Image Classification

Nikilas John | Mazidi

December 4th, 2022

Double-click (or enter) to edit

Importing the libraries

This section will import all of the libraries we need to complete this notebook

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import seaborn as sb
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential #importing our deep learing libraries
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Activat
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import imageio
from PIL import Image
from sklearn.model_selection import train_test_split
import cv2 as ocv
import random
```

Divide into train/test

As we can see by the output, the dataset is already split into train/test for us

```
print(os.listdir('//kaggle/input/dogvcat/dogVcat'))
train_dir = '/kaggle/input/dogvcat/dogVcat/training_set'
test_dir  = '/kaggle/input/dogvcat/dogVcat/test_set'
```

```
    ['test_set', 'training_set']
```

Reading the images into a dataframe

**This section is only here so that I can print a histogram based on how many dogs and cats there are in the training set they are a cat or a dog. This determination will be done by the file name, not by any classification algorithm

Reference: https://www.kaggle.com/code/bhuvanchennoju/hey-siri-is-it-a-or-class-f1-0-992/notebook

```
# category and filepath extraction helper functions
def category(path):
    return [file.split('.')[0] for file in os.listdir(path)]

def filename(path):
    return [file for file in os.listdir(path)]

# image names and labels
x_train_imgname = filename(train_dir)
x_test_imgname = filename(test_dir)
y_train_label = category(train_dir)

train_image_df = pd.DataFrame({ 'filename': x_train_imgname, 'category': y_train_label})
submission_image_df = pd.DataFrame({'filename': x_test_imgname})

# to see what the datafram looks like
train_image_df.head()
```

	filename	category
0	dog.3443.jpg	dog
1	dog.1942.jpg	dog
2	dog.375.jpg	dog
3	dog.3259.jpg	dog
4	cat.3498.jpg	cat

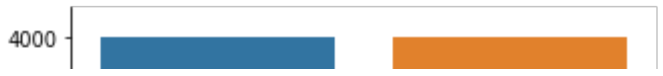
Histogram of the target classes

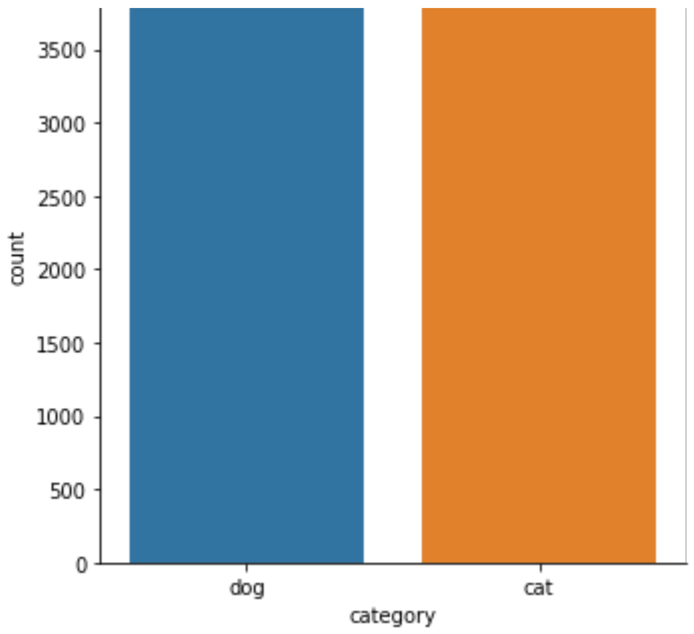
Now we will graph the target classes using a histogram

This will reveal that there are an equal amount of dogs and cats in this training file

```
sb.catplot(x="category", kind="count", data=train_image_df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f91bb21e4d0>
```





What to expect from this data set

With this dataset, I want to be able to train the algorithm to classify if the image is of a dog or a cat. I understand this is a binary decision, but this is also my first time experimenting with image classification, so I would rather tip my toe into the concept rather than dive into a multi-class classification.

Processing the data

We are only using the training set as the data set since it is large enough

Reference: https://www.kaggle.com/code/mohamedadel452/nn-binary-classifier-cats-vs-dogs

```
dir = '/kaggle/input/dogs-cats-images/dataset/training_set'
classes = ['cats','dogs']
num_of_img_per_class = 4000
#Reading data
data = []
labels = []
x_pixels=300
y_pixels=300
class_counter=1
for category in os.listdir(dir):
    if class_counter==1:
        print("Loading Data:", end=" ")
    new_dir = os.path.join(dir,category)
    for img in os.listdir(new_dir):
        if len(data)<(num_of_img_per_class*class_counter):
            img_path = os.path.join(new_dir,img)
            if (np.all(np.array(ocv.imread(img_path,0)))==0):
                data.append(ocv.resize(ocv.imread(img_path,0),(x_pixels,y_pixels)))
                labels.append(classes.index(category))
                if len(data)/(num_of_img_per_class*2)*100%20==0:
                    print(int(len(data)/(num_of_img_per_class*2)*100),"%", end="  ")
    class_counter=class_counter+1
    if class_counter==3:
        print(". Complete.", end=" ")
combined = list(zip(data,labels))
random.shuffle(combined)
data, labels = zip(*combined)
Data = np.array(data)/255
Labels = np.array(labels)
n_col = Data.shape[1]
n_rows = Labels.shape[0]
x_train,x_test,y_train,y_test = train_test_split(Data,Labels, train_size = .8)
```

```
     Loading Data: 20 %  40 %  60 %  . Complete.
```

Data Preparation

Time to prepare the data for training

This section was taken out of the textbook, but changed to fit the 300px/300px photos we are using

```
x_train, x_test = x_train / 300.0, x_test / 300.0
# convert class vectors to binary class matrices
y_train = tf.keras.utils.to_categorical(y_train, 2)
y_test = tf.keras.utils.to_categorical(y_test, 2)
```

Sequential modeling

Here we will use a general Sequential model to evaluate its accuracy

Building the Model

First, we are going to use the setup detailed in the textbook to see if that produced good results for this dataset

```
model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(300, 300)),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(2, activation='softmax'),
])
model.summary()
model.compile(loss='categorical_crossentropy',
optimizer='rmsprop',
metrics=['accuracy'])
```

```
    Model: "sequential_6"
    _________________________________________________________________
    Layer (type)                Output Shape              Param #
    =================================================================
    flatten_10 (Flatten)        (None, 90000)             0

    _________________________________________________________________
    dense_11 (Dense)            (None, 512)               46080512

    _________________________________________________________________
    dropout_2 (Dropout)         (None, 512)               0

    _________________________________________________________________
    dense_12 (Dense)            (None, 512)               262656

    _________________________________________________________________
    dropout_3 (Dropout)         (None, 512)               0

    _________________________________________________________________
    dense_13 (Dense)            (None, 2)                 1026
    =================================================================
    Total params: 46,344,194
    Trainable params: 46,344,194
    Non-trainable params: 0
    _________________________________________________________________
```

Train and Evaluate

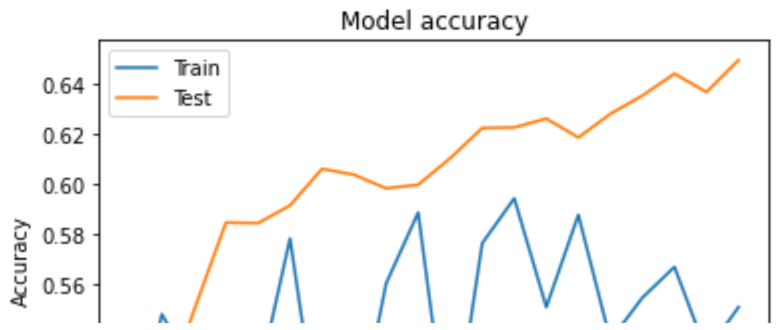
```
history = model.fit(x_train, y_train,
batch_size=128,
epochs=20,
verbose=1,
validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss: ', score[0])
print('Test accuracy: ', score[1])
```

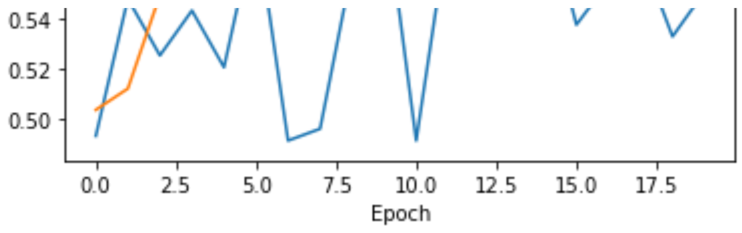
```
    Epoch 1/20
    34/34 [==============================] - 17s 484ms/step - loss: 0.7615 - accuracy: 0.
    Epoch 2/20
    34/34 [==============================] - 14s 422ms/step - loss: 0.6949 - accuracy: 0.
    Epoch 3/20
    34/34 [==============================] - 14s 408ms/step - loss: 0.6862 - accuracy: 0.
    Epoch 4/20
    34/34 [==============================] - 14s 419ms/step - loss: 0.6741 - accuracy: 0.
    Epoch 5/20
    34/34 [==============================] - 14s 409ms/step - loss: 0.6710 - accuracy: 0.
    Epoch 6/20
    34/34 [==============================] - 14s 408ms/step - loss: 0.6712 - accuracy: 0.
```

```
Epoch 7/20
34/34 [==============================] - 14s 417ms/step - loss: 0.6602 - accuracy: 0.
Epoch 8/20
34/34 [==============================] - 14s 408ms/step - loss: 0.6656 - accuracy: 0.
Epoch 9/20
34/34 [==============================] - 14s 417ms/step - loss: 0.6595 - accuracy: 0.
Epoch 10/20
34/34 [==============================] - 14s 406ms/step - loss: 0.6576 - accuracy: 0.
Epoch 11/20
34/34 [==============================] - 14s 420ms/step - loss: 0.6499 - accuracy: 0.
Epoch 12/20
34/34 [==============================] - 14s 406ms/step - loss: 0.6572 - accuracy: 0.
Epoch 13/20
34/34 [==============================] - 14s 417ms/step - loss: 0.6458 - accuracy: 0.
Epoch 14/20
34/34 [==============================] - 14s 412ms/step - loss: 0.6410 - accuracy: 0.
Epoch 15/20
34/34 [==============================] - 14s 410ms/step - loss: 0.6447 - accuracy: 0.
Epoch 16/20
34/34 [==============================] - 14s 426ms/step - loss: 0.6383 - accuracy: 0.
Epoch 17/20
34/34 [==============================] - 14s 408ms/step - loss: 0.6377 - accuracy: 0.
Epoch 18/20
34/34 [==============================] - 14s 421ms/step - loss: 0.6302 - accuracy: 0.
Epoch 19/20
34/34 [==============================] - 14s 405ms/step - loss: 0.6345 - accuracy: 0.
Epoch 20/20
34/34 [==============================] - 14s 418ms/step - loss: 0.6245 - accuracy: 0.
Test loss:  0.7491402626037598
Test accuracy:  0.5509433746337891
```

Plotting The Training History

```python
# Plot training & validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```





Evaluation of Code

This graph has a lot to unpack. First of all, we can see that the accuracy range is from 0.5-0.64. The peak of the model accuracy on the test data only reached up to 0.64. The accuracy also flucuates a lot for the training data. I can only assume that it is due my lack of knowledge on the subject. For a dataset that only has clear images of dogs and cats, I thought it would be better than that. Even worse than this, we can see that we have a high loss. This really was not a great model to classify the data.

CNN Model

Reshaping the data

We have to now reshape the data to make sure CNN runs correctly

```
x_train = np.expand_dims(x_train, axis=3)
x_test = np.expand_dims(x_test, axis=3)
```

Building the Model

```
model = tf.keras.models.Sequential([
tf.keras.Input(shape=(300, 300, 1)),
tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
tf.keras.layers.Flatten(),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(2, activation="softmax"),
])
model.summary()
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
     Model: "sequential_17"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_45 (Conv2D)           (None, 298, 298, 32)      320
_________________________________________________________________
max_pooling2d_46 (MaxPooling (None, 149, 149, 32)      0
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 147, 147, 64)      18496
_________________________________________________________________
max_pooling2d_47 (MaxPooling (None, 73, 73, 64)        0
_________________________________________________________________
flatten_26 (Flatten)         (None, 341056)            0
_________________________________________________________________
dropout_11 (Dropout)         (None, 341056)            0
_________________________________________________________________
dense_28 (Dense)             (None, 2)                 682114
=================================================================
Total params: 700,930
Trainable params: 700,930
Non-trainable params: 0
_________________________________________________________________
```

Train and Evaluate

```
history = model.fit(x_train, y_train,
                    batch_size=218,
                    epochs=20,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

```
Epoch 1/20
20/20 [==============================] - 211s 10s/step - loss: 0.6940 - accuracy: 0.4
Epoch 2/20
20/20 [==============================] - 206s 10s/step - loss: 0.6932 - accuracy: 0.5
Epoch 3/20
20/20 [==============================] - 205s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 4/20
20/20 [==============================] - 209s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 5/20
20/20 [==============================] - 209s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 6/20
20/20 [==============================] - 209s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 7/20
20/20 [==============================] - 206s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 8/20
20/20 [==============================] - 206s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 9/20
20/20 [==============================] - 210s 11s/step - loss: 0.6931 - accuracy: 0.5
Epoch 10/20
20/20 [==============================] - 206s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 11/20
20/20 [==============================] - 209s 10s/step - loss: 0.6931 - accuracy: 0.5
```

```
Epoch 12/20
20/20 [==============================] - 208s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 13/20
20/20 [==============================] - 206s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 14/20
20/20 [==============================] - 206s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 15/20
20/20 [==============================] - 208s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 16/20
20/20 [==============================] - 205s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 17/20
20/20 [==============================] - 207s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 18/20
20/20 [==============================] - 207s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 19/20
20/20 [==============================] - 206s 10s/step - loss: 0.6931 - accuracy: 0.5
Epoch 20/20
20/20 [==============================] - 210s 11s/step - loss: 0.6931 - accuracy: 0.5
```

```
print('Test loss: ', score[0])
print('Test accuracy: ', score[1])
```

```
Test loss:  0.7491402626037598
Test accuracy:  0.5509433746337891
```

Evaluation of the metrics

As we can see, this had the same accuracy and loss as the generic Sequential model. This shows that this algorithm only ended up taking longer with no improvement to the metrics.

```
### Image Classification Write-Up

Performance of Generic Sequential Modeling: <
Speed of execution: this was the fastest, tak
<br>
Loss: This value was extremely high on the tr
Accuracy: The accuracy of this algorithm was 
this was a binary classification, you statist
the correct prediction, and yet the algorithm
only this, but the dataset was also dogs vers
tell knowing that humans are able to tell the
blink of an eye.

Performance of the CNN Modeling: <br>
Speed of execution: this was extremely slow, 
complete execution, not only this, but all of
accuracy and loss did not fluctuate like the 
Loss: The loss was the exact same <br>
```

Image Classification Write-Up

Performance of Generic Sequential Modeling:
Speed of execution: this was the fastest, taking around 5 minutes to complete
Loss: This value was extremely high on the training data
Accuracy: The accuracy of this algorithm was surprisingly low. Knowing that this was a binary classification, you statistically have a 50% chance to get the correct prediction, and yet the algorithm still could not crack 80%. Not only this, but the dataset was also dogs versus

```
Accuracy: The accuracy was the exact same <br

Overall Thoughts/Questions: <br>
If it was not obvious enough, my understandin
work.
This assignment was extremely difficult for m
concepts at all. This was my best attempt at
```

cats. This should be an easy tell knowing that humans are able to tell the difference between them in a blink of an eye.

Performance of the CNN Modeling:
Speed of execution: this was extremely slow, taking around 66 minutes to complete execution, not only this, but all of the values were the same. The accuracy and loss did not fluctuate like the generic Sequential model did
Loss: The loss was the exact same
Accuracy: The accuracy was the exact same

Overall Thoughts/Questions:
If it was not obvious enough, my understanding of deep learning could use some work. This assignment was extremely difficult for me since I did not understand the concepts at all. This was my best attempt at compiling a presentable product

Colab paid products - Cancel contracts here