Comp219 Assignment 2 Document

## University ID: 201458436

Please read this document before running the files.

# Check list of marking criteria:

F1 - Implemented successfully

The following F2 criteria have not been implemented on the model from assignment 1 using knn, as my knn did not have a saved model

F2:

* Cross Validation – Implemented successfully
* Confusion Matrix – Implemented successfully
* ROC Curve – Attempted to implement but was unsuccessful (described why later in the document)

# Summary

I am going to train a neural network using the sequential model on the digits dataset (Optical recognition of handwritten digits dataset). On my machine it takes me around 9.2 seconds to run my program without the convolutional layer and around 11.8 seconds to run my program with the convolutional layer. The accuracy of my algorithm without a convolutional layer is around 97% and the accuracy of my algorithm with a convolutional layer is around 99.6%. My implementation of the KNN algorithm gives an accuracy of around 98.5% however, on my machine, it takes me 131.83121299743652 seconds to run.

I am running all files on an Intel i7-7500U CPU@ 2.90GHZ, with 8GB of RAM on Windows10.

# Submission files

1. CNN with Convolutional Layer
2. CNN without Convolutional Layer
3. CNN\_digit\_ConvLyr.hdf5
4. CNN\_digit\_NoLyr.hdf5
5. KNN algorithm (my own implementation)

Files 1 and 2 are my source code for the convolutional neural network without and with a convolutional layer respectively. Files 3 and 4 are the saved models. File 5 is my own implementation of the K-Nearest Neighbour (I will proceed with calling this KNN) algorithm, which will be used to some extent for comparison with files 1 and 2.

# Software dependencies

In order to run both programs, you will need to have the most up to date version of Python3.8 installed. Similarly, you will need to install scikit-learn, NumPy and Matplotlib. Along with this, you will need to have the most recent updates of Keras and Tensorflow. If you do not have an HDFView application, you will need to install one in order to view the saved models, or pip install h5py in the windows shell.

# How to run Comp219 Assignment 2 file 1 and 2

In order to run my first and second files, simply press f5 on the keyboard after opening the program in idle. This will load the digits dataset straight into the program and train our sequential model on the dataset, displaying each epoch, the loss and the accuracy of each epoch respectively. Then the program will display the summary of the model, which shows the model type, each layer (and their type), the output shape of each layer and the number of parameters of each layer. The program will also display the total number of parameters and trainable/non-trainable parameters. Then, each program will display the correct K-Folds cross validation, along with the correct confusion matrix. After this, the program will print the number of True Positives, False Positives, True Negatives and False Negatives, along with the False Positive Rate and True Positive Rate respectively. Finally, both of the programs will attempt to plot the ROC curve however, they will be unsuccessful. The reasoning for this will be explained in the implementation of File 2, when explaining the results() function. After you close the ROC curve, it will display how long it took the program to run.

# How to run Comp219 Assignment 2 file 3 and 4

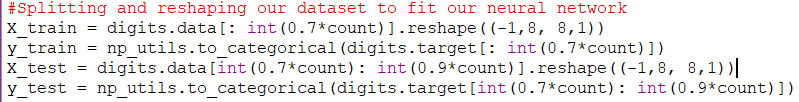
In order to view the saved models of files 1 and 2, simply right-click or long-press the file. Then click "Open with" and choose the HDFView application.

# How to run Comp219 Assignment 2 file 5

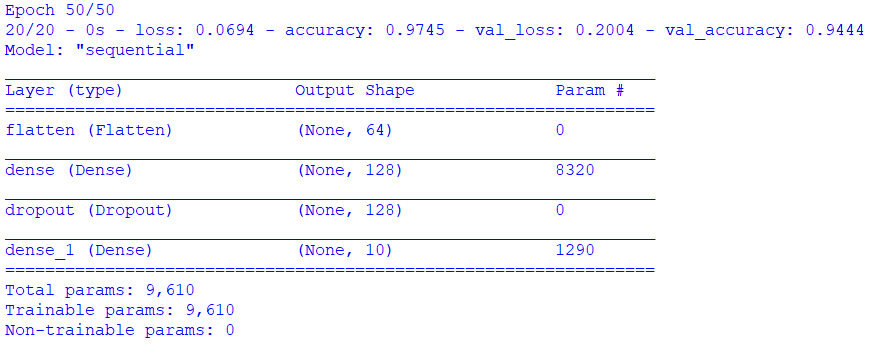
In order to run the second file, simply press f5 on your keyboard after opening the program in idle. This will load the digits dataset straight into the program and train our own knn algorithm using the predictClassification function, the getNeighbors function and the Euclidean function. When you run the program, it will display the accuracy of the knn algorithm on the training set, the accuracy of the knn algorithm on the test set and how long it takes for the program to run in seconds. There is clear commenting throughout the program, explaining each function. In summary, in order to correctly classify each digit, we need to calculate the nearest neighbours, which we can only do if we calculate the Euclidean distance between the test data point and all the training data.

# Implementation of File 2 (neural network without a convolutional layer)

I will describe File 2 implementation before File 1, as this is the neural network without a convolutional layer.

Frist of all, we need to load the dataset and define our target classes along with the data. Then, we create a variable called count. This is responsible for containing the length of our dataset. It is used for splitting our data. We then split our data into training data and testing data, reshaping our data. We reshape our images in 3 dimensions (height = 8px, width = 8px , canal = 1), the first dimension of is samples. Here we reshape our data into 3D matrices. Then we use one-hot encoding to label our 10 digit classes. This is done in order for our neural network to be able to predict the classes more accurately, and is an important step for a fully connected neural network.

After this, we create a neural network function called NN(), which takes no parameters. Here we select the Sequential() model, which allows us to build our neural network layer by layer. We use a Flatten() layer to transform our matrix into a vector. Then, we use a Dense() layer which applies the activation layer to the dot product rule of our input and the kernel considering weights. Then we add the Dropout() layer, which randomly selects neurons to dropout. This is a simple overfitting technique. We then add another Dense() layer but this time it considering the output of 10 possible classes and using SoftMax to classify our data. We then compile our model, using categorical cross entropy to calculate our loss, use the oprimizer=‘adam’ and the accuracy as our metrics. Finally we return our model.

We then call our neural network function, and provide a history and a summary of the model. The history is us actually fitting/training our classification model on our training set, using the testing data as our validation data, with a batch size of 64, 50 epochs and a verbose of 2. The image bellow displays the last epoch and the model summary.

We then save our model as a hdf5 file as required for F1. At this point, the F1 for file 2 is complete and we move onto F2. We begin this by creating 2 important variables; y\_pred\_classes and y\_true. The first variable, takes the model predictions on our testing data, and uses argmax to find the predicted classes of our model. The second variable uses argmax to find the true classes. Both of these are stored as arrays.

I then created an accuracy function, as the sklearn accuracy\_function did not work with our data, as I believe it does not work with the data types I provide in my next function; I will cover this after my crossVal() function.

To meet the first requirement of F2, I needed to provide a cross validation of my model on the dataset, which I did via the crossVal() function, which takes 2 parameters: the predicted classes and the true classes. I implemented the K-Folds cross validation by splitting the dataset into 5 folds and comparing the models accuracy my outputting the error rate. We use 5 folds as it is the recommended number for any K-Folds cross validation and by doing so we measure the performance of the model across the 5 folds, given the standard deviation. I then created an accuracy() function, which takes 2 parameters: the dataset and the predictions. Then the function essentially returns the error rate. We call the crossVal() function in the global sphere, outputting the K-Folds cross validation error rate of our model.

Then we move onto the second part of the F2 requirement, the confusion matrix. In order to meet this requirement, I created a confusion\_matrix() function, which takes 2 parameters: the true labels and the model’s predicted classes. This was a much simpler function to create, as there isn’t much really going on here. We make an array size 10x10 and input all of the unique elements which match or don’t match in the y\_true array with the y\_pred\_classes array. We then print the array.

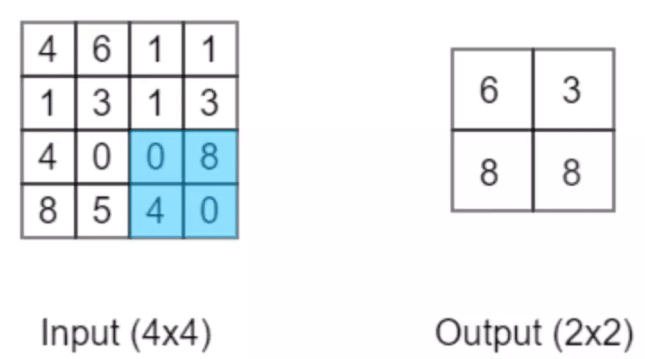
The next part of the F2 requirement was very difficult for me to implement, and even though I got very far in its development process, I was unable to complete it. I will display exactly how far I got and what my biggest challenge ended up being. This requirement is actually split up into 3 separate functions: perf\_measure(), TPR\_FPR() and result(). The first function, perf\_ measure(), takes 2 parameters, y\_actual and y\_pred, which are respectively representations of the variables y\_true and y\_pred\_classes. Here we calculate and return all of the true positives, false positive, true negatives and false negatives (I will continue referring to them as TP, FP, TN, FN respectively).

Then, I created a function called TPR\_FPR(), which takes 4 parameters: TP, FP, TN and FN. Then, from the equations which we already know, I calculate the false positive rate and true positive rate (I will continue calling these FPR and TPR respectively). I then call the perf\_measure() function in the global sphere, giving it the 2 parameters as the variables y\_true and y\_pred\_classes, labelling the returned values as TP, FP, TN, FN. I then print the values of TP, FP, TN and FN in order to understand if my function is working properly and the TPR and FPR are correctly. Similarly, I call the TPR\_FPR() function in the global sphere, giving it the 4 parameters as the stored values of TP, FP, TN, FN. I then store the returned values of FPR and TPR, printing them in the next line, in order to make sure that my TPR\_FPR() function is working effectively.

Finally, I created the results() function, which takes 2 parameters: x (which represents the FPR) and y (which represents the TPR). The purpose of the results() function is to plot the ROC curve, using the FPR, TPR and AUC (area under the curve). However, when creating the AUC variable, I attempted to use the np.trapz() function, to integrate the TPR by the TPR. This was unsuccessful, and resulted in an error, as the NumPy trapz() function could not preform this operation due to the list assignment function being apparently out of range. I have commented out these important pieces of code which were unsuccessful with 4 “#” for reference. I hope that this is taken into consideration when grading this section of F2. At the end of the program, I call the results() function and print the amount of time taken for the program to run.

# Implementation of File 1 (neural network without a convolutional layer)

This program is very similar to File 2, as most of the implementation is identical and the only real difference is the application of the convolutional layer, which I will describe bellow.

The main difference between File 1 and File 2 is the function NN() in File 2, is replaced with the CNN() function in File 1. The CNN() function takes no parameters and returns the fully functioning model, which is then called in the global sphere. We still use the Sequential() model and the same neural network layers as previously. However, here we use a number of convolutional functions to create a convolutional layer such as Conv2D() and MaxPooling2D(). The Conv2D() function is a 2D convolutional layer, which creates a convolutional kernel, which is wound with layers input, that help produce a tensor of outputs. The MaxPooling2D() function, is a maximum pooling operation, that calculates the maximum value in each patch of each feature map. This operation can be seen bellow as a summary. The bottom right quarter of the 4x4 input has the largest value of 8, just as the largest value in the bottom right quarter of the 2x2 output.

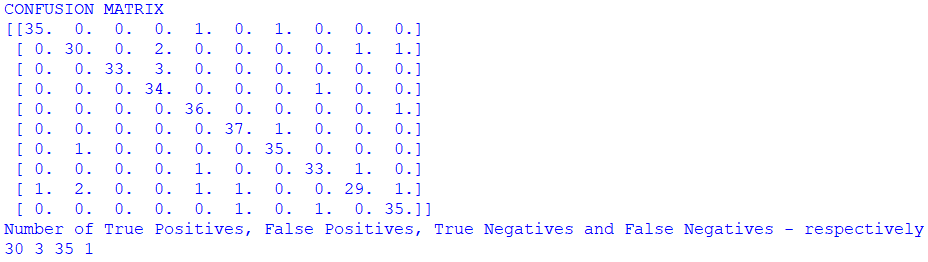
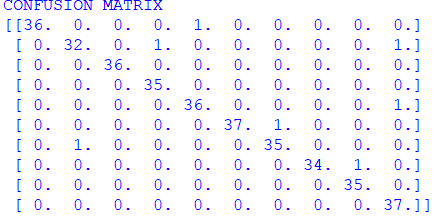
Then we add another Dropout() layer, to again avoid overfitting. The model is then compiled and returned when called in a global sphere.

I won’t cover the rest of the File 1 implementation as it is essentially exactly the same as the File 2 documentation, which has already been covered. Instead I will move on to the findings.

# The discovery

First, I will begin by discussing the accuracy of the files 2, 1 and 3 (in that order), as I am able to compare the accuracy statistics for all of these files. Then I will break down the error rate of files 2 and 1 based on the K-Folds cross validation. After that, I will compare the confusion matrix of the testing data and the number of TP, FP, TN, FN. Finally I will compare the FPR and TPR of files 2 and 1, as I am unable to compare the ROC curves and area under the curve.

The accuracy of File 2 is around 97%, taking around 9.2 seconds to run, whereas the accuracy of File 1 is around 99.6% and takes around 11.8 seconds to run. As expected, the neural network with a convolutional layer is more accurate takes longer to run, as there is more information to process, as well as being around 2.6% more accurate. This might seem like a small increase in accuracy however, this result becomes more interesting when looking at the confusion matrices. My own implementation of the KNN algorithm gave an accuracy of 98.5% however, this took 131.83121299743652 seconds to complete. It also needs to be considered that this is on a much smaller dataset, as it splits the training data into 1000 images, compared to File 1 and 2’s 1258 images. When the KNN algorithm was trained on the entire dataset of 1797 images, its accuracy was only 0.5% better however, this took approximately 9 minutes to complete. This shows that the convolutional neural network (File 1) is significantly more efficient and accurate than my implementation of the KNN algorithm.

 The error rate for File 2 was almost identical to the error rate found for File 1. Both of these hovered around 9.9% however, when comparing the confusion matrices for both of these files (File 2 on the left and File 1 on the right), we can see that the convolutional neural network has made significantly less incorrect classifications. The average number of TP, FP, TN and FN for File 2 were 30, 3, 35 and 1 respectively, whereas the average number of TP, FP, TN and FN for File 1 were 32 1 36 and 0 respectively. This shows that on average, the average number of TP and TN was significantly higher for the neural network with the convolutional layer however, the average number of FP and FN was lower. As expected, this shows that the neural network with a convolutional layer was more accurate at making correct classifications as well as avoiding incorrect classifications. The FPR of File 1 was 0.02702702702702703 and the TPR was 1.0, whereas the FPR for File 2 was 0.07894736842105263 and the TPR was 0.967741935483871. From this information, we can gather that the FPR of the neural network without a convolutional layer was approximately 5.19% higher than the FPR of the neural network with a convolutional layer. Similarly, the TPR of the neural network without a convolutional layer was approximately 3.23% lower than the TPR of the neural network with a convolutional layer.

From the information we have seen in this discovery, I believe it is obvious to conclude with the fact that the neural network with a convolutional layer (File 1) is significantly more accurate than the neural network without a convolutional layer (File 2). Even though File 2 took longer to run, I believe that the trade off in accuracy is worth it for such a small increase in length of time. Similarly, I believe that we can conclude that my own implementation of the KNN algorithm was not as accurate as the neural network with a convolutional layer, as well as taking far longer to complete. Even though File 5 was more accurate than File 2, it took over 13 times longer to complete and I believe that the amount of time taken was not worth the accuracy gained, when File 1 was still more than 10 times quicker and still more accurate.