Floating Assignment 1- Michael Keller

March 6, 2020

1 Introduction

I am most interested in the work of the connectionists and neural networks. In this report, I will attempt to answer the following question:

Given an image of a Pokemon, can a convolutional neural network model be used to classify the Pokemon by type?

I had this inspiration from a dataset I recently found on Kaggle, shown here:

https://www.kaggle.com/vishalsubbiah/pokemon-images-and-types

1.1 Background

I refer the reader to the following Wikipedia article for more information on Pokemon:

https://en.wikipedia.org/wiki/Pok%C3%A9mon_(video_game_series)

For the purposes of context for this report, Pokemon is shorthand for Pocket Monsters, and it is a Japanese video game series developed for Nintendo gaming systems. In it, the player goes on an adventure where they assemble a team of 6 creatures, train them up to become strong, and compete for the recognition of becoming the most powerful trainer in the game world. Players in Pokemon compete by battling them against each other. As of this writing, there are 890 unique Pokemon. The Kaggle dataset mentioned above contains only 809 Pokemon, and was not updated for the additional 81 Pokemon introduced in Pokemon Sword and Shield in November 2019.

Each Pokemon has a primary and possibly a secondary type. For the purposes of complexity, we will only use the primary type of a Pokemon as a class label. There are 18 unique types of Pokemon, which each type having its own strengths and weaknesses in battle with respect to other types. Examples of types include Fire, Water, Grass, Ground, or Electric. We will investigate in this report the ability of a CNN to distinguish Pokemon by type based on their appearance in images.

2 Implementation

First we load in the data:

[1]:
```
import numpy as np
import cv2
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
```

```python
import scipy
import os
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
```

[2]:
```python
os.chdir('C:/Users/mkell/Downloads/pokemon-images-and-types')
```

[3]:
```python
pokemon=pd.read_csv('pokemon.csv')
print(np.unique(pokemon['Type1'], return_counts=True))
pokemon=pokemon.sort_values('Name')
pokemon=pokemon.reset_index(drop=True)
pokemon
```

```
(array(['Bug', 'Dark', 'Dragon', 'Electric', 'Fairy', 'Fighting', 'Fire',
       'Flying', 'Ghost', 'Grass', 'Ground', 'Ice', 'Normal', 'Poison',
       'Psychic', 'Rock', 'Steel', 'Water'], dtype=object), array([ 72,  29,
27,  40,  18,  29,  53,   3,  27,  78,  32,  23, 105,
        34,  53,  46,  26, 114], dtype=int64))
```

[3]:
```
                Name    Type1   Type2
0          abomasnow    Grass     Ice
1               abra  Psychic     NaN
2              absol     Dark     NaN
3           accelgor      Bug     NaN
4    aegislash-blade    Steel   Ghost
..               …        …       …
804          zoroark     Dark     NaN
805            zorua     Dark     NaN
806            zubat   Poison  Flying
807         zweilous     Dark  Dragon
808       zygarde-50   Dragon  Ground

[809 rows x 3 columns]
```

This is the original dataset of 809 Pokemon. We will use the column 'Type1' for labeling. Next we load in the images:

[4]:
```python
images=np.empty((len(os.listdir('images/images')), 120, 120, 3))
count=0

for root, dirs, files in os.walk('images'):
    for i, file in enumerate(files):
        path = os.path.join(root, file)
        img=cv2.imread(path)
        images[count] = img
        count=count+1
```

```
        #print("Loaded file "+str(count)+ " of "+str(len(os.listdir('images/
↪images')))+ " ")
```

[5]:
```
images.shape
```

[5]:
```
(809, 120, 120, 3)
```

Some Pokemon types are intuitive and some are not. For example, the first and third Pokemon in the original dataset are Ice-type Pokemon, which is supported by their white, snowy appearance. However, the second Pokemon is Psychic-type, which is not immediately evident from its appearance. The difficulty of this task for humans is present because of this fact. Thus, we hope to see how difficult this task is for a neural network.

Note that the original dataset contains only 809 images in 18 classes. This is hardly enough data with which to train a model. Nevertheless, we will try and train a model on this small dataset to see what happens.

[6]:
```
os.chdir('C:/Users/mkell/Dropbox/Spring 2020/Artificial Intelligence/
↪pokemon-type-classifier/pokemon-classifier')
```

Next we preprocess the data:

[7]:
```
#create labels
image_labels=np.array(pokemon['Type1'])
```

[8]:
```
#normalize data
images/=255
images=images.astype('float32')
```

[9]:
```
# integer encode
label_encoder = LabelEncoder()
image_labels = label_encoder.fit_transform(image_labels)
# one hot encode
onehot_encoder = OneHotEncoder(sparse=False)
image_labels = image_labels.reshape(len(image_labels), 1)
image_labels = onehot_encoder.fit_transform(image_labels)
image_labels = np.asarray(image_labels)
```

[10]:
```
#split data into train/test sets
train_data, test_data, train_labels, test_labels=train_test_split(images,␣
↪image_labels, test_size=0.3, shuffle=True)
train_data, val_data, train_labels, val_labels=train_test_split(train_data,␣
↪train_labels, test_size=0.1, shuffle=True)
```

Next we define the model. Our input images are of size (120, 120, 3), as they are 120x120 RGB images. We use a Conv-Pool-Conv-Pool format for the network, doubling the number of filters in each convolutional layer. Once we have reduced the output to 512 1x1 images, we flatten the convolutional output, we use 3 Dense layers at the end of the network with 256, 128, and 64 nodes before pssing the output to our final softmax layer of 18 classes. All layers of the neural network except the final output layer have a Rectified Linear Unit, or ReLU activation function.

These choices for model architecture were based on past convolutional neural network designs in the field. For an optimizer, we use Adam, or adaptive gradient descent with momentum. This is the most widely accepted optimizer for convolutional neural networks in the literature. We use a learning rate of 0.0001 for the network. This was determined through trial and error of training the network. We train the network with 63% of the 809 Pokemon, validate it on 7% of the 809 Pokemon, and test it on 30% of the 809 Pokemon.

[11]:
```
model=tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(5, 5),␣
 ↪activation='relu', input_shape=(120, 120, 3)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3),␣
 ↪activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=(5, 5),␣
 ↪activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=(3, 3),␣
 ↪activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=(3, 3),␣
 ↪activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(18, activation='softmax'))

adam=tf.keras.optimizers.Adam(lr=10**-4)

model.compile(optimizer=adam, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 116, 116, 32)      2432

_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 58, 58, 32)        0

_________________________________________________________________
conv2d_1 (Conv2D)            (None, 56, 56, 64)        18496

_________________________________________________________________
```

```
max_pooling2d_1 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 128)       204928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 256)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 3, 512)         1180160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 1, 1, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 256)               131328
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 18)                1170
=================================================================
Total params: 1,874,834
Trainable params: 1,874,834
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
mc=tf.keras.callbacks.ModelCheckpoint('best_pokemon_model.hdf5',
 ↪monitor='val_loss', save_best_only=True)

hist=model.fit(train_data, train_labels, batch_size=1, epochs=50, verbose=1,
 ↪callbacks=[mc],
               validation_data=(val_data, val_labels))
```

[12]:
```
model=tf.keras.models.load_model('best_pokemon_model.hdf5')
test_results=model.evaluate(test_data, test_labels, verbose=0)
test_results
```

[12]:
```
[2.4113418594799905, 0.23045267]
```

3 Discussion

Given that we have 18 classes of Pokemon type, if a network were randomly guessing, it would achieve an accuracy of 1/18=0.055. Achieving a test accuracy of 23% thus means the network is doing better than randomly guessing, though still has fairly low accuracy. This supports the conclusion that there are noticeable, yet inconsistent patterns in Pokemon appearance that signify type.

There is also the possibility that a dataset of 809 instances is two small to properly train a model. It is possible to artificially inflate the dataset by making copies of the existing images or making rotated or reflected copies of the images. However, this will likely lead to overfitting, and it will be difficult to generalize what the network has learned to new Pokemon when they are released, as there is a large amount of variety in Pokemon design. Future work will investigate how rotations or reflections of these images affect the network. However, these initial findings will conclude the first draft of this assignment.

4 References

https://www.kaggle.com/vishalsubbiah/pokemon-images-and-types

https://en.wikipedia.org/wiki/Pok%C3%A9mon_(video_game_series)

https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7

https://www.youtube.com/watch?v=g2vlqhefADk&t=273s

https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/