Fashion-MNIST Image Classification using Deep Learning with Python

The Ultimate Guide for building Neural Networks to Classify Fashion Clothing Apparels using Keras and TensorFlow.

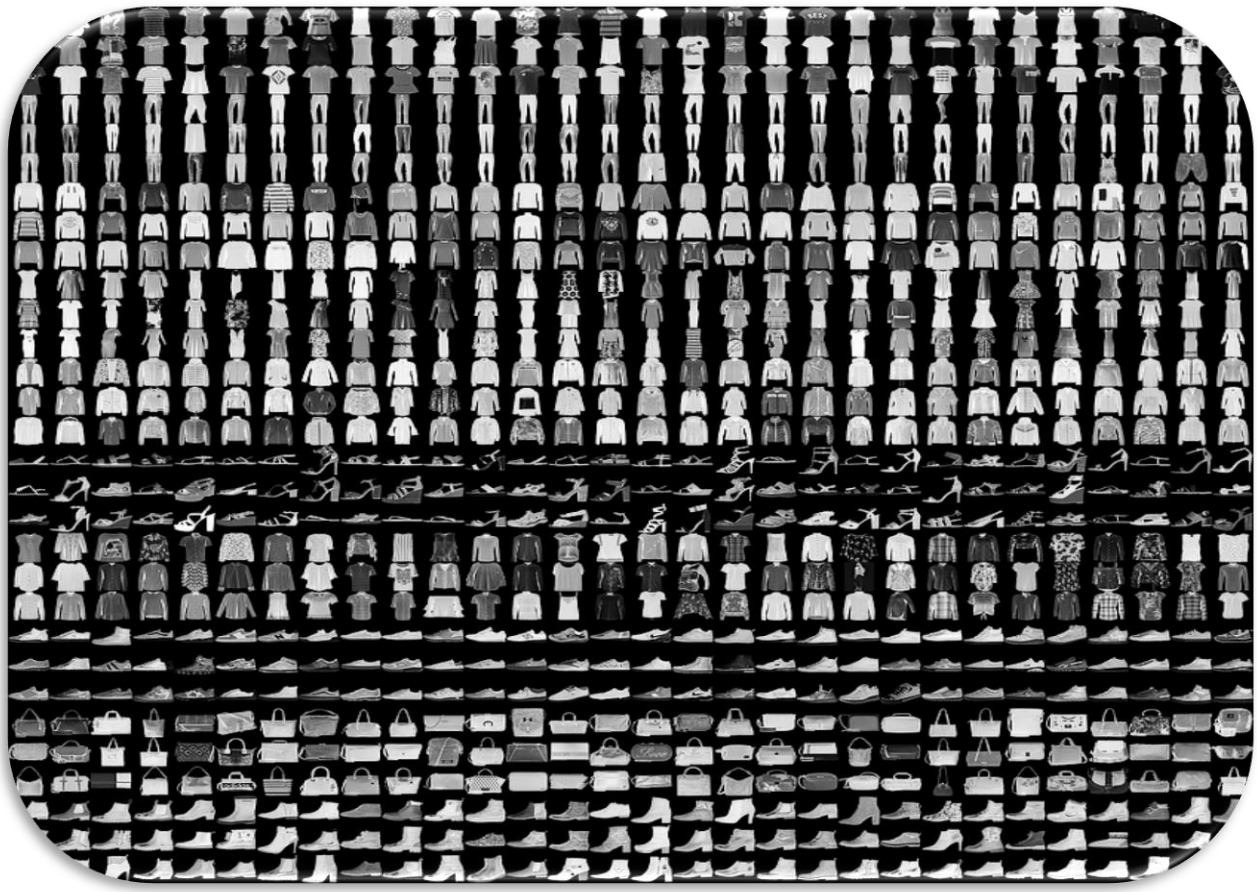


Image Source: Fashion-MNIST samples (by Zalando, MIT License).

Blog By: Chaithanya Vamshi Sai

Student Id: 21152797

Agenda: Steps to Build an Optimised Neural Network Image Classifier Model using Keras, TensorFlow & Hyperparameter Tuning with GridSearchCV

1. Problem Statement
2. Dataset Description
3. Importing Libraries
4. Loading the Fashion MNIST dataset from Keras API
5. Data Visualisation of Images in Training Data
6. Data Pre-processing
7. Model Building
8. Model Evaluation
9. Hyperparameter Tuning of the Neural Network Model using GridSearchCV
10. Model Building: Neural Network with the Best Hyperparameters
11. Model Evaluation: Neural Network with the Best Hyperparameters
12. Data Visualisation of Neural Network Model Loss and Accuracy Results
13. Predictions on Test Data

1. Problem Statement

The objective of this task is we are given a Fashion-MNIST dataset available through Keras API. Using this data, we will build an optimised Image Classifier and demonstrate how we can harness the power of Deep Learning using Keras and TensorFlow.

In this blog, I will walk you through the entire process of implementing a feedforward neural network model on the Fashion-MNIST dataset to classify images of clothing apparel on train data and make predictions on test data using GridSearchCV Hyperparameter tuning technique to achieve the best accuracy and performance.

2. Dataset Description

- The fashion-MNIST dataset consists of 60,000 training images and 10,000 test images of fashion product database images like Shirts, Bags, Sneakers etc.
- Fashion MNIST dataset can be accessed directly from Fashion MNIST TensorFlow using Keras API.
- Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total.
- Each pixel has a single pixel value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255.

3. Importing Libraries

We use Keras, a high-level API to build and train models in TensorFlow.

```
#Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import keras
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense,Flatten
from keras.layers import Dropout, InputLayer
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

4. Loading the Fashion MNIST dataset from Keras API

We use the Fashion MNIST dataset from Keras API using TensorFlow which contains 70,000 grayscale images. It has 10 different categories of fashion clothing apparel as shown below.

Label	Class
0	T-shirt/Top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

After loading the dataset from Keras API, it returns four NumPy arrays.

- X_train and y_train arrays are the training set arrays used to train the neural network model on training data images.
- X_test and y_test arrays are the testing set arrays used to make predictions on the testing data images.

```
#Loading Dataset from Keras API using TensorFlow
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

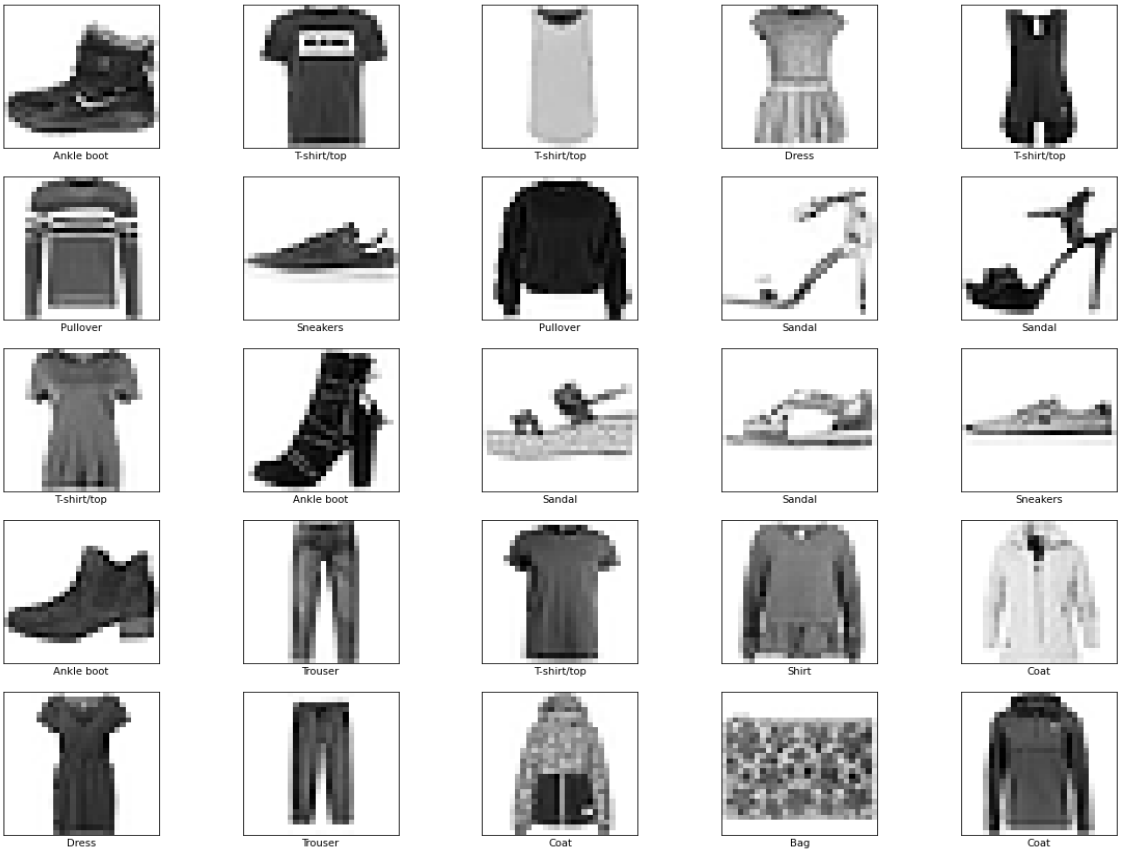
5. Data Exploration

```
# Explore the dataset
# Check the shape and size of X_train, X_test, y_train, y_test
print ("Number of observations in training data: " + str(len(X_train)))
print ("Number of labels in training data: " + str(len(y_train)))
print ("Dimensions of a single image in X_train:" + str(X_train[0].shape))
print("-------------------------------------------------------------\n")
print ("Number of observations in test data: " + str(len(X_test)))
print ("Number of labels in test data: " + str(len(y_test)))
print ("Dimensions of single image in X_test:" + str(X_test[0].shape))
```

- Number of observations in training data: **60000**
- Number of labels in training data: **60000**
- Dimensions of a single image in X_train: **(28, 28)**
- Number of observations in test data: **10000**
- Number of labels in test data: **10000**
- Dimensions of a single image in X_test: **(28, 28)**

6. Data Visualisation of Images in Training Data

Plotting the first 25 images from the training set and displaying the class name below each image.



7. Data Pre-processing

The data must be preprocessed before training the neural network model. If we check the first image in the training set, we will see that the pixel values fall in the range of 0 to 255.

Hence, we apply feature scaling to scale these values to a range of 0 to 1 before feeding them to the neural network model. So, divide the values by 255 and the training set and the testing set must be pre-processed in the same way.

```
#Feature Scaling
X_train = X_train / 255.0
X_test = X_test / 255.0

#Checking Dimension/Shape of Training Data
X_train.shape , y_train.shape
```

8. Model Building

Building the neural network model requires configuring the layers of the model and then compiling the model.

8.1 Neural Network Architecture

- The first layer in the network, tf. Keras. layers. Flatten, transforms the format of the images from a two-dimensional array of 28 x 28 pixels to a one-dimensional array of 28 x 28 = 784 pixels.
- After the pixels are flattened, the network consists of a sequence of two tf. Keras. layers. Dense layers. These are densely connected, or fully connected neural layers and it has 128 neurons.
- The last layer returns a logits array with a length of 10. Each node contains a score that indicates the current image belongs to one of the 10 classes/categories in the dataset.

```
#Neural Network Architecture
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)])
```

8.2 Compile the Neural Network Model

- **Loss function:** It measures how accurate the model is during training. We want to minimize this function to move the model in the right direction.
- **Optimizer:** This is how the model is updated based on the data it sees and its loss function.
- **Metrics:** Used to monitor the training and testing steps. We use accuracy as an evaluation metric to check how accurately the images are classified.

```
#Complie Neural Network Model
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(
from_logits=True),metrics=['accuracy'])
```

8.3 Model Training

To start training, we use the model.fit method which is called because it "fits" the model to the training data and parameter with epochs = 50.

```
Epoch 45/50
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1059 - accuracy: 0.9596
Epoch 46/50
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1068 - accuracy: 0.9588
Epoch 47/50
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1048 - accuracy: 0.9610
Epoch 48/50
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1050 - accuracy: 0.9602
Epoch 49/50
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1002 - accuracy: 0.9633
Epoch 50/50
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0979 - accuracy: 0.9638
<keras.callbacks.History at 0x7f3a8f63df50>
```

8.4 Model Evaluation

As the model training is completed, the loss and accuracy metrics are displayed.

```
#Model Accuracy Results
print("Results:")
print("----------------------")
scores_train = model.evaluate(X_train, y_train, verbose= 2)
print("Training Accuracy: %.2f%%\n" % (scores_train[1] * 100))
scores_test = model.evaluate(X_test, y_test, verbose= 2)
print("Testing Accuracy: %.2f%%\n" % (scores_test[1] * 100))
```

Results:

1875/1875 - 2s - loss: 0.0884 - accuracy: 0.9678 - 2s/epoch - 1ms/step

Training Accuracy: 96.78%

313/313 - 1s - loss: 0.4963 - accuracy: 0.8908 - 623ms/epoch - 2ms/step

Testing Accuracy: 89.08%

We can observe that the accuracy of the Training dataset is **96.78%** and the testing dataset is **89.08%** which depicts the test dataset's accuracy is less than the accuracy on the training dataset. This gap between training accuracy and test accuracy is called **Overfitting**.

Overfitting happens when a deep learning model performs worse on new previously unseen data than it does on the training data.

Most of the deep learning models tend to be good at fitting to the training data, but the real challenge is **Generalization**, not fitting.

Hence, to counter overfitting, we use different strategies and one of the techniques we use in this project is Hyperparameter tuning using **GridSearchCV**.

9. Hyperparameter Tuning of the Neural Network Model using GridSearchCV

Hyperparameter Tuning

The process of selecting the right set of hyperparameters for the ML/DL model is called Hyperparameter tuning.

Hyperparameters are the variables that govern the training process and the topology of a model. These variables remain constant over the training process and directly impact the performance of the model.

Grid Search

Grid Search uses a different combination of all the specified hyperparameters and calculates the performance for each combination and selects the best value for the hyperparameters.

Cross-Validation

In GridSearchCV, along with Grid Search, cross-validation is also performed. Cross-Validation is used while training the model. As we know that before training the model with data, we divide the data into two parts – train data and test data.

In cross-validation, the process divides the train data further into two parts – the train data and the validation data.

The most popular type of Cross-validation is **K-fold Cross-Validation**. It is an iterative process that divides the train data into k partitions.

Hence, Grid Search along with the cross-validation (GridSearchCV) technique takes huge time cumulatively to evaluate the best hyperparameters and build an optimised model.

9.1 Implementation of Hyperparameter Tuning of Neural Network Model using GridSearchCV

1. Hyperparameter Tuning "**Epochs**"
2. Hyperparameter Tuning "**Batch Size**"
3. Hyperparameter Tuning "**Learning Rate**" and "**Dropout Rate**"
4. Hyperparameter Tuning "**Activation Function**" and "**Kernel Initializer**"
5. Hyperparameter Tuning "**Hidden Layer Neuron 1**" and "**Hidden Layer Neuron 2**"
6. Hyperparameter Tuning "**Optimizers**"

The best hyperparameters obtained on the Fashion-MNIST dataset are shown below after GridSearchCV tuning.

Python Implementation of Hyperparameter tuning of the Neural Network Model using GridSearchCV is available on **GitHub**

Hyperparameter	Hyperparameter Values	Best Hyperparameter Values
Epochs	[5,10,50,100]	50
Batch Size	[10,20,50]	50
Learning Rate	[0.00001, 0.0001, 0.001, 0.01, 0.1]	0.001
Dropout Rate	[0.0, 0.1, 0.2, 0.3]	0.1
Activation Function	['softmax', 'relu', 'tanh', 'linear']	tanh
Kernel Initializer	['uniform', 'normal', 'zero']	normal
Hidden Layer Neuron 1	[8, 16,32,64,128]	16
Hidden Layer Neuron 2	[8,16,32]	8
Optimizers	['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']	Nadam

10. Model Building with the Best Hyperparameters Obtained using GridSearchCV

10.1 Splitting Dataset into Training & Validation set

```
# Train Test Split the Training Data to 80% and Validation Data to 20%
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train,X_val,y_train,y_val = train_test_split(X_train,y_train,test_size=0.2,random_state= 100)

# shape of training and validation set
(X_train.shape, y_train.shape), (X_val.shape, y_val.shape)

Output:

(((48000, 28, 28), (48000,)), ((12000, 28, 28), (12000,)))
```

10.2 Neural Network Architecture

```
# number of hidden layers and hidden neurons
# Applying hyperparameters obtained using GridSearch CV
# define hidden layers and neuron in each layer
number_of_hidden_layers = 2
neuron_hidden_layer_1 = 16
neuron_hidden_layer_2 = 8

# defining the Neural network architecture of the model
model_final = Sequential()
model_final.add(Flatten(input_shape=(28, 28)))
model_final.add(Dense(units=neuron_hidden_layer_1, kernel_initializer = 'normal',
activation='tanh'))
model_final.add(Dropout(0.1))
model_final.add(Dense(units=neuron_hidden_layer_2,kernel_initializer = 'normal',
activation='tanh'))
model_final.add(Dropout(0.1))
model_final.add(Dense(units=output_neurons,activation='softmax'))
```

```
[ ]   # summary of the neural network model
      model_final.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 flatten_1 (Flatten)          (None, 784)               0

 dense_2 (Dense)              (None, 16)                12560

 dropout (Dropout)            (None, 16)                0

 dense_3 (Dense)              (None, 8)                 136

 dropout_1 (Dropout)          (None, 8)                 0

 dense_4 (Dense)              (None, 10)                90

=================================================================
Total params: 12,786
Trainable params: 12,786
Non-trainable params: 0
```

10.3 Compile the Neural Network Model

```
# compiling the model
# loss as "sparse_categorical_crossentropy", since we have multi class classification problem
# defining the optimizer as "Nadam" obtained in GridSearhCV
# Evaluation metric as "accuracy"
#define learning rate obtained in GridSearhCV

learn_rate = 0.001
import tensorflow as tf
opt = tf.keras.optimizers.Adam(learning_rate = learn_rate)
model_final.compile(loss='sparse_categorical_crossentropy',optimizer='Nadam',metrics=['accuracy']
```

10.4 Model Training

After applying Hyperparameter tuning using GridSearchCV, we can observe that the accuracy of the Training dataset is **89.83%** and the validation dataset is **86.96%.**

In comparison with the neural network model accuracy obtained without hyperparameter tuning, there was a high gap between training and testing data but after applying the hyperparameter tuning GridSearchCV technique we were able to reduce the overfitting.

Hence, we can say that our Neural Network model with hyperparameter tuning is more generalized and prevented overfitting.

```
Epoch 45/50
960/960 [==============================] - 2s 2ms/step - loss: 0.3821 - accuracy: 0.8679 - val_loss: 0.3725 - val_accuracy: 0.8693
Epoch 46/50
960/960 [==============================] - 2s 2ms/step - loss: 0.3816 - accuracy: 0.8679 - val_loss: 0.3731 - val_accuracy: 0.8698
Epoch 47/50
960/960 [==============================] - 2s 2ms/step - loss: 0.3807 - accuracy: 0.8666 - val_loss: 0.3715 - val_accuracy: 0.8701
Epoch 48/50
960/960 [==============================] - 2s 2ms/step - loss: 0.3780 - accuracy: 0.8684 - val_loss: 0.3798 - val_accuracy: 0.8689
Epoch 49/50
960/960 [==============================] - 2s 2ms/step - loss: 0.3761 - accuracy: 0.8691 - val_loss: 0.3898 - val_accuracy: 0.8632
Epoch 50/50
960/960 [==============================] - 2s 2ms/step - loss: 0.3780 - accuracy: 0.8678 - val_loss: 0.3786 - val_accuracy: 0.8696
```

Results:

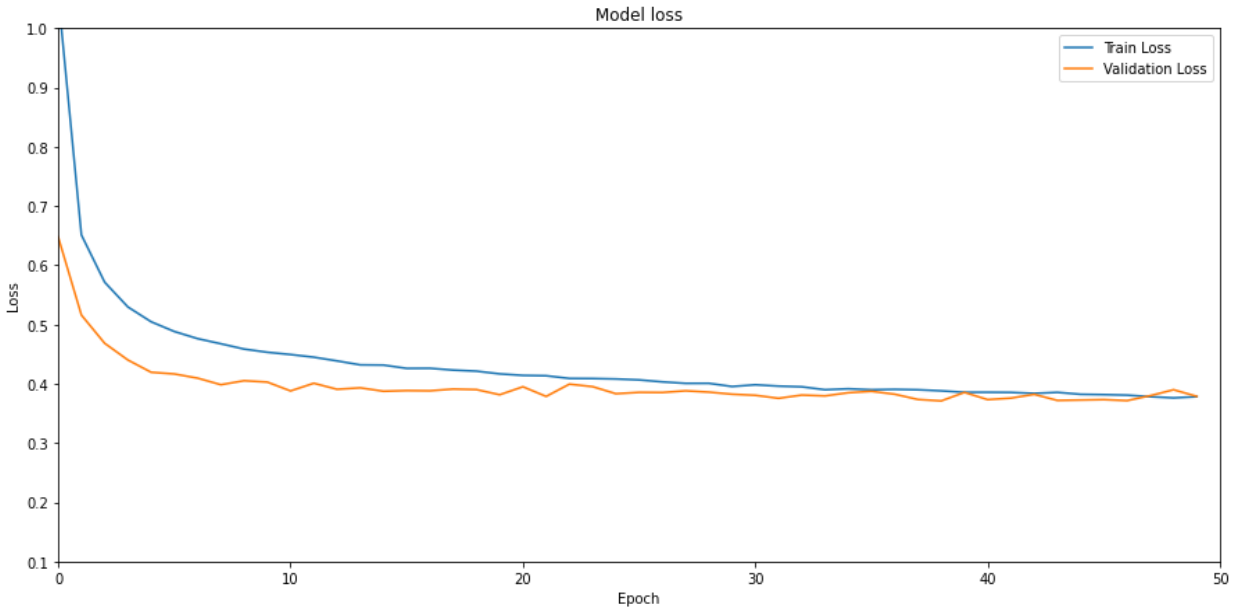
1500/1500 - 2s - loss: 0.2915 - accuracy: 0.8983 - 2s/epoch - 2ms/step

Training Accuracy: 89.83%

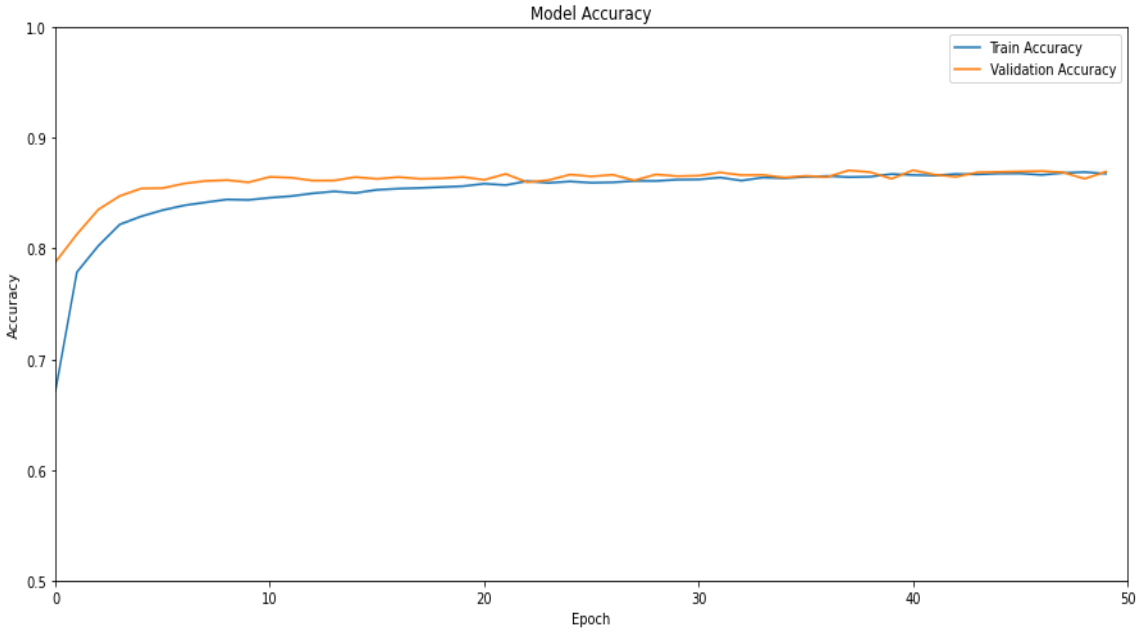
375/375 - 1s - loss: 0.3786 - accuracy: 0.8696 - 593ms/epoch - 2ms/step

Validation Accuracy: 86.96%

11. Neural Network Model Loss on Train and Validation Data



12. Neural Network Model Accuracy on Train and Validation Data



13. Predictions on Test Data

13.1 Overall Model Accuracy Results Summary

With the model trained with the best hyperparameters, we can use it to make predictions on test data. Accuracy given by the Training set is **89.83%** and Accuracy given by the Testing set is **85.66%.** Hence, we can say that the neural network model is more generalized (learns well) and even performs better on testing data.

NN Model	Hyperparameter Tuning - GridSearchCV	Training Accuracy	Testing Accuracy
model	**No**	**96.78%**	**89.08%**
model_final	**Yes**	**89.83%**	**86%**

13.2 Model Evaluation: Classification Report on Test Data

	precision	recall	f1-score	support
T-shirt/top	0.82	0.81	0.81	1000
Trouser	0.98	0.95	0.96	1000
Pullover	0.72	0.81	0.76	1000
Dress	0.84	0.88	0.86	1000
Coat	0.77	0.75	0.76	1000
Sandal	0.97	0.92	0.94	1000
Shirt	0.67	0.58	0.62	1000
Sneakers	0.93	0.94	0.93	1000
Bag	0.94	0.96	0.95	1000
Ankle boot	0.92	0.96	0.94	1000
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

13.3 Plotting Predictions on Test Data Images

Plotting the first 25 test images with their predicted labels, and the true labels.

14. Conclusion

In this blog, we discussed how to approach the image classification problem by implementing a Neural networks model using Keras, TensorFlow and GridSearchCV.

We can explore this work further by trying to improve the accuracy by using advanced Deep Learning algorithms like Convolutional Neural Networks (CNN).

15. References

[1] https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

[2] https://www.tensorflow.org/tutorials/keras/classification

[3] https://www.tensorflow.org/api_docs/python/tf/keras/datasets/fashion_mnist/load_data