Jorge Quiroz

Professor Erik Grimmelmann

Introduction to Machine Learning - Final Project

May 21, 2020

Image Recognition Using Deep Learning

Introduction

Image classification refers to a process in computer vision that allows a computer to classify an image based on its visual content. While detecting objects in images may be trivial for humans, powerful and accurate image recognition is still a work in progress in computer vision applications.

How does it work?

Image classification is a supervised learning problem. The objects to be identified in each image are defined within a set of target classes and a model is trained to recognize them using labeled example training photographs. A popular type of models that have emerged to tackle these problems involve the use of deep neural networks, more popularly known as deep learning. Deep learning and deep neural networks have gained popularity within the past decade, and with breakthroughs in research, the performance of large-scale image recognition saw large improvements in accuracy and efficiency.

Motivation

I am interested in creating a simple deep neural network to solve an image recognition problem that fascinated me. The problem I am tackling is to try to classify whether a given image is of a chihuahua (dog breed) or a muffin. I am using TensorFlow and Keras for building and running my neural network and using various Python libraries such as numpy, matplotlib (for visualizing image results), and Pillow (for image processing).

Dataset

A sample of the two class types we will be distinguishing from the photographs comprising our data.

Establishing and Loading our Data

The method I decided to use to process the images was to convert each one into an 8-bit grayscale image. I then cropped all images to have a uniform aspect ratio to be accurately converted into numpy arrays. The function below demonstrates the process done to each image:

```
# Use Pillow to convert an input jpeg to a 8 bit grey scale image array for processing.
def jpeg_to_8_bit_greyscale(path, maxsize):
    img = Image.open(path).convert('L')   # convert image to 8-bit grayscale
    # Make aspect ratio as 1:1, by applying image crop.
    WIDTH, HEIGHT = img.size
    if WIDTH != HEIGHT:
        m_min_d = min(WIDTH, HEIGHT)
        img = img.crop((0, 0, m_min_d, m_min_d))
    # Scale the image to the requested maxsize by Anti-alias sampling.
    img.thumbnail(maxsize, PIL.Image.ANTIALIAS)
    return np.asarray(img)
```

Once images are preprocessed, we separate each image into its particular class type by identifying its label. We will use these classified images to train our model. The function below demonstrates how each image is classified:

```
def load_image_dataset(path_dir, maxsize):
    images = []
    labels = []
    os.chdir(path_dir)
    for file in glob.glob("*.jpg"):
        img = jpeg_to_8_bit_greyscale(file, maxsize)
        if re.match('chihuahua.*', file):
            images.append(img)
            labels.append(0)
        elif re.match('muffin.*', file):
            images.append(img)
            labels.append(1)
    return (np.asarray(images), np.asarray(labels))
```

Loading our Training and Testing Datasets

Once the necessary preprocessing functions are defined, I split the data into a training set which is used to train my neural network on identifying images based on the class types as well as a testing set which is used to validate my model's performance on unseen data. The function below demonstrates the splitting:

```
maxsize = 100, 100

(train_images, train_labels) = load_image_dataset(
    path_dir='/home/jorge/Desktop/code/python/Machine Learning/Final Project/image-recognition-tensorflow/chihuahua-muffin',
    maxsize=maxsize)

(test_images, test_labels) = load_image_dataset(
    path_dir='/home/jorge/Desktop/code/python/Machine Learning/Final Project/image-recognition-tensorflow/chihuahua-muffin/test_set',
    maxsize=maxsize)
```

Exploring our Data

- As mentioned previously, the images are being classified by two distinct class types. The types are defined using the **class_names** variable. Since each image can either be a chihuahua or a muffin, or class types are as follows:

```
class_names = ['Chihuahua', 'Muffin']
```

- Our training set consists of 26 image examples (13 chihuahua's, 13 muffins), with a uniform resolution of 100x100.

```
print(train_images.shape)
(26, 100, 100)
```

- Each image has its respective label, either a 0 or 1, identifying its class type.
    - 0: indicates **class_names[0]** = a chihuahua
    - 1: indicates **class_names[1]** = a muffin

```
print(train_labels)
[1 0 0 0 0 0 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 1 0]
```

- Similarly for our testing set, we have 14 image examples.

```
print(test_images.shape)
print(test_labels)
(14, 100, 100)
[0 1 0 1 0 0 1 0 1 0 1 1 1 0]
```

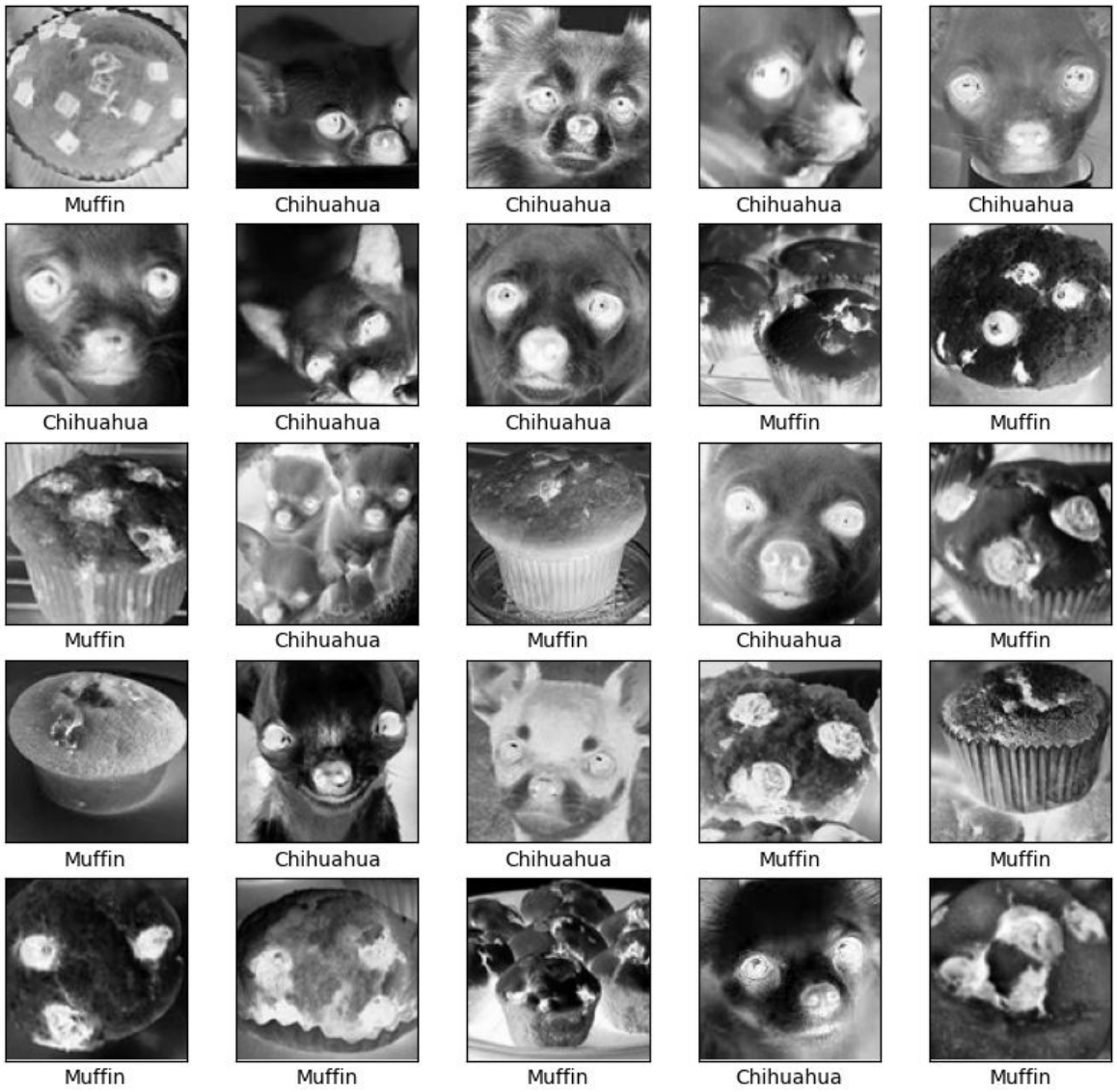
Visualizing our Data

I used matpotlib to display the images comprising our training set while also displaying the corresponding class labels associated with each image. The function below displays the results:

```
def display_images(images, labels):
    plt.figure(figsize=(10,10))
    grid_size = min(25, len(images))
    for i in range(grid_size):
        plt.subplot(5, 5, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(images[i], cmap=plt.cm.binary)
        plt.xlabel(class_names[labels[i]])


display_images(train_images, train_labels)
plt.show()
```

Displayed results of the training set:



Building the Model

Once the images have been preprocessed and training and testing sets have been established, the neural network is ready to be built. My network consists of four layers. The first layer flattens the data set into a single array and does not receive training. The other 3 layers are dense and use sigmoid as the activation function.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(100, 100)),
        keras.layers.Dense(128, activation=tf.nn.sigmoid),
        keras.layers.Dense(16, activation=tf.nn.sigmoid),
    keras.layers.Dense(2, activation=tf.nn.softmax)
])
```

The model is then compiled and uses stochastic gradient descent (SGD) as the optimizer.

```
sgd = keras.optimizers.SGD(lr=0.01, decay=1e-5, momentum=0.7, nesterov=True)

model.compile(optimizer=sgd,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Training the Model

Fitting the model using the training set, we see the following training iterations as the result:

```
model.fit(train_images, train_labels, epochs=100)
```

```
Epoch 98/100
1/1 [==============================] - 0s 268us/step - loss: 0.4134 - accuracy: 0.9231
Epoch 99/100
1/1 [==============================] - 0s 271us/step - loss: 0.4109 - accuracy: 0.9231
Epoch 100/100
1/1 [==============================] - 0s 259us/step - loss: 0.4085 - accuracy: 0.9231
```

- The network displays an accuracy of about 92% on the training set.

Testing the Model

Once the network has received training, we are ready to evaluate the results of our network performing on unseen data. The model is then evaluated on the testing set.

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

```
1/1 [==============================] - 0s 226us/step - loss: 0.5951 - accuracy: 0.7143
Test accuracy: 0.7142857313156128
```

- The network displays an accuracy of about 71% on the testing set.

Predictions

To discover more on how the images are being classified, we can take a look at the prediction values that our model is giving on the testing set images.

```
predictions = model.predict(test_images)
print(predictions)
```

```
[[0.71601105 0.28398895]
 [0.37262228 0.6273777 ]
 [0.5883091  0.41169083]
 [0.4239632  0.5760368 ]
 [0.4731513  0.52684873]
 [0.7188232  0.2811768 ]
 [0.18757671 0.81242335]
 [0.5654694  0.43453062]
 [0.49324018 0.5067598 ]
 [0.5429351  0.45706493]
 [0.6388649  0.36113515]
 [0.46500257 0.53499746]
 [0.52504635 0.47495368]
 [0.64418787 0.35581213]]
```

‘CHIHUAHUA’ ‘MUFFIN’

- As we can see from the chart above, for each image in our testing set, it is given a percentage value on the possibility of it being from the associated class type.
- For the first image (top array), it is given a value of about 72% being a chihuahua and a 28% being a muffin. The greater of the two is chosen to ultimately classify the image.

- It is interesting to point out that for a few of the images the percentages come very close to each other (both values coming close to 50%). These close values can attribute directly to inaccurate predictions given by the model.

Results

Finally, using matplotlib we can see the results of the network performing on the testing set.

```
display_images(test_images, np.argmax(predictions, axis = 1))
plt.show()
```

- As we can see, some of these predictions are inaccurate.

Conclusion

As shown previously from the results of running the training and testing sets on the neural network, the training accuracy was about 92% but the testing accuracy was significantly lower at about 71%. This is a clear indication that the model has overfit the data. Overfitting occurs when a model displays a high accuracy on the training data but performs much poorly when evaluated against a testing or unseen data set. Some elements of my model that I can attribute to the performance and the occurrence of overfitting are:

- Model Size - Experimenting with varying model sizes would have been beneficial in determining the correct size to run on the network. If the model size is too large, it is likely to learn the features/patterns not relevant to the problem and thus overfit the data. A larger model will not generalize well, and a smaller model will underfit the data.
- Number of Epochs - The number of epochs plays an important role in avoiding overfitting and overall model performance. It is critical to determine the correct point to stop before the model overfits the data.
- Lack of Training Data - With only 26 training image examples, the model performed reasonably well. However, it may be beneficial to run the network on a larger dataset to determine its true accuracy.

Building and running this simple deep neural network was a lot of fun and a true learning experience on how powerful these machine learning models can be. While this network is not perfect, it still demonstrates how accurate computers can be even with small datasets. I'm

interested to pursue further research on image classification using more sophisticated models such as convolutional neural networks (CNN) and even other advanced uses of neural networks.

Works Cited

- https://www.tensorflow.org/tutorials/images/classification
- https://www.tensorflow.org/tutorials/keras/classification
- https://www.freecodecamp.org/news/chihuahua-or-muffin-my-search-for-the-best-computer-vision-api-cbda4d6b425d/
- https://www.youtube.com/watch?v=cAICT4Al5Ow
- https://arxiv.org/pdf/1801.09573.pdf