Cat vs Dog Predictor Model

September 2, 2023

[1]:

```
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings('ignore')

from tensorflow import keras
from keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.utils import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras.preprocessing import image_dataset_from_directory

import os
import matplotlib.image as mpimg
```

[2]:

```
path = "C:/Users/hp.pc/Python/dataset"
classes = os.listdir(path)
classes
```

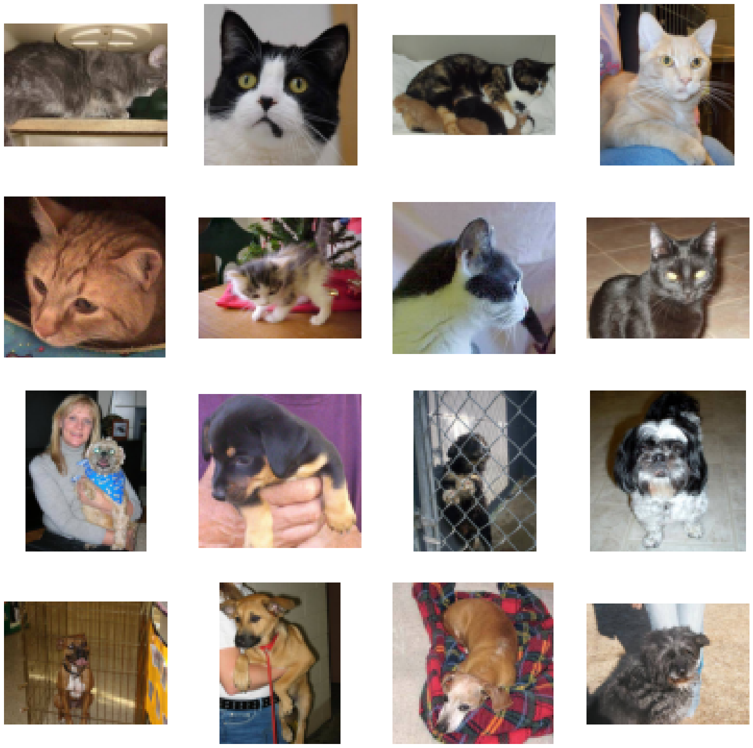
[2]:

```
['cats', 'dogs']
```

[3]:

```
fig = plt.gcf()
fig.set_size_inches(5, 5)

cat_dir = os.path.join('C:/Users/hp.pc/Python/dataset/cats')
dog_dir = os.path.join('C:/Users/hp.pc/Python/dataset/dogs')
cat_names = os.listdir(cat_dir)
dog_names = os.listdir(dog_dir)

pic_index = 210

cat_images = [os.path.join(cat_dir, fname)
                        for fname in cat_names[pic_index-8:pic_index]]
```

```
dog_images = [os.path.join(dog_dir, fname)
                        for fname in dog_names[pic_index-8:pic_index]]

for i, img_path in enumerate(cat_images + dog_images):
        sp = plt.subplot(4, 4, i+1)
        sp.axis('Off')

        img = mpimg.imread(img_path)
        plt.imshow(img)

plt.show()
```



[4]:

```
base_dir = 'C:/Users/hp.pc/Python/dataset'

# Create datasets
train_datagen = image_dataset_from_directory(base_dir,



 ↪= 1,


 ↪25,
 ↪32)
test_datagen = image_dataset_from_directory(base_dir,




 ↪= 1,

 ↪25,

 ↪32)
```

```
Found 8005 files belonging to 2 classes.
Using 6004 files for training.
Found 8005 files belonging to 2 classes.
Using 2001 files for validation.
```

[5]:

```
model = tf.keras.models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)),
        layers.MaxPooling2D(2, 2),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D(2, 2),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D(2, 2),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D(2, 2),

        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.BatchNormalization(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.1),
        layers.BatchNormalization(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.2),
        layers.BatchNormalization(),
        layers.Dense(1, activation='sigmoid')
])
```

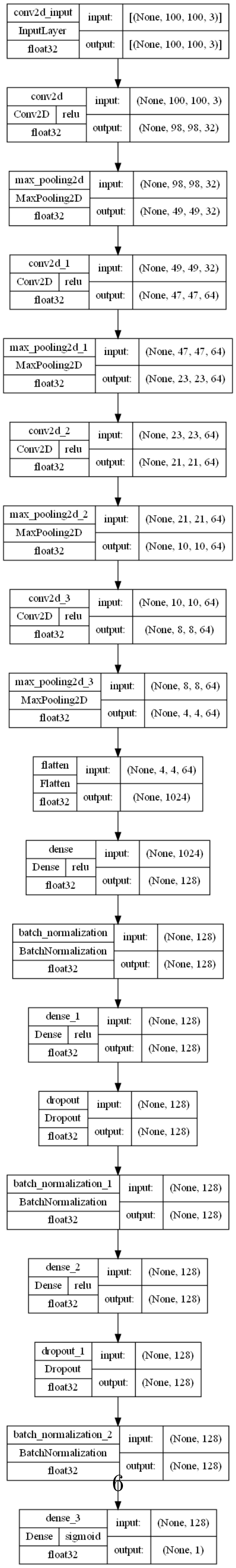
[6]:

```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 32)        896


 max_pooling2d (MaxPooling2  (None, 49, 49, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 47, 47, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 23, 23, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 21, 21, 64)        36928

 max_pooling2d_2 (MaxPoolin  (None, 10, 10, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 8, 8, 64)          36928

 max_pooling2d_3 (MaxPoolin  (None, 4, 4, 64)          0
 g2D)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 128)               131200

 batch_normalization (Batch  (None, 128)               512
 Normalization)

 dense_1 (Dense)             (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 batch_normalization_1 (Bat  (None, 128)               512
 chNormalization)

 dense_2 (Dense)             (None, 128)               16512

 dropout_1 (Dropout)         (None, 128)               0

 batch_normalization_2 (Bat  (None, 128)               512
 chNormalization)

 dense_3 (Dense)             (None, 1)                 129

=================================================================
Total params: 259137 (1012.25 KB)
Trainable params: 258369 (1009.25 KB)
Non-trainable params: 768 (3.00 KB)
_________________________________________________________________
```

[7]:
```
keras.utils.plot_model(
        model,
        show_shapes=True,
        show_dtype=True,
        show_layer_activations=True
)
```

[7]:


conv2d_input | InputLayer | float32 | input: [(None, 100, 100, 3)] | output: [(None, 100, 100, 3)]
conv2d | Conv2D | relu | float32 | input: (None, 100, 100, 3) | output: (None, 98, 98, 32)
max_pooling2d | MaxPooling2D | float32 | input: (None, 98, 98, 32) | output: (None, 49, 49, 32)
conv2d_1 | Conv2D | relu | float32 | input: (None, 49, 49, 32) | output: (None, 47, 47, 64)
max_pooling2d_1 | MaxPooling2D | float32 | input: (None, 47, 47, 64) | output: (None, 23, 23, 64)
conv2d_2 | Conv2D | relu | float32 | input: (None, 23, 23, 64) | output: (None, 21, 21, 64)
max_pooling2d_2 | MaxPooling2D | float32 | input: (None, 21, 21, 64) | output: (None, 10, 10, 64)
conv2d_3 | Conv2D | relu | float32 | input: (None, 10, 10, 64) | output: (None, 8, 8, 64)
max_pooling2d_3 | MaxPooling2D | float32 | input: (None, 8, 8, 64) | output: (None, 4, 4, 64)
flatten | Flatten | float32 | input: (None, 4, 4, 64) | output: (None, 1024)
dense | Dense | relu | float32 | input: (None, 1024) | output: (None, 128)
batch_normalization | BatchNormalization | float32 | input: (None, 128) | output: (None, 128)
dense_1 | Dense | relu | float32 | input: (None, 128) | output: (None, 128)
dropout | Dropout | float32 | input: (None, 128) | output: (None, 128)
batch_normalization_1 | BatchNormalization | float32 | input: (None, 128) | output: (None, 128)
dense_2 | Dense | relu | float32 | input: (None, 128) | output: (None, 128)
dropout_1 | Dropout | float32 | input: (None, 128) | output: (None, 128)
batch_normalization_2 | BatchNormalization | float32 | input: (None, 128) | output: (None, 128)
dense_3 | Dense | sigmoid | float32 | input: (None, 128) | output: (None, 1)

[8]:

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

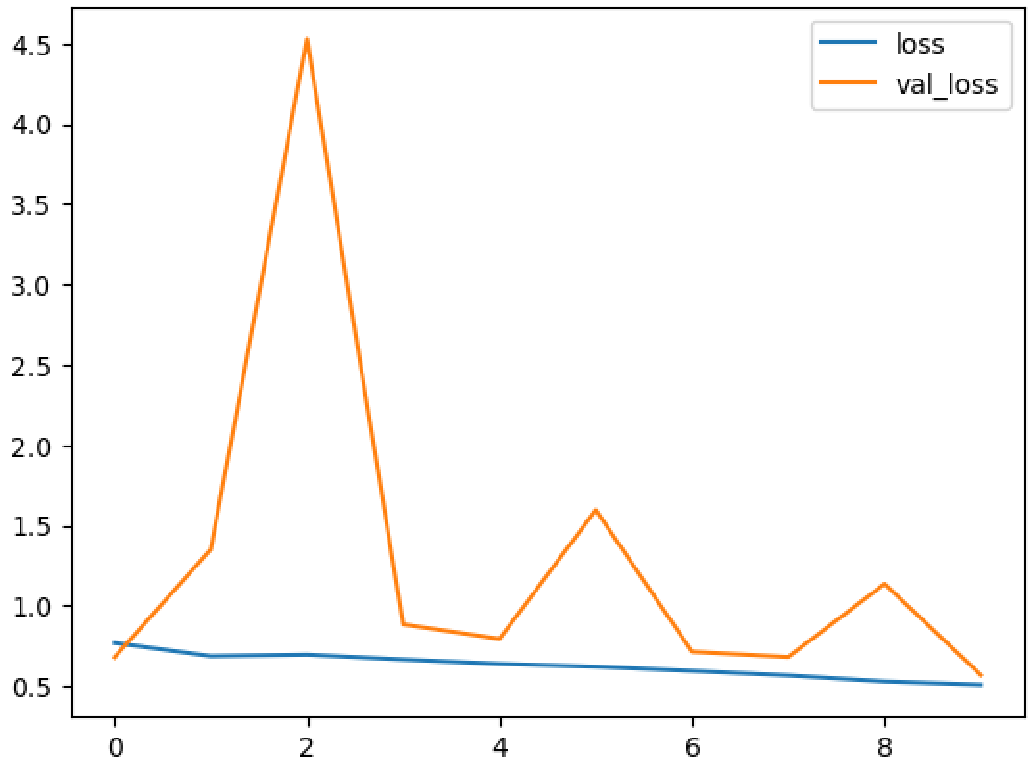
[9]:

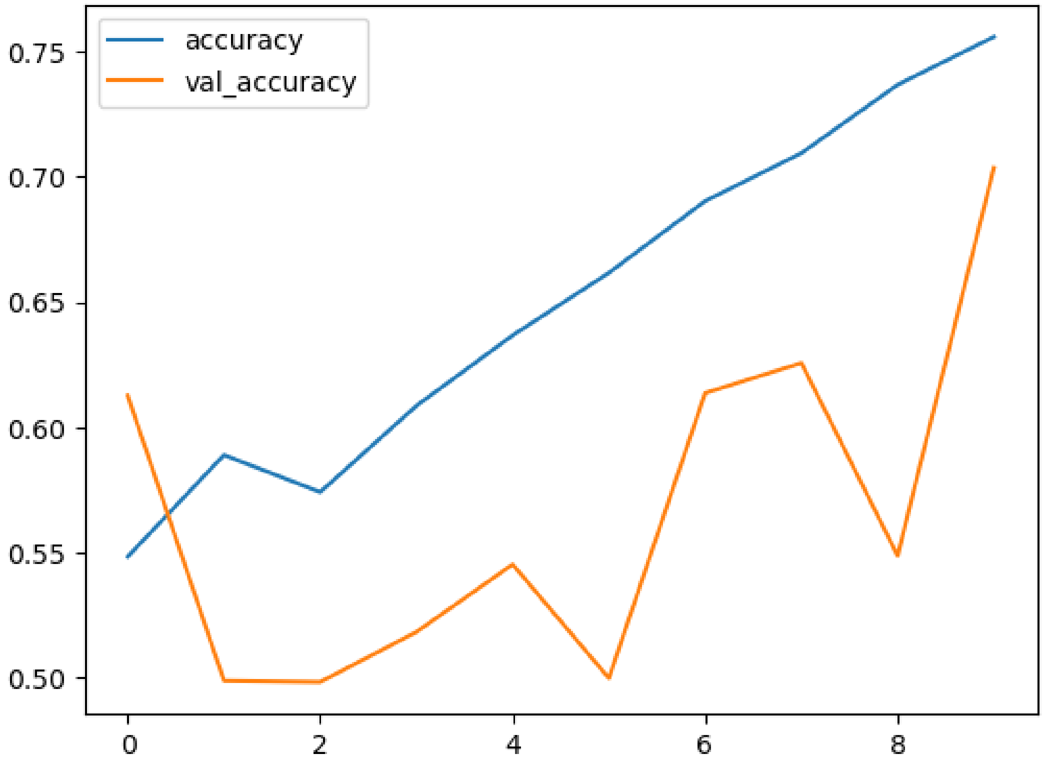
```
history = model.fit(train_datagen,
                epochs=10,
                validation_data=test_datagen)
```

```
Epoch 1/10
188/188 [==============================] - 111s 553ms/step - loss: 0.7683 -
accuracy: 0.5483 - val_loss: 0.6786 - val_accuracy: 0.6127
Epoch 2/10
188/188 [==============================] - 104s 552ms/step - loss: 0.6865 -
accuracy: 0.5889 - val_loss: 1.3504 - val_accuracy: 0.4988
Epoch 3/10
188/188 [==============================] - 104s 551ms/step - loss: 0.6949 -
accuracy: 0.5741 - val_loss: 4.5273 - val_accuracy: 0.4983
Epoch 4/10
188/188 [==============================] - 104s 552ms/step - loss: 0.6650 -
accuracy: 0.6086 - val_loss: 0.8833 - val_accuracy: 0.5182
Epoch 5/10
188/188 [==============================] - 104s 553ms/step - loss: 0.6380 -
accuracy: 0.6367 - val_loss: 0.7936 - val_accuracy: 0.5452
Epoch 6/10
188/188 [==============================] - 104s 553ms/step - loss: 0.6205 -
accuracy: 0.6617 - val_loss: 1.5957 - val_accuracy: 0.4998
Epoch 7/10
188/188 [==============================] - 105s 554ms/step - loss: 0.5941 -
accuracy: 0.6904 - val_loss: 0.7132 - val_accuracy: 0.6137
Epoch 8/10
188/188 [==============================] - 104s 552ms/step - loss: 0.5664 -
accuracy: 0.7095 - val_loss: 0.6806 - val_accuracy: 0.6257
Epoch 9/10
188/188 [==============================] - 104s 552ms/step - loss: 0.5289 -
accuracy: 0.7368 - val_loss: 1.1360 - val_accuracy: 0.5487
Epoch 10/10
188/188 [==============================] - 104s 550ms/step - loss: 0.5085 -
accuracy: 0.7560 - val_loss: 0.5685 - val_accuracy: 0.7036
```

[10]:

```
history_df = pd.DataFrame(history.history)
history_df.loc[:, ['loss', 'val_loss']].plot()
history_df.loc[:, ['accuracy', 'val_accuracy']].plot()
plt.show()
```

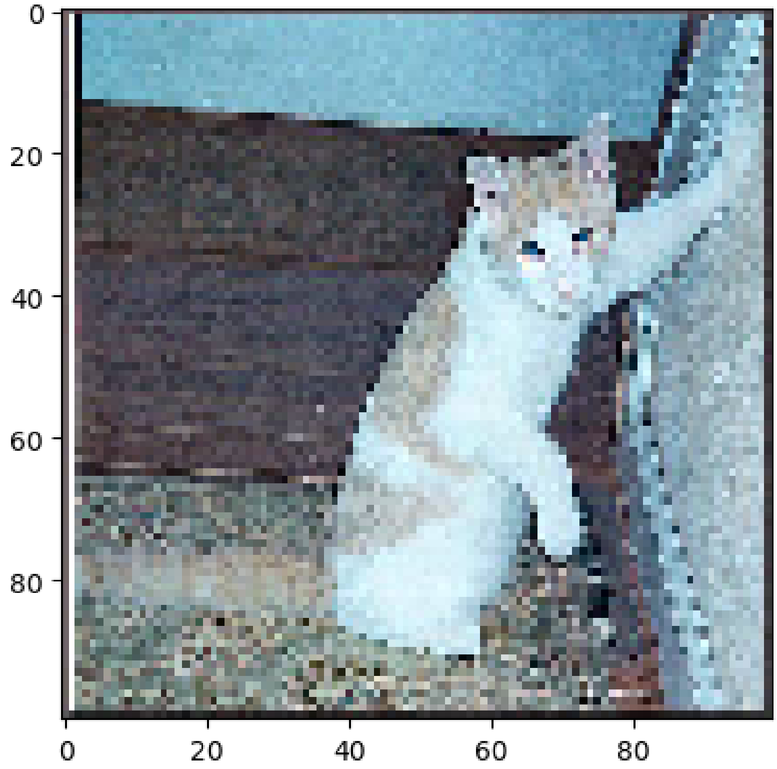

4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
2
4
6
8
loss
val_loss


0.75
0.70
0.65
0.60
0.55
0.50
0
2
4
6
8
accuracy
val_accuracy

[24]:

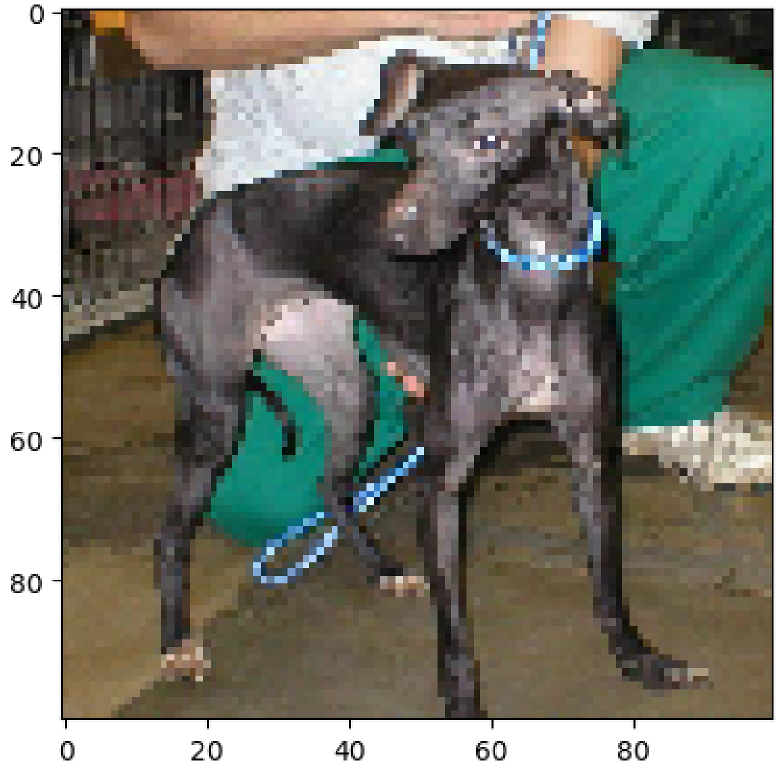
```
from keras.preprocessing import image

#Input image
test_image = image.load_img('C:/Users/hp.pc/Python/test_set/cats/cat.4001.
 ↪jpg',target_size=(100,100))

#For show image
plt.imshow(test_image)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image,axis=0)

# Result array
result = model.predict(test_image)


#Mapping result array with the main name list

if result >= 0.5:
    print("Cat")
else:
    print("Dog")
```

```
1/1 [==============================] - 0s 64ms/step
Cat
```



[16]:
```
test_image = image.load_img('C:/Users/hp.pc/Python/test_set/dogs/dog.4004.jpg',␣
 ↪target_size=(100, 100))

# For show image
plt.imshow(test_image)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)

# Result array
result = model.predict(test_image)
# Mapping result array with the main name list

if result >= 0.8:
    print("Dog")
else:
    print("Cat")
```

```
1/1 [==============================] - 0s 47ms/step
Dog
```


0
20
40
60
80
0
20
40
60
80

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```