**CIND860 Advanced Data Analytics Project:**

**Fashion MNIST - Evaluating the Efficacy of CNN against Traditional ML Models**

**Initial Results & Code**

**By: Hasib Chowdhury (500439135)**

**Supervisor: Dr. Tamer Abdou**

**GitHub Link:** [**https://github.com/Hasib147/CIND860-Capstone-Project**](https://github.com/Hasib147/CIND860-Capstone-Project)

**Date of Submission: November 10, 2023**

**Exploratory Data Analyses:**

An embedded technique known as the Random Forest Importance was used to take the top 20 pixels based on the ‘Importance’ value of the Fashion-MNIST train dataset (60,000 rows and 785 columns with column 1 being the dependent variable). The reason top 20 was used was because it is a decent value and the system would not crash (due to limited capabilities) or take a long time to generate a panda profiling report. The following 20 pixels on the train dataset had the highest importance based on the correlation of the dependent variable (label) and other independent variables (the remaining number of pixels). They do contain a high number of zeros, which were taken into account but of a lesser amount than the other pixels.

Rank Pixel # Importance

1 pixel547 0.009806

2 pixel603 0.009392

3 pixel491 0.008809

4 pixel575 0.007279

5 pixel263 0.006986

6 pixel407 0.006045

7 pixel435 0.005642

8 pixel687 0.005324

9 pixel519 0.005292

10 pixel631 0.005287

11 pixel401 0.005284

12 pixel180 0.005119

13 pixel234 0.005010

14 pixel236 0.004944

15 pixel68 0.004934

16 pixel611 0.004891

17 pixel96 0.004734

18 pixel555 0.004687

19 pixel594 0.004502

20 pixel427 0.004482

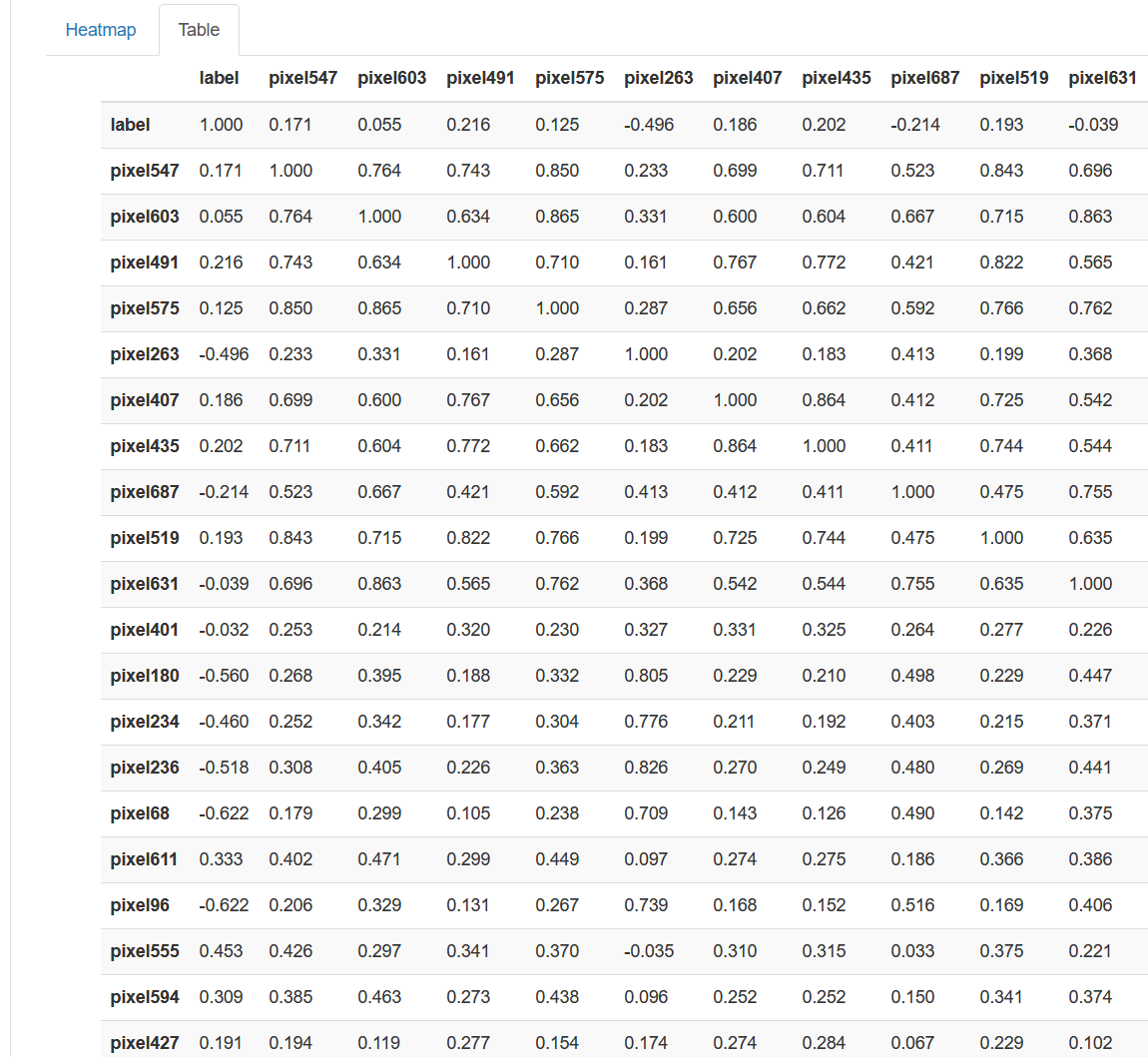
The number of trees used to generate this output was 100 trees based on the Random Forest Importance (RFI). I chose this value is because it fits in nicely with the 60,000 total rows of the train dataset. Also in practice, the number of trees can be tuned using techniques like cross-validation or grid search to find the optimal value for specific problems within deep learning. According to ChatGPT, “Using more trees can lead to a more robust model but might increase computational complexity but using fewer trees might lead to a less complex model but could lead to the model overfitting. The optimal number of trees can depend on factors like the dataset size, complexity, and available computational resources” [1]. I will experiment with the different number of trees and see which value is the most sufficient to use for this specific dataset of the top 20 pixels. In addition, this will be compared to other ML models such as SVM , XGBoost and the traditional simple-CNN model (with the basic parameters done on a simple scaled test set).

**Data Preparation:**

A screenshot of a computer

Description automatically generated

The above graph shows the heatmap and how each pixel correlates to one another and the dependent variable (label). This specific heatmap was generated from the pandaprofiling report using the combined dataset of both training and testing together (70,000 total rows), consisting of the top 20 pixels (ranked according to Importance based on RFI 100 trees). The pixels near the centre of the diagram, are light to dark blue indicating strong correlation between the pixels, whereas the other pixels are somewhat light and sort of faded, indicating a weak correlation.



The above table (taken from Pandaprofiling) shows that most of the independent variables (pixels) have a positive correlation with one another and some have negative correlation. They can vary between weak correlation (0.100 to 0.399) to medium (0.400-0.699) to high correlation (0.700-0.999). Each variable has a perfect correlation (1.00) with one another in the horizontal and vertical column.

**Modelling:**

To answer one of the research questions from the modified abstract of this project. I compared the simple-CNN model with the basic parameters and compared it with different machine learning models such as Random Forest, SVM and XGBoost. And to compare their accuracies based off the exact same dataset, top 20 pixels from the combined dataset based on RFI 100 trees (with the label column and 70,000 rows).

The following is from the simple-CNN-model:

A screenshot of a computer

Description automatically generated

It gave 75.68% as the test accuracy, which is much lower than expected mainly cause of the small sample size (70,00 rows) and in the research paper there were over 110,000 trainable parameters listed on p. 992 of the article (they also received a test accuracy using 2 different TensorFlow versions) and in my case I only used the physical dataset with the numbers and one TensorFlow version (most recent version) 2.14.0 and Keras version 2.14.0. In addition, the research paper that I am replicating (on section 5.3 cnn-simple, page 992) indicates that “All those models were modeled based on Keras Sequential model. Convolutional and dense layers used Rectified Linear Unit (ReLU) activation functions, except by the last dense layer on each model (output layer), were Softmax was used. The optimizer used was Adadelta, Batch size was 128 and we trained the models for 12 epochs” [2]. I did this differently from what they did. The reason I chose ‘Adam’ was because I had used it in the past for this exact same dataset in the CIND850 course and in general, it is a popular and effective optimizer that works well generally for many deep learning tasks such as image classification (which is used in this case). The choice of optimizer is problem-specific and I used that but other optimizers like what they used (Adadelta) or other optimizers such as ‘SGD’ or ‘RMSprop’ may also work.

As for batch size and the number of epochs, as mentioned earlier I am using only 70,000 rows so I used lower values than the paper did. I used 10 for number of epochs is cause I’m working with a limited number of rows and columns and I didn’t want it to set it too low, which could cause underfitting or setting it too high which can lead to overfitting. I think 10 is a decent value for number of epochs in this particular situation, however if it was over 100,000 rows I may have done 100 epochs to speed up the process. For batch size, 64 is a decent value and I think it will work well for this particular dataset, reason being too high of a batch size can speed up the training but use too much computer memory (which I am limited to) and too low of a batch size such as 16 or 32 can be decent as well since a smaller batch size allows for more frequent weight updates and can help the model converge faster, but it may also require more epochs. That is why I chose 64 as the batch size value.

For the loss values/accuracy, steps 1-10 epochs gave 10 different loss & accuracy values along with the final test accuracy when the model was completed. Based on the observations, the numbers on the accuracy kept increasing in every single succeeding epoch (from 1 to 2, 2 to 3, etc.), whereas the loss value decreased in every single step (from 1-10 epochs). The loss value, in particular is obtained by the loss function that is defined when compiling the model. The value is calculated by assessing the difference between the predicted value and the actual value. A common loss function is 'categorical cross-entropy' (which I used) for multi-class classification tasks like Fashion-MNIST. As for the accuracy, it is reported in the training output, and it’s calculated by the model's evaluation on the provided data (70,000 rows, 21 columns) in each epoch. It represents the proportion of correct predictions the model makes. During training, the accuracy is calculated based on the model's predictions for the training dataset, comparing them with the true labels. There may have been trial & error involved in this due to many factors such as the sample size being used and only the top 20 pixels being used to train the whole dataset. This could have been higher if more epochs/batch size were used but due to memory constraints only the top 20 were considered. According to ChatGPT, “The loss and accuracy metrics provides with more details into the model's learning and generalization to new data. The goal is to reach an optimal point where the model isn't overfitting or underfitting (somewhere in between) and has a good generalization ability” [1].

Lastly, in terms of the parameters that were used when making the simple-CNN-model, there were 2 hidden layers (the ‘Conv2D’ and ‘MaxPooling2D’). In the code, the input layer is created using the Conv2D method: ‘layers.Conv2D(32, (3, 1), activation='relu', input\_shape=(20, 1, 1))’. This sets up the initial convolutional layer with 32 filters/kernels applied to the original data (70,000 rows) and each filter/kernel generates a different output each time, the (3, 1) is the filter size (a 3x1 matrix), using the ReLU activation function, and an input\_shape of (20, 1, 1), which indiactes the dimensions of the input data (20 x 1 x 1, in other words 20 rows, 1 column and black-and-white image) depicting the Fashion-MNIST dataset. As for the Max pooling, the ‘(2,1)’ represents the pooling size (2 rows 1 column pooling matrix). There were 2 dense layers, with 2 different activation functions, the values of ‘128’ and ‘10’ are used as the number of neurons in each respective dense layer. 10 is used on the 2nd dense layer because there are 10 classes of fashion images (labels 0-9, representing different fashion attire). In the first dense layer, it uses ‘ReLU’ as the activation function for the hidden layer with 128 units and it is effective in training deep neural networks. In the second dense layer, ‘Softmax’ is used as the output layer's activation function for multi-class classification (10 classes), generating class probabilities. The class with the highest probability is usually predicted.

The following is from the Random Forest method:

A screenshot of a computer

Description automatically generated

I ran 3 different values for the ‘n\_estimators’ to try out 3 different values for the trees and see which gives the most significant results. They were very close with one another (less than 3 percent difference), 10 trees gave an accuracy of 73.51%, 50 trees gave an accuracy of 75.41%, whereas 100 trees gave a 76.11% test accuracy. These values can differ each time due to the specific trees being chosen to be evaluated and it is very high due to only the top 20 pixels being chosen from test set (784 total pixels in total). If 100-200 pixels were used instead of 20, the test accuracy may have been lower but can’t be used due to the system crashing. In the research paper, it was not indicated how many trees were used but it gave a 87% accuracy result on the table.

In addition, I also created a graph (full code on the .ipynb file) depicting the graph of the random forest trees based on it’s test accuracy on the 70,000 rows of data (top 20 pixels). I tested on a wide interval, from 0-500 trees:

A graph with a line going up

Description automatically generated

A grand total of 12 different values for trees were used in this interval (as shown on the graph above, 12 blue points) and from the looks of this, it stabilizes after 400 trees were compiled. The following was the output of the results:

Test accuracy (n\_estimators=10): 73.51%

Test accuracy (n\_estimators=20): 74.70%

Test accuracy (n\_estimators=50): 75.54%

Test accuracy (n\_estimators=100): 76.11%

Test accuracy (n\_estimators=150): 76.13%

Test accuracy (n\_estimators=200): 76.37%

Test accuracy (n\_estimators=250): 76.36%

Test accuracy (n\_estimators=300): 76.41%

Test accuracy (n\_estimators=350): 76.48%

Test accuracy (n\_estimators=400): 76.52%

Test accuracy (n\_estimators=450): 76.59%

Test accuracy (n\_estimators=500): 76.49%

When it compiles the ‘500’ tree value, it goes down but 0.10% from the ‘450’ tree value and this output was generated in intervals of 50 trees at a time. So most likely, 450 trees may have been the optimal value to use as this point in the graph is where the line flattens and the accuracy is more or less the same moving forward in the graph. In my situation however, due to the memory constraint I stuck with 100 trees but for better results 450 is more beneficial to use but it can take a long time to compile and run. There is less than one percent difference between 100 and 500 trees, so depending on how much memory one’s CPU has, any value between can work.

The following code is from the SVM & XGBoost model respectively:

A screenshot of a computer

Description automatically generated

Out of the 4 models compared, the test accuracy were very close to one another in the low to mid 70’s range. The following table summarizes the results of the accuracy done on the combined dataset (70,000 rows) of the top 20 pixels based on RFI – 100 trees:

|  |  |
| --- | --- |
| **Model** | **Test Accuracy** |
| CNN-simple-model | 75.68% |
| Random Forest – 100 trees (my choice) | 76.11% |
| Random Forest – 450 trees (optimal choice) | 76.59% |
| SVM | 72.61% |
| XGBoost | 76.41% |

The XGBoost outperforms all the other models. In the research paper, they tested 17 different ML models whereas here I only compared 4. I could have experimented with 3 different dropout method like they did (CNN-dropout-1, CNN-dropout-2, and CNN-dropout-3), but will do so on the next stage of the project. In their situation (with several parameters from the dataset) they got results ranging from low 80’s to high 90’s in 12 out of their 17 models. I ended with the low to mid 70% range accuracy cause of the limited CPU memory and only working on a sample 70,000 total rows from the dataset (60,000 training and 10,000 testing). There were several factors that lead to the results, the main being the limited dataset and only certain parameters being involved in training and testing as well as using only top 20 of the total 784 pixels. In the research paper, it more than likely took into account all the pixels (although not quite specified but a huge number of trainable parameters were accounted for).

References

[1] OpenAI. (2023). ChatGPT (GPT-3.5 model) [Large language model]. <https://chat.openai.com/chat>

[2] LEITHARDT, V. (2021). Classifying garments from fashion-MNIST dataset through CNNs. *Advances in Science, Technology and Engineering Systems Journal*, *6*(1), 989-994.