Convolutional Neural Network for Classifying Pair Trading Equity Curves

Abstract

In this paper, a convolutional neural network is created for the purpose of classifying accepted equity curves resulting from the brute force pair trading program. The model serves as a filter for the results of the brute force program. 90.4% accuracy is achieved by the model. The model can be saved in a file containing the weights and later loaded in other scripts to perform the tasks.

Introduction

The process of how neural networks work can be simplified in Figure 1 below. Basically, we have an input image that is fed into the CNN model, and we have an output which in our case classifies the image (simply, in our case, it suggests whether the pair in the image is useful for trading or not). [1]

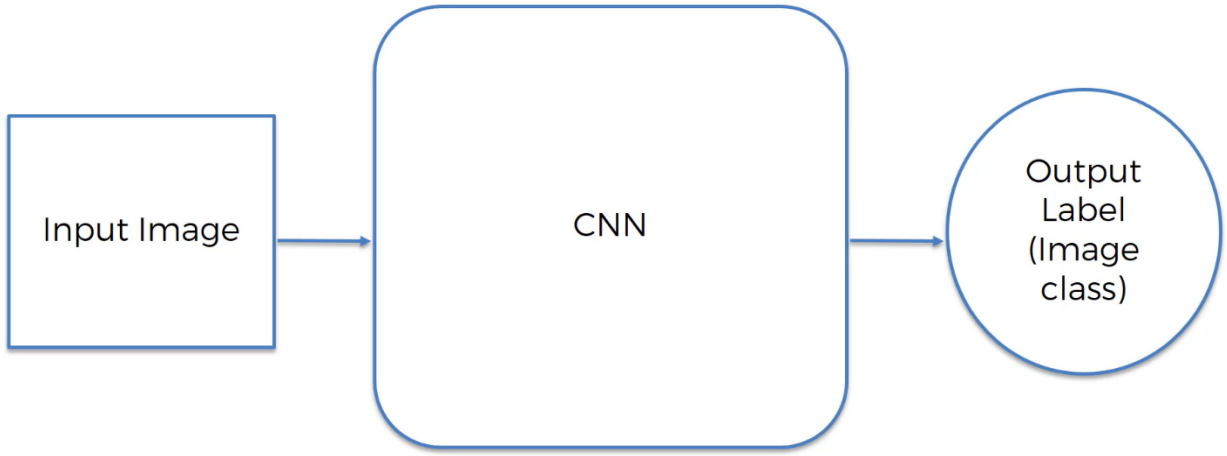

Figure 1 Simplified CNN Classifier Model Process [1]

The ability for a neural network to recognize images in the first place and training a model as such stems from the fact that it can process the digital representation of an image (i.e. pixels). Suppose for simplicity, we have a 2 by 2 px image in black and white (Figure 2). The digital representation of the 2 by 2 black and white image, therefore, is a two-dimensional 2 by 2 array. Each pixel then has a value between 0 and 255 (8 bits of information, $2^8 = 256$), which represents the intensity of the color. 0 would represent a completely black pixel and 255 would be a completely white pixel and between them the gray scale range.

In Fig.3, a colored 2 by 2 pixels image is digitally depicted. In computational terms, the image is a 3D array, with the 3rd array being that depicting color (blue, red, green).

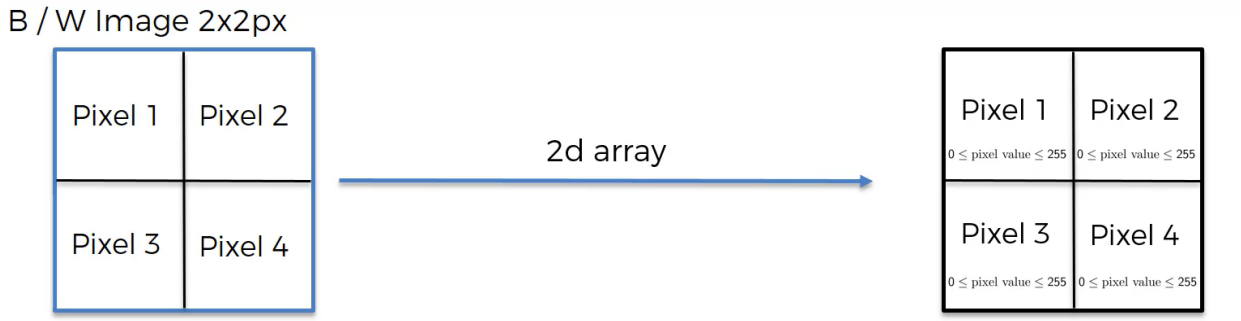

Figure 2 Digital depiction of a 2 x 2 pixels Image (B&W)

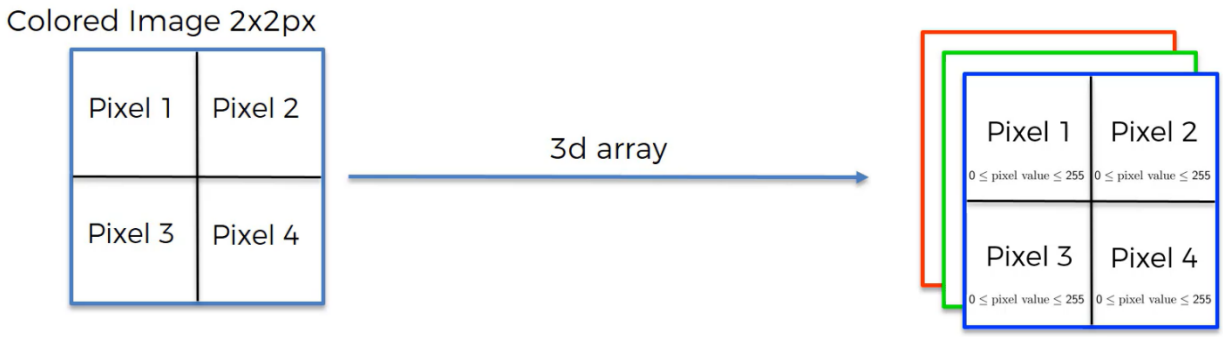

Figure 3 Digital depiction of a 2 x 2 pixels Image (colored)[1]

Steps of CNN Modeling

Step 1 – Convolution

In this step, we have the input image, a feature detector (or kernel, convention: 3 x 3, though it can differ), and the resulting feature map (Figure 4). A convolution operation is basically an element-based multiplication of the 2D feature detector array with a sub 2D array of the input image resulting in filling up the Feature Map with the resulting calculations. In the example of Figure 4, we start at the top left of the input image. Then, suppose we take one step (also called stride, in this case, one-pixel stride) downwards, the calculation is repeated, and the results are added to the Feature Map.

The main reasons for this step are:

- Reducing image size (array size shrinks)
- The higher the stride, the smaller the resulting feature map.
- Detects features that are integral

The purpose of the feature detector is to determine certain features. The highest number in the feature map corresponds to the position where the feature is likely to be detected.

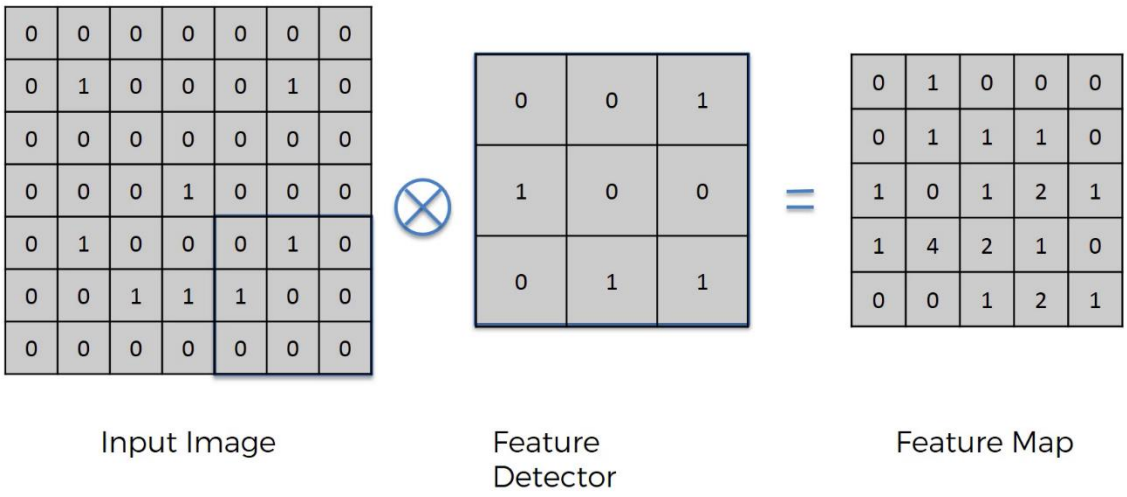

Figure 4 Feature Map Calculation [1]

Multiple feature maps are created to account for certain features in an image (Figure 5). The CNN algorithm defines those features through training. Feature maps are found by applying different filters on the image.

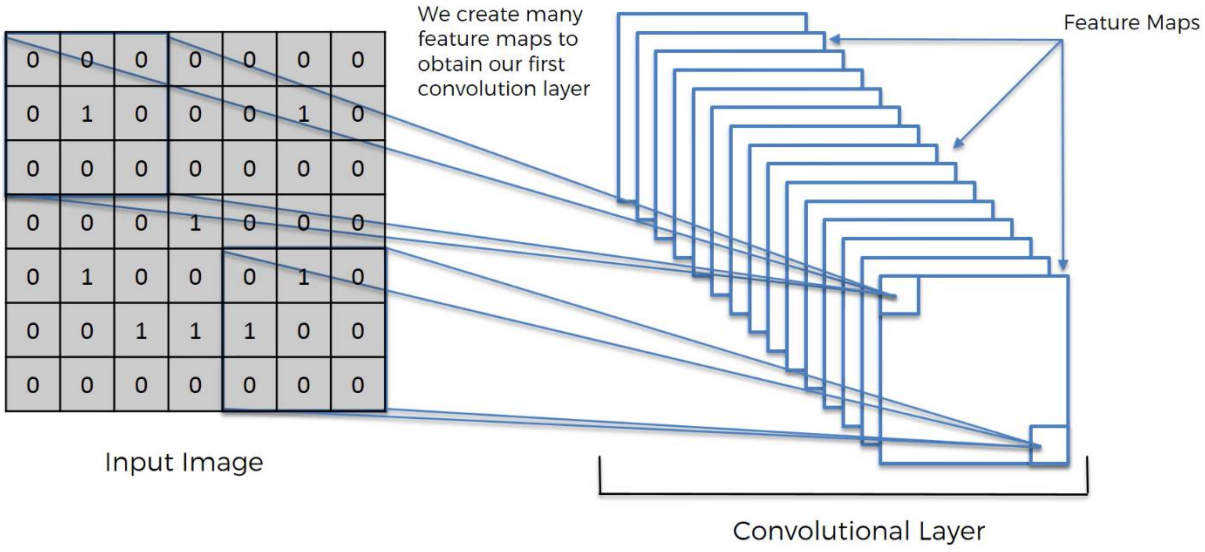

Figure 5 Multiple Feature Maps in the Convolution Step [1]

Step 2 – ReLU Layer

In this step, a rectifier is applied on top of the convolutional layer to increase nonlinearity in the model (Figure 6). The reason for that is due to the inherent nonlinearity of images (i.e. the transition between adjacent pixels is often going to be nonlinear e.g. borders, different colors etc…

When applying a mathematical operation such as convolution, we risk creating linear results, and this step helps break the nonlinearity.

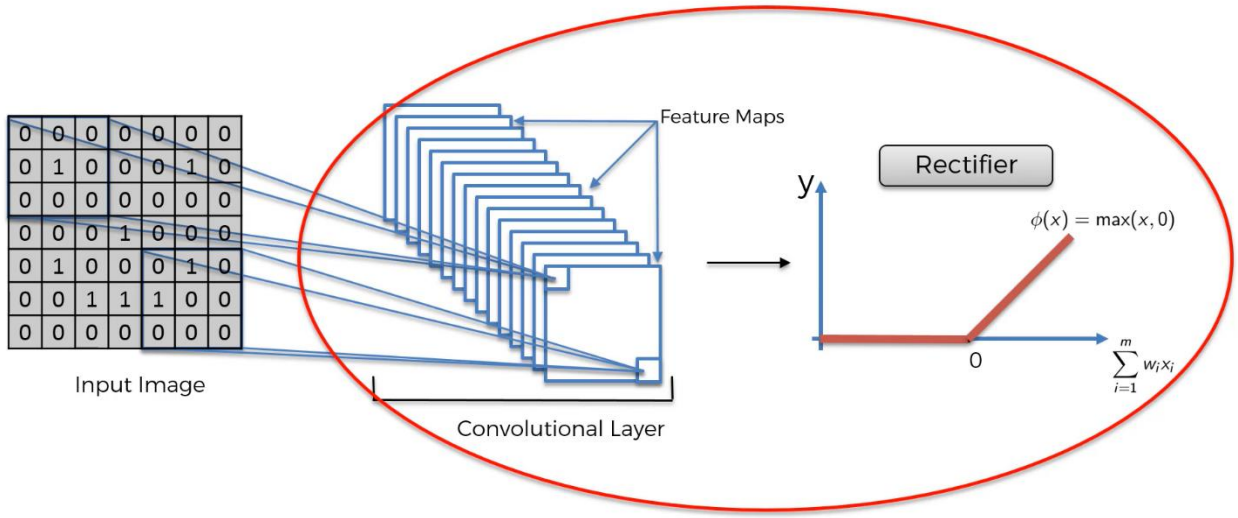

Figure 6 ReLU Layer to add nonlinearity [1]

Step 3: Pooling

In this step, we ensure the neural network is spatially invariant. That is, the model would be able to detect the feature even though it might for example be tilted, expanded, or distorted in any way. Therefore, it adds flexibility to the CNN model. An example pooled feature map is shown in Figure 7. In this example, a 3 x 3 pooled feature map is created by running a 2 x 2 array on top of the feature map (5 x 5). The 2 x 2 array spans 2 x 2 subsets of the original feature map and extracts the value, which is highest within that subset, and adds that value in the corresponding element of the pooled feature map as shown (the stride can also vary here, a stride of two is chosen in this explanatory figure).

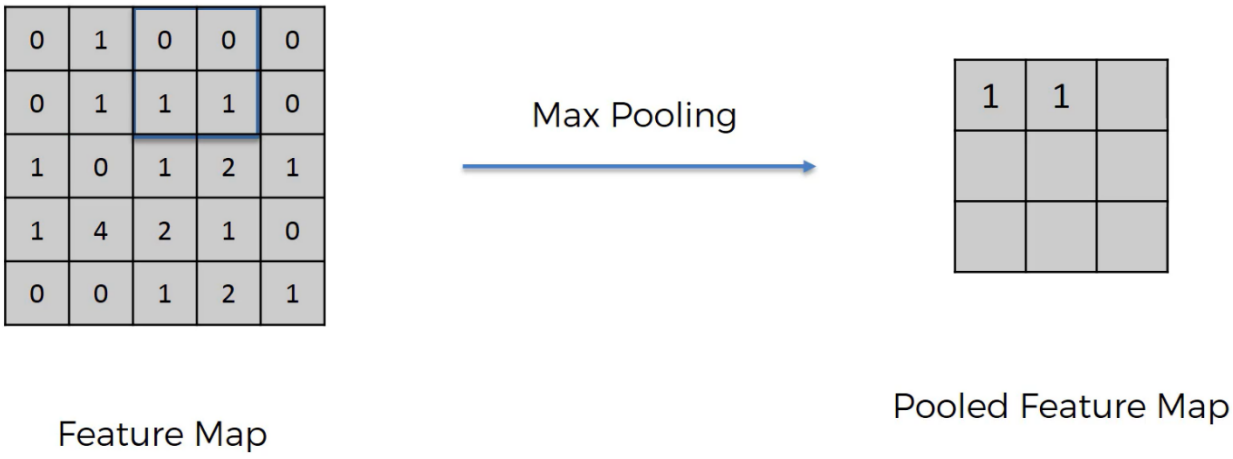

Figure 7 Max Pooling Step [1]

Step 4: Flattening

In this step, the pooled feature maps are flattened into a column row by row (Figure 8). The reason for this step is to prepare the pooled feature maps to be fed as input to an artificial neural network for further processing.

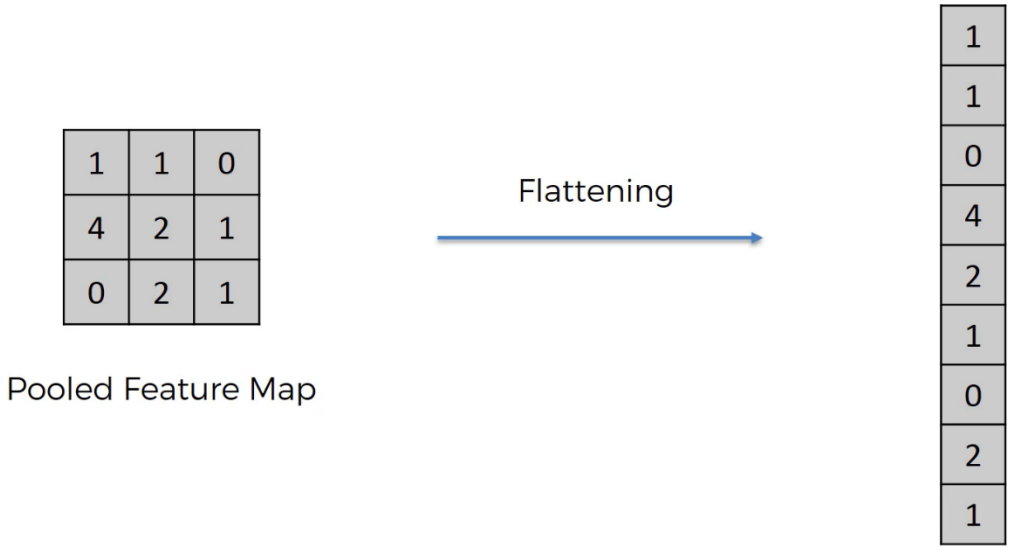

Figure 8 Flattening Layer Basic Schematic [1]

Step 5: Full Connection

In this step, an artificial neural network is added to the previous network of operations (Figure 9). The reason for this step is to combine the features into more attributes for a yet better prediction.

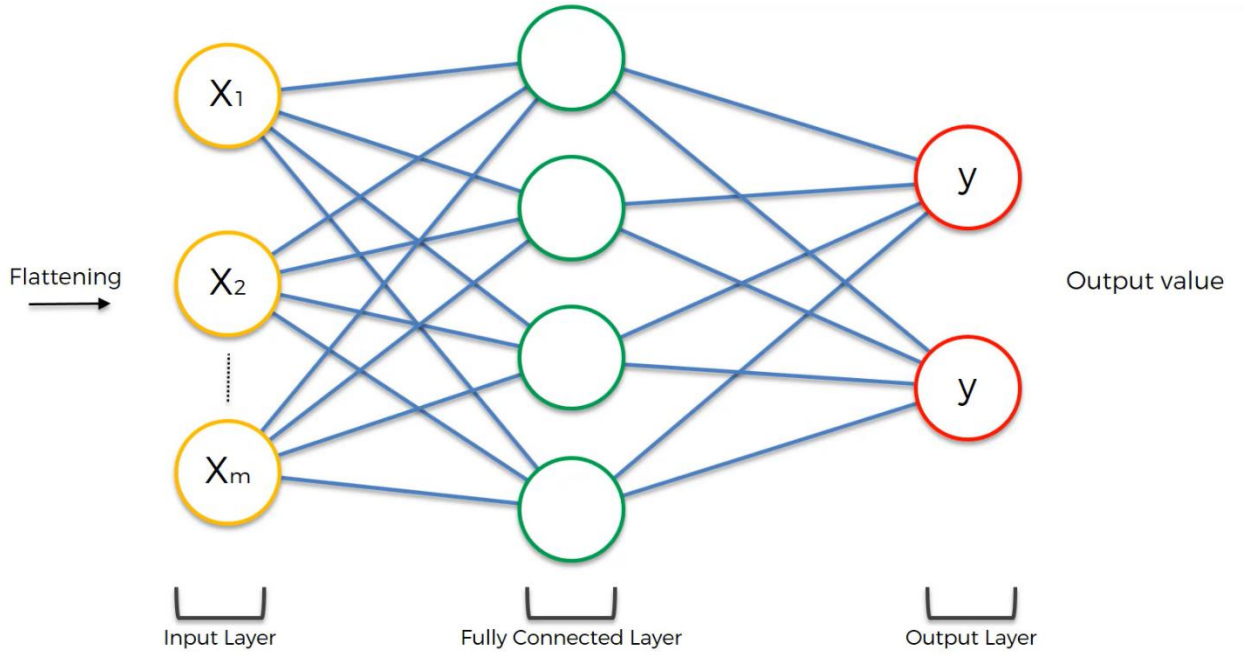

Figure 9 Full Connection Step [1]

Pairs Trading CNN Model library and dataset

The steps of the CNN training are applied to the resulting equity curves from the pairs trading program. The keras library in Python is used to import and train a CNN model.

In order to train, the keras CNN model is fed a dataset containing training and test folders. Within each of these folders, we have YES or NO folders containing the equity curve images that can be traded and those that are not suitable for trading (Figure 10).

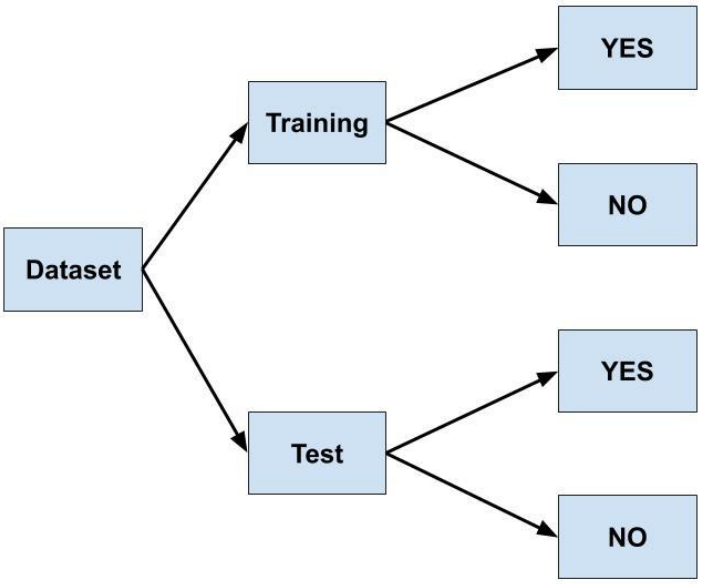

Figure 10 CNN Model dataset folder format

The dataset as depicted in Figure 10 is split into 2/3rd training and 1/3rd testing with corresponding ratios for the files in the YES and NO folders within the training and test folders.

Training the CNN model

After creating the folders necessary to feed into the model, we start building the python code for training our model (cnn_model_building.py). The first step is to import the libraries necessary for initializing CNN, convolution, pooling, flattening, and adding ANN. (refer to code)

Importing Libraries:

```
36 # Part 1 - Building the CNN
37
38 # Importing the Keras libraries and packages
39 from keras.models import Sequential
40 from keras.layers import Convolution2D
41 from keras.layers import MaxPooling2D
42 from keras.layers import Flatten
43 from keras.layers import Dense
44
```

Initializing the CNN:

```
46 # Initialising the CNN
47 classifier = Sequential()
```

Adding the CNN steps:

Step 1 - Convolution

```
49 # Step 1 - Convolution
50 classifier.add(Convolution2D(32, 3, 3, input_shape = (64, 64, 3), activation = 'relu'))
```

Step 2 – Pooling

```
56 # Step 2 - Pooling
57 classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

Adding a second convolutional layer

```
62 # Adding a second convolutional layer
63 classifier.add(Convolution2D(32, 3, 3, activation = 'relu'))
64 classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

Step 3 - Flattening

```
69 # Step 3 - Flattening
70 classifier.add(Flatten())
```

Step 4 - Full connection

```
72 # Step 4 - Full connection
73 classifier.add(Dense(output_dim = 128, activation = 'relu'))
74 classifier.add(Dense(output_dim = 1, activation = 'sigmoid'))
```

Compiling the CNN:

```
76 # Compiling the CNN
77 classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Fitting the CNN model to the images:

```
81 from keras.preprocessing.image import ImageDataGenerator
82
83 train_datagen = ImageDataGenerator(rescale = 1./255,
84                                    shear_range = 0.2,
85                                    zoom_range = 0.2,
86                                    horizontal_flip = True)
87
88 test_datagen = ImageDataGenerator(rescale = 1./255)
89
90 training_set = train_datagen.flow_from_directory('dataset/training_set',
91                                                  target_size = (64, 64),
92                                                  batch_size = 32,
93                                                  class_mode = 'binary')
94
```

```
95 test_set = test_datagen.flow_from_directory('dataset/test_set',
96                                             target_size = (64, 64),
97                                             batch_size = 32,
98                                             class_mode = 'binary')
99
100 classifier.fit_generator(training_set,
101                          samples_per_epoch = 8000,
102                          nb_epoch = 8,
103                          validation_data = test_set,
104                          nb_val_samples = 2000)
```

Saving the CNN model as classifier.h5 file to be loaded in other scripts:

```
106 classifier.save('classifier.h5')
```

Classifying a folder with the model

After training the CNN model with keras and saving the file as “classifier.h5”, we can load the classifier file in another python script (Load_test_model.py) to test on images from a folder of our choice. In this python script, however, the YES and NO folders in the test set are used to from as source folders for images to predict and classify using our model (classifier.h5). The equity curves are then classified into YES and No folders within the folder named “cnn_predictions”.

The code “Load_test_model.py” is explained here:

First, we import the libraries, specify the path, and load the model (classifier.h5):

```
import numpy as np
from keras.preprocessing import image
import os
from keras.models import load_model

path = r"C:\Users\amink\Desktop\AI_in_Finance\pair_trading_cnn_project\new_cnn_keras_algorithm\latest_code"
os.chdir(path)
classifier = load_model('classifier.h5')
# Appending the YES testing dataset into yes and no folders in the cnn_predictions folder
```

Second, a loop is created to go over each image in the specified source folder, classify it using the CNN model we loaded, and cut and paste in the destination folder cnn_predictions under the predicted classification (yes or no). An example of that is the loop applied over the test_set dataset containing YES (or suitable for trading) pairs:

```
i = 0
path = r"C:\Users\amink\Desktop\AI_in_Finance\pair_trading_cnn_project\new_cnn_keras_algorithm\latest_code\dataset\test_set\YES"
os.chdir(path)
for filename in os.listdir(r"C:\Users\amink\Desktop\AI_in_Finance\pair_trading_cnn_project\new_cnn_keras_algorithm\latest_code\dataset\test_set\YES\\"):
    test_image = image.load_img(filename, target_size = (64,64))
    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis = 0)
    result = classifier.predict(test_image)
   # training_set.class_indices
    if result[0][0] >= 0.5:
        prediction = 'YES'
        dst = prediction + str(i) + ".png"
        dst =r"C:\Users\amink\Desktop\AI in Finance\pair trading cnn project\new cnn keras algorithm\latest code\cnn predictions\yes\\"+ dst
    else:
        prediction = 'NO'
        dst = prediction + str(i) + ".png"
        dst =r"C:\Users\amink\Desktop\AI_in_Finance\pair_trading_cnn_project\new_cnn_keras_algorithm\latest_code\cnn_predictions\no\\"+ dst

    src =r"C:\Users\amink\Desktop\AI_in_Finance\pair_trading_cnn_project\new_cnn_keras_algorithm\latest_code\dataset\test_set\YES\\"+ filename
    # rename() function will
    # rename all the files
    if not os.path.exists(dst): # we might need to delete this and unindent the two lines below
        os.rename(src, dst)
        i += 1
```

The python script also contains a loop for the NO in the test_set as well. However, the source and destination folders can be modified to include any source folder to apply the model to and classify its results to a new folder.

Results

Running the results of the modeling code with 250 steps and 4 epochs gives the following results:

Loss	0.1523
Training Accuracy	**0.9321**
Test Accuracy	**0.9043**

Therefore, our classifier's accuracy is ~ 90.4%

Future Work

- Regularization can be considered as the model fares well but might be considered slightly over trained.
- Training the model on more equity curves. The only pairs considered in this study are the currency pairs. Adding equity curves as to balance the number of each classification might help increase the accuracy of the model.
- Merging the code script with the brute force pair generating code and considering tests like White's reality check and Dickie-Hellman and filtering for those as well.

References

[1] "Machine Learning A-Z" Kirill Eremenko, Retrieved from: www.udemy.com
[2] "Convolutional Neural Network Explained Step by Step". Retrieved from: https://medium.com/pylessons/convolutional-neural-networks-cnn-explained-step-by-step-69137a54e5e7