Nathan Burwig
Math 87 HW 7
Due 11/12/2022

What follows is the complete output of my jupyter notebook file, formatted to make it a decent big more readable. At the end, I will answer all of the questions individually, though I believe most of the questions are answered throughout the notebook.

Is it a hot dog, or not (dog)?¶

This is an attempt at classifying a set of images as either hot dogs or not hot dogs. I'll be using tensorflow, and a hotdog image set with training data found online.

```
import os
from os.path import join
import tensorflow as tf
import matplotlib.pyplot as plt
from IPython.display import Image, display
import numpy as np
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.11.0
```

```
image_size = 224

## imports images and outputs an array
def prep_images(img_paths, img_height=image_size, img_width=image_size):
    ## loading images in the image paths
    imgs = [load_img(img_path, target_size=(img_height, img_width)) for img_path in img_paths]



    ## converting images to arrays and returning
    img_array = np.array([img_to_array(img) for img in imgs])
    output = preprocess_input(img_array)
    print(np.shape(imgs[0]))
    print(np.shape(img_array[0]))
    return (output)
```

We can now get the training data, they will be in categories either "hot dog" or "not hot dog"

```
train_hotdog_dir    = 'C:\\Users\\nburw\\OneDrive\\Desktop\\train\\hot_dog'
train_nothotdog_dir = 'C:\\Users\\nburw\\OneDrive\\Desktop\\train\\not_hot_dog'
```

Quick test to make sure directories are okay

```
print(train_hotdog_dir)
print(train_nothotdog_dir)
```

```
C:\Users\nburw\OneDrive\Desktop\train\hot_dog
C:\Users\nburw\OneDrive\Desktop\train\not_hot_dog
```

```
train_hotdog_filenames      = os.listdir(train_hotdog_dir)
train_nothotdog_filenames   = os.listdir(train_nothotdog_dir)
```

```
hotdog_paths       = [(train_hotdog_dir    + '\\' + train_hotdog_filenames[i])
                      for i in range(len(train_hotdog_filenames))]
nothotdog_paths    = [(train_nothotdog_dir + '\\' + train_nothotdog_filenames[i])
                      for i in range(len(train_nothotdog_filenames))]
```

Cool, so now we have a way to get paths for individual images, so we can start getting tensorflow out now. That was a bit of a lie, but first we need to get the training set setup. Training sets are gonna be an array of x,y pairs where x is an image (as an array) and y is a value from 0 to 1 where 1 means it's a hot dog, and 0 means it's not a hot dog.

```
prepped_hotdogs     = prep_images(hotdog_paths)
prepped_nothotdogs  = prep_images(nothotdog_paths)
```

Now we convert into the xy pair

```
x_train = np.concatenate((prepped_hotdogs, prepped_nothotdogs))
y_train = np.array([(1 if n <= 249 else 0) for n in range(len(x_train))])

print(np.shape(x_train), np.shape(y_train))
```

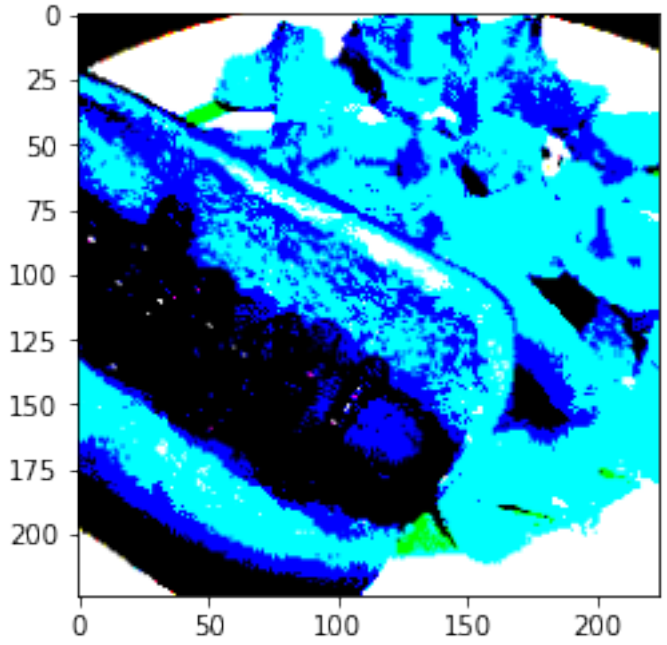
```
(498, 224, 224, 3) (498,)
```

```
## now we shuffle them in unison
assert len(x_train) == len(y_train)

##    Example    ##
a = np.array([1, 2, 3, 4, 5])
b = np.array([-1, -2, -3, -4, -5])

p = np.random.permutation(len(a))
print(a[p])
print(b[p])
## Example (end) ##

## actual unison shuffle of data
shuffler = np.random.permutation(len(x_train))
y_train = y_train[shuffler]
x_train = x_train[shuffler]
```

```
[4 1 5 2 3]
[-4 -1 -5 -2 -3]
```

```
import matplotlib.pyplot as plt
plt.imshow(prepped_hotdogs[1])
print(np.shape(prepped_hotdogs[0]))
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
(224, 224, 3)
```



Okay so now we have our training data setup. We are going to setup our model using tf.keras now.

Developing Models¶

For our first attempt, we will be using a convolutional network

```
model_v1 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, kernel_size=(3,3), activation='relu', input_shape=(224,224,3)),
    tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(2)
    ])
```

Here we define a loss function

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

```
predictions = model_v1(x_train[:1]).numpy()
print(predictions)
```

```
[[-12.270783   -7.8165493]]
```

```
tf.nn.softmax(predictions).numpy()
```

Out[ ]:

```
array([[0.01149554, 0.98850447]], dtype=float32)
```

Does this mean the model is predicting that this image is a hotdog? I don't completely understand...

Compile the model

```
model_v1.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
```

Time to train

```
model_v1.fit(x_train, y_train, epochs=6)
```

```
Epoch 1/6
16/16 [==============================] - 33s 2s/step - loss: 2092.5833 - accuracy: 0.5482
Epoch 2/6
16/16 [==============================] - 26s 2s/step - loss: 78.5949 - accuracy: 0.6265
Epoch 3/6
16/16 [==============================] - 28s 2s/step - loss: 6.6297 - accuracy: 0.8112
Epoch 4/6
16/16 [==============================] - 26s 2s/step - loss: 0.8133 - accuracy: 0.9478
Epoch 5/6
16/16 [==============================] - 27s 2s/step - loss: 0.2848 - accuracy: 0.9759
Epoch 6/6
16/16 [==============================] - 28s 2s/step - loss: 0.0389 - accuracy: 0.9940
```

```
<keras.callbacks.History at 0x29fa7008700>
```

Okay, so now we want to test our data, so we need to create a testing set. After that, we will evaluate our model, and play around with a few different model setups to determine the impact on our accuracy.

Developing Test Data¶

So now we need to go through and develop a test data set. What we are going to do is grab images from our training data set which contains no shared images of hotdogs, so the model can't just identify something it already was trained on. We will quickly run through the same steps we did in the beginning.

```
test_hotdog_dir    = 'C:\\Users\\nburw\\OneDrive\\Desktop\\test\\hot_dog'
test_nothotdog_dir = 'C:\\Users\\nburw\\OneDrive\\Desktop\\test\\not_hot_dog'

test_hotdog_filenames      = os.listdir(test_hotdog_dir)
test_nothotdog_filenames   = os.listdir(test_nothotdog_dir)

hotdog_paths_test       = [(test_hotdog_dir    + '\\' + test_hotdog_filenames[i])
                           for i in range(len(test_hotdog_filenames))]
nothotdog_paths_test    = [(test_nothotdog_dir + '\\' + test_nothotdog_filenames[i])
                           for i in range(len(test_nothotdog_filenames))]
```

```
prepped_hotdogs_test     = prep_images(hotdog_paths_test)
prepped_nothotdogs_test  = prep_images(nothotdog_paths_test)
```

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

```
x_test = np.concatenate((prepped_hotdogs_test, prepped_nothotdogs_test))
y_test = np.array([(1 if n <= 249 else 0) for n in range(len(x_test))])

print(np.shape(x_test), np.shape(y_test))
```

```
(500, 224, 224, 3) (500,)
```

```
## now we shuffle them in unison
assert len(x_test) == len(y_test)

## actual unison shuffle of data
shuffler = np.random.permutation(len(x_test))
y_test = y_test[shuffler]
x_test = x_test[shuffler]
```

```
model_v1.evaluate(x_test,  y_test, verbose=1)
```

```
16/16 [==============================] - 5s 238ms/step - loss: 12.7981 - accuracy: 0.5040
```

```
[12.79814624786377, 0.5040000081062317]
```

OK so now that we have a model, and we've tested it (and it frankly worked out quite poorly) we are gonna start playing around with different models.

Playing with Different Models¶

First and foremost, I'm going to print out the model_v1 summary so we know what layers we had originally

```
model_v1.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 222, 222, 16)      448

 conv2d_7 (Conv2D)           (None, 220, 220, 32)      4640

 flatten_5 (Flatten)         (None, 1548800)           0

 dropout_5 (Dropout)         (None, 1548800)           0

 dense_7 (Dense)             (None, 2)                 3097602

=================================================================
Total params: 3,102,690
Trainable params: 3,102,690
Non-trainable params: 0
_________________________________________________________________
```

Immediately I'm curious about what happens if I don't use convolutions, and instead use a much simpler model. I'll be using just a 4 layer sequential model to see how it goes. My immediate prediction is that I will have a worse accuracy, so let's see what happens.

Model_v2¶

```
model_v2 = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(224, 224, 3)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(2)
])

model_v2.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])


model_v2.fit(x_train, y_train, epochs=6)
```

```
Epoch 1/6
16/16 [==============================] - 4s 223ms/step - loss: 114.7943 - accuracy: 0.8373
Epoch 2/6
16/16 [==============================] - 3s 214ms/step - loss: 237.4119 - accuracy: 0.8373
Epoch 3/6
16/16 [==============================] - 3s 211ms/step - loss: 78.6632 - accuracy: 0.8655
Epoch 4/6
16/16 [==============================] - 3s 215ms/step - loss: 138.4023 - accuracy: 0.8454
Epoch 5/6
16/16 [==============================] - 3s 209ms/step - loss: 45.1477 - accuracy: 0.8976
```

```
Epoch 6/6
16/16 [==============================] - 3s 206ms/step - loss: 60.4015 - accuracy: 0.8896

<keras.callbacks.History at 0x29fa6d1d5b0>
```

Immediately we can tell this is going worse. I expected we'd need more epochs to get to a similar level of accuracty for the training data, and I was right. I don'tknow if adding more is useful becuase it seems to platue

```
model_v2.evaluate(x_test, y_test, verbose=1)
```

```
16/16 [==============================] - 1s 20ms/step - loss: 1760.8885 - accuracy: 0.5000

[1760.8885498046875, 0.5]
```

Surprisingly close to the original convolutional model... Let's try another convolutional mode, but which has more dense layers. I'm going to include a dense layer before the first conv layer to see if that helps at all.

Model_v3¶

```
## Our first attempt will be a generic sequential model with 4 layers
model_v3 = tf.keras.models.Sequential([
    tf.keras.layers.Dense(128, input_shape=(224, 224, 3)),
    tf.keras.layers.Conv2D(16, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(2)
    ])

model_v3.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])

model_v3.fit(x_train, y_train, epochs=6)
```

```
Epoch 1/6
16/16 [==============================] - 108s 6s/step - loss: 368.6917 - accuracy: 0.5201
Epoch 2/6
16/16 [==============================] - 134s 8s/step - loss: 33.3654 - accuracy: 0.6225
Epoch 3/6
16/16 [==============================] - 119s 7s/step - loss: 3.8088 - accuracy: 0.8173
Epoch 4/6
16/16 [==============================] - 125s 8s/step - loss: 0.5800 - accuracy: 0.9378
Epoch 5/6
16/16 [==============================] - 124s 8s/step - loss: 0.1246 - accuracy: 0.9900
Epoch 6/6
16/16 [==============================] - 121s 8s/step - loss: 0.0804 - accuracy: 0.9819

<keras.callbacks.History at 0x29fa5b24be0>
```

```
model_v3.evaluate(x_test, y_test, verbose=1)
```

```
16/16 [==============================] - 17s 996ms/step - loss: 8.9771 - accuracy: 0.4580

[8.977143287658691, 0.4580000042915344]
```

Okay so clearly that was a bad idea! The model performed worse than any of the models thus far. It also took forever, and I'm wondering if I should have flattened after my first dense layer and then after the second convolutional layer. In either case, I'm going to try something else out.

I want to try using max pooling to see if that offers a better output.

Model_v4¶

```
model_v4 = tf.keras.Sequential([
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(2)
])
```

```
model_v4.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])

model_v4.fit(x_train, y_train, epochs=6)
```

```
Epoch 1/6
16/16 [==============================] - 15s 782ms/step - loss: 110.2860 - accuracy: 0.5201
Epoch 2/6
16/16 [==============================] - 12s 739ms/step - loss: 0.8192 - accuracy: 0.5201
Epoch 3/6
16/16 [==============================] - 12s 765ms/step - loss: 0.6178 - accuracy: 0.7209
Epoch 4/6
16/16 [==============================] - 13s 812ms/step - loss: 0.4802 - accuracy: 0.7952
Epoch 5/6
16/16 [==============================] - 13s 831ms/step - loss: 0.3183 - accuracy: 0.9137
Epoch 6/6
16/16 [==============================] - 13s 808ms/step - loss: 0.1621 - accuracy: 0.9598
```

```
<keras.callbacks.History at 0x29f9af65160>
```

```
model_v4.evaluate(x_test, y_test, verbose=1)
```

```
16/16 [==============================] - 4s 223ms/step - loss: 1.1091 - accuracy: 0.5300
```

```
[1.1090584993362427, 0.5299999713897705]
```

There you have it. Best attempt so far only offers 53% accuracy.

I have considered attempting to do keras tuning to tune up the hyper parameters, but I don't really know how to do that for convolutional layers. I could probably tune just the dense layer though...

Questions

That concludes my jupyter notebook output. I'll now answer the questions individually...

1. My code can be seen above
2. The models I used are linked in the code as Model_v2, Model_v3, and Model_v4. I talk a bit more about the speed and whatnot in the next question.
3. So the models performed similarly, not getting more than 50% on average. And also worth noting is that model_v3 ran the slowest, and also performed the worst. This makes sense as I basically just randomly stuck a very large Dense layer at first, and didn't make any attempt to flatten it after that, so I think it took a while to pass things along after that layer.

   It was also surprising to see that model 2, a non convolutional model, was able to discern with better accuracty between hot dogs and not hot dogs.

   Unfortunately the highest I was able to get my accuracy was to 53%. I don't know how that compares to other models, and I'm sure if I used kera tune to tune parameters a little better, I could have gone higher, but this was what I was able to do with just three extra models.
4. Dropout is the percentage of the training data that doesn't get utilized in training. It helps prevent overfitting in a model. The only model I didn't use dropout on was for model 4 and it did score the highest. I don't know exactly why that is, but I wonder about if having even a few more images can help train the data better than with the 20% dropout. I imagine the effect would be more noticeable with higher dropout rate and larger training data sets.