INTERNSHIP REPORT ON DEEP LEARNING FOR SIGNATURE FORGERY VERIFICATION

MENTOR:

MARY ANITHA

RAJAM

BY:

D.DAYA LOKESH

T.SAI JOSHITH

S.SANDEEP KUMAR

INTRODUCTION:-

Deep learning is a part of machine learning which uses multiple layers consisting of neurons to progressively extract higher level features from the raw input it is mainly used to process images with higher accuracy.

In image processing lower layers may identify edges while higher layers may identify the concepts relevant to human such as digits or letters.

There are many types of Deep Learning networks which are based on artificial neural networks, the most prominent being convolutional neural networks. CNN contains more than one convolution layer since it contains a convolution layer in the network with fewer parameters, it is very efficient for image recognization and identifying different image patterns.

Transfer Learning:-

Transfer learning is a machine learning method in which a model is developed for a task and is reused as a starting point in an another model for a different task.

This method allows the user to train a model using less data but getting better results by using the already trained models as a starting point. It also speeds up the process of training a model for a new task.

PRE TRAINED MODELS :-

Pre Trained Models are created by different user but has the experience of the problem that is given to it by the user. These models are generally used to gain higher accuracy without building a model from scratch. Models like resnet 50 vgg 16, efficient net etc., are some of Pre Trained Models that can be imported using keras.

People using these Models have the option to train the given model or not depending on their choice but training these Models takes a huge amount of time since these model have minimum of 50 layers so people generally don’t trained Models.

VGG 16

VGG 16 is a kind of artificial neural networks that is used in the object detection and classification algorithm which is able to classify thousand images of thousand different categories with 92% accuracy.

RESNET

Resent 50 is an artificial neural network which has 50 layers. It is a gate less variant of high way net, the first working feed forward neural network. It is used in computer visioned algorithms.

EFFICIENT NET

Efficient Net is a CNN and is a scaling method that uniformly scales all dimensions using a compound co-efficient. This scaling method scales the resolution, width, depth with a set of fixed scaling co-efficients.

There are many functions in efficient net ranging from B0 to B7 each having a different purpose.

CEDAR DATA SET

This is a data set containing 1320 real and forgeries of signatures.

USING THESE MODELS TO DEVELOP AND NEW MODEL

Step 1:- Import all necessary libraries.

Step 2:- Building the new model using Pre Trained Model.

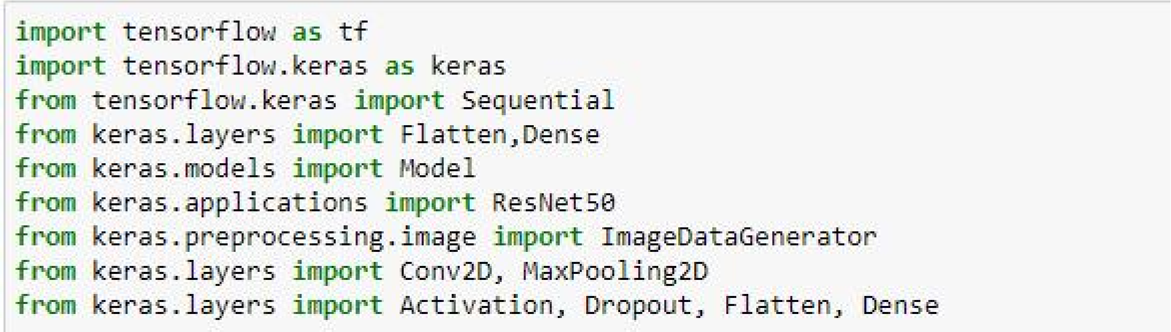
Step 3:- Importing all of the images from a directory.

Step 4:- Compiling and Running the given model with the data set.

Step 5:- Increasing the accuracy using different parameters.

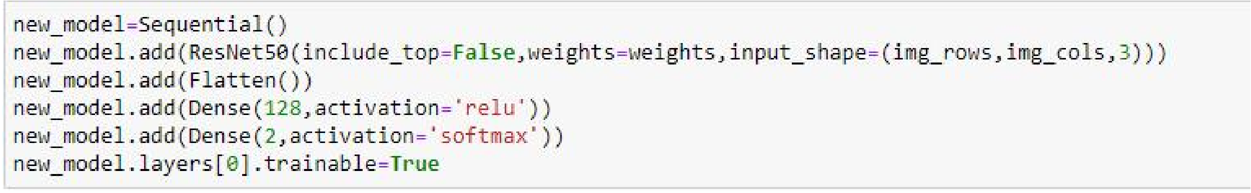
Step 6 :- Predicting the given test data

Step 1:-



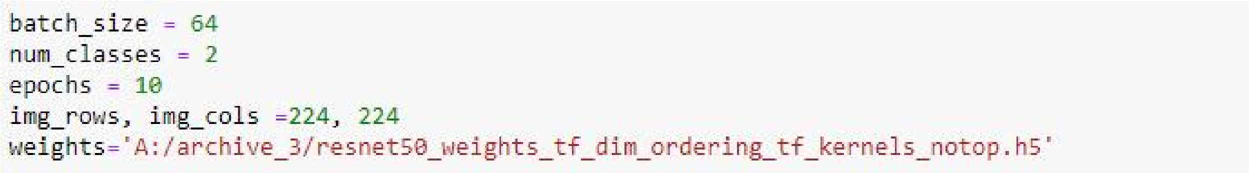
```
import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras import Sequential
from keras.layers import Flatten,Dense
from keras.models import Model
from keras.applications import ResNet50
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
```

Step 2:-

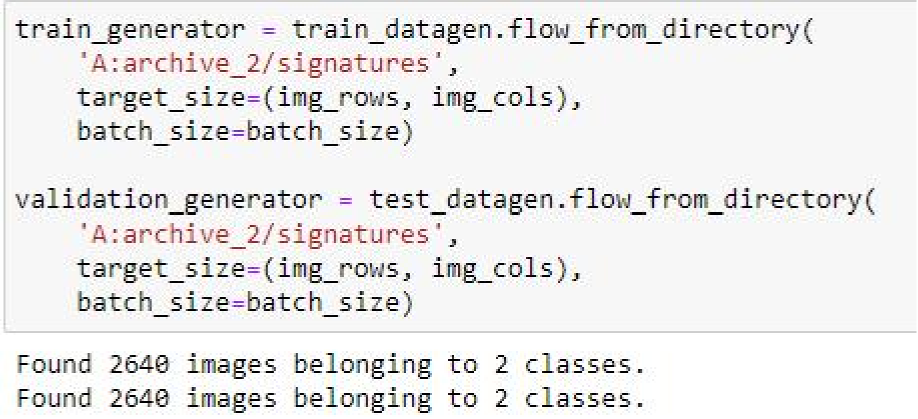


```
new_model=Sequential()
new_model.add(ResNet50(include_top=False,weights=weights,input_shape=(img_rows,img_cols,3)))
new_model.add(Flatten())
new_model.add(Dense(128,activation='relu'))
new_model.add(Dense(2,activation='softmax'))
new_model.layers[0].trainable=True
```

Step 3:-



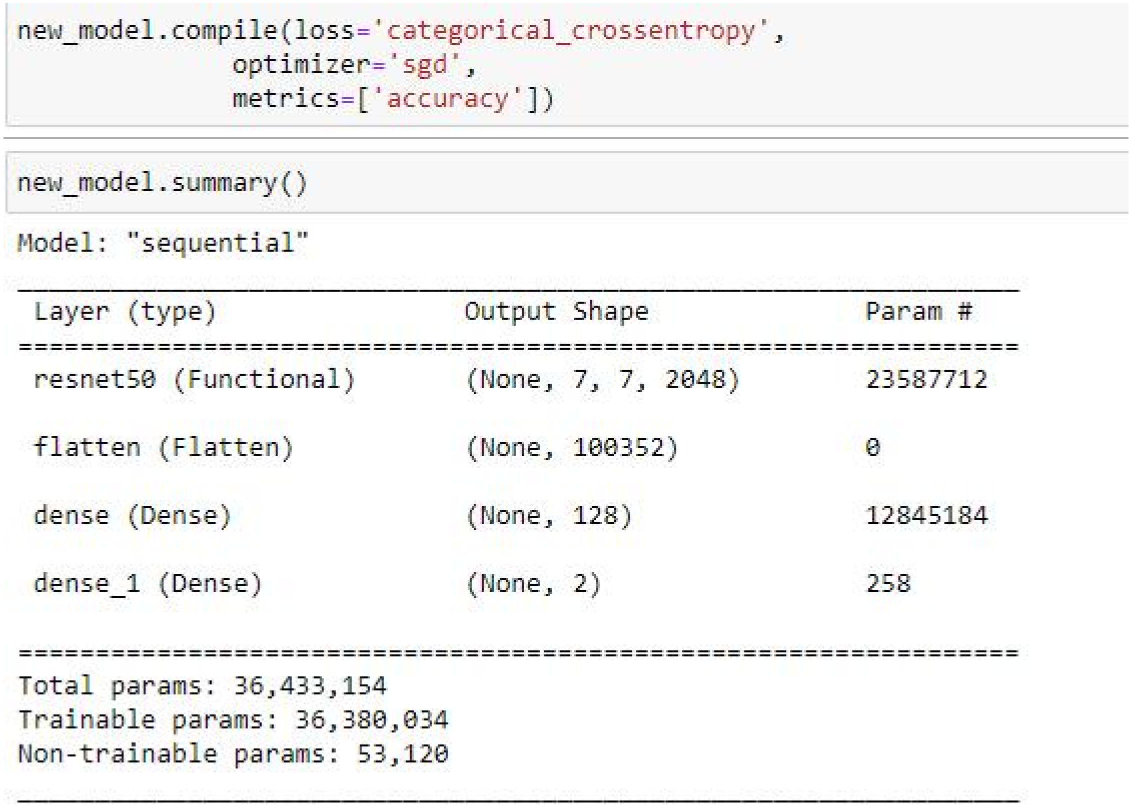
```
batch_size = 64
num_classes = 2
epochs = 10
img_rows, img_cols =224, 224
weights='A:/archive_3/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5'
```



```
train_generator = train_datagen.flow_from_directory(
    'A:archive_2/signatures',
    target_size=(img_rows, img_cols),
    batch_size=batch_size)

validation_generator = test_datagen.flow_from_directory(
    'A:archive_2/signatures',
    target_size=(img_rows, img_cols),
    batch_size=batch_size)
```

```
Found 2640 images belonging to 2 classes.
Found 2640 images belonging to 2 classes.
```

Step 4:-

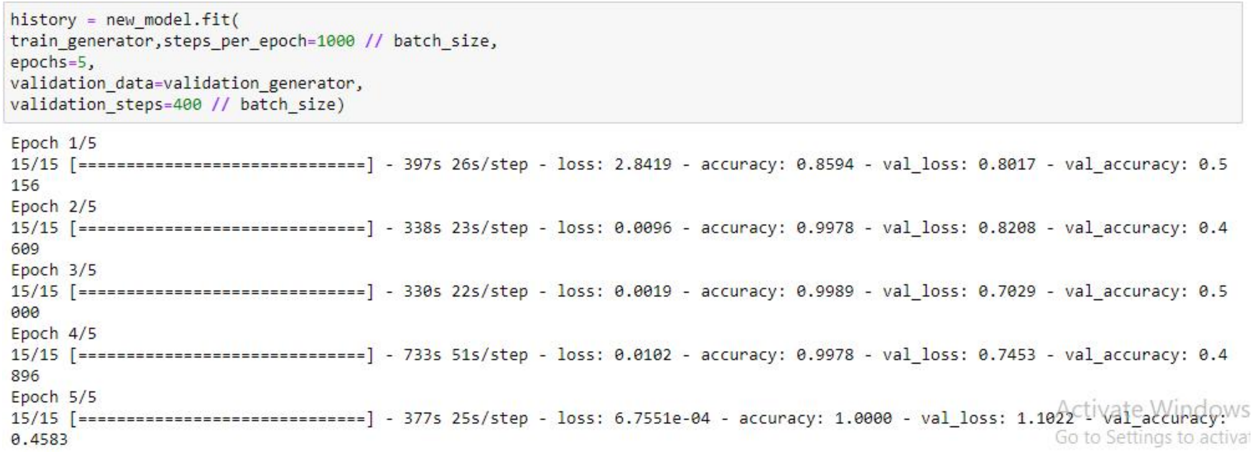


```
new_model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

```
new_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 flatten (Flatten)           (None, 100352)            0

 dense (Dense)               (None, 128)               12845184

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 36,433,154
Trainable params: 36,380,034
Non-trainable params: 53,120
_________________________________________________________________
```

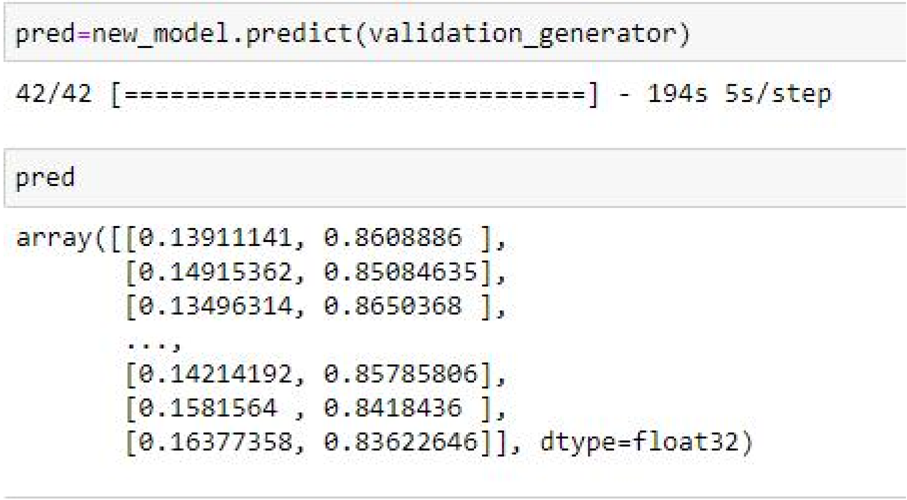
Step 5:-



```
history = new_model.fit(
train_generator,steps_per_epoch=1000 // batch_size,
epochs=5,
validation_data=validation_generator,
validation_steps=400 // batch_size)
```

```
Epoch 1/5
15/15 [==============================] - 397s 26s/step - loss: 2.8419 - accuracy: 0.8594 - val_loss: 0.8017 - val_accuracy: 0.5
156
Epoch 2/5
15/15 [==============================] - 338s 23s/step - loss: 0.0096 - accuracy: 0.9978 - val_loss: 0.8208 - val_accuracy: 0.4
609
Epoch 3/5
15/15 [==============================] - 330s 22s/step - loss: 0.0019 - accuracy: 0.9989 - val_loss: 0.7029 - val_accuracy: 0.5
000
Epoch 4/5
15/15 [==============================] - 733s 51s/step - loss: 0.0102 - accuracy: 0.9978 - val_loss: 0.7453 - val_accuracy: 0.4
896
Epoch 5/5
15/15 [==============================] - 377s 25s/step - loss: 6.7551e-04 - accuracy: 1.0000 - val_loss: 1.1022 - val_accuracy:
0.4583
```

Step 6:-



```
pred=new_model.predict(validation_generator)
```

```
42/42 [==============================] - 194s 5s/step
```

```
pred
```

```
array([[0.13911141, 0.8608886 ],
       [0.14915362, 0.85084635],
       [0.13496314, 0.8650368 ],
       ...,
       [0.14214192, 0.85785806],
       [0.1581564 , 0.8418436 ],
       [0.16377358, 0.83622646]], dtype=float32)
```

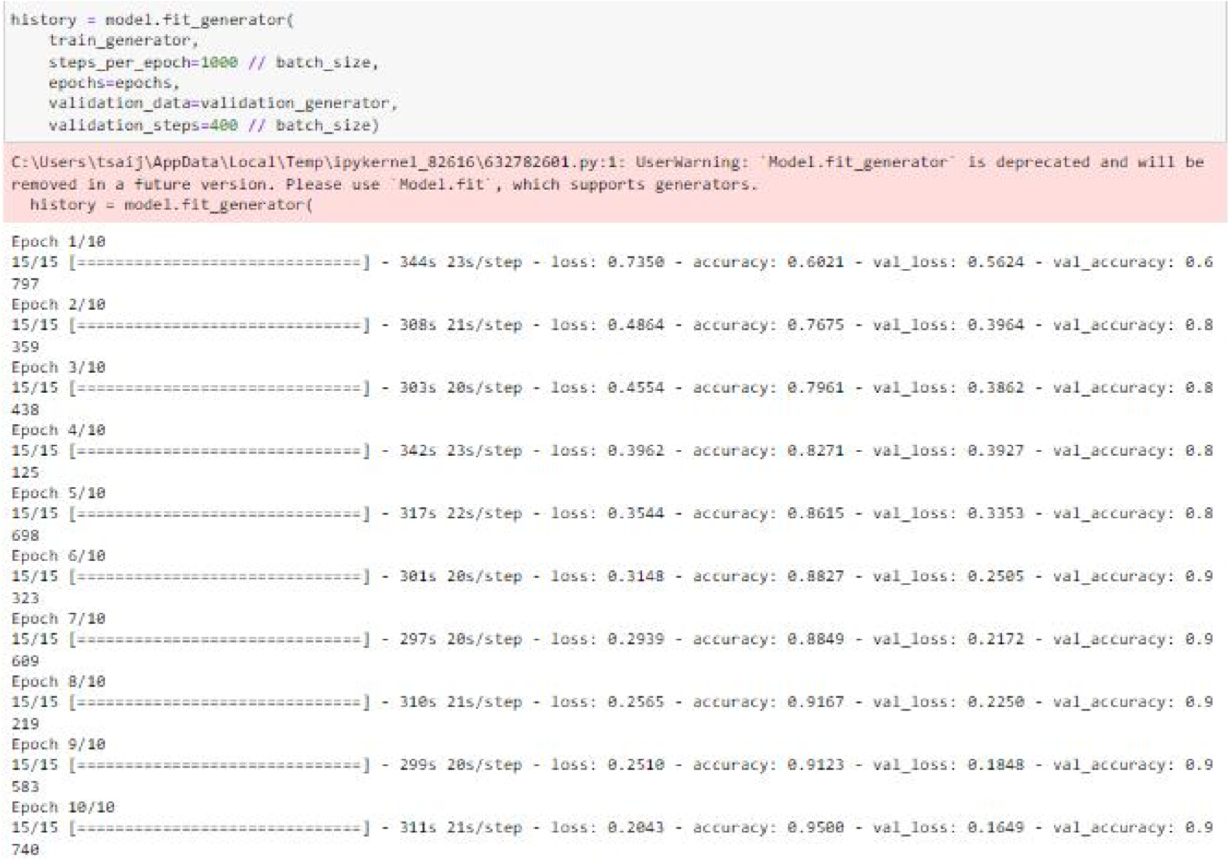
The above steps have been repeated for different models except for the pretrained model we are importing from keras.

As mentioned earlier the pretrained models can be trained again by the user if they wish to which is what we did in the case of resnet.

We decided to train the model in an attempt to increase the accuracy, It worked but at the cost of taking more time.

Here are some other cases where we did not train the models(pretrained).

VGG16:-

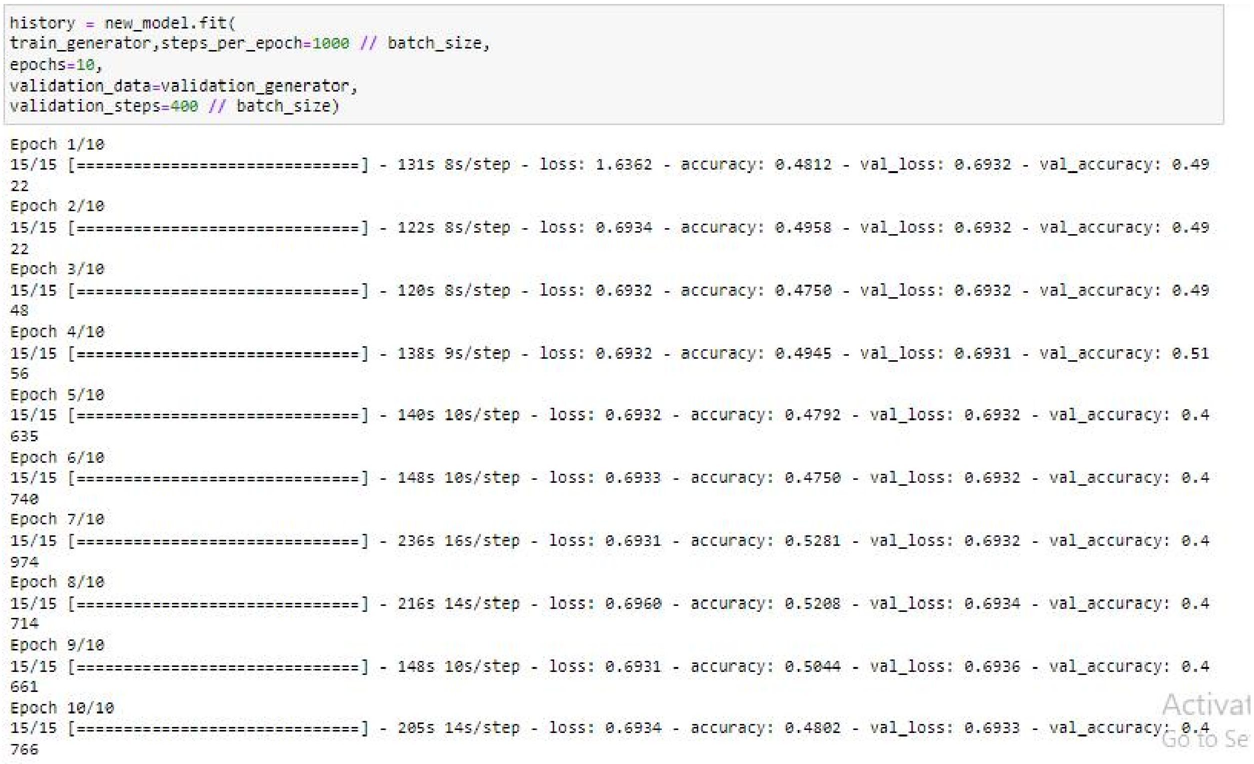


```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=1000 // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=400 // batch_size)
```

```
C:\Users\tsaij\AppData\Local\Temp\ipykernel_82616\632782601.py:1: UserWarning: `Model.fit_generator` is deprecated and will be
removed in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(
```

```
Epoch 1/10
15/15 [==============================] - 344s 23s/step - loss: 0.7350 - accuracy: 0.6021 - val_loss: 0.5624 - val_accuracy: 0.6
797
Epoch 2/10
15/15 [==============================] - 308s 21s/step - loss: 0.4864 - accuracy: 0.7675 - val_loss: 0.3964 - val_accuracy: 0.8
359
Epoch 3/10
15/15 [==============================] - 303s 20s/step - loss: 0.4554 - accuracy: 0.7961 - val_loss: 0.3862 - val_accuracy: 0.8
438
Epoch 4/10
15/15 [==============================] - 342s 23s/step - loss: 0.3962 - accuracy: 0.8271 - val_loss: 0.3927 - val_accuracy: 0.8
125
Epoch 5/10
15/15 [==============================] - 317s 22s/step - loss: 0.3544 - accuracy: 0.8615 - val_loss: 0.3353 - val_accuracy: 0.8
698
Epoch 6/10
15/15 [==============================] - 301s 20s/step - loss: 0.3148 - accuracy: 0.8827 - val_loss: 0.2505 - val_accuracy: 0.9
323
Epoch 7/10
15/15 [==============================] - 297s 20s/step - loss: 0.2939 - accuracy: 0.8849 - val_loss: 0.2172 - val_accuracy: 0.9
609
Epoch 8/10
15/15 [==============================] - 310s 21s/step - loss: 0.2565 - accuracy: 0.9167 - val_loss: 0.2250 - val_accuracy: 0.9
219
Epoch 9/10
15/15 [==============================] - 299s 20s/step - loss: 0.2510 - accuracy: 0.9123 - val_loss: 0.1848 - val_accuracy: 0.9
583
Epoch 10/10
15/15 [==============================] - 311s 21s/step - loss: 0.2043 - accuracy: 0.9500 - val_loss: 0.1649 - val_accuracy: 0.9
740
```

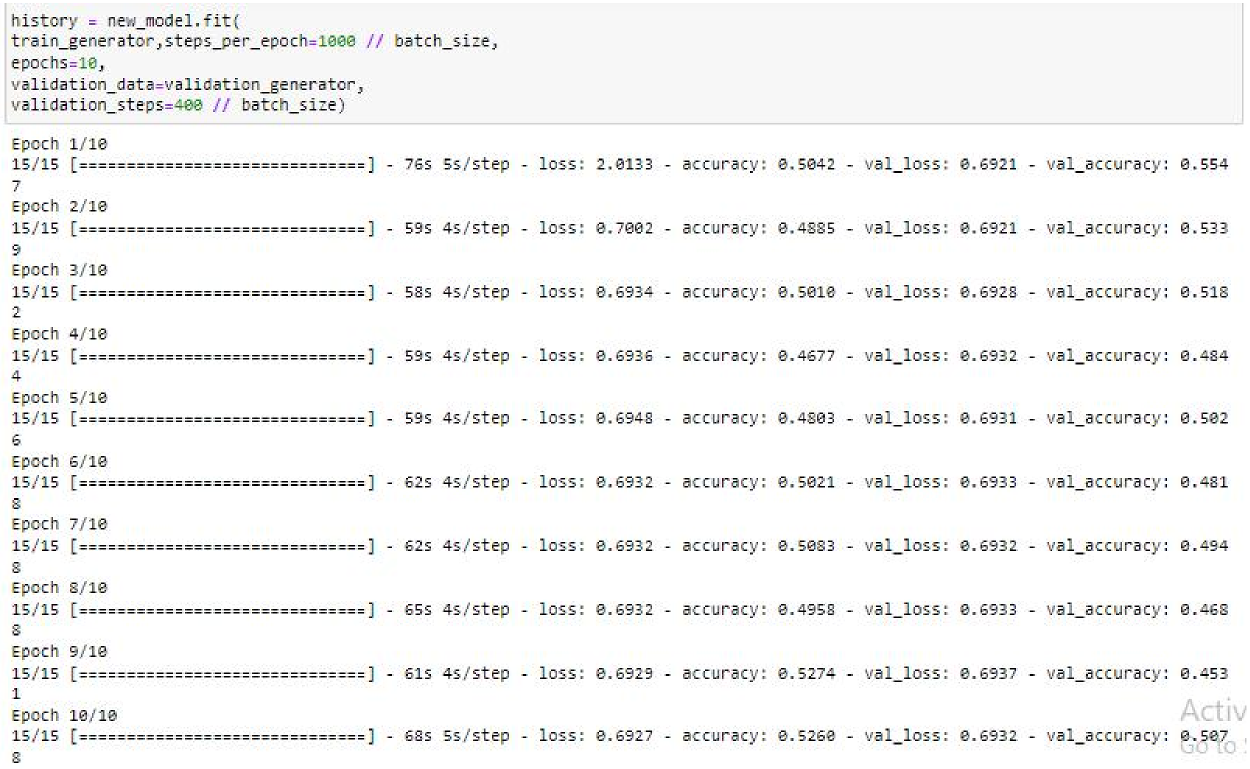
EFFICIENTNETB3:



```
history = new_model.fit(
train_generator,steps_per_epoch=1000 // batch_size,
epochs=10,
validation_data=validation_generator,
validation_steps=400 // batch_size)
```

```
Epoch 1/10
15/15 [==============================] - 131s 8s/step - loss: 1.6362 - accuracy: 0.4812 - val_loss: 0.6932 - val_accuracy: 0.49
22
Epoch 2/10
15/15 [==============================] - 122s 8s/step - loss: 0.6934 - accuracy: 0.4958 - val_loss: 0.6932 - val_accuracy: 0.49
22
Epoch 3/10
15/15 [==============================] - 120s 8s/step - loss: 0.6932 - accuracy: 0.4750 - val_loss: 0.6932 - val_accuracy: 0.49
48
Epoch 4/10
15/15 [==============================] - 138s 9s/step - loss: 0.6932 - accuracy: 0.4945 - val_loss: 0.6931 - val_accuracy: 0.51
56
Epoch 5/10
15/15 [==============================] - 140s 10s/step - loss: 0.6932 - accuracy: 0.4792 - val_loss: 0.6932 - val_accuracy: 0.4
635
Epoch 6/10
15/15 [==============================] - 148s 10s/step - loss: 0.6933 - accuracy: 0.4750 - val_loss: 0.6932 - val_accuracy: 0.4
740
Epoch 7/10
15/15 [==============================] - 236s 16s/step - loss: 0.6931 - accuracy: 0.5281 - val_loss: 0.6932 - val_accuracy: 0.4
974
Epoch 8/10
15/15 [==============================] - 216s 14s/step - loss: 0.6960 - accuracy: 0.5208 - val_loss: 0.6934 - val_accuracy: 0.4
714
Epoch 9/10
15/15 [==============================] - 148s 10s/step - loss: 0.6931 - accuracy: 0.5044 - val_loss: 0.6936 - val_accuracy: 0.4
661
Epoch 10/10
15/15 [==============================] - 205s 14s/step - loss: 0.6934 - accuracy: 0.4802 - val_loss: 0.6933 - val_accuracy: 0.4
766
```

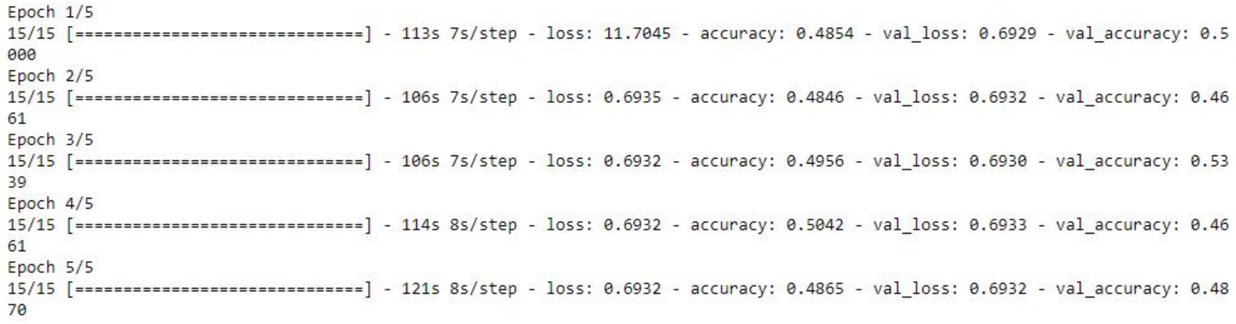
EFFICIENTNETB0:



```
history = new_model.fit(
train_generator,steps_per_epoch=1000 // batch_size,
epochs=10,
validation_data=validation_generator,
validation_steps=400 // batch_size)
```

```
Epoch 1/10
15/15 [==============================] - 76s 5s/step - loss: 2.0133 - accuracy: 0.5042 - val_loss: 0.6921 - val_accuracy: 0.554
7
Epoch 2/10
15/15 [==============================] - 59s 4s/step - loss: 0.7002 - accuracy: 0.4885 - val_loss: 0.6921 - val_accuracy: 0.533
9
Epoch 3/10
15/15 [==============================] - 58s 4s/step - loss: 0.6934 - accuracy: 0.5010 - val_loss: 0.6928 - val_accuracy: 0.518
2
Epoch 4/10
15/15 [==============================] - 59s 4s/step - loss: 0.6936 - accuracy: 0.4677 - val_loss: 0.6932 - val_accuracy: 0.484
4
Epoch 5/10
15/15 [==============================] - 59s 4s/step - loss: 0.6948 - accuracy: 0.4803 - val_loss: 0.6931 - val_accuracy: 0.502
6
Epoch 6/10
15/15 [==============================] - 62s 4s/step - loss: 0.6932 - accuracy: 0.5021 - val_loss: 0.6933 - val_accuracy: 0.481
8
Epoch 7/10
15/15 [==============================] - 62s 4s/step - loss: 0.6932 - accuracy: 0.5083 - val_loss: 0.6932 - val_accuracy: 0.494
8
Epoch 8/10
15/15 [==============================] - 65s 4s/step - loss: 0.6932 - accuracy: 0.4958 - val_loss: 0.6933 - val_accuracy: 0.468
8
Epoch 9/10
15/15 [==============================] - 61s 4s/step - loss: 0.6929 - accuracy: 0.5274 - val_loss: 0.6937 - val_accuracy: 0.453
1
Epoch 10/10
15/15 [==============================] - 68s 5s/step - loss: 0.6927 - accuracy: 0.5260 - val_loss: 0.6932 - val_accuracy: 0.507
8
```

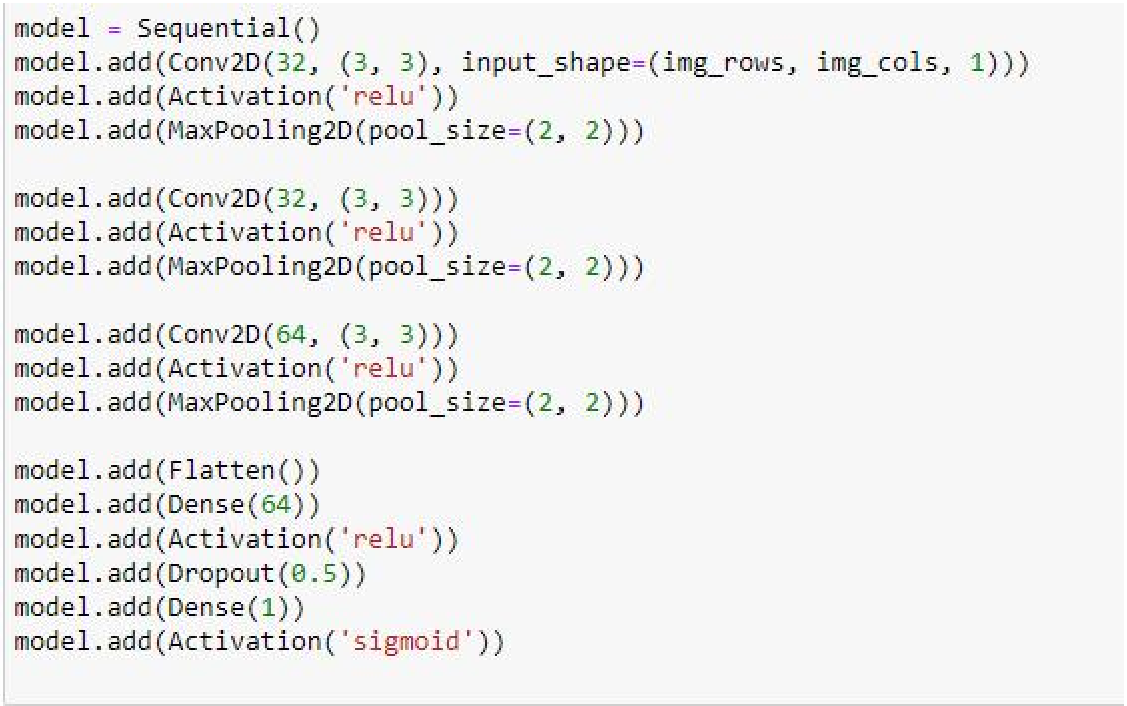
ResNet50:



```
Epoch 1/5
15/15 [==============================] - 113s 7s/step - loss: 11.7045 - accuracy: 0.4854 - val_loss: 0.6929 - val_accuracy: 0.5
000
Epoch 2/5
15/15 [==============================] - 106s 7s/step - loss: 0.6935 - accuracy: 0.4846 - val_loss: 0.6932 - val_accuracy: 0.46
61
Epoch 3/5
15/15 [==============================] - 106s 7s/step - loss: 0.6932 - accuracy: 0.4956 - val_loss: 0.6930 - val_accuracy: 0.53
39
Epoch 4/5
15/15 [==============================] - 114s 8s/step - loss: 0.6932 - accuracy: 0.5042 - val_loss: 0.6933 - val_accuracy: 0.46
61
Epoch 5/5
15/15 [==============================] - 121s 8s/step - loss: 0.6932 - accuracy: 0.4865 - val_loss: 0.6932 - val_accuracy: 0.48
70
```

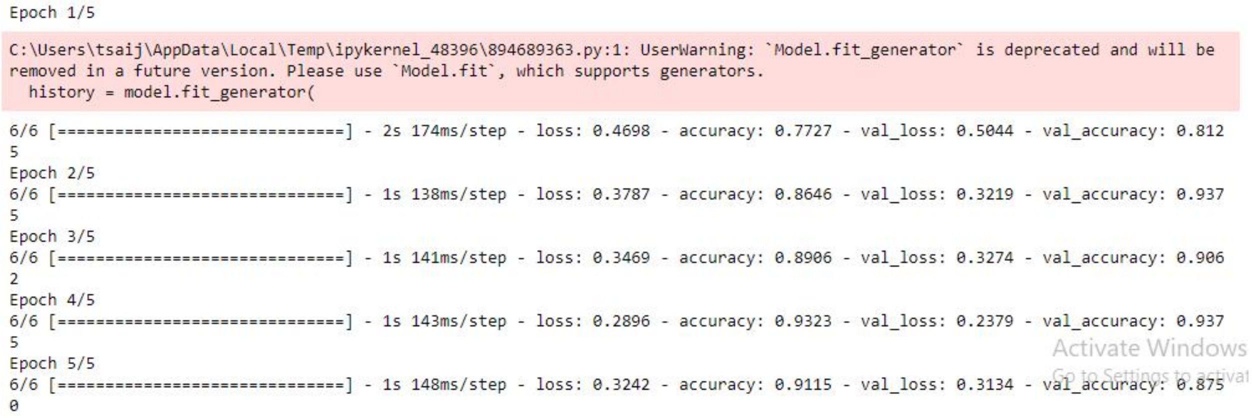
Using an new model:

Model:



```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(img_rows, img_cols, 1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

Fitting the model:



```
Epoch 1/5
C:\Users\tsaij\AppData\Local\Temp\ipykernel_48396\894689363.py:1: UserWarning: `Model.fit_generator` is deprecated and will be
removed in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(
6/6 [==============================] - 2s 174ms/step - loss: 0.4698 - accuracy: 0.7727 - val_loss: 0.5044 - val_accuracy: 0.812
5
Epoch 2/5
6/6 [==============================] - 1s 138ms/step - loss: 0.3787 - accuracy: 0.8646 - val_loss: 0.3219 - val_accuracy: 0.937
5
Epoch 3/5
6/6 [==============================] - 1s 141ms/step - loss: 0.3469 - accuracy: 0.8906 - val_loss: 0.3274 - val_accuracy: 0.906
2
Epoch 4/5
6/6 [==============================] - 1s 143ms/step - loss: 0.2896 - accuracy: 0.9323 - val_loss: 0.2379 - val_accuracy: 0.937
5
Epoch 5/5
6/6 [==============================] - 1s 148ms/step - loss: 0.3242 - accuracy: 0.9115 - val_loss: 0.3134 - val_accuracy: 0.875
0
```

Accuracy of different pre trained models:

VGG16	**95%(without training model)**
ResNet50	**100%(training the model) , 48%(without training the model)**
EfficientnetB0	**52%(without training the model)**
EfficientnetB3	**48%(without training the model)**
New model	**91%**

Observation:-

The most time efficient, most accurate method is to use vgg16 here as we didn’t train the model and it gave the highest accuracy even though it was not trained.

Training a pretrained model(In this case ResNet) gave us the best result but it will take a lot of time to run and doesn’t always guarantee good results.

The training time of a model which did not have any pretrained models is much higher than the models that used transfer learning.

Conclusion:-

For training smaller data sets a model that is built from scratch may be useful but if the data is much larger then usage of transfer learning is recommended.

But the user should have to know which model to use since each model has been trained for a different purpose.