IMAGE CLASSIFICATION PROJECT

Image Classification Model Deployment

Compiled By: Muchammad Malik

Notes:

1. Dataset taken from Kaggle, on July 15, 2021, with link: https://www.kaggle.com/ashishsaxena2209/animal-image-datasetdog-cat-and-panda
2. To load the dataset directly from the runtime, input the Kaggle account username and password 3. The highest accuracy in the last experiment was 0.9351 for training data and 0.9267 for data validation

In [1]:
```
import os
os.environ['KAGGLE_USERNAME'] = #INPUT USERNAME
os.environ['KAGGLE_KEY'] = #INPUT PASSWORD

!kaggle datasets download -d ashishsaxena2209/animal-image-datasetdog-cat-and-panda
```

```
Downloading animal-image-dataset-dog-cat-and-panda.zip to /content 100% 375M/376M [00:12<00:00,
39.0MB/s]
100% 376M/376M [00:12<00:00, 32.2MB/s]
```

In [2]:
```
!unzip -q animal-image-datasetdog-cat-and-panda.zip -d .
```

In [3]:
```
os.listdir('/content/animals')
```

Out[3]:
```
['animals', 'dogs', 'images', 'cats', 'panda']
```

In [4]:
```
os.listdir('/content/animals/animals')
```

Out[4]:
```
['dogs', 'cats', 'pandas']
```

In [5]:
```
#Checking the number of files print('cats:',
len(os.listdir('/content/animals/animals/cats'))) print('dogs:',len(os.listdir('/ content/animals/animals/
dogs'))) print('panda:',len(os.listdir('/content/animals/animals/panda')))
```

```
cats: 1000
dogs: 1000
pandas: 1000
```

In[6]:
```
base_dir= '/content/animals/animals/'
```

In [7]:
```
#separating files into training files and validation files !pip install split-folders tqdm #splitfolder
 installation

import splitfolders base_dir
= '/content/animals/animals/' splitfolders.ratio(base_dir,
output=base_dir, ratio=(.8, .2))
```

```
Collecting split-folders Downloading
    https://files.pythonhosted.org/packages/b8/5f/3c2b2f7ea5e047c8cdc3bb00ae58
2c5438fcdbbedcc23b3cc1c2c7aae642/split_folders-0.4.3-py3-none-any.whl Requirements already satisfied:
tqdm in /usr/local/lib/python3.7/dist-packages (4.41.1)
```

```
Installing collected packages: split-folders Successfully installed split-
folders-0.4.3
```

```
Copying files: 3000 files [00:00, 3029.62 files/s]
```

In [8]:
```
os.listdir('/content/animals/animals/')
```

Out[8]:
```
['train', 'dogs', 'cats', 'panda', 'val']
```

In [9]:
```
os.listdir('/content/animals/animals/train') os.listdir('/content/animals/animals/
val')
```

Out[9]:
```
['dogs', 'cats', 'pandas']
```

In [10]:
```
#Checking the amount of training data and validation train_panda = os.listdir('/
content/animals/animals/train/panda') train_cats = os.listdir('/content/animals/animals/train/cats')
train_dogs = os.listdir('/content/animals/animals/train/dogs') val_panda = os.listdir('/content/
animals/animals/val/panda') val_cats = os.listdir('/content/animals/ animals/val/cats') val_dogs =
os.listdir('/content/animals/animals/val/dogs')



print('Total data training pandas:', len(train_panda)) print('Total data training cats:',
len(train_cats)) print('Total data training dogs:', len(train_dogs)) print('Total data
validation pandas:', len(val_panda)) print('Total data validation cats:', len(val_cats))
print('Total data validation dogs:', len(val_dogs))
```

```
Number of data training pandas: 800
Number of data training cats: 800
Number of data training dogs: 800
Number of data validation pandas: 200
Number of data validation cats: 200
Number of data validation dogs: 200
```

In [11] :
```
import numpy as np import
pandas as pd import
tensorflow as tf from tensorflow . _
```

In [12]:
```
data_training_generator = ImageDataGenerator( rescale=1./255,
                                              rotation_range=20,
                                              horizontal_flip=True,
                                              shear_range = 0.2, fill_mode =
                                              'nearest')


data_testing_generator = ImageDataGenerator(
                                              rescale=1./255,
                                              rotation_range=20,
                                              horizontal_flip=True, shear_range
                                              = 0.2, fill_mode = 'nearest')
```

In [13]:
```
#initiate training data variables and data testing
train_dir = os.path.join(base_dir, 'train') test_dir =
os.path.join(base_dir, 'val')
```

In [14]:
```
training_generator = data_training_generator.flow_from_directory(
                     train_dir, # training data directory target_size=(100, 100),
                     # change the resolution of all images to 150x150 px batch_size=9,
```

```
class_mode='categorical')

testing_generator = data_testing_generator.flow_from_directory( test_dir, # data directory testing target_size=(100,
                    100), # change resolution of all images to 150x150 pixels
                    batch_size=9, class_mode='categorical')



#class directory
training_generator.class_indices
```

```
Found 2400 images belonging to 3 classes.
Found 600 images belonging to 3 classes. {'cats': 0, 'dogs': 1, 'pandas':
 2}
```

Out[14]:

In [19]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.applications import ResNet152V2

#Developing the neural network architecture using the transfer learning model ResNet152V2 model = tf.keras.models.Sequential([

        ResNet152V2(weights="imagenet", include_top =False, input_tensor=Input(shape=(100 , 10 tf . .MaxPooling2D(2,2), #tf.keras.layers.Dropout(0.3 ) ,
        tf . .layers.Dropout(0.3), tf.keras.layers.Dense(3, activation='softmax', name='class_output')









            #the softmax activation function is suitable for data with more than 2 classes
])
model. layers[0].trainable = False

#model illustration
model.summary()

#Selection of loss, optimizer, and metrics
model.compile(loss='categorical_crossentropy', #data is multiclass optimizer='adam', metrics=['accuracy'])



#stop training when accuracy reaches 92% with callback class myCallback(tf.keras.callbacks.Callback): def
on_epoch_end(self, epoch, logs={ }):

      if(logs.get('accuracy')>0.92 and logs.get('val_accuracy')>0.92): print("\nTraining and validation accuracy has
        reached >92%!") self.model.stop_training = True


history = model.fit( training_generator,
            # displays the accuracy of training data steps_per_epoch=50, # number of steps for each epoch
            training data epochs=100, # total number of epochs validation_data=testing_generator, # displays accuracy
            of data testing validation_steps=50, verbose=1, # verbose value
            1 to add loading animation callbacks = myCallback() )
```

```
Model: "sequential_2"
_________________________________________________________________
Layers (types)                 Output Shapes              param#
=================================================================
resnet152v2 (Functional)       (None, 4, 4, 2048)         58331648
_________________________________________________________________
conv2d (Conv2D)                (None, 2, 2, 64)           1179712
_________________________________________________________________
```

max_pooling2d_9 (MaxPooling2	(None, 1, 1, 64)	0
flatten_2 (Flatten)	(None, 64)	0
dense_2 (Dense)	(None, 512)	33280
dropout_2 (Dropout)	(None, 512)	0
class_output (Dense)	(None, 3)	1539

Total params: 59,546,179
Trainable params: 1,214,531
Non-trainable params: 58,331,648

Epoch 1/100 50/50
[==========] - 24s 260ms/step - loss: 1.4255 - accuracy : 0.751 1 - val_loss: 0.6817 - val_accuracy: 0.8533 Epoch 2/100 50/50
[==========] - 10s 203ms/step - loss: 0.7030 -
accuracy: 0.811 1
- val_loss: 0.3113 - val_accuracy: 0.8600 Epoch 3/100 50/50 [========== ===] - 10s 202ms/step - loss: 0.5014 - accuracy: 0.833 3 - val_loss: 0.3846 - val_accuracy: 0.8556 Epoch 4/100 50/50 [=========

==========] - 10s 203ms/step - loss: 0.5579 - accuracy: 0.788 9 - val_loss: 0.8651 - val_accuracy: 0.8133 Epoch 5/100 50/ 50
[==========] - 10s 204ms/step - loss: 0.4751 -
accuracy: 0.846 7
- val_loss: 0.4625 - val_accuracy: 0.8644 Epoch 6/100 50/50 [==========] - 10s 203ms/step - loss: 0.5178 - accuracy: 0.824 4 - val_loss: 0.3704 - val_accuracy: 0.8489 Epoch 7/100 50/50

[========== ====] - 10s 202ms/step - loss: 0.4254 - accuracy: 0.846 7 - val_loss: 0.4256 - val_accuracy: 0.8511 Epoch 8/100 50/50
[======== ==========] - 10s 202ms/step - loss: 0.3830 - accuracy:
0.826 7 - val_loss:
0.3767 - val_accuracy: 0.8556 Epoch 9/100 50/50 [==== ==========] - 10s 201ms/step - loss: 0.5282 - accuracy: 0.811 1 - val_loss: 0.2901 - val_accuracy: 0.8711 Epoch 10/100 50/50

[==========] - 10s 202ms/step - loss: 0.3887 - accuracy: 0.840 0 - val_loss: 0.3725 - val_accuracy: 0.8356 Epoch 11/100 50/50
[==========] - 10s 205ms/step - loss: 0.4358 -
accuracy: 0.870 2
- val_loss: 0.2787 - val_accuracy: 0.8778 Epoch 12/100 50/50 [========== =========] - 10s 204ms/step - loss: 0.2625 - accuracy: 0.882 2 - val_loss: 0.3776 - val_accuracy: 0.8867 Epoch 13/100 50/50 [===

==========] - 10s 203ms/step - loss: 0.3894 - accuracy: 0.857 8 - val_loss: 0.3043 - val_accuracy: 0.8756 Epoch 14/100 50/50
[==========] - 10s 204ms/step - loss: 0.2552 -
accuracy: 0.886 7 -
val_loss: 0.3243 - val_accuracy: 0.8600 Epoch 15/100 50/50 [==========] - 10s 203ms/step - loss : 0.4139 - accuracy: 0.865 8 - val_loss: 0.3233 - val_accuracy: 0.8800 Epoch 16/100 50/50

[========== ===] - 10s 203ms/step - loss: 0.3857 - accuracy: 0.866 7 - val_loss: 0.3483 - val_accuracy: 0.8867 Epoch 17/100 50/50
[========== ==========] - 10s 204ms/step - loss: 0.4086 -
accuracy: 0.865 8 -
val_loss: 0.4442 - val_accuracy: 0.8511 Epoch 18/100

50/50 [==========] - 10s 205ms/step - loss: 0.3151 - accuracy: 0.902 2 - val_loss: 0.3665 - val_accuracy: 0.8756 Epoch 19/100 50/50

[==========] - 10s 204ms/ step - loss: 0.3175 - accuracy: 0.873 3 - val_loss: 0.3006 - val_accuracy: 0.8800 Epoch 20/100 50/50

[========== ======] - 10s 205ms/step - loss: 0.3005 - accuracy: 0.880 0 - val_loss: 0.2579 - val_accuracy: 0.9133 Epoch 21/100 50/50 [======

==========] - 10s 204ms/step - loss: 0.3298 - accuracy: 0.880 0 - val_loss: 0.3485 - val_accuracy: 0.8844 Epoch 22/100 50/50 [== ==========] - 10s 203ms/step - loss: 0.3203 - accuracy: 0.872 5 - val_loss: 0.4299 - val_accuracy: 0.8756 Epoch 23/100 50/50 [==========] - 10s 203ms/step - loss: 0.3414 - accuracy: 0.863 5 - val_loss: 0.3099 - val_accuracy: 0.8622 Epoch 24/100 50/50 [==========] - 10s 203ms/step - loss: 0.2718 - accuracy: 0.891 1 - val_loss: 0.3435 - val_accuracy: 0.8578 Epoch 25/100 50/50 [========== ====] - 10s 203ms/step - loss: 0.3011 - accuracy: 0.902 2 - val_loss: 0.3160 - val_accuracy: 0.8889 Epoch 26/100 50/50 [======== ==========] - 10s 204ms/step - loss: 0.3430 - accuracy: 0.870 2 - val_loss: 0.3773 - val_accuracy: 0.8644 Epoch 27/100 50 /50

[==========] - 10s 205ms/step - loss: 0.2804 - accuracy: 0.900 0 - val_loss : 0.3433 - val_accuracy: 0.8956 Epoch 28/100 50/50

[==========] - 10s 207ms/step - loss: 0.3012 - accuracy: 0.906 7 - val_loss: 0.3189 - val_accuracy: 0.8667 Epoch 29/100 50/50

[========== =====] - 10s 204ms/step - loss: 0.3427 - accuracy: 0.880 0 - val_loss: 0.2917 - val_accuracy: 0.8622 Epoch 30/100 50/50 [=======

==========] - 10s 204ms/step - loss: 0.3745 - accuracy: 0.875 6 - val_loss: 0.3585 - val_accuracy: 0.8778 Epoch 31/100 50/50 [=== ==========] - 10s 204ms/step - loss: 0.2848 - accuracy: 0.866 7 - val_loss: 0.3091 - val_accuracy: 0.8800 Epoch 32/100 50/50 [==========] - 10s 205ms/step - loss: 0.2645 - accuracy : 0.904 4 - val_loss: 0.3033 - val_accuracy: 0.8756 Epoch 33/100 50/50 [==========] - 10s 206ms/step - loss: 0.3154 - accuracy: 0.917 8 - val_loss: 0.3854 - val_accuracy: 0.8711 Epoch 34/100 50/50 [========== ===] - 10s 205ms/step - loss: 0.3259 - accuracy: 0.908 9 - val_loss: 0.3429 - val_accuracy: 0.8756 Epoch 35/100 50/50 [========= ==========] - 10s 204ms/step - loss: 0.2904 - accuracy: 0.875 6 - val_loss: 0.2550 - val_accuracy: 0.9000 Epoch 36/100 50/ 50

[==========] - 10s 206ms/step - loss: 0.3858 - accuracy: 0.893 3 - val_loss: 0.3822 - val_accuracy: 0.8600 Epoch 37/100 50/50

[==========] - 10s 205ms/step - loss: 0.2727 - accuracy: 0.913 3 - val_loss: 0.3535 - val_accuracy: 0.8956 Epoch 38/100 50/50

[========== ====] - 10s 207ms/step - loss: 0.2442 - accuracy: 0.917 8 - val_loss: 0.3195 - val_accuracy: 0.8867 Epoch 39/100 50/50 [========

==========] - 10s 204ms/step - loss: 0.2355 - accuracy: 0.908 3 - val_loss: 0.2758 - val_accuracy: 0.8867 Epoch 40/100

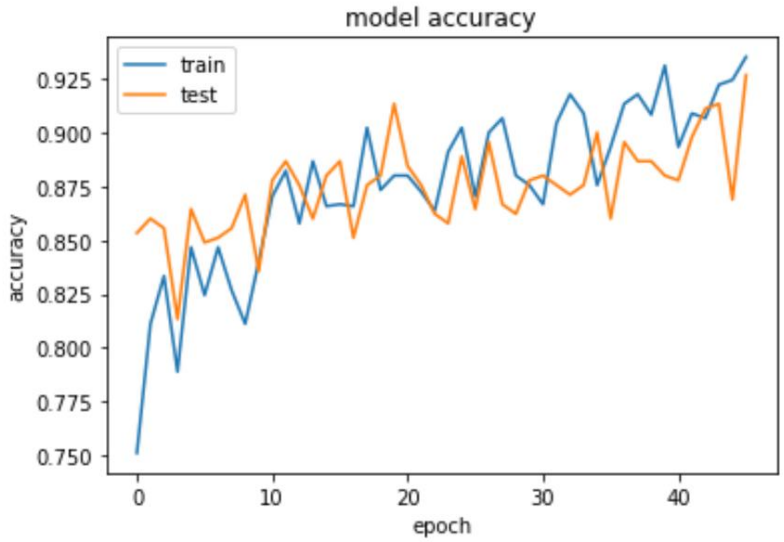
```
50/50 [==============================] - 10s 204ms/step - loss: 0.1964 - accuracy: 0.931 1 - val_loss: 0.3358 - val_accuracy:
0.8800 Epoch 41/100 50/50 [==============================]
- 10s 204ms/ step
- loss: 0.3620 - accuracy: 0.893 3 - val_loss: 0.2750 - val_accuracy: 0.8778 Epoch 42/100 50/50 [==========================
======] - 10s 205ms/step - loss: 0.2491 - accuracy: 0.908 9 -
val_loss: 0.3095 -
val_accuracy: 0.8978 Epoch 43/100 50/50 [====== ==================] - 10s 205ms/step - loss: 0.2793 - accuracy: 0.906 7 -
val_loss: 0.2734 - val_accuracy: 0.9111 Epoch 44/100 50/50 [==

==============================] - 10s 204ms/step - loss: 0.2852 - accuracy: 0.922 2 - val_loss: 0.2756 - val_accuracy: 0.9133
Epoch 45/100 50/50 [==============================] -
10s 205ms/step -
loss: 0.2458 - accuracy: 0.924 4 - val_loss: 0.3566 - val_accuracy: 0.8689 Epoch 46/100 50/50 [============================== ]
- 10s 204ms/step - loss: 0.2237 - accuracy: 0.935 1 - val_loss:
0.2003 -
val_accuracy: 0.9267
```

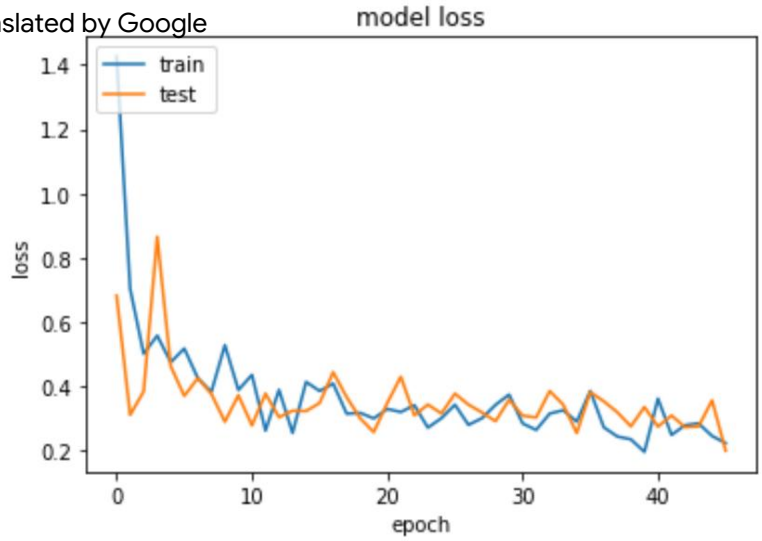
Training and validation accuracy has reached >92%!

In [24]:
```
import matplotlib. pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy']) plt.title('model accuracy')
plt.ylabel('accuracy') plt.xlabel('epoch' )
plt.legend(['train', 'test'], loc='upper
left') plt.show()
```



In [25]:
```
plt. plot(history. history['loss'])
plt.plot(history.history['val_loss']) plt.title('model loss')
plt.ylabel('loss') plt.xlabel('epoch')
plt.legend(['train', 'test' ],
loc='upper left') plt.show()
```



In [26]:
```
models
```

Out[26]:
```
<tensorflow.python.keras.engine.sequential.Sequential at 0x7f610bb2cc90>
```

In [27]:
```
# Model conversion.
converter = tf.lite.TFLiteConverter.from_keras_model(model) tflite_model =
converter.convert()
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/utils/generic_utils.py:49 7: CustomMaskWarning: Custom mask layers
require a config and must override get_config.
When loading, the custom mask layer must be passed to the custom_objects argument.
    category=CustomMaskWarning)
INFO:tensorflow:Assets written to: /tmp/tmp66f40mi9/assets
```

In [28]:
```
with tf.io.gfile.GFile('model.tflite', 'wb') as f:
    f.write(tflite_model)
```

In [ ]:
```
```