Deep Learning

CISC 867

Project 2

By:

Name: Bilal Mohamed Fetouh Mohamed Morsy

email: 22kytb@queensu.ca

ID:20398551

Supervised by:

Dr. Hazem Abbas

Eng. **Asif Mahfuz**

Part 1:

Description of the data:

Fashion MNIST dataset consists of 70000 images divided into 60000 training and 10000 testing samples.
Dataset sample consists of 28x28 grayscale image, associated with a label from 10 classes.
The 10 classes are as follows:

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

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.
Although the dataset is relatively simple, it can be used as the basis for learning and practicing how to develop, evaluate, and use deep

convolutional neural networks for image classification from scratch.

Problem statement

The objective is to identify (predict) different fashion products from the given images using different transferred learning models:

- **Try using CNN model (LeNet5 Model)**
- **Try using CNN model (DenseNet169 Model)**
- **Try using CNN model (AResNet152 V2 Model)**
- **Try using CNN model (VGG-16 Model)**

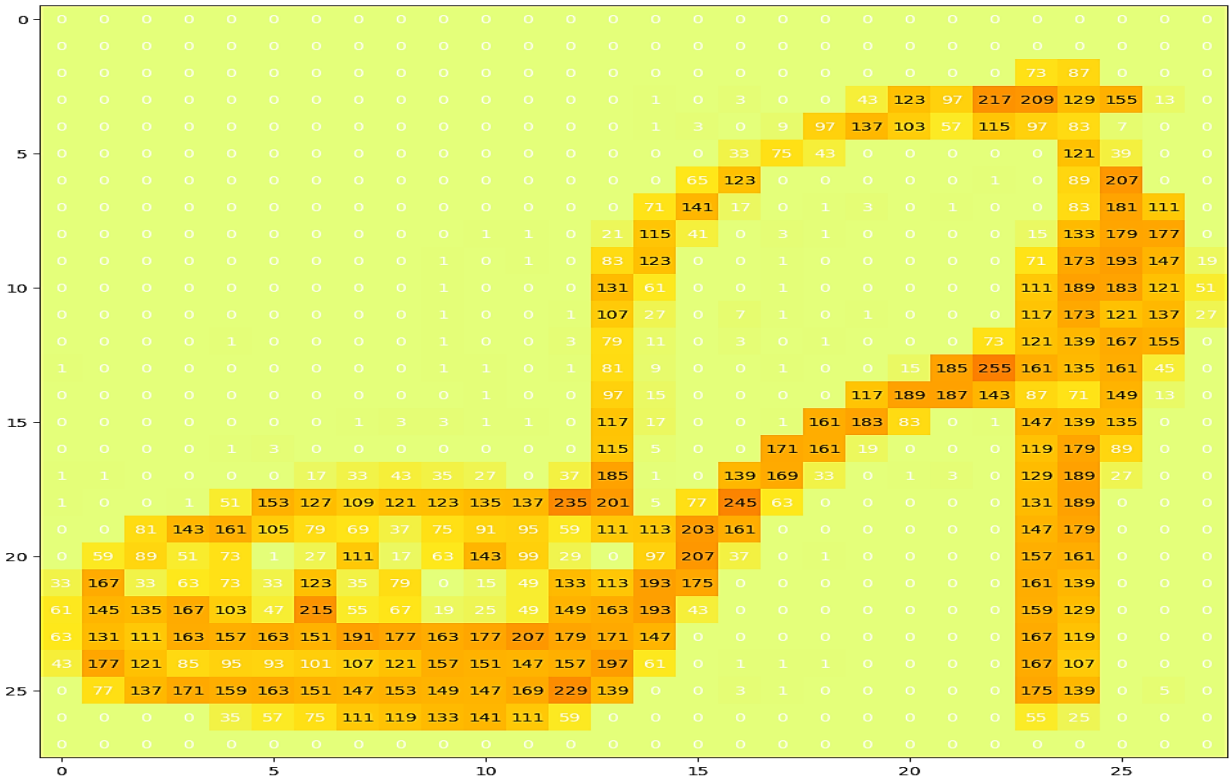
with some preprocessing for each model of them. The target dataset has 10 class labels:

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

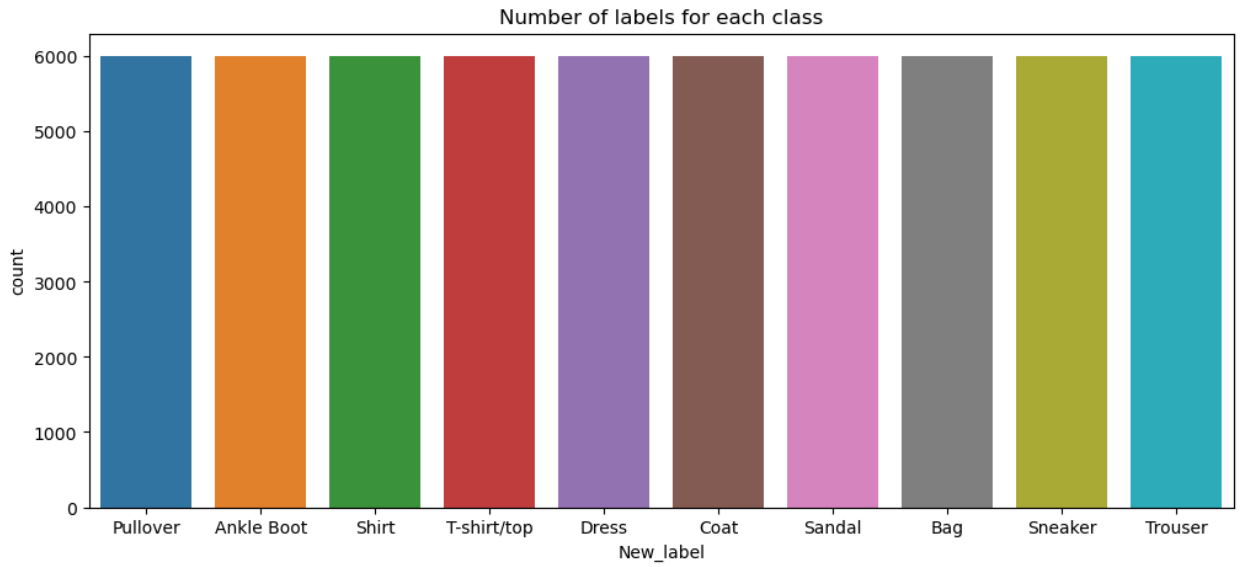
Given the images of the articles, we need to classify them into one of these classes, hence, it is essentially a 'Multi-class Classification' problem. We will be using CNN to come up with a model for this problem and will use "Accuracy" as the performance measure.

Draw some of the images:


Image label is: 9
Image label is: 0
Image label is: 0
Image label is: 3
Image label is: 0
Image label is: 2
Image label is: 7
Image label is: 2
Image label is: 5
Image label is: 5
Image label is: 0
Image label is: 9
Image label is: 5
Image label is: 5
Image label is: 7
Image label is: 9
Image label is: 1
Image label is: 0
Image label is: 6
Image label is: 4
Image label is: 3
Image label is: 1
Image label is: 4
Image label is: 8
Image label is: 4


0
5
10
15
20
25

Encode the labels



Carry out any required preprocessing operations on the data

- **Split the data into train/validation/test data sets**
- **Training data** - used for training the model
- **Validation data** - used for tuning the hyperparameters and evaluate the models (It will be done during hyperparameter tuning)
- **Test data** - used to test the model after the model has gone through initial vetting by the validation set.
- Prepare data in a format that CNN could accept.
- Reshape input data from (28, 28) to (28, 28, 1)
- Converts a class vector (integers) to binary class matrix (One-hot encode the labels).
- Normalization, scale these values to a range of 0 to 1 before feeding them to the neural network model.

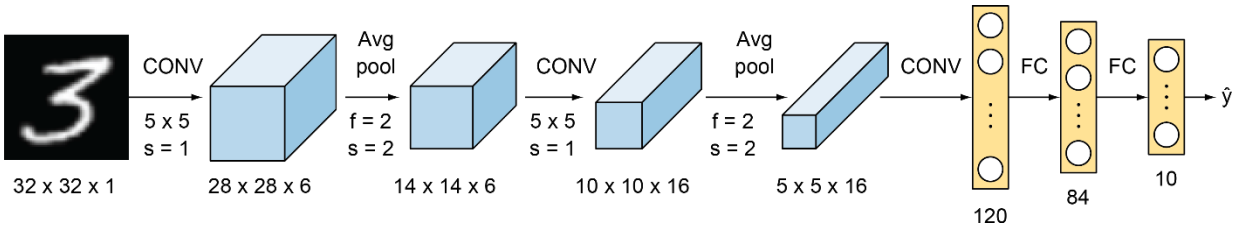
- Normalize the data dimensions so that they are of approximately the same scale, To do so, divide the values by 255.

Part 2: Training a CNN neural network

We will build the following these DL models:

- **Try using CNN model (LeNet5 Model)**
- **Try using CNN model (DenseNet169 Model)**
- **Try using CNN model (AResNet152 V2 Model)**
- **Try using CNN model (VGG-16 Model)**

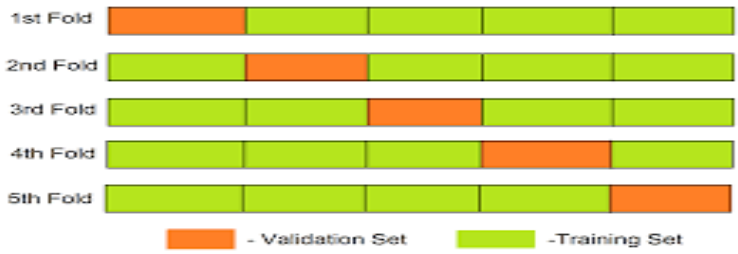
For LeNet5 Model



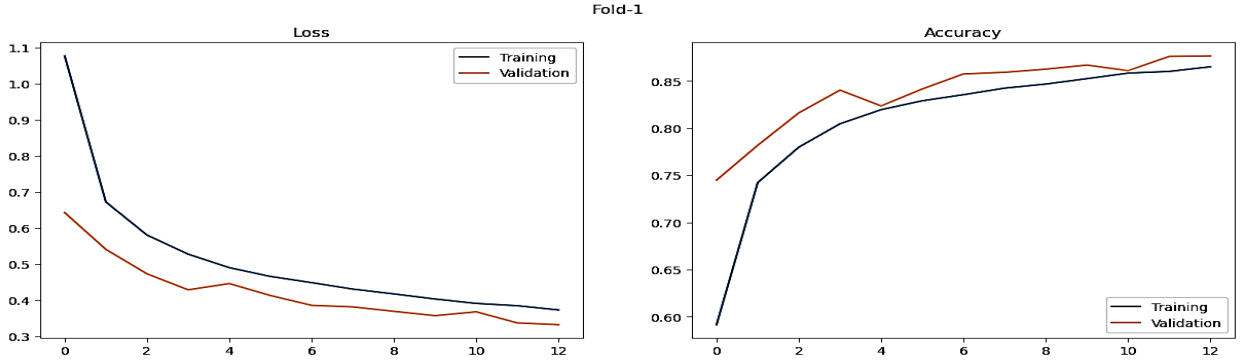
LeNet-5 has following features:

- No. of filters of each CONV layer are: C1 = 6, C3 = 16, C5 = 120 layers.
- Kernel size of each layers is 5 x 5.
- A subsampling POOL layer is added after each subsequent CONV layer. Its receptive field is 2 x 2 & this pooling is average pooling rather than the max pooling.

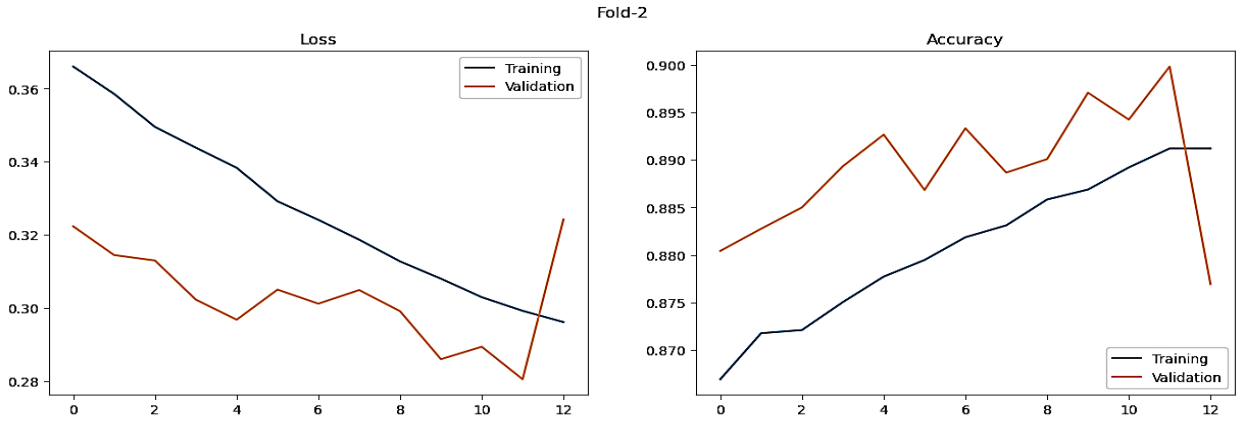
- Activation function used: tanh(x) [But I have used 'relu' for better accuracy]
- Last layer is a Dense Layer with activation func: softmax
- Modify hyperparameters to get to the best performance you can achieve:
    - Loss = 'categorical_crossentropy'
    - optimizer = SGD (learning_rate=0.1, momentum=0.0)
    - metrics = ['accuracy']
- Evaluate the model using 5-fold cross-validation.



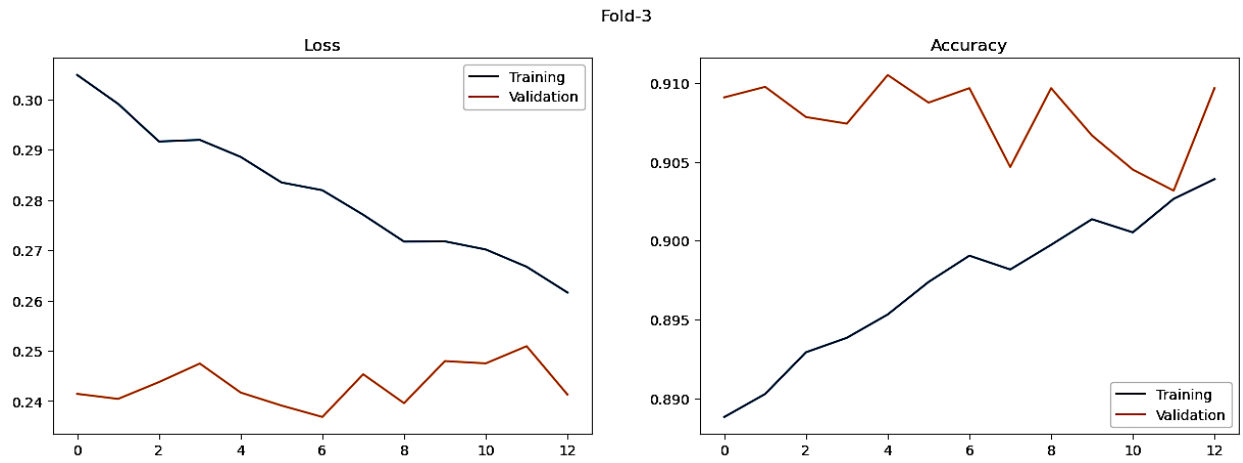
- For each fold we evaluate the accuracy and loss
    - Fold-1: 87.66%
    - Fold-2: 88.21%
    - Fold-3: 90.48%
    - Fold-4: 90.68%
    - Fold-5: 91.29%
- For fold (1)
    - loss: 0.3269 - accuracy: 0.8766



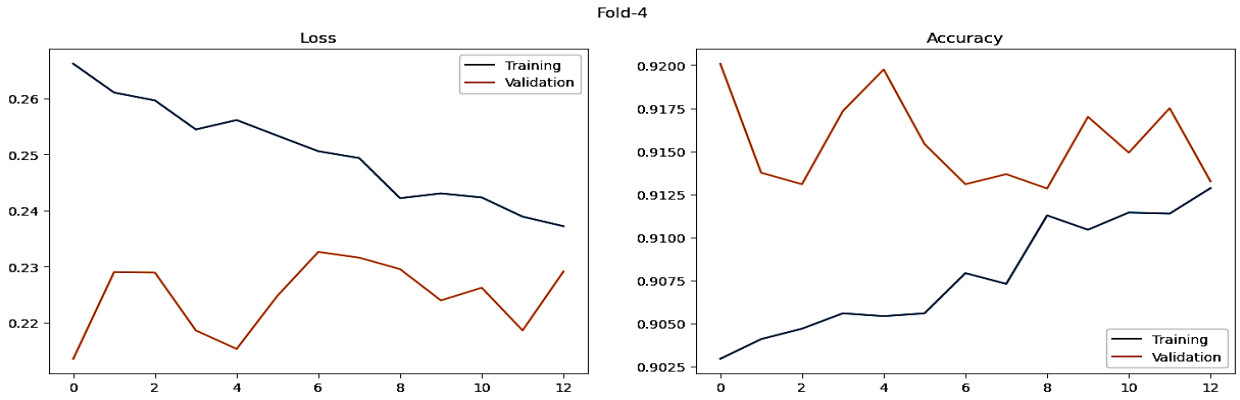
- For Fold (2)
  - loss: 0.3125 - accuracy: 0.8821



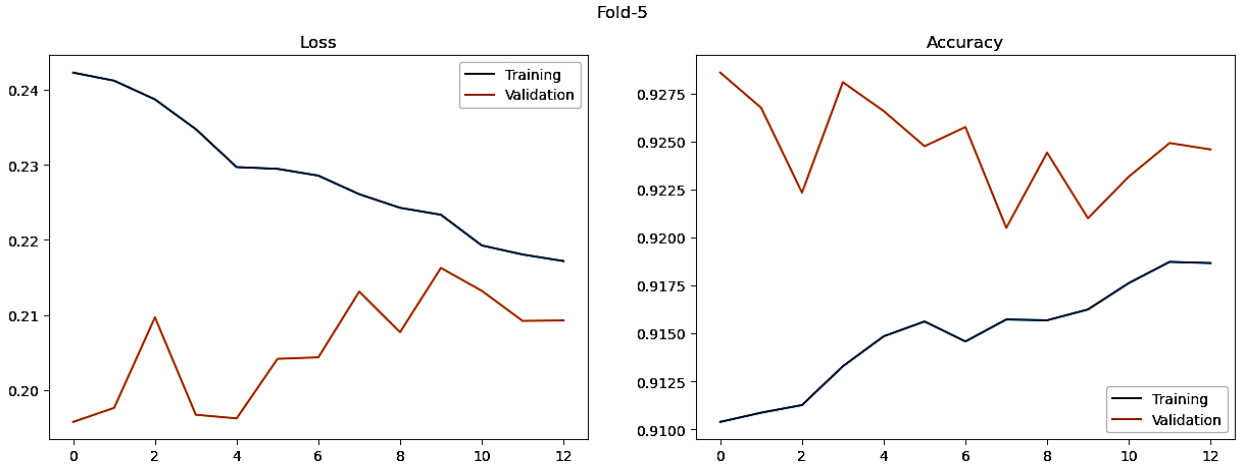
- For Fold (3)
  - loss: 0.2614 - accuracy: 0.9048



- For Fold (4)
  - loss: 0.2536 - accuracy: 0.9068

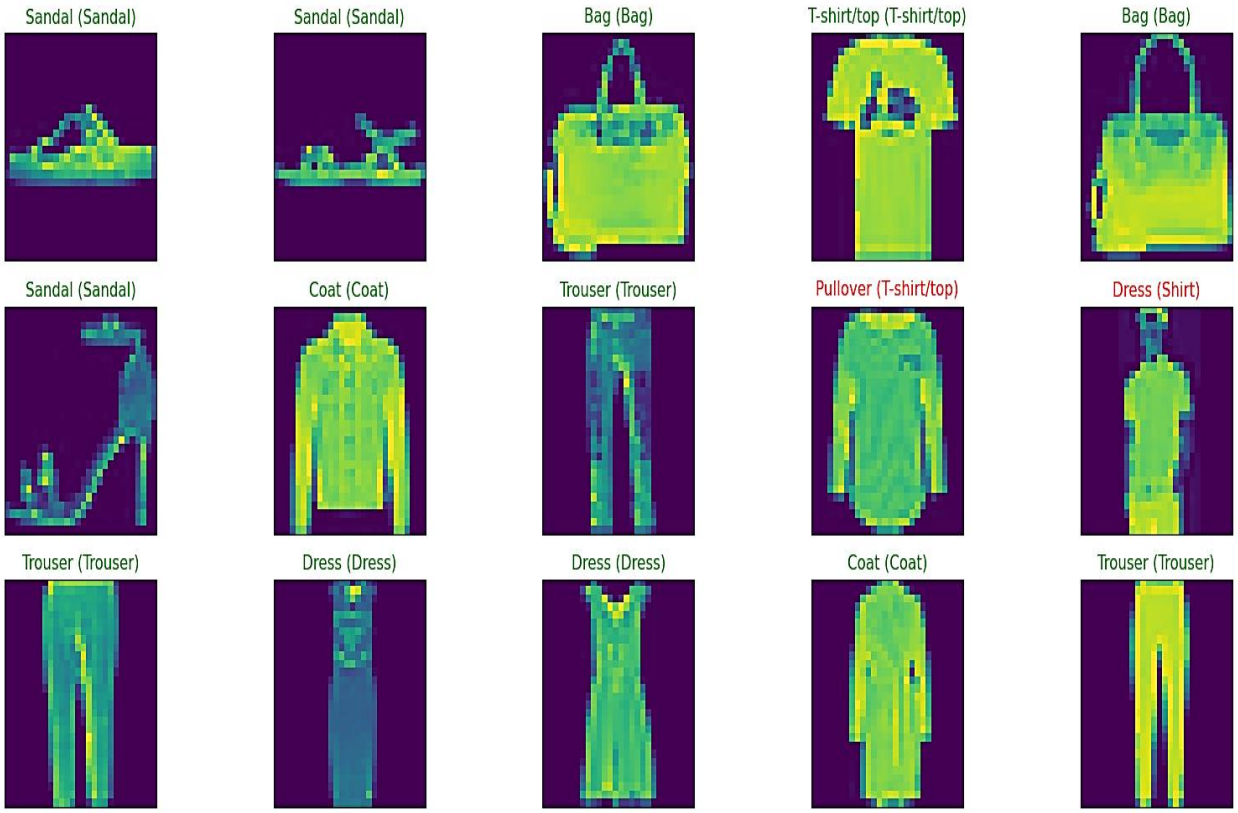


- For Fold (5)
  - loss: 0.2454 - accuracy: 0.9129



The best one is the fold 5 with 91.29% Accuracy

Visualize the classified images (LeNet-5 Model)

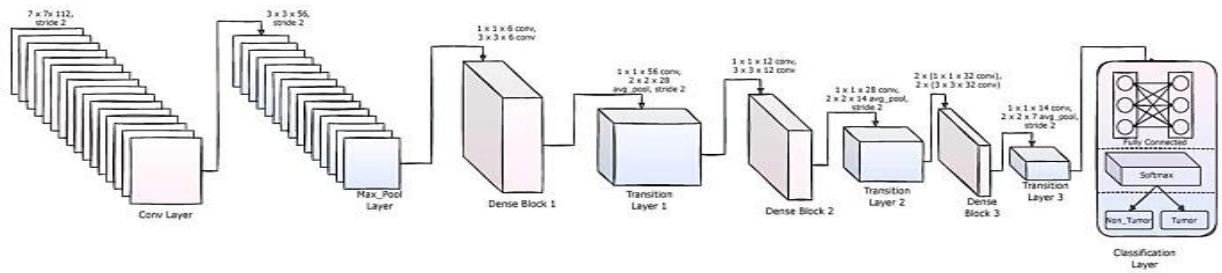


CNN models using Transfer learning

Training this kind of architectures in big datasets is time-consuming. We might need to train for several hours to start getting good results. That is why Keras offers several pre-trained models on the ImageNet dataset. Those models can be used directly for image prediction, image classification, feature extraction, or fine-tuning, among others, without the need of spending long hours of training. Even if the task or the dataset is different, using the pre-trained weights as initialization for the training process provides usually quite better results (as well as less time needed for training and better accuracy) than using a random initialization.

Try using CNN model (DenseNet169 Model)

DenseNet-169 is one of the architectures of the DenseNet family with 169 layers and is a widely used architecture for DL classification tasks. It has far less trainable parameters when compared to its fellow DenseNet architectures with fewer layers. DenseNet-169 and the other DenseNet architectures have the ability to overcome the vanishing gradient problem, have a strong feature propagation strategy, minimize the number of trainable parameters, and encourage the reuse of features, thus making them a family of very reliable DL architectures.

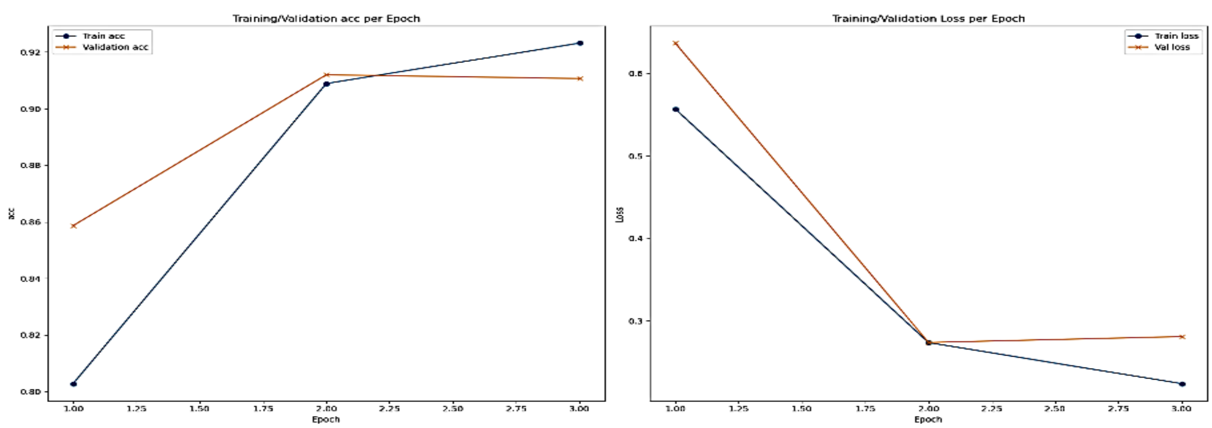


```python
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++DenseNet169+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
from keras import Model
from keras.applications.densenet import DenseNet169
preTrainedModelDenseNet169 = DenseNet169(input_shape =(48, 48, 3), include_top = False, weights='imagenet')

for layer  in range(len(preTrainedModelDenseNet169.layers)-250):
    preTrainedModelDenseNet169.layers[layer].trainable = False

# Build Fine Tuning Layers
d = Flatten()(preTrainedModelDenseNet169.output)
#Fully Connection Layers
d = Dense(1024, activation="relu")(d) # FC1
d = Dropout(0.4)(d) #Dropout to avoid overfitting effect
d = Dense(1024, activation="relu")(d) # FC2
d = Dense(1024, activation="relu")(d) # FC3
d = Dropout(0.2)(d) #Dropout to avoid overfitting effect
d = Dense(512, activation="relu")(d) # FC4
d = Dense(512, activation="relu")(d) # FC5
d = Dropout(0.2)(d) #Dropout to avoid overfitting effect
d = Dense(256, activation="relu")(d) # FC6
d = Dense(256, activation="relu")(d) # FC7
d = Dropout(0.2)(d) #Dropout to avoid overfitting effect
d = Dense(128, activation="relu")(d) # FC8
d = Dense(10,activation="softmax")(d) #output layer
modelDenseNet169 = Model(preTrainedModelDenseNet169.input,d) #concatenation layers
modelDenseNet169.compile(optimizer = "adam", loss="categorical_crossentropy",metrics=['accuracy']) #compile DenseNet169 Model
```

```python
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++DenseNet169+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#fit DenseNet169 Model
historyDenseNet169Model=modelDenseNet169.fit(X_Tr,Y_Tr,validation_data=(X_Val , Y_Val),
                                             epochs=3,
                                             batch_size = 512,
                                             callbacks=tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience = 2),
                                             verbose=1)
ACC_Loss_plot(historyDenseNet169Model) # plot the loss and accuarcy
#Evaluate DenseNet169 Model
print("Evaluate DenseNet169 Model")
modelDenseNet169.evaluate(X_Ts,Y_Ts)
```

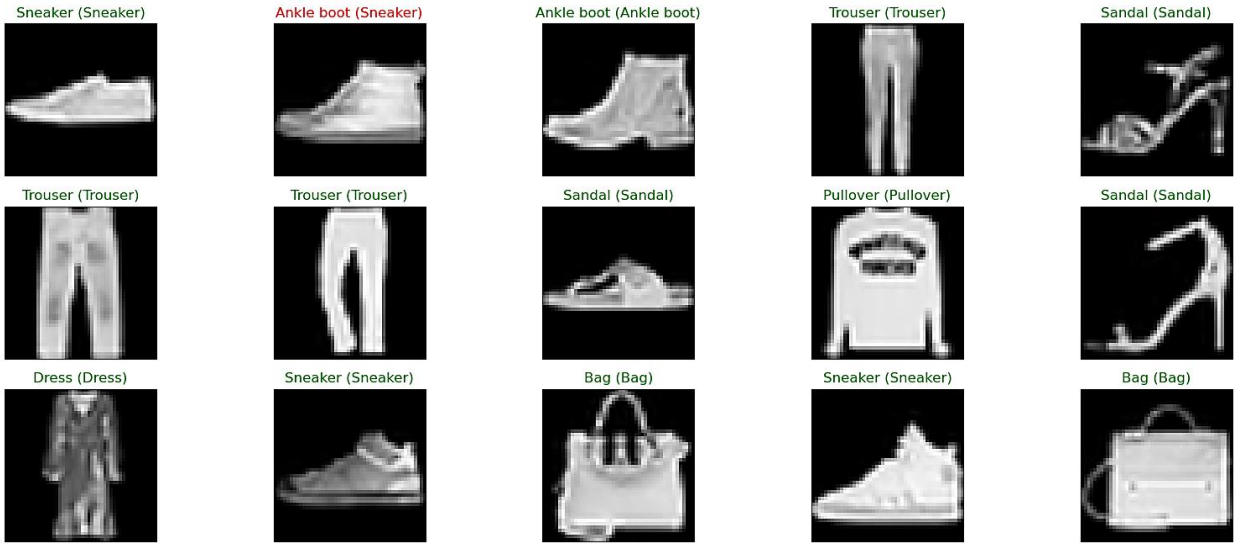



```
Evaluate DenseNet169 Model
313/313 [==============================] - 8s 24ms/step - loss: 0.2805 - accuracy: 0.9146
[0.28046563267707825, 0.9146000146865845]
```

Evaluate DenseNet169 Model

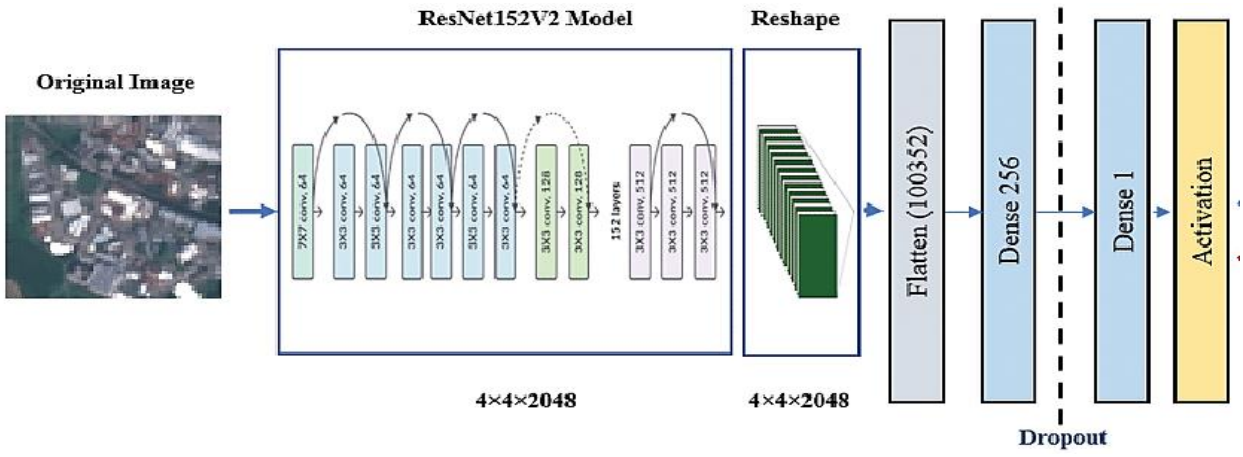
loss: 0.2805 - accuracy: 0.9146

Visualize the classified images (DenseNet169 Model Model)



Try using CNN model (ResNet152 V2 Model)

ResNet152V2 deep model is used as a feature extraction model, as shown in Fig. The model has initial weights because it is a pre-trained model, which can help to gain acceptable accuracy faster than a traditional CNN. The model architecture consists of the ResNet152V2 model followed by a reshape layer, a flatten layer, a dense layer with 128 neurons, a dropout layer, and finally a dense layer with Softmax activation function to classify the image into its corresponding class.



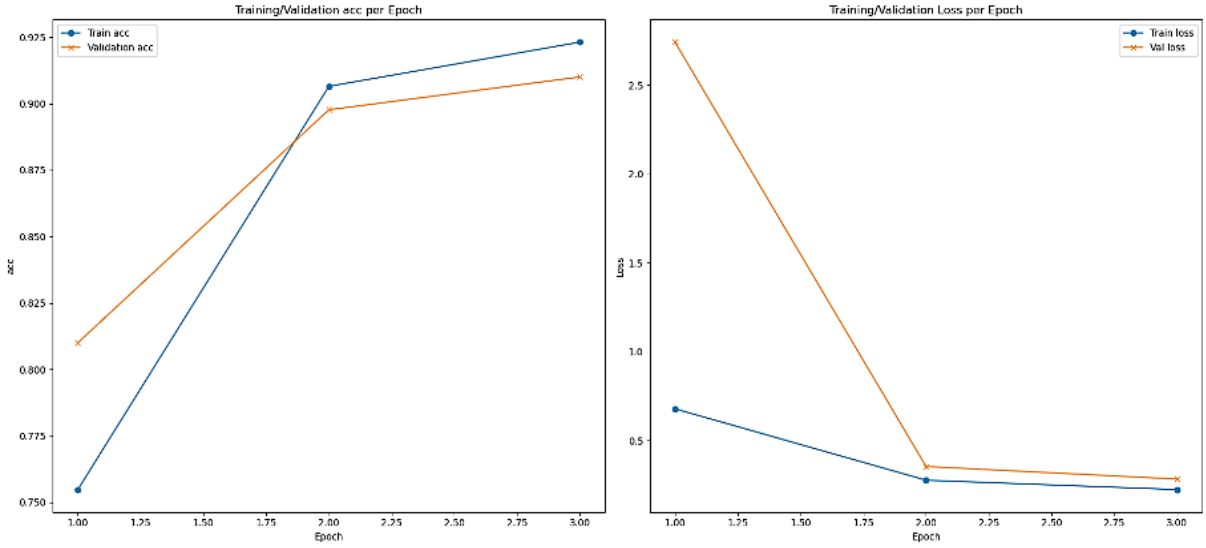
```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ResNet152 V2 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

from keras.applications.resnet_v2 import ResNet152V2
preTrainedModelResNet152V2  = ResNet152V2 (input_shape = (48, 48, 3), include_top = False, weights="imagenet")

for layer  in range(len(preTrainedModelResNet152V2.layers)-64):
    preTrainedModelResNet152V2.layers[layer].trainable = False

# Build Fine Tuning Layers
r = Flatten()(preTrainedModelResNet152V2.output)
#Fully Connection Layers
r = Dense(1024, activation="relu")(r) # FC1
r = Dense(1024, activation="relu")(r) # FC2
r = Dense(1024, activation="relu")(r) # FC3
r = Dense(1024, activation="relu")(r) # FC4
r = Dropout(0.2)(r) #Dropout to avoid overfitting effect
r = Dense(512, activation="relu")(r) # FC5
r = Dense(512, activation="relu")(r) # FC6
r = Dense(256, activation="relu")(r) # FC7
r = Dense(256, activation="relu")(r) # FC8
r = Dropout(0.2)(r) #Dropout to avoid overfitting effect
r = Dense(10,activation="softmax")(r) #output layer
modelResNet152V2=Model(preTrainedModelResNet152V2.input,r) #concatenation layers
modelResNet152V2.compile(optimizer = "adam", loss="categorical_crossentropy",metrics=['accuracy']) #compile ResNet152V2 Model
```

```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ResNet152 V2 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#fit ResNet152 V2 Model
historyResNet152V2Model=modelResNet152V2.fit(X_Tr,Y_Tr,validation_data=(X_Val , Y_Val) ,
                                             epochs=3 ,
                                             batch_size =512 ,
                                             callbacks=tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience = 2),
                                             verbose=1)
ACC_Loss_plot(historyResNet152V2Model)
#Evaluate ResNet152V2 Model
print("Evaluate ResNet152V2 Model")
modelResNet152V2.evaluate(X_Ts,Y_Ts)
```



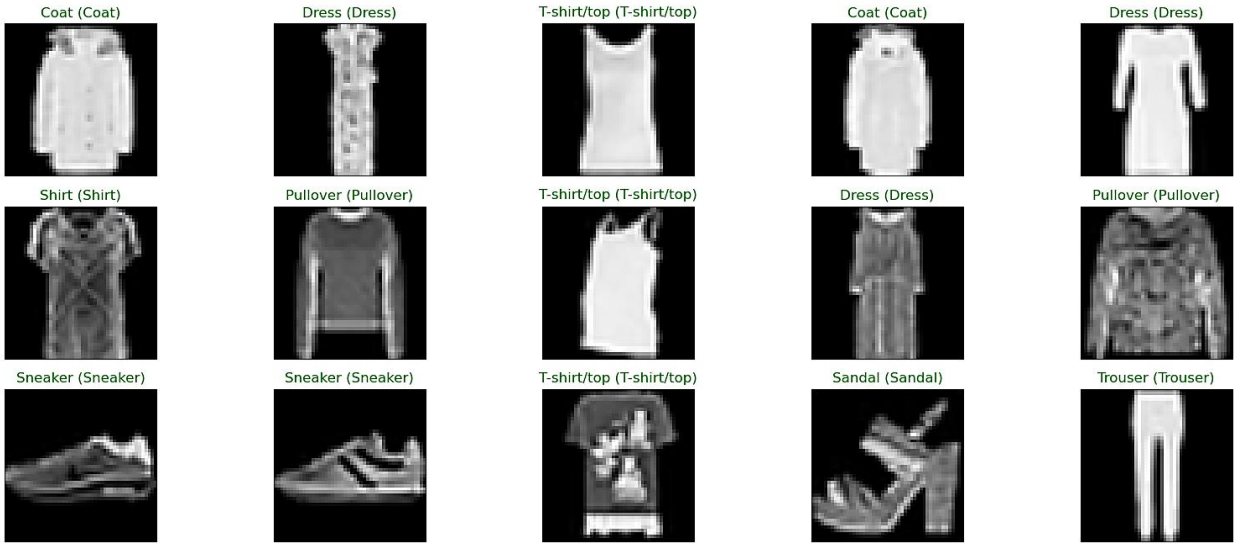

```
Evaluate ResNet152V2 Model
313/313 [==============================] - 12s 38ms/step - loss: 0.2678 - accuracy: 0.9117

[0.2678053379058838, 0.9117000102996826]
```

Evaluate ResNet152V2 Model

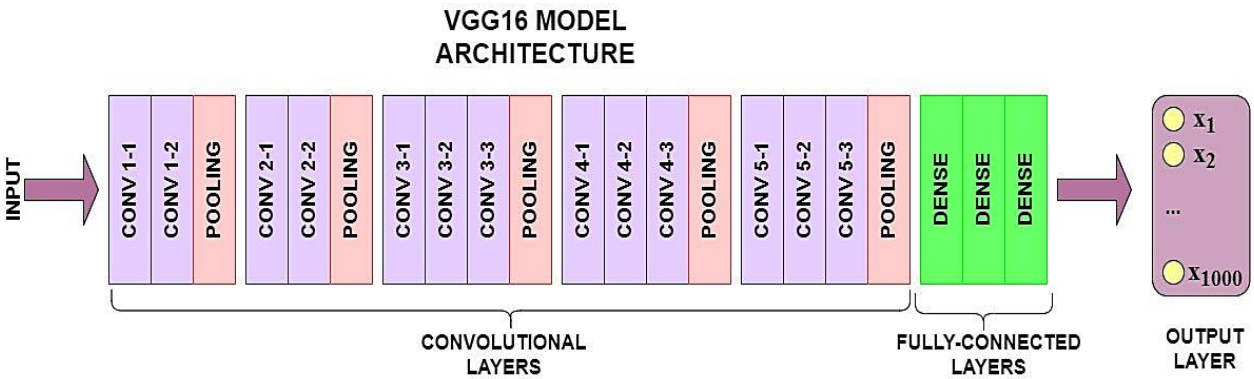
loss: 0.2678 - accuracy: 0.9117

Visualize the classified images (ResNet152V2 Model)



Try using CNN model (VGG-16 Model)

VGG16 is a convolutional neural network trained on a subset of the ImageNet dataset, a collection of over 14 million images belonging to 22,000 categories. K. Simonyan and A. Zisserman proposed this model in the 2015 paper, Very Deep Convolutional Networks for Large-Scale Image Recognition. The VGG16 Model has 16 Convolutional and Max Pooling layers, 3 Dense layers for the Fully-Connected layer, and an output layer of 1,000 nodes.



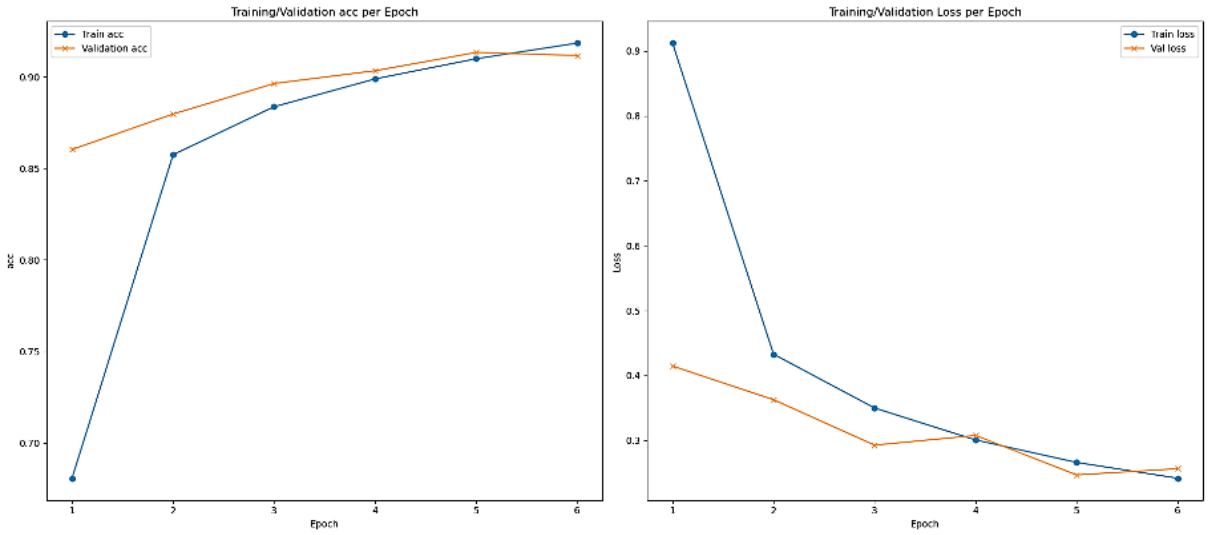
```python
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ VGG-16 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

from keras.applications.vgg16 import VGG16
preTrainedModelVgg16 = VGG16(input_shape = (48, 48, 3), include_top = False, weights='imagenet')

for layer  in range(len(preTrainedModelVgg16.layers)-5):
       preTrainedModelVgg16.layers[layer].trainable = False

# Build Fine Tuning Layers
V = Flatten()(preTrainedModelVgg16.output)
#Fully Connection Layer
V = Dense(1024, activation="relu")(V) # FC1
V = Dense(1024, activation="relu")(V) # FC2
V = Dense(1024, activation="relu")(V) # FC3
V = Dropout(0.5)(V) #Dropout to avoid overfitting effect
V = Dense(512, activation="relu")(V) # FC4
V = Dense(512, activation="relu")(V) # FC5
V = Dropout(0.4)(V) #Dropout to avoid overfitting effect
V = Dense(256, activation="relu")(V) # FC6
V = Dense(64, activation="relu")(V) # FC7
V = Dense(64, activation="relu")(V) # FC8
V = Dropout(0.2)(V) #Dropout to avoid overfitting effect
V = Dense(10,activation="softmax")(V) #output layer
modelVgg16=Model(preTrainedModelVgg16.input,V) #concatenation layers
modelVgg16.compile(optimizer=optRMSProp, loss="categorical_crossentropy",metrics=['accuracy']) #compile Vgg16 Model
```

```python
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ VGG-16 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#fit Vgg16 Model
historyVgg16Model=modelVgg16.fit(X_Tr,Y_Tr,validation_data=(X_Val , Y_Val) ,
                                 epochs=6 ,
                                 batch_size =256 ,
                                 callbacks=tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience = 2),
                                 verbose=1)
ACC_Loss_plot(historyVgg16Model) # plot the loss and accuarcy
#Evaluate Vgg16 Model
print("Evaluate Vgg16 Model")
modelVgg16.evaluate(X_Ts,Y_Ts)
```



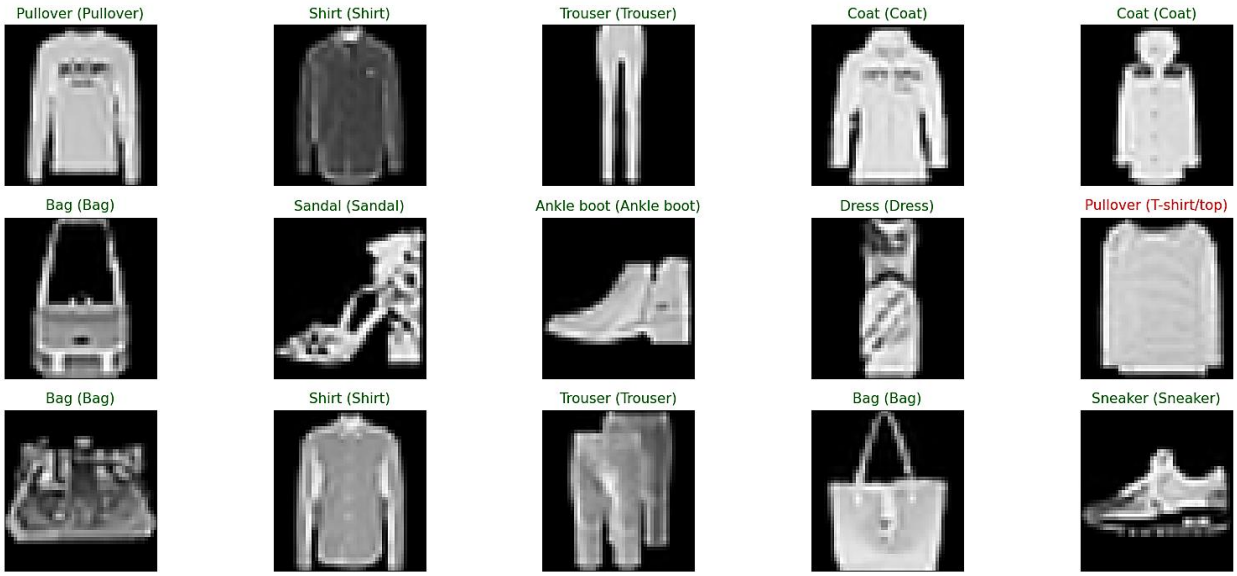

```
Evaluate Vgg16 Model
313/313 [==============================] - 3s 10ms/step - loss: 0.2478 - accuracy: 0.9093

[0.2477997988462448, 0.9093000292778015]
```

Evaluate Vgg16 Model

loss: 0.2478 - accuracy: 0.9093

Visualize the classified images (VGG16 Model)



Conclusion

Model	Evaluation - Loss	Evaluation – Acc
LeNet-5(Fold-5)	24.54%	91.29%
DenseNet169	28.05%	91.46%
ResNet152V2	26.78%	91.17%
VGG16	24.78%	90.93%

- As we see from all the previous models, the LeNet-5 is the best
  - It is one of the two highest evaluation score.
  - It has the least evaluation loss.