Team Name: Fast And Furious

Name : M. PAVAN GANESH

import the libraries as shown below

In [1]:
```
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.resnet50 import ResNet50
#from keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,loa
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
```

In [2]:
```
# re-size all the images to this
IMAGE_SIZE = [224, 224]

train_path = 'Datasets/train'
valid_path = 'Datasets/test'
```

In [ ]:
```
#[224,224]+[3] rgb
```

In [3]:
```
# Import the Vgg 16 library as shown below and add preprocessing layer
# Here we will be using imagenet weights

resnet = ResNet50(input_shape=IMAGE_SIZE + [3], weights='imagenet', inc
```

In [4]:
```
# don't train existing weights
for layer in resnet.layers:
    layer.trainable = False
```

In [5]:
```
# useful for getting number of output classes
folders = glob('Datasets/train/*')
```

In [6]:
```
# our layers - you can add more if you want
x = Flatten()(resnet.output)
```

In [7]:
```
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=resnet.input, outputs=prediction)
```

In [8]:
```
# view the structure of the model
model.summary()
```

```
conv1_pad (ZeroPadding2D)       (None, 230, 230, 3)  0           i
nput_1[0][0]

__________________________________________________________________________________________________
conv1_conv (Conv2D)             (None, 112, 112, 64) 9472        c
onv1_pad[0][0]

__________________________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, 112, 112, 64) 256         c
onv1_conv[0][0]

__________________________________________________________________________________________________
conv1_relu (Activation)         (None, 112, 112, 64) 0           c
onv1_bn[0][0]

__________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 114, 114, 64) 0           c
onv1_relu[0][0]

__________________________________________________________________________________________________
```

In [9]:
```
# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

In [10]:
```
# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

In [11]:
```
# Make sure you provide the same target size as initialied for the imag
training_set = train_datagen.flow_from_directory('Datasets/train',
                                                 target_size = (224, 22
                                                 batch_size = 32,
                                                 class_mode = 'categori
```

```
Found 64 images belonging to 3 classes.
```

In [12]:
```
test_set = test_datagen.flow_from_directory('Datasets/test',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

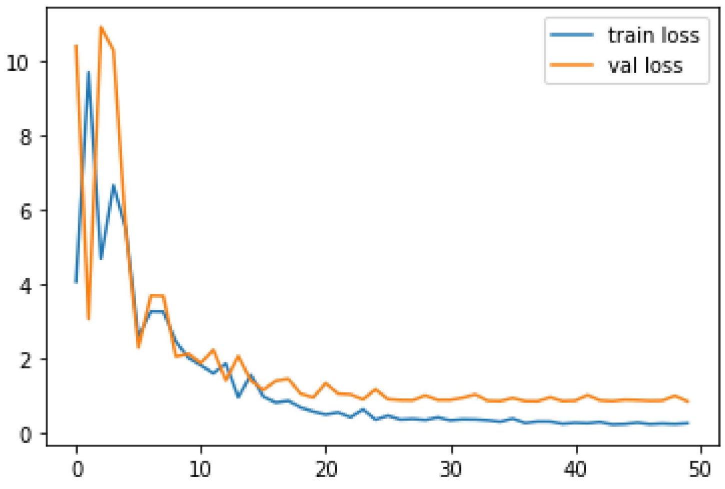
```
Found 58 images belonging to 3 classes.
```

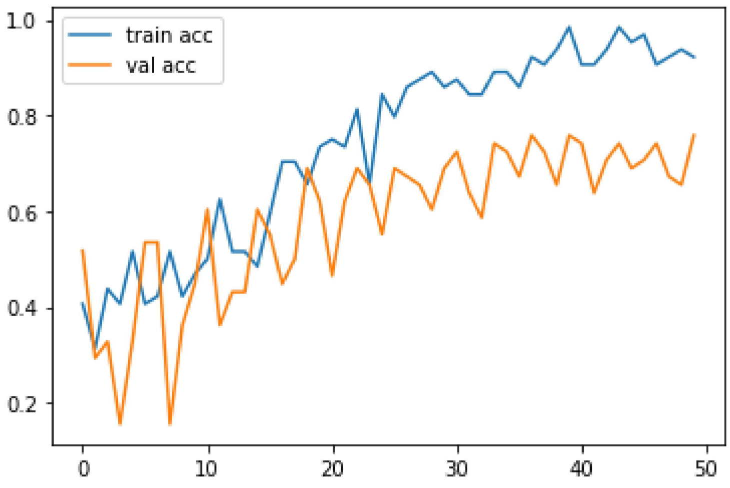
In [13]:
```
# fit the model
# Run the cell. It will take some time to execute
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=50,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

```
accuracy: 0.4219 - val_loss: 2.0428 - val_accuracy: 0.3621
Epoch 10/50
2/2 [==============================] - 7s 3s/step - loss: 2.0080 -
accuracy: 0.4688 - val_loss: 2.1211 - val_accuracy: 0.4483
Epoch 11/50
2/2 [==============================] - 7s 3s/step - loss: 1.8102 -
accuracy: 0.5000 - val_loss: 1.8676 - val_accuracy: 0.6034
Epoch 12/50
2/2 [==============================] - 7s 3s/step - loss: 1.5928 -
accuracy: 0.6250 - val_loss: 2.2227 - val_accuracy: 0.3621
Epoch 13/50
2/2 [==============================] - 6s 3s/step - loss: 1.8562 -
accuracy: 0.5156 - val_loss: 1.4006 - val_accuracy: 0.4310
Epoch 14/50
2/2 [==============================] - 7s 3s/step - loss: 0.9487 -
accuracy: 0.5156 - val_loss: 2.0536 - val_accuracy: 0.4310
Epoch 15/50
2/2 [==============================] - 8s 4s/step - loss: 1.5506 -
accuracy: 0.4844 - val_loss: 1.4023 - val_accuracy: 0.6034
Epoch 16/50
```

In [14]:
```
# plot the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```





```
<Figure size 432x288 with 0 Axes>
```

In [15]:
```
# save it as a h5 file


from tensorflow.keras.models import load_model

model.save('model_resnet50.h5')
```

In [16]:
```
y_pred = model.predict(test_set)
```

In [17]:

```
y_pred
```

```
Out[17]: array([[6.03424728e-01, 3.94489110e-01, 2.08615768e-03],
                [5.95550612e-02, 9.31573093e-01, 8.87183752e-03],
                [2.72485558e-02, 7.71507263e-01, 2.01244146e-01],
                [9.58901346e-01, 2.98168212e-02, 1.12818116e-02],
                [3.10736131e-02, 1.21991960e-02, 9.56727207e-01],
                [6.25616685e-02, 6.61905289e-01, 2.75532931e-01],
                [2.31756046e-02, 3.76151711e-01, 6.00672722e-01],
                [7.35052481e-06, 2.08769273e-03, 9.97905016e-01],
                [1.17803516e-03, 9.73448217e-01, 2.53737103e-02],
                [1.01372600e-02, 7.22803831e-01, 2.67058939e-01],
                [3.00410632e-02, 5.38501322e-01, 4.31457669e-01],
                [1.64399464e-02, 7.13688433e-01, 2.69871712e-01],
                [4.14143223e-03, 8.10631216e-01, 1.85227275e-01],
                [3.78132216e-03, 9.72671390e-01, 2.35472806e-02],
                [5.59390523e-02, 5.21750515e-03, 9.38843369e-01],
                [1.24186557e-02, 8.37701559e-01, 1.49879828e-01],
                [4.34832238e-02, 6.38708353e-01, 3.17808479e-01],
                [1.09004855e-01, 7.11206436e-01, 1.79788738e-01],
                [7.88908824e-03, 9.49634850e-01, 4.24761102e-02],
                [1.21959951e-02, 9.64727521e-01, 2.30764691e-02],
                [1.01567312e-04, 9.97872829e-01, 2.02554092e-03],
                [3.02698016e-02, 9.15531039e-01, 5.41991256e-02],
                [5.29895306e-01, 1.62291750e-01, 3.07812959e-01],
                [1.32341921e-01, 4.07876283e-01, 4.59781826e-01],
                [8.50102723e-01, 1.12333536e-01, 3.75637375e-02],
                [6.71292539e-04, 9.98043418e-01, 1.28530478e-03],
                [8.11699685e-03, 9.91407335e-01, 4.75681707e-04],
                [8.07938818e-03, 9.71131325e-01, 2.07892507e-02],
                [1.16976667e-02, 9.41856921e-01, 4.64454070e-02],
                [2.00887527e-02, 5.54612139e-03, 9.74365175e-01],
                [6.37757732e-03, 9.64011908e-01, 2.96104886e-02],
                [3.53689641e-01, 5.30432284e-01, 1.15878105e-01],
                [5.83457351e-01, 3.66402268e-01, 5.01403436e-02],
                [1.52392602e-02, 3.43331814e-01, 6.41428888e-01],
                [6.47052407e-01, 3.53682451e-02, 3.17579359e-01],
                [4.38970840e-03, 9.80929136e-01, 1.46812070e-02],
                [1.58958137e-01, 6.43497527e-01, 1.97544366e-01],
                [3.83536182e-02, 7.35449016e-01, 2.26197347e-01],
                [7.75442570e-02, 8.30377638e-01, 9.20781791e-02],
                [4.48868722e-02, 4.56214435e-02, 9.09491658e-01],
                [8.70088339e-01, 9.77741927e-02, 3.21374573e-02],
                [1.61685213e-01, 8.09954822e-01, 2.83599291e-02],
                [2.74146907e-02, 3.01918422e-04, 9.72283363e-01],
                [5.94166166e-04, 8.52464497e-01, 1.46941260e-01],
                [2.61779525e-03, 9.91158068e-01, 6.22404134e-03],
                [1.27695873e-01, 3.78242284e-02, 8.34479868e-01],
                [5.59063116e-03, 6.21087134e-01, 3.73322248e-01],
                [1.03011817e-01, 2.05884457e-01, 6.91103697e-01],
                [2.07957219e-05, 1.65727222e-04, 9.99813497e-01],
                [3.43784727e-02, 2.16879800e-01, 7.48741746e-01],
                [3.89950238e-02, 6.04600072e-01, 3.56404930e-01],
                [2.63514221e-02, 8.39421034e-01, 1.34227589e-01],
                [4.93413746e-01, 3.73394608e-01, 1.33191660e-01],
                [1.85855418e-01, 5.55812418e-01, 2.58332133e-01],
                [1.60492118e-02, 9.16722953e-01, 6.72277883e-02],
                [4.82625049e-03, 7.55395889e-02, 9.19634223e-01],
                [6.14218414e-02, 8.42604756e-01, 9.59734619e-02],
                [1.90324057e-02, 7.30597973e-01, 2.50369668e-01]], dtype=float
         32)
```

In [18]:
```
import numpy as np
y_pred = np.argmax(y_pred, axis=1)
```

In [19]:
```
y_pred
```

Out[19]:
```
array([0, 1, 1, 0, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,
1,
       0, 2, 0, 1, 1, 1, 1, 2, 1, 1, 0, 2, 0, 1, 1, 1, 1, 2, 0, 1, 2,
1,
       1, 2, 1, 2, 2, 2, 1, 1, 0, 1, 1, 2, 1, 1], dtype=int64)
```

Prediction

In [20]:
```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

In [21]:
```
model=load_model('model_resnet50.h5')
```

In [23]:
```
img=image.load_img('Datasets/Test/lamborghini/11.jpg',target_size=(224,
```

In [24]:
```
x=image.img_to_array(img)
x
```

Out[24]:
```
array([[[252., 252., 252.],
        [252., 252., 252.],
        [252., 252., 252.],
        ...,
        [196., 187., 172.],
        [217., 208., 193.],
        [243., 234., 219.]],

       [[252., 252., 252.],
        [252., 252., 252.],
        [252., 252., 252.],
        ...,
        [245., 245., 237.],
        [243., 243., 235.],
        [242., 242., 234.]],

       [[252., 252., 252.],
        [252., 252., 252.],
        [252., 252., 252.],
        ...,
        [240., 249., 248.],
        [242., 251., 250.],
        [242., 251., 250.]],

       ...,

       [[189., 207., 229.],
        [190., 206., 229.],
        [190., 206., 229.],
        ...,
        [171., 180., 187.],
        [171., 180., 187.],
        [171., 180., 187.]],

       [[185., 206., 227.],
        [185., 206., 227.],
        [185., 206., 227.],
        ...,
        [171., 180., 187.],
        [171., 180., 187.],
        [171., 180., 187.]],

       [[185., 206., 227.],
        [185., 206., 227.],
        [185., 206., 227.],
        ...,
        [171., 180., 187.],
        [171., 180., 187.],
        [171., 180., 187.]]], dtype=float32)
```

In [25]:
```
x.shape
```

Out[25]:
```
(224, 224, 3)
```

```
In [26]:  x=x/255
```

```
In [27]:  x=np.expand_dims(x,axis=0)
          img_data=preprocess_input(x)
          img_data.shape

Out[27]:  (1, 224, 224, 3)
```

```
In [28]:  model.predict(img_data)

Out[28]:  array([[0.08106431, 0.80465674, 0.1142789 ]], dtype=float32)
```

```
In [29]:  a=np.argmax(model.predict(img_data), axis=1)
```

```
In [30]:  a==1

Out[30]:  array([ True])
```

```
In [ ]:
```