Recognize The Fashion-MNIST Digits

Introduction:

In this project we will implement a LeNet-5 network to recognize the Fashion-MNIST digits:

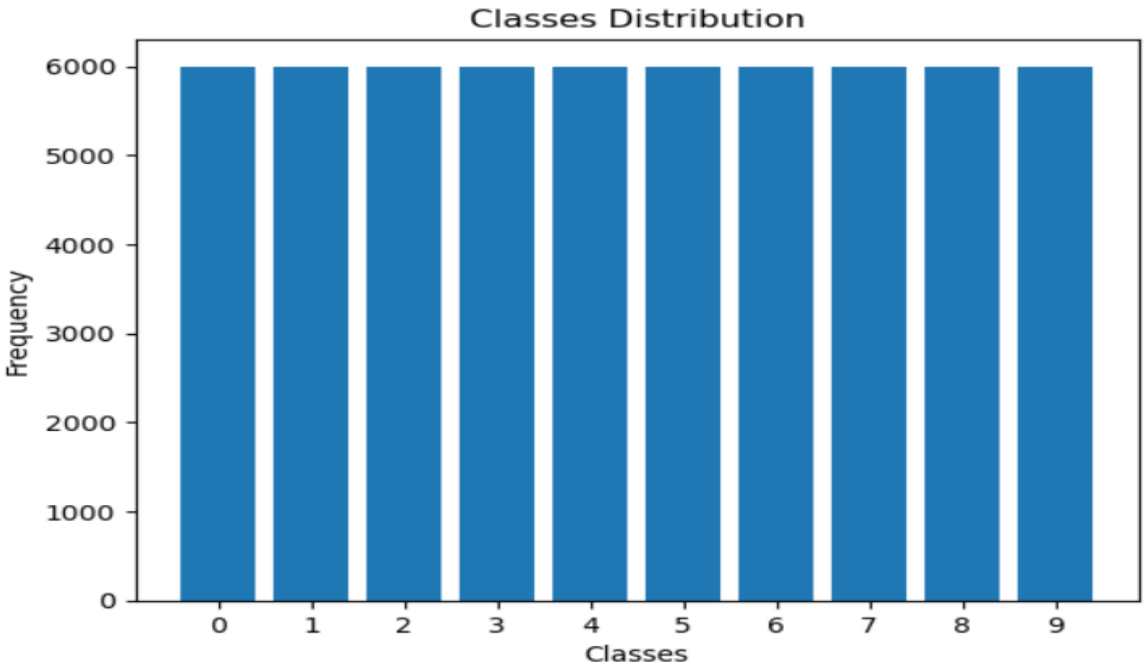
- We will modify hyperparameters to get to the best performance you can achieve.
- Evaluate the model using 5-fold cross-validation.
- We will try to use other three CNN models (using transfer learning) and compare the results with the full trained LeNet-5.

The competition and data link here https://www.kaggle.com/datasets/zalando-research/fashionmnist

Preprocessing And EDA:

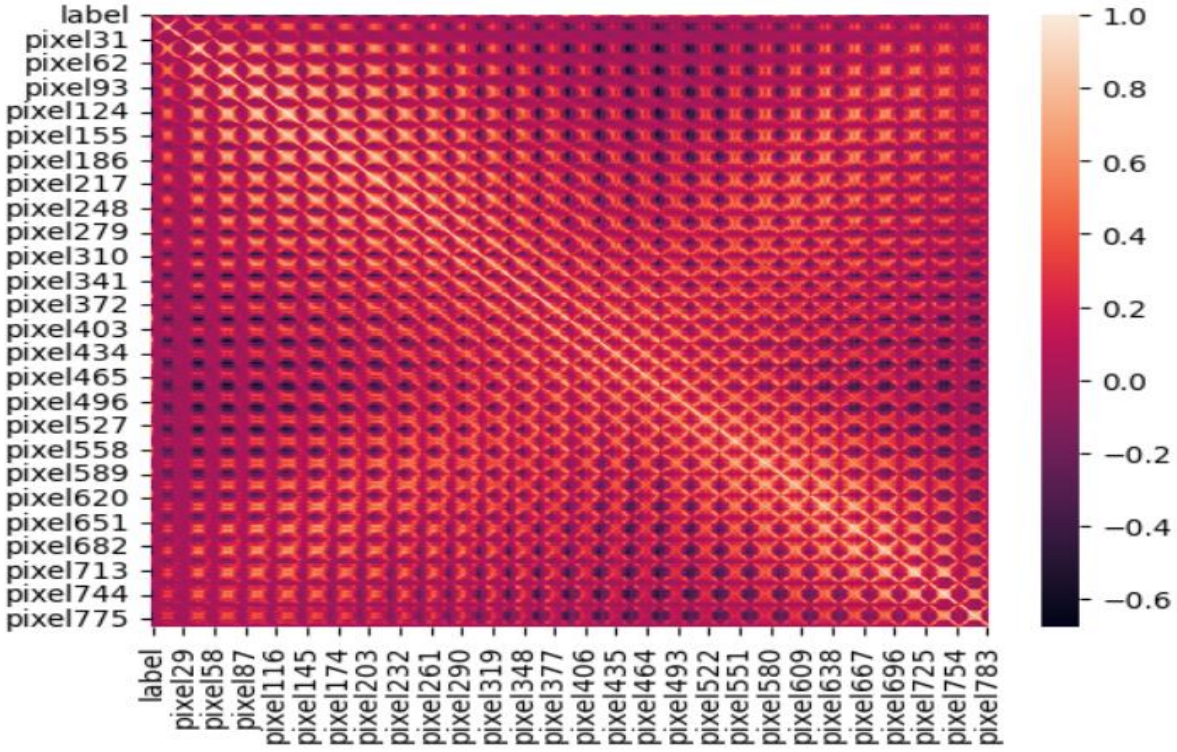
- There are 60,000 grayscale images, That need to be classified into 10 classes.
- There is no any missing value.
- There are some duplicated values, they are 43 images, that we dropped them.
- Here are some samples of these images:

- The classes distribution is balanced, The distribution is:



- The label classes are already encoded (ordinal encoding).
- The correlations between features have a pattern:

```
# correlation analysis
corr = df.corr()
sns.heatmap(corr);
```



- There are some features that have strong correlations and others have weak ones.

Solutions:

We will use different approaches to solve this problem. They are:

1. Building LeNet-5.
2. RESNet152V2 (transfer).
3. Xception(transfer).
4. VGG16(transfer.

1. Building LeNet-5:

- We will build the structure of LeNet-5 and train it:

```
# this function to build the model (LeNet 5 Structure)
def build_model(hyper):
    model = Sequential()
    model.add(Conv2D(filters=6, kernel_size=(5, 5), activation='relu', input_shape=(32, 32, 1)))
    model.add(MaxPool2D(strides=2))
    model.add(Conv2D(filters=16, kernel_size=(5, 5), activation='relu'))
    model.add(MaxPool2D(strides=2))
    model.add(Flatten())

    # values to tune the two hidden layers nodes
    units_1 = hyper.Choice('units_1', values=[128, 256])
    units_2 = hyper.Choice('units_2', values=[32, 64, 128, ])
    model.add(Dense(units=units_1, activation='relu'))
    model.add(Dense(units=units_2, activation='relu'))
    model.add(Dense(10, activation='softmax'))

    # values to tune learning rate
    learning_rate = hyper.Choice('learning_rate',
                                 values=[1e-2, 1e-3, 1e-4])

    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss=SparseCategoricalCrossentropy(),
                  metrics=['accuracy'])

    return model
```

- We got very good accuracy and loss on training and validation with hyperparameters tuning:

```
In [44]: model_tuner = find_best(build_model, X_train, y_train, X_val, y_val)

         Trial 16 Complete [00h 00m 09s]
         val_accuracy: 0.8430609703063965

         Best val_accuracy So Far: 0.883824348449707
         Total elapsed time: 00h 02m 48s
```

```
In [46]: best_hypers = model_tuner.get_best_hyperparameters()[0]
         # print the best hyperparameters
         print(f"Best hyperparameters: {best_hypers.values}")

         Best hyperparameters: {'units_1': 256, 'units_2': 32, 'learning_rate': 0.001, 'tuner/epochs': 2, 'tuner/initial_epoch': 0, 'tuner/bracket': 4, 'tuner/round': 0}
```

- As we can see the best hyperparameters are:
  - No. units of first hidden layer: 256
  - No. units of second hidden layer: 32
  - learning_rate: 0.001

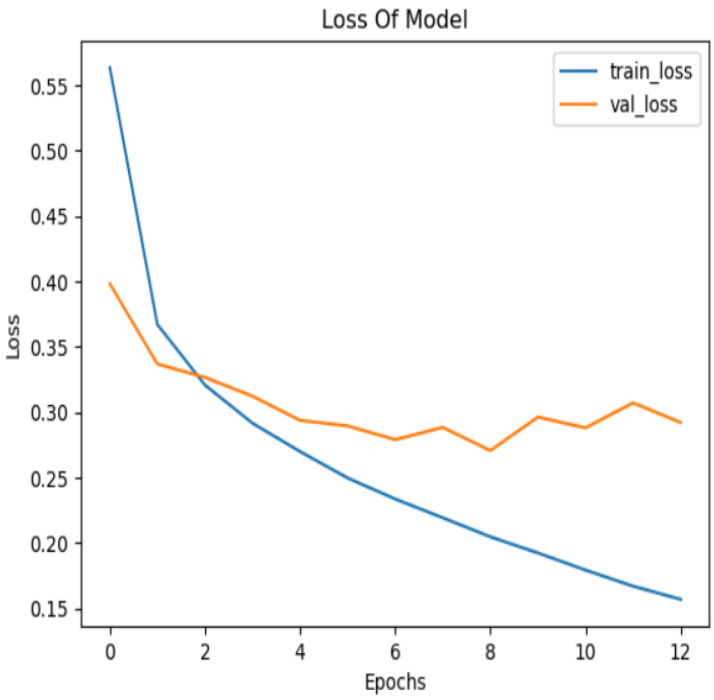
- Check the model performace in general(Generalization of model)

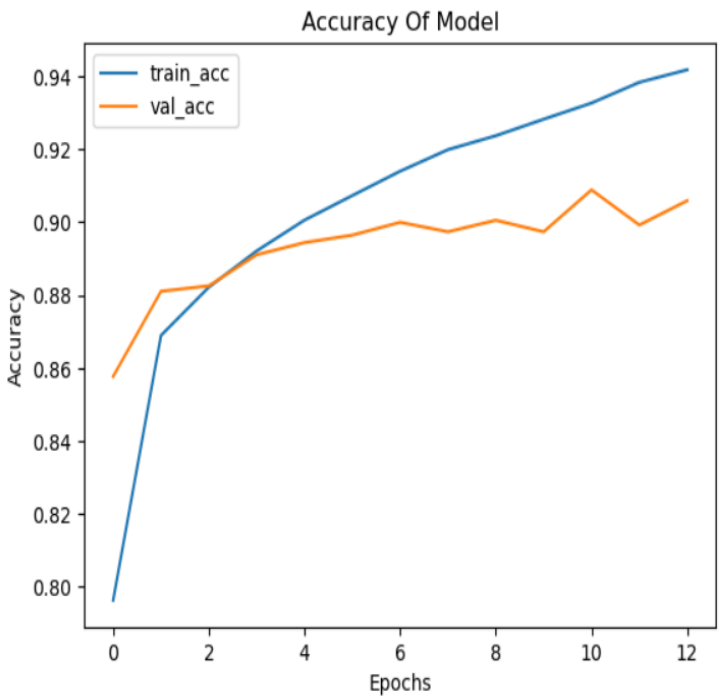
```
# evaluate the model on test data
model.evaluate(X_test, y_test)
```

```
188/188 [==============================] - 1s 3ms/step - loss: 0.3125 - accuracy: 0.8983
[0.31249210238456726, 0.8982654809951782]
```

- As we can see the model generalizes well on test data:
  - loss: 0.31
  - accuracy: 0.898

- We visualize the loss and accuracy curves on training and testing, they look very good:



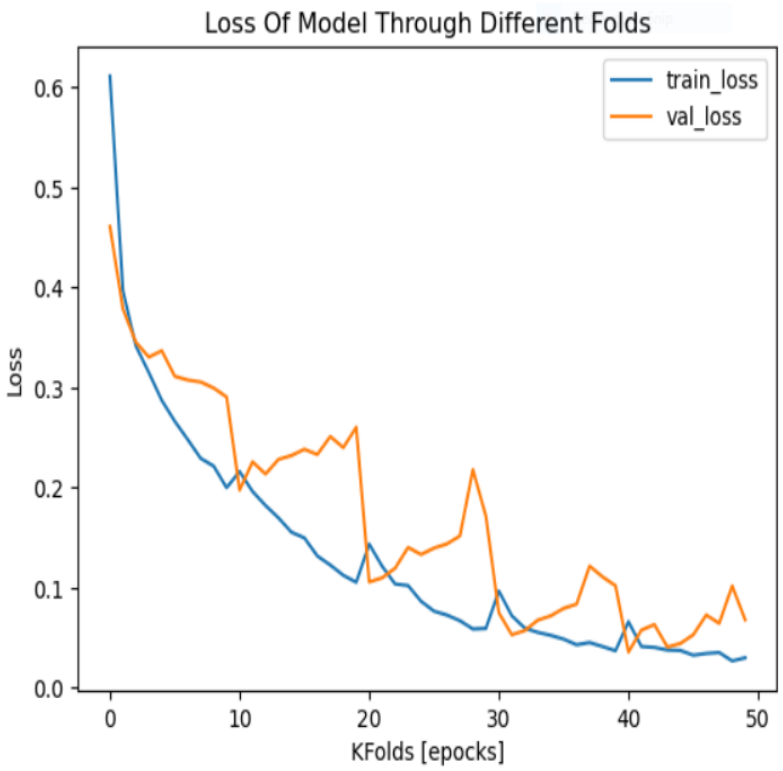


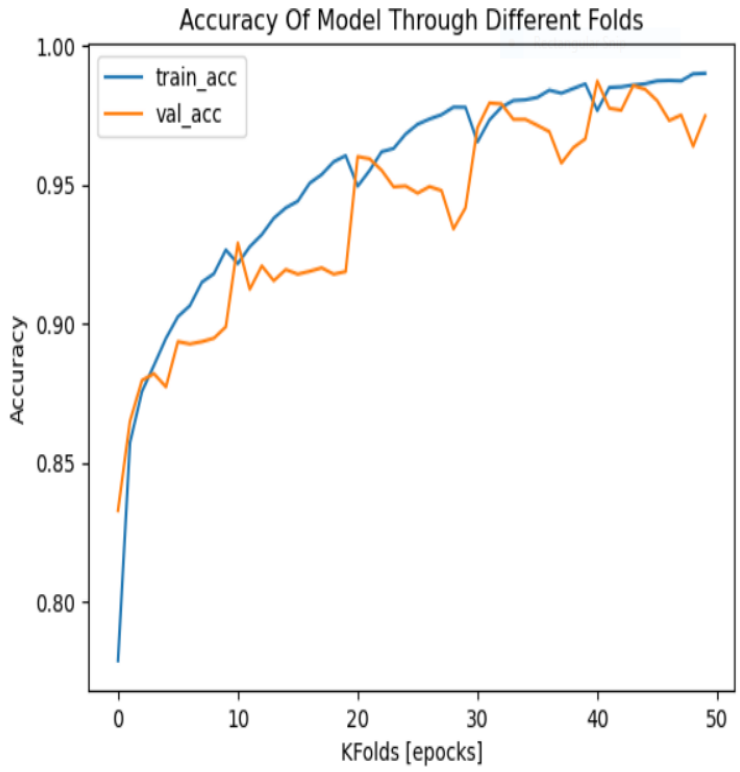
- We used Cross-Validation to evaluate the LeNet-5 too, we got very good results:

```
In [118]: print(f"the average losses through validation folds: {round(np.mean(losses), 2)}\n")
          print(f"the average accuracies through validation folds: {round(np.mean(accs), 2)}\n")

          the average losses through validation folds: 0.18

          the average accuracies through validation folds: 0.94
```

- As we can see the model generalizes well on Cross-Validation Folds too.

- We visualized the loss and accuracy of Cross-Validation evaluation:





- As we can notice the LeNet-5 generalizes well on Cross-Validation.

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

- I think LeNet-5 improves the accuracy.
- explanation:
  - A hand-built model(LeNet-5 in our case) involves designing and training a neural network from scratch, where the architecture, hyperparameters, and weights are all learned on the specific task and dataset at hand. This approach can be effective when the dataset is large and diverse, and when the task at hand is well-defined and specific. Hand-built models also offer greater flexibility and control over the network architecture and training process, which can be useful for specialized applications.

2. RESNet152V2 (transfer):

- We imported the RESNet15v2 as a transfer algorithm to train our data on it we got a little bit good results:

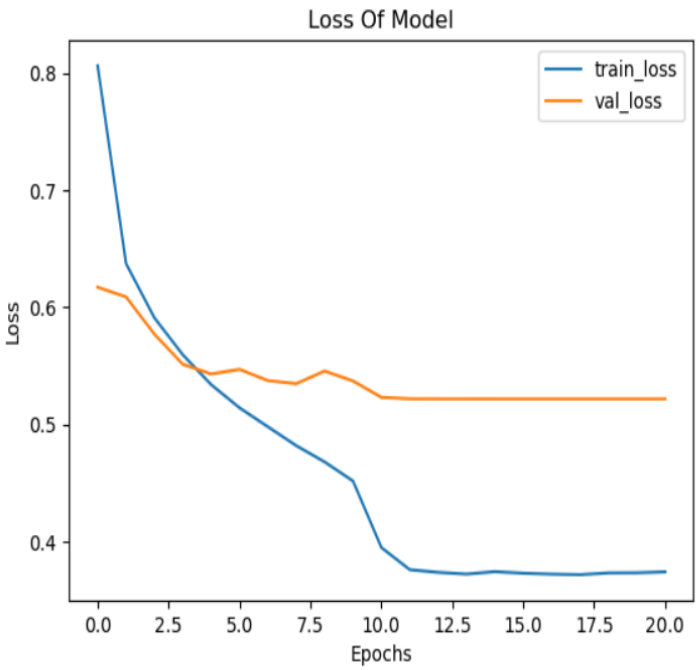
- evalute the model to check its generalization

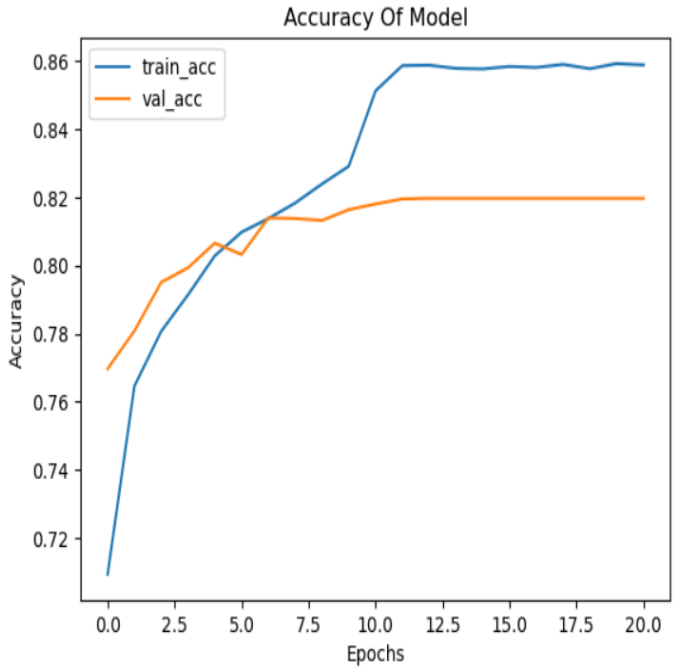
```
# evalute the model on test data
model.evaluate(X_test, y_test)
```

```
188/188 [==============================] - 7s 36ms/step - loss: 0.5628 - accuracy: 0.8074

[0.5627942085266113, 0.8073715567588806]
```

- As we can see the model achieved good accuracy and loss(it generalizes well):
  - loss: 0.5628
  - accuracy: 0.8074
- But the "LeNet 5" was better than this model.

- When we visualized the curve of loss and accuracy on training and validation:





- As we can notice the RESNet generalizes well, but LeNet-5 is better than it.

3. Xception (transfer):

- We imported the Xception as a transfer algorithm to train our data on it we got bad results:

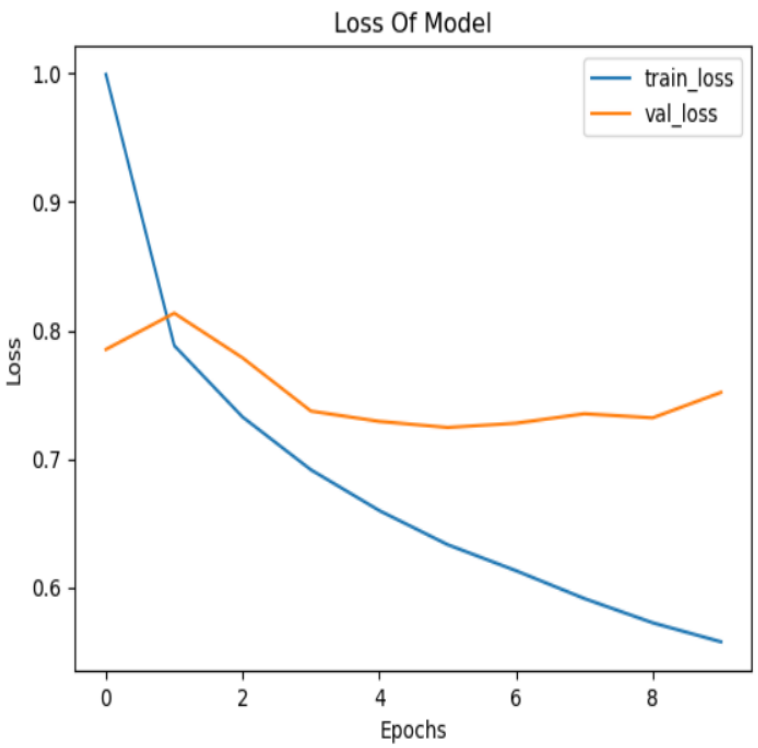
- evalute the model to check its generalization

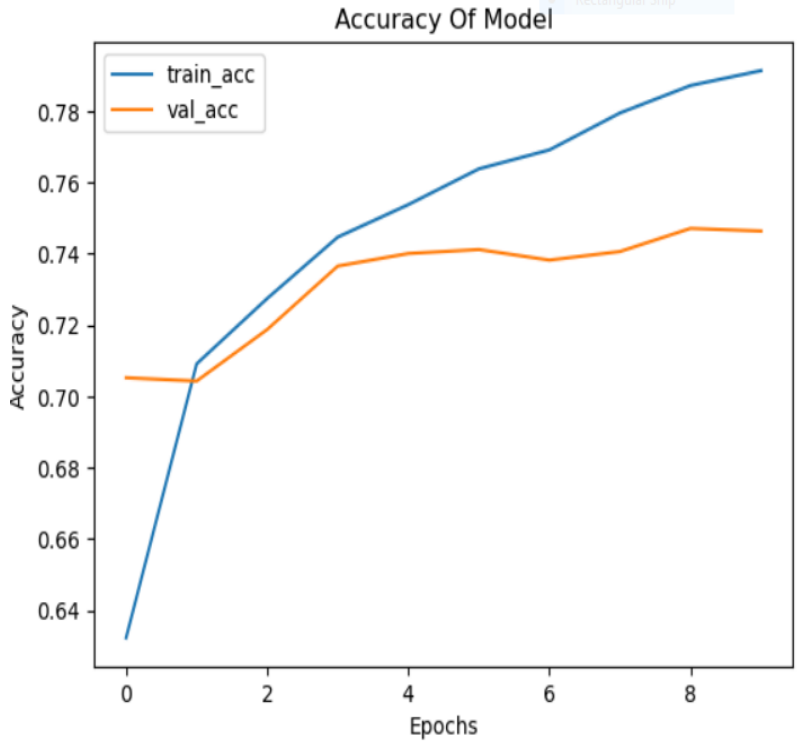
```
# evalute the model on test data
model.evaluate(X_test, y_test)
```

```
188/188 [==============================] - 3s 14ms/step - loss: 0.7386 - accuracy: 0.7470
[0.7386459112167358, 0.7469980120658875]
```

- Unfortunaitely the model performace is very bad, where the accuracy and loss:
  - loss: 0.7386
  - accuracy: 0.7470
- The "LeNet 5" and "RESNet152V2" are better than this model.

- When we visualized the curve of loss and accuracy on training and validation:





- As we can notice the Xception model is not doing well on validation and testing, it is worse than LeNet-5 and RESNet.

4. VGG16 (transfer):

- We imported the VGG16 as a transfer algorithm to train our data on it we got very good results:

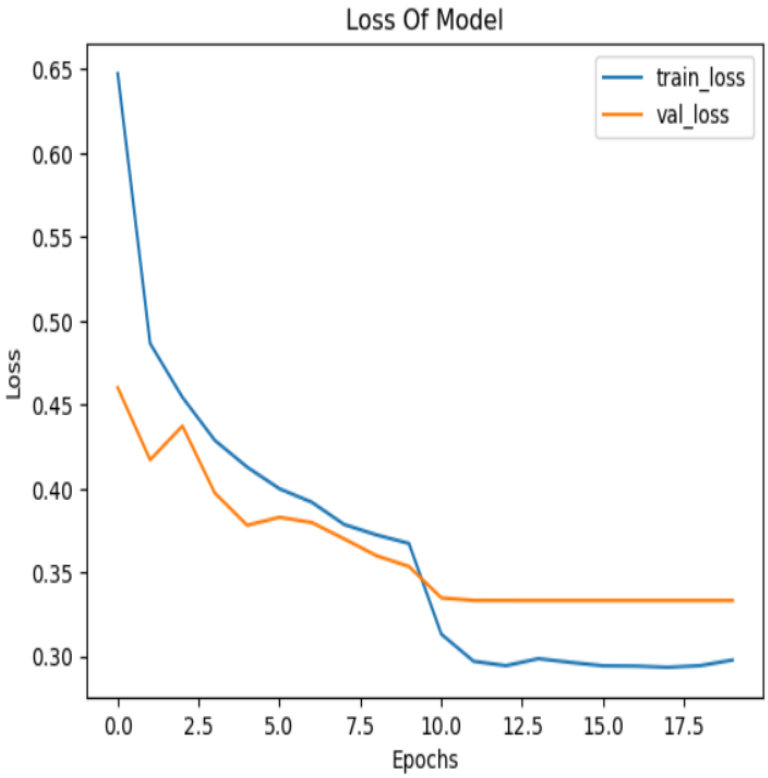
- evalute the model to check its generalization

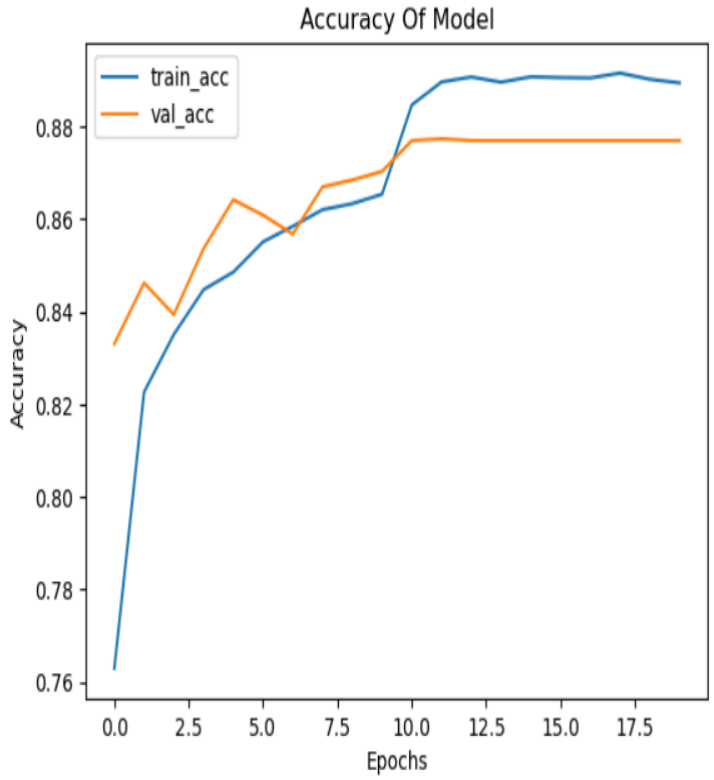
```
# evalute the model on test data
model.evaluate(X_test, y_test)
```

```
188/188 [==============================] - 2s 8ms/step - loss: 0.3254 - accuracy: 0.8839
[0.32541623711586, 0.8839226365089417]
```

- The model performace is very good, where the accuracy and loss:
  - loss: 0.3254
  - accuracy: 0.8839
- It's better than "RESNet152V2", and "Xception" for our data.

- When we visualized the curve of loss and accuracy on training and validation:





- As we can notice the VGG16 model is doing well on validation and testing, it is better than Xception and RESNet, Its results are near to LeNet-5 results of evaluation (LeNet-5 is the best on our trials).