```
!nvidia-smi -L
```

```
     GPU 0: Tesla K80 (UUID: GPU-0d4a5a09-f94e-d96a-a98d-5060f1e203ad)
```

```
#!mkdir ~/.kaggle
```

```
#!cp /kaggle.json ~/.kaggle/
```

```
#!chmod 600 ~/.kaggle/kaggle.json
```

```
#! kaggle datasets download -d slothkong/10-monkey-species
```

```
#! unzip /content/10-monkey-species.zip
```

```
## Creating CNN Using Scratch And Transfer Learning
```

```
# import the libraries as shown below

from tensorflow.keras.layers import Input, Lambda, Dense, Flatten,Conv2D
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg19 import VGG19
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 glob import glob
import matplotlib.pyplot as plt
```

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

train_path = '/content/training/training'
valid_path = '/content/validation/validation'
```

```
# Import the Vgg 19 library as shown below and add preprocessing layer to the front of VGG
# Here we will be using imagenet weights

vgg19 = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
# don't train existing weights
for layer in vgg19.layers:
    layer.trainable = False
```

```
  # useful for getting number of output classes
folders = glob('/content/training/training/*')
```

```
folders
```

```
#·our·layers·-·you·can·add·more·if·you·want
x·=·Flatten()(vgg19.output)
```

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

# create a model object
model = Model(inputs=vgg19.input, outputs=prediction)
```

```
#·view·the·structure·of·the·model
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_1 (InputLayer)        [(None, 224, 224, 3)]     0

_________________________________________________________________
block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

_________________________________________________________________
block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

_________________________________________________________________
block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
```

```
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 10)                250890
=================================================================
Total params: 20,275,274
Trainable params: 250,890
Non-trainable params: 20,024,384
_________________________________________________________________
```

```
from·tensorflow.keras.layers·import·MaxPooling2D
```

```
###·Create·Model·from·scratch·using·CNN
model=Sequential()
model.add(Conv2D(filters=16,kernel_size=2,padding="same",activation="relu",input_shape=(224,224,3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation·="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Flatten())
model.add(Dense(500,activation="relu"))
model.add(Dense(10,activation="softmax"))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 224, 224, 16)      208       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 16)      0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 112, 112, 32)      2080      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 56, 56, 64)        8256      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 64)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 50176)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 500)               25088500  
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5010      
=================================================================
Total params: 25,104,054
Trainable params: 25,104,054
Non-trainable params: 0
_________________________________________________________________
```

```
#·tell·the·model·what·cost·and·optimization·method·to·use
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/training/training',
·················································target_size·=·(224,·224),
·················································batch_size·=·32,
·················································class_mode·=·'categorical')
```

```
     Found 1098 images belonging to 10 classes.
```

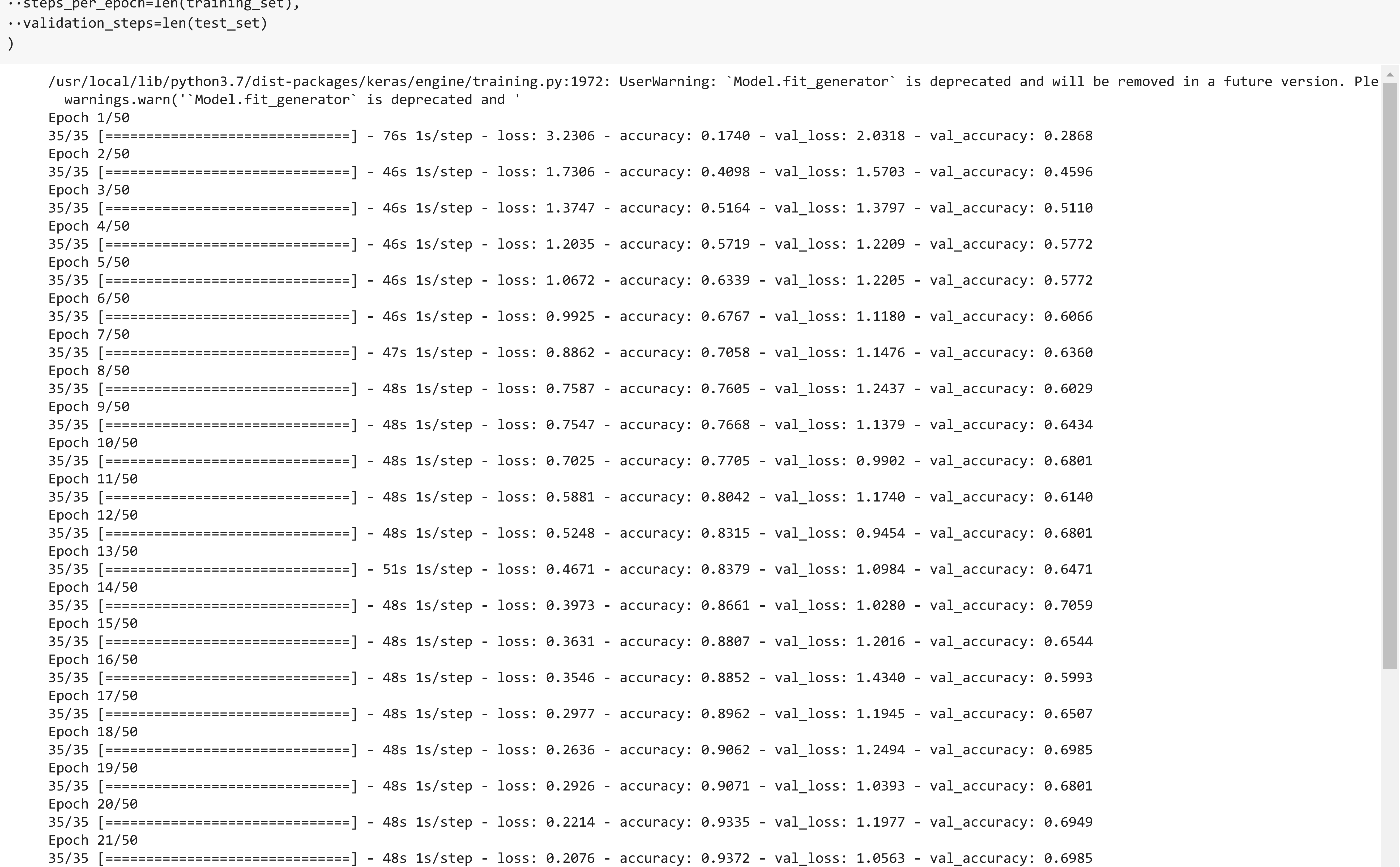
```
training_set
```

```
     <keras.preprocessing.image.DirectoryIterator at 0x7fa510410590>
```

```
test_set·=·test_datagen.flow_from_directory('/content/validation/validation',
············································target_size·=·(224,·224),
············································batch_size·=·32,
············································class_mode·=·'categorical')
```

```
     Found 272 images belonging to 10 classes.
```

```
#·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),
```



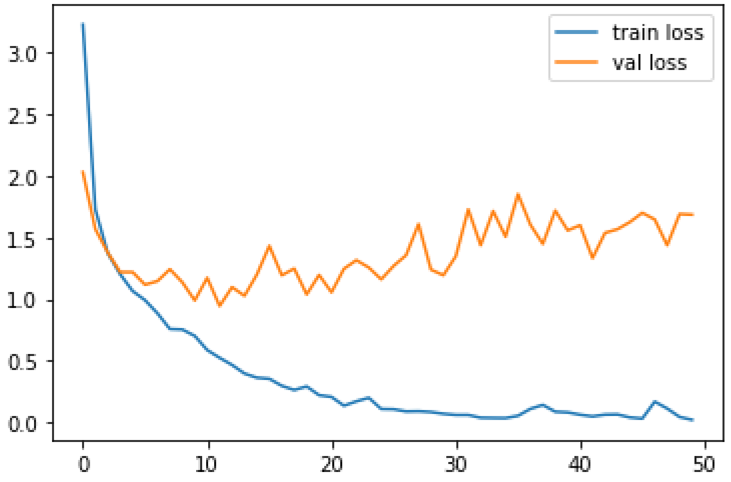
```
··steps_per_epoch=len(training_set),
··validation_steps=len(test_set)
)
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/training.py:1972: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Ple
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/50
35/35 [==============================] - 76s 1s/step - loss: 3.2306 - accuracy: 0.1740 - val_loss: 2.0318 - val_accuracy: 0.2868
Epoch 2/50
35/35 [==============================] - 46s 1s/step - loss: 1.7306 - accuracy: 0.4098 - val_loss: 1.5703 - val_accuracy: 0.4596
Epoch 3/50
35/35 [==============================] - 46s 1s/step - loss: 1.3747 - accuracy: 0.5164 - val_loss: 1.3797 - val_accuracy: 0.5110
Epoch 4/50
35/35 [==============================] - 46s 1s/step - loss: 1.2035 - accuracy: 0.5719 - val_loss: 1.2209 - val_accuracy: 0.5772
Epoch 5/50
35/35 [==============================] - 46s 1s/step - loss: 1.0672 - accuracy: 0.6339 - val_loss: 1.2205 - val_accuracy: 0.5772
Epoch 6/50
35/35 [==============================] - 46s 1s/step - loss: 0.9925 - accuracy: 0.6767 - val_loss: 1.1180 - val_accuracy: 0.6066
Epoch 7/50
35/35 [==============================] - 47s 1s/step - loss: 0.8862 - accuracy: 0.7058 - val_loss: 1.1476 - val_accuracy: 0.6360
Epoch 8/50
35/35 [==============================] - 48s 1s/step - loss: 0.7587 - accuracy: 0.7605 - val_loss: 1.2437 - val_accuracy: 0.6029
Epoch 9/50
35/35 [==============================] - 48s 1s/step - loss: 0.7547 - accuracy: 0.7668 - val_loss: 1.1379 - val_accuracy: 0.6434
Epoch 10/50
35/35 [==============================] - 48s 1s/step - loss: 0.7025 - accuracy: 0.7705 - val_loss: 0.9902 - val_accuracy: 0.6801
Epoch 11/50
35/35 [==============================] - 48s 1s/step - loss: 0.5881 - accuracy: 0.8042 - val_loss: 1.1740 - val_accuracy: 0.6140
Epoch 12/50
35/35 [==============================] - 48s 1s/step - loss: 0.5248 - accuracy: 0.8315 - val_loss: 0.9454 - val_accuracy: 0.6801
Epoch 13/50
35/35 [==============================] - 51s 1s/step - loss: 0.4671 - accuracy: 0.8379 - val_loss: 1.0984 - val_accuracy: 0.6471
Epoch 14/50
35/35 [==============================] - 48s 1s/step - loss: 0.3973 - accuracy: 0.8661 - val_loss: 1.0280 - val_accuracy: 0.7059
Epoch 15/50
35/35 [==============================] - 48s 1s/step - loss: 0.3631 - accuracy: 0.8807 - val_loss: 1.2016 - val_accuracy: 0.6544
Epoch 16/50
35/35 [==============================] - 48s 1s/step - loss: 0.3546 - accuracy: 0.8852 - val_loss: 1.4340 - val_accuracy: 0.5993
Epoch 17/50
35/35 [==============================] - 48s 1s/step - loss: 0.2977 - accuracy: 0.8962 - val_loss: 1.1945 - val_accuracy: 0.6507
Epoch 18/50
35/35 [==============================] - 48s 1s/step - loss: 0.2636 - accuracy: 0.9062 - val_loss: 1.2494 - val_accuracy: 0.6985
Epoch 19/50
35/35 [==============================] - 48s 1s/step - loss: 0.2926 - accuracy: 0.9071 - val_loss: 1.0393 - val_accuracy: 0.6801
Epoch 20/50
35/35 [==============================] - 48s 1s/step - loss: 0.2214 - accuracy: 0.9335 - val_loss: 1.1977 - val_accuracy: 0.6949
Epoch 21/50
35/35 [==============================] - 48s 1s/step - loss: 0.2076 - accuracy: 0.9372 - val_loss: 1.0563 - val_accuracy: 0.6985
```

```
Epoch 22/50
35/35 [==============================] - 48s 1s/step - loss: 0.1359 - accuracy: 0.9645 - val_loss: 1.2475 - val_accuracy: 0.7132
Epoch 23/50
35/35 [==============================] - 48s 1s/step - loss: 0.1715 - accuracy: 0.9353 - val_loss: 1.3173 - val_accuracy: 0.6801
Epoch 24/50
35/35 [==============================] - 48s 1s/step - loss: 0.2006 - accuracy: 0.9262 - val_loss: 1.2576 - val_accuracy: 0.6985
Epoch 25/50
35/35 [==============================] - 48s 1s/step - loss: 0.1095 - accuracy: 0.9690 - val_loss: 1.1618 - val_accuracy: 0.7243
Epoch 26/50
35/35 [==============================] - 48s 1s/step - loss: 0.1075 - accuracy: 0.9636 - val_loss: 1.2711 - val_accuracy: 0.6949
Epoch 27/50
35/35 [==============================] - 48s 1s/step - loss: 0.0895 - accuracy: 0.9745 - val_loss: 1.3578 - val_accuracy: 0.7022
Epoch 28/50
35/35 [==============================] - 48s 1s/step - loss: 0.0913 - accuracy: 0.9718 - val_loss: 1.6105 - val_accuracy: 0.6544
Epoch 29/50
```

```
#·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')
```



```
# save it as a h5 file


from tensorflow.keras.models import load_model

model.save('model_vgg19.h5')
```

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

```
y_pred
```

```
array([[7.4200456e-05, 5.7106816e-11, 7.0140527e-16, ..., 4.4123065e-12,
        1.5433450e-08, 9.9992573e-01],
       [5.2844331e-04, 6.8873609e-04, 2.0556271e-07, ..., 1.3399301e-01,
        8.6287284e-01, 3.0296171e-04],
       [8.0836227e-04, 1.4947353e-01, 6.7373565e-03, ..., 1.9687573e-06,
        1.9595947e-02, 8.2338220e-01],
       ...,
       [1.3942763e-06, 3.2462629e-03, 4.7707577e-05, ..., 8.0205145e-04,
        7.8215022e-03, 1.1042499e-05],
       [3.1408689e-13, 8.5698248e-06, 4.4368718e-08, ..., 2.6772225e-06,
        8.6317552e-05, 4.2417053e-18],
       [9.9915016e-01, 5.2499849e-10, 3.4223052e-12, ..., 3.7409705e-13,
        4.0357299e-05, 8.0755574e-04]], dtype=float32)
```

```
# Evaluating model on validation data
# Evaluating model on validation data
evaluate·=·model.evaluate(test_set)
print(evaluate)
```

```
9/9 [==============================] - 8s 743ms/step - loss: 1.6870 - accuracy: 0.7132
[1.6869858503341675, 0.7132353186607361]
```

```
from·sklearn.metrics·import·confusion_matrix
```

```
def·give_accuracy():
····p=model.predict(test_set)
····cm=confusion_matrix(y_true=test_set.classes,y_pred=np.argmax(p,axis=-1))
····acc=cm.trace()/cm.sum()
····print('The·Classification·Report·\n',·cm)
····print(f'Accuracy:·{acc*100}')
```

```
give_accuracy()
```

```
The Classification Report
 [[4 3 5 2 2 2 3 2 2 1]
 [3 1 3 3 3 1 2 5 0 7]
 [1 0 2 3 5 2 3 1 5 5]
 [3 6 1 3 5 3 1 4 2 2]
 [0 5 0 6 2 2 0 5 3 3]
 [1 3 6 3 5 2 1 0 2 5]
 [2 3 2 2 3 3 2 2 4 3]
 [3 5 3 3 1 1 1 4 3 4]
 [2 1 3 1 2 3 2 3 7 3]
 [4 5 1 3 1 1 2 2 3 4]]
Accuracy: 11.397058823529411
```

```
import·numpy·as·np
from·tensorflow.keras.preprocessing·import·image
test_image·=·image.load_img('/content/validation/validation/n7/n701.jpg',·target_size·=·(224,224))
test_image·=·image.img_to_array(test_image)
test_image=test_image/255
test_image·=·np.expand_dims(test_image,·axis·=·0)
result·=·model.predict(test_image)
```

```
test·=·np.array(test_image)
```

```
#·making·predictions
#prediction·=·np.argmax(cnn.predict(test_image),·axis=-1)
prediction·=·np.argmax(model.predict(test_image))
```

```
prediction
```

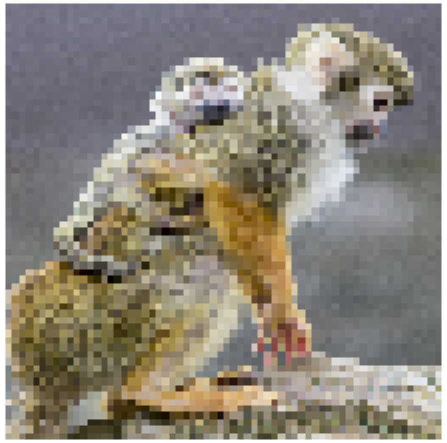
```
     4
```

```
output·=·{·0:'mantled_howler',1:'patas_monkey',2:'bald_uakari',3:'japanese_macaque',4:'pygmy_marmoset',5:'white_headed_cap
```

```
print("The·prediction·Of·the·Image·is·:·",·output[prediction])
```

```
     The prediction Of the Image is :  pygmy_marmoset
```

```
#·show·the·image
import·matplotlib.pyplot·as·plt
test_image·=·image.load_img('/content/validation/validation/n7/n701.jpg',·target_size·=·(64,64))
plt.axis('off')
plt.imshow(test_image)
plt.show()
```



✓ 0s completed at 2:00 PM ● ✕