	1	0	6	2	54	4	932

True label \ Predicted label	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	807	1	36	30	12	1	99	0	14	0
Trouser	2	962	0	25	4	0	4	0	3	0
Pullover	6	0	777	9	127	0	75	0	6	0
Dress	20	12	28	880	34	0	22	0	2	2
Coat	1	1	100	40	736	0	118	0	4	0
Sandal	0	0	0	3	0	944	0	30	3	20
Shirt	155	1	133	33	107	0	544	0	27	0
Sneaker	0	0	0	0	0	27	0	924	0	49
Bag	1	0	8	3	8	7	14	2	956	1
Ankle boot	0	0	0	0	1	10	1	43	1	944

Figure 3.6: Deep CNN Model Confusion Matrix

From the above confusion matrix, class 'Shirt' seems to be confused quite often with other classes like 'T-shirt/top', 'Pullover' and 'Coat'. Similarly, class 'T-shirt/top' with class 'Shirt' and class 'Coat' with class 'Shirt'. Both the model shows the similar behaviour. But model with adam optimizer is more confused with the class 'Shirt' and 'T-shirt/top', 'Pullover', 'Coat'. Similarly, class 'Coat' with 'Shirt'.

3.3. CNN model with Batch Normalization and Dropout

This model is structurally similar to basic CNN with batch normalization and dropout added to the model. This model is using adam optimizer only. And rest of the learning parameters are same.
Below is the parameter summary of the model:

Layer (type)	**Output Shape**	**Param #**
gaussian_noise_10 (GaussianNoise)	(None, 28, 28, 1)	0
conv2d_31 (Conv2D)	(None, 28, 28, 64)	640
batch_normalization_20 (BatchNormalization)	(None, 28, 28, 64)	256
activation_20 (Activation)	(None, 28, 28, 64)	0
max_pooling2d_31 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_20 (Dropout)	(None, 14, 14, 64)	0
flatten_27 (Flatten)	(None, 12544)	0
dense_59 (Dense)	(None, 128)	1,605,760
batch_normalization_21 (BatchNormalization)	(None, 128)	512
activation_21 (Activation)	(None, 128)	0
dropout_21 (Dropout)	(None, 128)	0
dense_60 (Dense)	(None, 10)	1,290

Table 3.10: CNN (with Dropout) Model Parameter Summary

Accuracy for CNN model (with Dropout) is shown below:

Model	**Test Accuracy**	**Test Loss**
extraModel	0.818	0.583

Table 3.11: CNN (with Dropout) Model Accuracy

The table compares the performance of the model having dropout and batch normalization. The model is using Adam optimizer to provide accuracy of around 82% and loss of around 0.58.

The below graph shows the accuracy and loss plot of training and validation data for SGD and Adam optimized models:

Accuracy vs Loss :

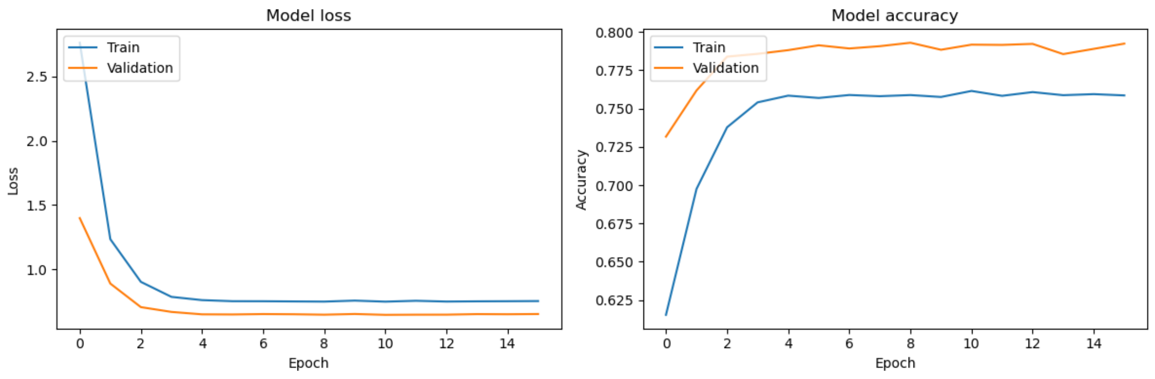


Figure 3.7: CNN Model (with Dropout) Line Plot

The graph shows the loss and accuracy plot for the model. The loss for both training and validation is decreasing with the increase in epoch and remains steady after some epoch. Whereas accuracy increases with the increase in epoch and remain stable after some epoch for both training and validation dataset.

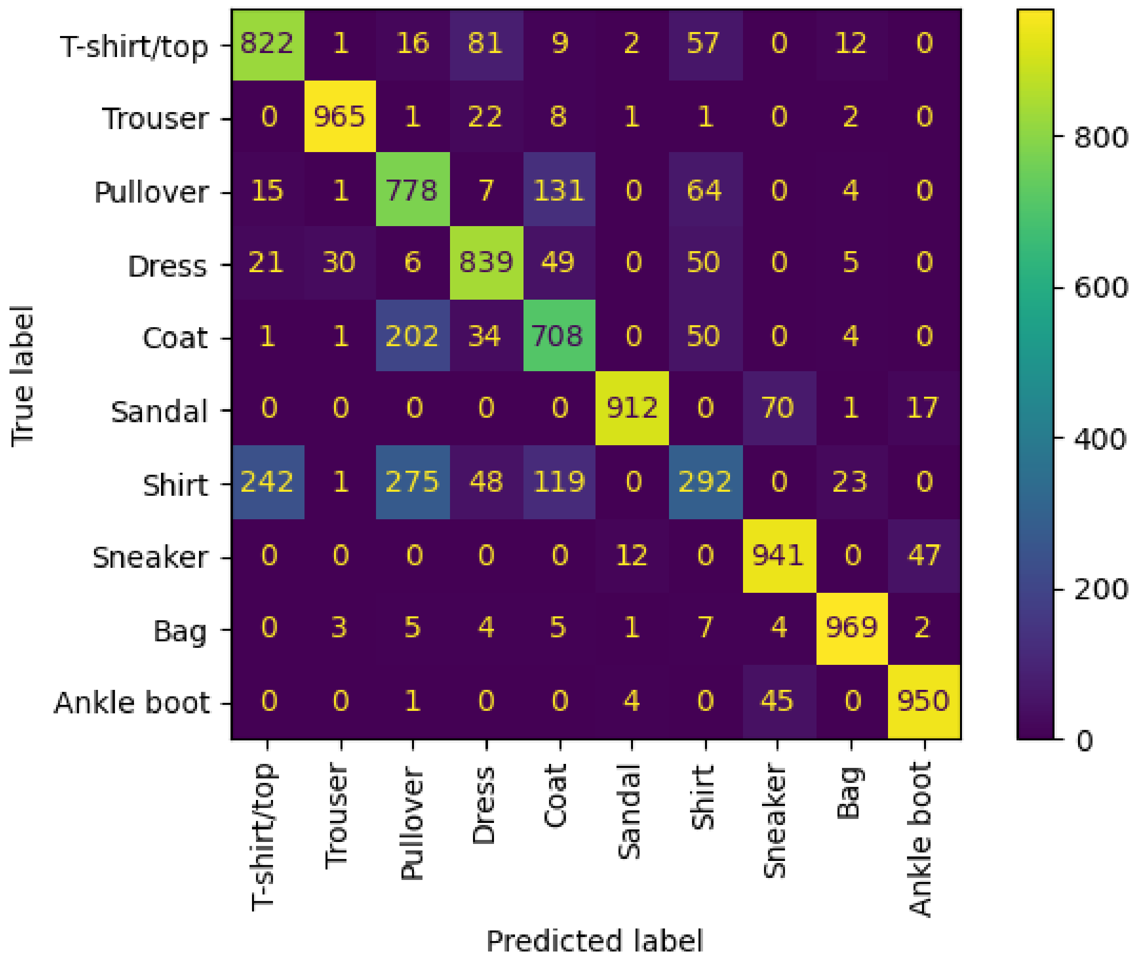
Precision, Recall and F1 Score for CNN model (with Dropout) is shown below:

Metric	Adam Optimizer		
Metric Type	**Precision**	**Recall**	**F1 Score**
Accuracy			0.82
Macro Avg	0.81	0.82	0.81
Weighted Avg	0.81	0.82	0.81

Table 3.12: Classification Reports for Deep CNN

All the performance metrics for the model are around 0.82.

Confusion Matrix for CNN Model (with Dropout) is shown below:



True label \ Predicted label	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	822	1	16	81	9	2	57	0	12	0
Trouser	0	965	1	22	8	1	1	0	2	0
Pullover	15	1	778	7	131	0	64	0	4	0
Dress	21	30	6	839	49	0	50	0	5	0
Coat	1	1	202	34	708	0	50	0	4	0
Sandal	0	0	0	0	0	912	0	70	1	17
Shirt	242	1	275	48	119	0	292	0	23	0
Sneaker	0	0	0	0	0	12	0	941	0	47
Bag	0	3	5	4	5	1	7	4	969	2
Ankle boot	0	0	1	0	0	4	0	45	0	950

Figure 3.8: CNN Model (with Dropout) Confusion Matrix

From the above confusion matrix, class ’Shirt’ seems to be confused quite often with other classes like ’T-shirt/top’, ’Pullover’ and ’Coat’. Similarly, class ’T-shirt/top’ with class ’Shirt’ and class ’Coat’ with class ’Shirt’ and ’Dress’. Also, class ’Pullover’ with class ’Coat’.

3.4. Neural Network (with Skip Connection) model:

This model is structurally similar to shallow Neural Network but this network model takes inputs from previous layers unlike other neural models which take sequential input from immediate previous layer. This model has dropout to address overfitting problem.

Layer (type)	Output Shape	Param #	Connected to
input_layer_31 (InputLayer)	(None, 28, 28, 1)	0	-
flatten_28 (Flatten)	(None, 784)	0	input_layer_31[0][0]
dense_61 (Dense)	(None, 784)	615,440	flatten_28[0][0]
batch_normalization_22 (BatchNormalization)	(None, 784)	3,136	dense_61[0][0]
activation_22 (Activation)	(None, 784)	0	batch_normalization_22[0][0]
dropout_23 (Dropout)	(None, 784)	0	activation_22[0][0]
reshape (Reshape)	(None, 28, 28, 1)	0	dropout_23[0][0]
concatenate (Concatenate)	(None, 28, 28, 2)	0	input_layer_31[0][0], reshape[0][0]
flatten_29 (Flatten)	(None, 1568)	0	concatenate[0][0]
dense_63 (Dense)	(None, 10)	15,690	flatten_29[0][0]

Table 3.13: Model Architecture with Connected Layers

Accuracy for Neural Network skip connection is shown below:

Model	Test Accuracy	Test Loss
residualModel	0.876	0.351

Table 3.14: Neural Network (skip connection) Model Summary

The residual model (skip connection) accuracy is around 87% with loss of around 0.351.

The below graph shows the accuracy and loss plot of training and validation data:

Accuracy vs loss :

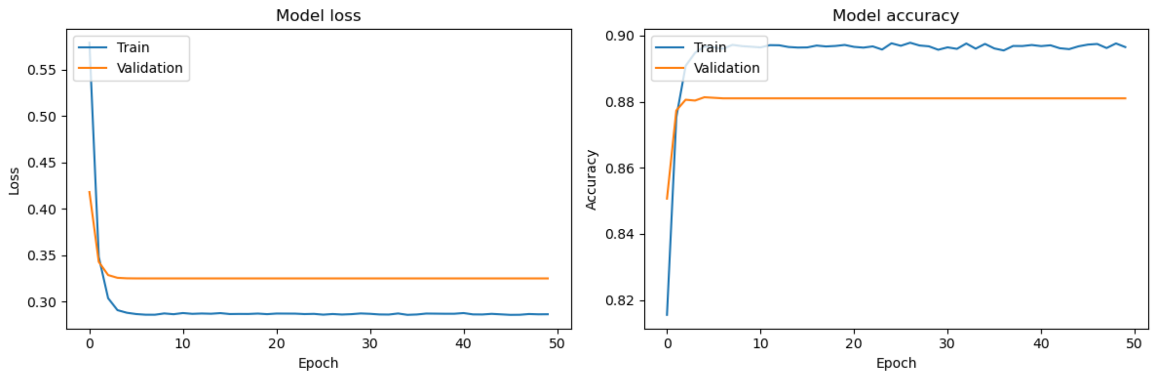


Figure 3.9: Neural Network (skip connection) Line Graph

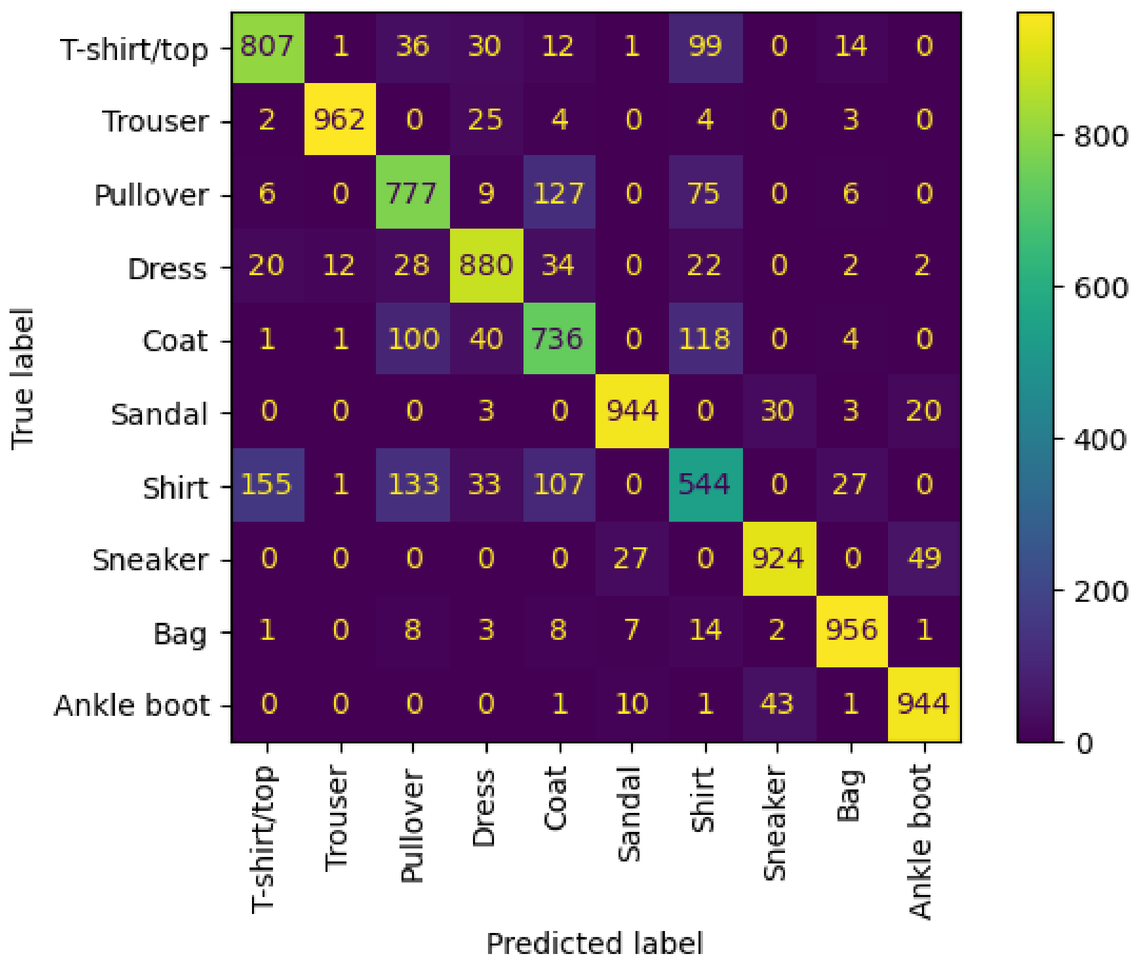
Precision, Recall and F1 Score for Residual Neural Network (skip connection) model is shown below:

Metric	Adam Optimizer		
Metric Type	**Precision**	**Recall**	**F1 Score**
Accuracy			0.88
Macro Avg	0.88	0.88	0.88
Weighted Avg	0.88	0.88	0.88

Table 3.15: Classification Reports for Deep CNN

The performance metrics are around 0.85 for all.

Confusion Matrix for Residual (skip connection) Neural Network is shown below:



True label \ Predicted label	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	807	1	36	30	12	1	99	0	14	0
Trouser	2	962	0	25	4	0	4	0	3	0
Pullover	6	0	777	9	127	0	75	0	6	0
Dress	20	12	28	880	34	0	22	0	2	2
Coat	1	1	100	40	736	0	118	0	4	0
Sandal	0	0	0	3	0	944	0	30	3	20
Shirt	155	1	133	33	107	0	544	0	27	0
Sneaker	0	0	0	0	0	27	0	924	0	49
Bag	1	0	8	3	8	7	14	2	956	1
Ankle boot	0	0	0	0	1	10	1	43	1	944

Figure 3.10: Residual Neural Model Confusion Matrix

From the above confusion matrix, class 'Shirt' seems to be confused quite often with other classes like 'T-shirt/top', 'Pullover' and 'Coat'. Similarly, class 'T-shirt/top' with class 'Shirt' and class 'Coat' with class 'Pullover' and 'Shirt'.

3.5. Comparision between Different Model Structures (Vertical Comparison)

3.5.1. Committee Comparison

The models are divided into two committees based on the optimizer they used. So, it consists of SGD committee and Adam committee. Lets see the model performance.

Committee	Model Name	Individual Accuracy	Committee Accuracy
SGD	shallowModel_sgd	0.871	0.9125
	basicCNNModel_sgd	0.911	
	deepCNNModel_sgd	0.869	
Adam	shallowModel_adam	0.866	0.896
	basicCNNModel_adam	0.906	
	deepCNNModel_adam	0.847	

Table 3.16: Accuracy of Individual Models and Committees

This table summarizes the performance of two committees of models using different optimization methods (SGD and Adam) for training. Here's a breakdown of its elements Above table summarize the performance of two committee which consists of model cluster categorized based on the optimizer user. The model which are using SGD as an optimizing algorithm performs slightly better on average. Models using Adam optimization algorithm has average accuracy of 0.896 slightly lower than the SGD committee average 0.912. This shows that as a SGD committee performs better.

3.5.2. Individual Model Comparison

Model Type	Model Name	Trainable Parameters	Optimizer	Loss			Accuracy		
				Training	Validation	Test	Training	Validation	Test
Shallow NN	shallow-Model_sgd	101,770	SGD	0.048	0.736	0.761	0.982	0.875	0.871
	shallow-Model_adam	101,770	Adam	0.306	0.347	0.370	0.887	0.873	0.866
Basic CNN	basic-CNNModel_sgd	693,962	SGD	6.247	0.876	0.920	1.000	0.915	0.911
	basicCNN-Model_adam	693,962	Adam	0.171	0.251	0.262	0.936	0.910	0.906
Deep CNN	deepCNN-Model_sgd	292,618	SGD	0.051	1.066	1.108	0.985	0.874	0.869
	deepCNN-Model_adam	292,618	Adam	0.362	0.395	0.416	0.864	0.853	0.847
CNN (Dropout)	extraModel	1,608,074	Adam	0.749	0.658	0.588	0.759	0.786	0.811
Residual NN	residualModel	632,698	N/A	0.286	0.325	0.351	0.896	0.881	0.876

Table 3.17: Model Training and Testing Results with Hierarchical Columns for Loss and Accuracy

The table above shows the individual model performance. The column ”Trainable Parameters” indicates the number of parameters that the model created and trained during the model-building process. It can be seen from the table that the accuracy and loss on the validation and test data are closely related. Accuracy on the training data is high for most models, except for the model with dropouts. Among the models, the Basic CNN model performs best compared to the others. The basic model performs well with both optimizing algorithms, achieving 90% accuracy. The Basic CNN consists of only one convolutional layer without any batch normalization or dropouts. Additionally, the model appears to overfit the training data, with 100% and 90% accuracy, respectively. However, the accuracy on the training set is not low enough to clearly classify the model as overfitting.

The Deep CNN model achieves accuracies of 86% and 84% with the SGD and Adam optimizers, respectively. It consists of three convolutional layers and max-pooling, without any batch normalization or dropout. Considering the model's complexity, its performance is good. Models with dropouts, batch normalization, and data augmentation perform competitively, showing a good tradeoff between loss and accuracy. The CNN (Dropout) model is fed with data augmentation to reduce overfitting and other biases in the model. This model demonstrates good performance, achieving around 82

4. Conclusion

The following section contains key findings from the project:

- **Model Architectures:**
  - The shallow neural network, simple CNN, deep CNN, and custom neural network architectures showed varying performance.
  - Advanced architectures like the Deep CNN and Custom Neural Network showed balanced complexity and regularization. Whereas simpler models like the Basic CNN delivered high performance with minimal complexity.
- **Hyperparameter Consistency:**
  - All models were trained using consistent hyperparameters, such as:
    - Learning rate: **0.01**
    - Loss function: **Categorical Crossentropy**
    - Evaluation metric: **Accuracy**
  - Optimizers included **SGD with momentum and NAG**, and **Adam with exponential decay**.
- **Committee Performance:**
  - The SGD Committee achieved slightly higher accuracy (**91.25%**) compared to the Adam Committee (**89.6%**), indicating SGD's effectiveness for the given dataset.
  - Individual model performance within committees highlighted the superior performance of the Basic CNN model under both optimization techniques.
- **Individual Model Analysis:**
  - **Shallow NN:** Performed reasonably well but lagged in accuracy compared to CNN-based architectures.
  - **Basic CNN:** Achieved the best results with 91% accuracy, benefiting from its simplicity but showing signs of overfitting.
  - **Deep CNN:** Delivered competitive results (86-87% accuracy) with more complex feature extraction but fell slightly behind in comparison to simpler models.

  - **Custom CNN (Dropout):** Demonstrated robustness with good tradeoffs between loss and accuracy in spite of data augmentation and regularization techniques like dropout and batch normalization.
  - **Residual NN:** Balanced feature preservation and complex feature extraction but did not outperform simpler CNN architectures.

- **Performance Insights:**
  - Models incorporating regularization (e.g., dropout, batch normalization) effectively reduced overfitting and biases.
  - The Basic CNN achieved high training and test accuracy, but its performance gap suggests possible overfitting.

- **Key Findings:**
  - Simpler architectures (Basic CNN) excelled in performance but risked overfitting.
  - More complex architectures (Deep CNN, Custom CNN) offered better generalization and were more robust.
  - SGD proved slightly superior in optimizing model performance compared to Adam.
  - It is equally true that adam also provide competitive results. Due to its exponential decaying learning rate quickly reaches optimal performance in lower epoch cycle where as to achieve the same accuracy adam is taking more epoch cycle.
  - Considering computational time and power Adam is a better choice for optimizing algorithm.

- **Future Directions:**
  - Incorporating data augmentation, ensemble learning, or additional regularization methods properly could further improve generalization.
  - Further hyperparameter tuning and experimentation with larger datasets could yield better insights and performance improvements.