BT4012 Assignment 2 Report

A0192263E

November 2020

<u>Anomaly Detection in Images</u>

Abstract: This paper walks through this assignment of detecting anomalies in the MNIST digits dataset. It starts with understanding the dataset and libraries required. The report also addresses the imbalanced dataset issue and focuses on supervised and unsupervised models that can produce the highest ROC score. It explains the architecture of the models along with the training, validation and testing process.

Introduction: The dataset we had to work with consists of MNIST handwritten digits dataset. As per Kaggle, the normal data were labelled as 0 and the anomalous data were labelled as 1. Normal Data consists of the digit '0' while the anomaly data consists of digits other than '0'. Techniques of data science and predictive analytics can be used to predict the probability of data being normal/anomaly.

Project Objective: To detect anomalous entities in a set of images with a high ROC

Data preprocessing and feature engineering:

To preprocess and engineer the available data, we must first understand the dataset. We are provided with the train_images.csv dataset containing 6,427 data points. Each data point represents the label (0 or 1) corresponding to the image in the train images folder. The test images folder consists 1070 images for which we need to predict the probabilities and submit on Kaggle. Exploring the train dataset, we can observe that there are 5923 normal images and 504 anomalies. This leads to an imbalanced dataset problem.

The only features we had for this project were the MNIST handwritten digits. It is common knowledge that the images are of size 28 by 28 pixels. We can use the appropriate packages to read and store those images in a data structure (array). Keras' image library was made use of to obtain an image of the shape (28, 28, 1) and matplotlib's imread function was used for obtaining an image of the shape (28, 28, 3). These arrays were then stored in cache on my google colab notebook, to save time on reading the images every time the kernel had to be restarted.

*The project must be built using only the given information of the arrangement of the pixels of the images and their corresponding labels. In terms of **feature engineering**, all the pixels were normalized between the values of 0 and 1 while being read.5*

Important Libraries: Tensorflow (Keras), Numpy, Pandas, Sklearn, Matplotlib

Addressing the Imbalanced Dataset Challenge: Before moving on to the models, we need to do away with the imbalanced, skewed dataset issue. For that, we can make use of the class weight hyper parameter in the Keras' sequential model method. We can allocate weights such that the weight is inversely related to the number of entries for each class. It can be done with the following formula:

```
Weight for class 0 = (1 / length of zeroes) * (total length)/2.0
Weight for class 1 = (1 / length of ones)  * (total length)/2.0
```

This will ensure that the model is not biased or skewed towards the majority class.

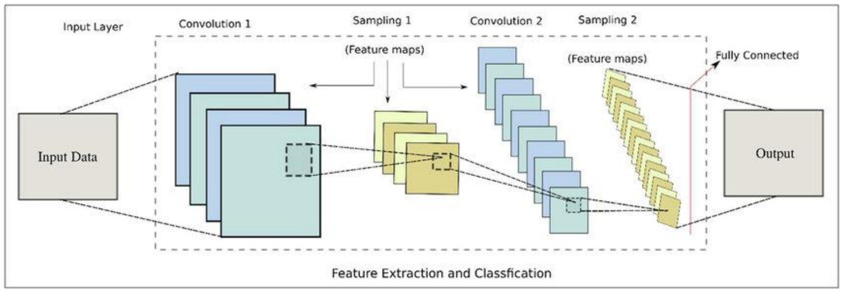
Kaggle ROC score: Our objective is to ensure that the submission of our predictions on Kaggle receives a high accuracy. The receiver operating characteristics curve (ROC) plots the true positive rate against the false-positive rate at any probability threshold. This implies that the model we use should be able to accurately differentiate the anomalous data from the normal data and have a clear threshold between both the classes. Keeping this in mind, we can proceed with the Model Building.

Model Building: Two Models were chosen for this task. One was Convolutional Neural Network (supervised learning) and another was Deep Autoencoder (unsupervised learning).

Architecture:

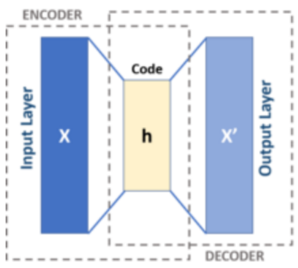
Convolutional Neural Network

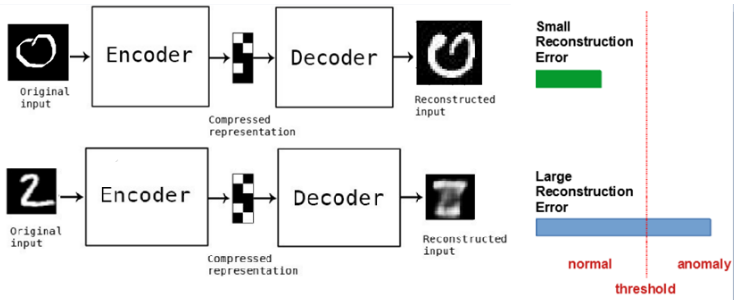
CNN also contains Convolutional layers and Max Pooling layers. Convolutional layers contain many feature maps, which are two dimensional hidden nodes. Every feature map owns a weight matrix called kernel, and different feature maps owns different kernels. Kernels do convolutional operation with every feature map in previous layer, then we sum them up and put it into sigmoid function. The output is the pixel value in the next layer. Once a feature has been detected, its exact location becomes less important. Only its approximate position relative to other features is relevant. Max pooling layers do sub-sampling which reduces the resolution of feature maps and the sensitivity of the output to distortions, so that the model will be more robust.



Autoencoder

An autoencoder is basically an encoder-decoder system that reconstructs the input as the output. This is achieved by two subsystems: the encoder takes the image and maps it into an internal representation, while the decoder takes the internal representation and maps it back to the original input. Auto-encoders are used to generate embeddings that describe inter and extra class relationships.





The autoencoder reconstructs the normal data(top) with a smaller error and the anomaly data with a larger error (bottom).

Why were the models chosen?:

The CNN is a kind of deep architecture which has achieved great performance in image recognition. It is also extremely robust. On the other hand, autoencoder, does not require labels. Hence it is not affected by imbalanced dataset, lack of labelled data and moreover, is less susceptible to noise.

Training, Exploration and Validation of Models:

Convolutional Neural Network: 10% of the data was set aside as test data. The remaining data was used for training the model. Class weights was used to account for the imbalanced dataset. Hyperparameters such as number of layers, units per layer, kernel size, pool size, dropout rate were based on the week 6 lecture code. It stated that first hidden layer of 256 nodes; second hidden layer of 128 nodes and dropout regularization rate of 0.2 resulted in a 98.2% accuracy in the test set for the very same MNIST digit classification problem. The intermediate layers made use of relu activation function. The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better. For the output we need a value between 0 and 1 and hence we make use of sigmoid function with 1 unit for the binary classification. Moreover, I reduced the number of convolutional layers from the original code and played around with the kernel size and pool size. This gave birth to an *alternative cnn model*. As for kernel size, the common choice is to set it at 3x3 or 5x5. The first convolutional layer is often kept larger. Hence the first layer of the alternative model has a kernel size of 5x5 size in the first layer and 3x3 in the second layer. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 down samples at every depth slice in the input by 2 along both width and height. To take each pixel into higher consideration, in the second layer, I have used a 1x1 filter size. *Early stopping was also used in all models to prevent overfitting*. Binary cross entropy is the best optimiser for a classification model whose output is a probability value between 0 and 1. These models were also used in a *K-fold cross-validation structure*. Cross-validation will give us a more accurate estimate of a model's performance with the lesser amount of available data.

Convolutional Autoencoder: For this model, the Training Dataset consisted of 4500 normal images used to train the convolutional autoencoder, Testing Dataset had 500 normal images and the Anomaly Dataset contained 500 anomalous images. The testing and anomaly dataset would be used for verifying if there is a significant difference between the average loss of the normal data and anomalous data to check if the model is overfitting. In this model we consider cross-entropy as the reconstruction error between the original image and the reconstructed image. The normal data would have lower loss and lesser reconstruction error as the model is trained against the normal data. However, the very same model will face a higher reconstruction error and loss when tested against anomalous dataset. Similar to the CNN case, 2 variations of this model were used. One was autoencoder with 2 layers and another was a Deep autoencoder with 4 layers.

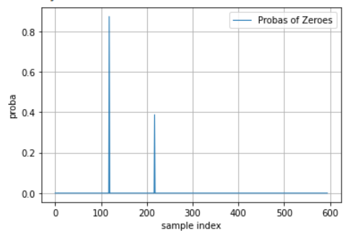
Mean squared error was used as opposed to binary cross entropy in the CNN model. In terms of activation functions, inner layers made use of relu while the outermost layers made use of a single sigmoid function, like the CNN model. Adam optimizer was used in both CNN and autoencoder models. It is an excellent optimization algorithm for stochastic gradient descent for training deep learning models.

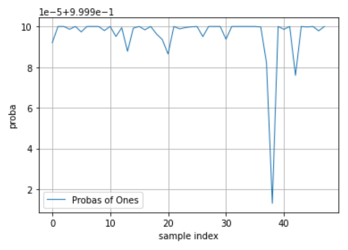
Test Outcome and Result: Highly important part of this project is the testing of the trained model against the test data. By predicting the proba on unseen data in the case of CNN or by calculating the reconstruction error in the autoencoder model, we can observe how well the models can distinguish between both the classes.

Simple CNN Model

```
[25] plot_probas(model_1, x_test, y_test)

     Accuracy score is: 1.0
```

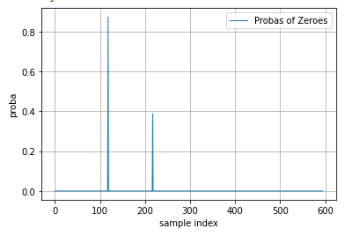


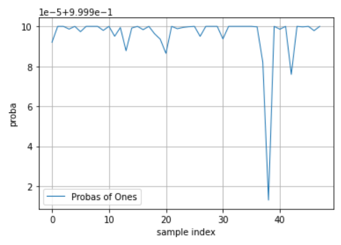


Simple CNN Model with KFold CV

```
[27] plot_probas(model_3, x_test, y_test)

     Accuracy score is: 1.0
```





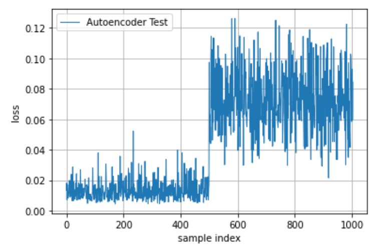
CNN model. Based on the test set, it seems like the model can perfectly distinguish between both the classes. The K-fold seems to have no difference as the initial model (with its 50% validation split) is already good enough. But the test data is relatively small, so the performance of the model can't be measured clearly. This is the model I submitted.
CNN (left), K-Fold CNN (right)

```
------------------------------------------------------------------------
verage scores for all folds:
› Accuracy: 99.82722878456116 (+- 0.26756416490750595)
› Loss: 0.028923493074944417
------------------------------------------------------------------------
```

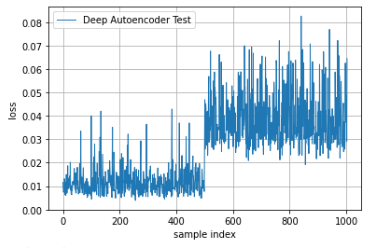
```
Epoch 10/10
91/91 [==============================] - 3s 31ms/step - loss: 2.1027e-04 - accuracy: 1.0000 - val_loss: 0.0022 - val_accuracy: 0.9993
<tensorflow.python.keras.callbacks.History at 0x7ff9e896b5f8>
```

(Deep) Autoencoder model: While fitting the model on the test and anomalous data, it seems like the autoencoder model (left) has a clearer threshold in the reconstruction error/loss between both the classes than the deep autoencoder model (right). Either way, both models are superior to the CNN model due to the clear segregation between the classes. However, I could only get this model without any errors after the deadline on Kaggle.

```
losses_autoencoder = compute_losses(model_auto, x_concat, 'Autoencoder Test')
```



```
losses_deep = compute_losses(model_deep, x_concat, 'Deep Autoencoder Test')
```



```
Epoch 10/10
71/71 [==============================] - 0s 5ms/step - loss: 0.0116
<tensorflow.python.keras.callbacks.History at 0x7ff9cd98da58>
```

Conclusion

Supervised learning requires labels on all the images which is not only labor-intensive but also potentially noisy. As a result, unsupervised learning could be a reasonable approach or companion in some anomaly detection problems. As observed in this case, both CNN model and the autoencoder model seem to have been able to differentiate both the classes, given the right hyperparameters. It seems that both of these deep learning models are good at anomaly detection in an image classification problem. This can be observed from CNN's train accuracy, validation accuracy and out-of-sample test accuracy. In the case of autoencoder, it can be observed from the clear divide between the losses of reconstructing the normal and anomalous datasets. Based on the Kaggle submissions, it seems like the models can reach up to a 98% accuracy on the test set with the autoencoder model and up to 96.6% with the convolutional neural network model.