X-Ray CNN Update **Ryan Taylor**
November 8, 2019

Python Adaptation, pt. II

The data has been correctly assigned now, with `X_train` and `X_test` occupying a rank four shape of `(x, 512, 512, 1)` that will be correctly processed for future uses. I also chose to implement methods from the Keras API to get a sense of how the data must be processed and ideally achieve preliminary results--this would preferably become a point from which I can reverse-engineer parts of the process that perform poorly from others. The current state of the code has been pasted below:

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  "
   Created on Fri Nov 15 12:11:22 2019

   @author: ryan
   "
4  import numpy as np
5  import pandas as pd
6  from skimage.io import imread
7  from matplotlib import pyplot as plot
8
9  '''Instantiates test data'''
10 train_locs = pd.io.parsers.read_csv("wrist_train_image_paths.csv")
11 batch_size = 25
12 X_train = list()
13 Y_train = list()
14 i = 0
15 for row in train_locs['Location'] :
16 img = imread(row, as_gray = True)
17 if len(img[0]) < 499 :
18 Y_train.append([1, 0] if ('positive' in row) else [0, 1])
19 img = np.reshape(img, (512, img.shape[1], 1))
20 index = int((512-len(img[0]))/2) +1
21 end_index = 512-index
22 fin = np.full((512,512,1), 0)
23 for r in range(512) :
24 for c in range(index, end_index) :
25 fin.itemset((r,c,0), img.item((r,c-index,0)))
26 X_train.append(fin)
27 i += 1
```

```
    if(i > batch_size) :
    break

'''Instantiates validation data'''
train_locs = pd.io.parsers.read_csv("wrist_valid_image_paths.csv")
X_test = list()
Y_test = list()
i = 0
for row in train_locs['Locations'] :
img = imread(row, as_gray = True)
if len(img[0]) < 499 :
Y_test.append([1, 0] if ('positive' in row) else [0, 1])
img = np.reshape(img, (512, img.shape[1], 1))
index = int((512-len(img[0]))/2) +1
end_index = 512-index
fin = np.full((512,512,1), 0)
for r in range(512) :
for c in range(index, end_index) :
fin.itemset((r,c,0), img.item((r,c-index,0)))
X_test.append(fin)
i += 1
if(i > batch_size) :
break

X_train = np.asarray(X_train)
Y_train = np.asarray(Y_train)
X_test = np.asarray(X_test)
Y_test = np.asarray(Y_test)

print("Data initialized.")

import keras
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalization
from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

```
X_train /= 255
X_test /= 255

#Y_train = keras.utils.to_categorical(Y_train, 2)
#Y_test = keras.utils.to_categorical(Y_test, 2)
print(Y_train.shape, ",", Y_test.shape)

data_generator = ImageDataGenerator(rotation_range=90,
width_shift_range=0.1,
height_shift_range=0.1,
featurewise_center=True,
featurewise_std_normalization=True,
horizontal_flip=True)

data_generator.fit(X_train)

for i in range(len(X_test)):
X_test[i] = data_generator.standardize(X_test[i])

model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=X_train.shape[1:]))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), padding='same'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('elu'))
model.add(BatchNormalization())
```

```
108 model.add(Conv2D(128, (3, 3), padding='same'))
109 model.add(Activation('elu'))
110 model.add(BatchNormalization())
111 model.add(MaxPooling2D(pool_size=(2, 2)))
113
114 model.add(Flatten())
115 model.add(Dense(2, activation='softmax', input_shape=(2,)))
116
117 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
118
119 model.fit_generator(
120 generator=data_generator.flow(x=X_train,
121 y=Y_train,
122 batch_size=batch_size),
123 steps_per_epoch=len(X_train) // batch_size,
124 epochs=1,
125 validation_data=(X_test, Y_test),
126 workers=4)
```

This segment of code is very similar to the code attached in the last update; however, there are notable additions as far as the handling of the data. I have written a summary of the lines marked in bold below:

74-79: Creates an ImageDataGenerator object that is able to manipulate the images as needed for the convolutional and multilayer-perceptron layers to function correctly. This allows images to be rotated up to 90°, shifted by 10%, flipped horizontally, or normalized as features are determined to better achieve accurate results.
81: Fits the training data, but not the training labels, to the data_generator.
84-85: Standardizes the training data to fit the format necessary for the SequentialModel.
86-94: Creates a SequentialModel as the basis for the entire summed network; these lines implement the following functional layers in order:

1. 2D Convolution with 32 filters, operating with a 3x3 kernel on the X_train dataset
2. Exponential Linear Units activation model; similar to previously mentioned ReLU but with a different second parameter in the max() function
3. Batch Normalization, which ensures that the data is normalized following each pass of the above activation function [this will apply to all subsequent activation functions]
4. Second Convolution layer with identical parameters to the first
5. Activation
6. Normalization
7. Pooling: this has been previously mentioned as a means of shrinking the data while maintaining relevant features; it functions by taking, in this case, 2x2 segments of the processed data and returning a single entry that corresponds to the maximum value within that square, effectively shrinking the data by a factor of 2.

96-102: Similar function as the previous set of layers, but with additional filters (operating on a smaller set of data)
105-111: Same as above
114: Flattens the output data of the convolutional layers for use with the fully-connected dense MLP
115: Creates the fully-connected dense model, using softmax activation and the correct input shape corresponding to the effective output (2x1 ndarray)
117: Compiles the model before running the data through it; truthfully, this is an aspect of Keras that I do not fully understand and thus am unlikely to implement in the final result.
119-126: The most important line of the code; effectively starts up the network, using the previously-created data_generator with the X, Y training data, the corresponding batch_size (in this case, 25), the number of epochs (full runs through the entire dataset, in this case 1 but ideally much greater), the corresponding X, Y test data, and the number of threads being operated on.

Current State

As of the project's current state, data is read in and processed correctly, and implementations of the Keras API are correct in their ability to process and forward propagate as well as backpropagate through both the 3 convolutional layers and the fully-connected dense multilayer-perceptron model. However, one small caveat still presents itself--the current model is able to achieve a whopping 0% accuracy, which I have highlighted below in the console output. This suggests that there are still errors to be fixed. There are a considerable amount of hyperparameters to be taken into account, most notably the size of the dataset, the number of filters used, the size of pooling and kernel operations in convolution, and many others. Additionally, the number of epochs, the number of samples per epoch (currently 25, at most can be ~8000) and the number of steps per epoch (should be equal to the batch size) must be increased in order to achieve desirable results.

```
runfile('/Users/ryan/MURA-Network.py', wdir='/Users/ryan')
Data initialized.
(26, 2) , (26, 2)
Epoch 1/1
1/1 [==============================] - 30s 30s/step - loss: 5.8285 -
accuracy: 0.0000e+00 - val_loss: 0.1052 - val_accuracy: 1.0000
```