```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
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,load_img
from tensorflow.keras.models import Sequential
import numpy as np
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
```

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

train_path = '/content/drive/MyDrive/car data/Train'
valid_path = '/content/drive/MyDrive/car data/Test'
```

```
#Import the Vgg 16 library as shown below and add preprocessing layer to the front of VGG
# Here we will be using imagenet weights

resnet=ResNet50(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 0s 0us/step
```

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

```
# useful for getting number of output classes
folders = glob('/content/drive/MyDrive/car data/Train/*')
```

```
x = Flatten()(resnet.output)
```

```
prediction = Dense(len(folders), activation='softmax')(x)

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

```
model.summary()
```

```
 rmalization)

 conv5_block3_1_relu (Activ  (None, 7, 7, 512)          0         ['conv5_block3_1_bn[0][0]']
 ation)

 conv5_block3_2_conv (Conv2  (None, 7, 7, 512)          2359808   ['conv5_block3_1_relu[0][0]']
 D)

 conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)          2048      ['conv5_block3_2_conv[0][0]']
 rmalization)

 conv5_block3_2_relu (Activ  (None, 7, 7, 512)          0         ['conv5_block3_2_bn[0][0]']
 ation)

 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)         1050624   ['conv5_block3_2_relu[0][0]']
 D)

 conv5_block3_3_bn (BatchNo  (None, 7, 7, 2048)         8192      ['conv5_block3_3_conv[0][0]']
 rmalization)

 conv5_block3_add (Add)      (None, 7, 7, 2048)         0         ['conv5_block2_out[0][0]',
                                                                   'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activati  (None, 7, 7, 2048)         0         ['conv5_block3_add[0][0]']
 on)

 flatten_4 (Flatten)         (None, 100352)             0         ['conv5_block3_out[0][0]']

 dense_3 (Dense)             (None, 3)                  301059    ['flatten_4[0][0]']

==================================================================================================
Total params: 23888771 (91.13 MB)
Trainable params: 301059 (1.15 MB)
Non-trainable params: 23587712 (89.98 MB)
__________________________________________________________________________________________________
```

```
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

```
# 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)
```

```
 # Make sure you provide the same target size as initialied for the image size
training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/car data/Train',
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
```

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

```
test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/car data/Test',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

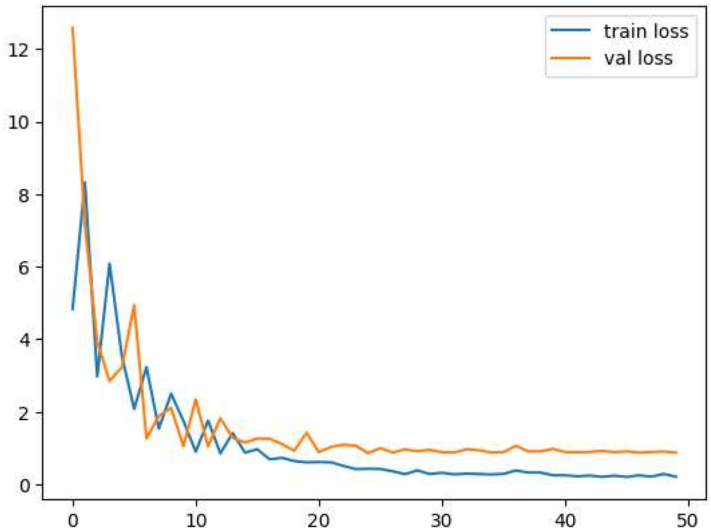
```
Found 58 images belonging to 3 classes.
```

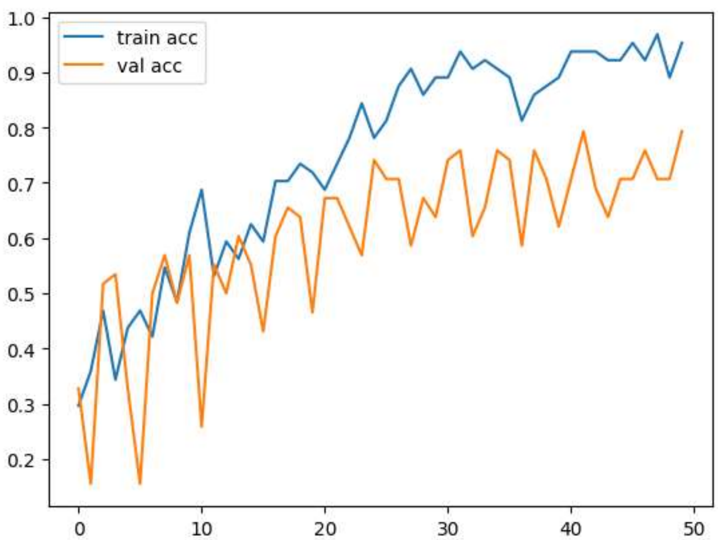
```
# fit the model
# Run the cell. It will take some time to execute
r = model.fit(
  training_set,
  validation_data=test_set,
  epochs=50,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

```
Epoch 22/50
2/2 [==============================] - 1s 833ms/step - loss: 0.6045 - accuracy: 0.7344 - val_loss: 1.0302 - val_accuracy: 0.6724
Epoch 23/50
2/2 [==============================] - 1s 840ms/step - loss: 0.5054 - accuracy: 0.7812 - val_loss: 1.0929 - val_accuracy: 0.6207
Epoch 24/50
2/2 [==============================] - 1s 814ms/step - loss: 0.4199 - accuracy: 0.8438 - val_loss: 1.0630 - val_accuracy: 0.5690
Epoch 25/50
2/2 [==============================] - 1s 833ms/step - loss: 0.4281 - accuracy: 0.7812 - val_loss: 0.8599 - val_accuracy: 0.7414
Epoch 26/50
2/2 [==============================] - 2s 1s/step - loss: 0.4223 - accuracy: 0.8125 - val_loss: 0.9934 - val_accuracy: 0.7069
Epoch 27/50
2/2 [==============================] - 2s 1s/step - loss: 0.3574 - accuracy: 0.8750 - val_loss: 0.8763 - val_accuracy: 0.7069
Epoch 28/50
2/2 [==============================] - 1s 883ms/step - loss: 0.2794 - accuracy: 0.9062 - val_loss: 0.9642 - val_accuracy: 0.5862
Epoch 29/50
2/2 [==============================] - 1s 813ms/step - loss: 0.3837 - accuracy: 0.8594 - val_loss: 0.9113 - val_accuracy: 0.6724
Epoch 30/50
2/2 [==============================] - 1s 828ms/step - loss: 0.2841 - accuracy: 0.8906 - val_loss: 0.9506 - val_accuracy: 0.6379
Epoch 31/50
2/2 [==============================] - 1s 850ms/step - loss: 0.3174 - accuracy: 0.8906 - val_loss: 0.8833 - val_accuracy: 0.7414
Epoch 32/50
2/2 [==============================] - 1s 857ms/step - loss: 0.2740 - accuracy: 0.9375 - val_loss: 0.8788 - val_accuracy: 0.7586
Epoch 33/50
2/2 [==============================] - 1s 833ms/step - loss: 0.2942 - accuracy: 0.9062 - val_loss: 0.9666 - val_accuracy: 0.6034
Epoch 34/50
2/2 [==============================] - 2s 1s/step - loss: 0.2831 - accuracy: 0.9219 - val_loss: 0.9384 - val_accuracy: 0.6552
Epoch 35/50
2/2 [==============================] - 2s 1s/step - loss: 0.2705 - accuracy: 0.9062 - val_loss: 0.8776 - val_accuracy: 0.7586
Epoch 36/50
2/2 [==============================] - 1s 855ms/step - loss: 0.2865 - accuracy: 0.8906 - val_loss: 0.8859 - val_accuracy: 0.7414
Epoch 37/50
2/2 [==============================] - 1s 831ms/step - loss: 0.3792 - accuracy: 0.8125 - val_loss: 1.0600 - val_accuracy: 0.5862
Epoch 38/50
2/2 [==============================] - 1s 827ms/step - loss: 0.3251 - accuracy: 0.8594 - val_loss: 0.9061 - val_accuracy: 0.7586
Epoch 39/50
2/2 [==============================] - 1s 833ms/step - loss: 0.3233 - accuracy: 0.8750 - val_loss: 0.9102 - val_accuracy: 0.7069
Epoch 40/50
2/2 [==============================] - 1s 824ms/step - loss: 0.2494 - accuracy: 0.8906 - val_loss: 0.9763 - val_accuracy: 0.6207
Epoch 41/50
2/2 [==============================] - 1s 803ms/step - loss: 0.2492 - accuracy: 0.9375 - val_loss: 0.8879 - val_accuracy: 0.7069
Epoch 42/50
2/2 [==============================] - 1s 801ms/step - loss: 0.2210 - accuracy: 0.9375 - val_loss: 0.8848 - val_accuracy: 0.7931
Epoch 43/50
2/2 [==============================] - 2s 1s/step - loss: 0.2410 - accuracy: 0.9375 - val_loss: 0.8922 - val_accuracy: 0.6897
Epoch 44/50
2/2 [==============================] - 2s 929ms/step - loss: 0.2061 - accuracy: 0.9219 - val_loss: 0.9202 - val_accuracy: 0.6379
Epoch 45/50
2/2 [==============================] - 1s 845ms/step - loss: 0.2378 - accuracy: 0.9219 - val_loss: 0.8843 - val_accuracy: 0.7069
Epoch 46/50
2/2 [==============================] - 1s 793ms/step - loss: 0.2022 - accuracy: 0.9531 - val_loss: 0.9098 - val_accuracy: 0.7069
Epoch 47/50
2/2 [==============================] - 1s 824ms/step - loss: 0.2487 - accuracy: 0.9219 - val_loss: 0.8714 - val_accuracy: 0.7586
Epoch 48/50
2/2 [==============================] - 1s 849ms/step - loss: 0.2102 - accuracy: 0.9688 - val_loss: 0.8899 - val_accuracy: 0.7069
Epoch 49/50
2/2 [==============================] - 1s 836ms/step - loss: 0.2847 - accuracy: 0.8906 - val_loss: 0.9044 - val_accuracy: 0.7069
Epoch 50/50
2/2 [==============================] - 2s 1s/step - loss: 0.2099 - accuracy: 0.9531 - val_loss: 0.8720 - val_accuracy: 0.7931
```

```
# 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 640x480 with 0 Axes>
```

```
from tensorflow.keras.models import load_model
model.save('car_model_resnet50.h5')
```

```
y_pred = model.predict(test_set)
```

```
2/2 [==============================] - 2s 205ms/step
```

```
y_pred
```

```
array([0, 2, 1, 0, 0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 0, 1,
       2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 0, 1, 1, 2, 1, 2,
       0, 2, 1, 1, 0, 2, 2, 1, 1, 1, 2, 1, 1, 1])
```

```
model=load_model('car_model_resnet50.h5')
```

```
img=image.load_img('/content/drive/MyDrive/car data/Test/lamborghini/10.jpg',target_size=(224,224))
```

```
x=image.img_to_array(img)
```

```
x
```

```
array([[[ 17.,   7.,   0.],
        [ 17.,   7.,   0.],
        [ 17.,   7.,   0.],
        ...,
        [ 13.,   2.,   0.],
        [ 14.,   4.,   3.],
```

```
         [ 14.,   4.,   3.]],

        [[ 17.,   7.,   0.],
         [ 17.,   7.,   0.],
         [ 18.,   8.,   0.],
         ...,
         [ 14.,   3.,   1.],
         [ 14.,   4.,   3.],
         [ 14.,   4.,   3.]],

        [[ 18.,   8.,   0.],
         [ 18.,   8.,   0.],
         [ 19.,   9.,   0.],
         ...,
         [ 14.,   3.,   1.],
         [ 14.,   4.,   3.],
         [ 14.,   4.,   3.]],

        ...,

        [[209., 129.,  92.],
         [210., 130.,  93.],
         [212., 132.,  95.],
         ...,
         [216., 132.,  96.],
         [214., 129.,  92.],
         [213., 129.,  92.]],

        [[208., 128.,  91.],
         [208., 128.,  91.],
         [211., 131.,  94.],
         ...,
         [215., 131.,  95.],
         [209., 129.,  94.],
         [207., 129.,  93.]],

        [[206., 128.,  90.],
         [206., 128.,  90.],
         [209., 131.,  93.],
         ...,
         [214., 130.,  94.],
         [203., 128.,  96.],
         [203., 128.,  96.]]], dtype=float32)
```

```
x.shape
```

```
(224, 224, 3)
```

```
x=x/255
```

```
x=np.expand_dims(x,axis=0)
img_data=preprocess_input(x)
img_data.shape
```

```
(1, 224, 224, 3)
```

```
model.predict(img_data)
```

```
1/1 [==============================] - 97s 97s/step
array([[0.03226576, 0.06799173, 0.8997425 ]], dtype=float32)
```

```
a=np.argmax(model.predict(img_data), axis=1)
```

```
1/1 [==============================] - 0s 44ms/step
```

+ Code + Text

Start coding or generate with AI.