Total words count: 3000



**Machine Learning- COIY065H7**

coursework report

STUDENT NAME: Bernard fromson

Mres Computer science

**Emails:** [bfroms01@mail.bbk.ac.uk](mailto:bfroms01@mail.bbk.ac.uk)

bhfromson@gmail.com

11 – May - 2020

# *Academic Declaration*

# *I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software.*

# **Introduction**

The aim of this work is to evaluate the WAME optimisation methodology described in [1]. This has been done using the Landsat dataset [2] consisting of 6435 instances, split into two groups, a training group of 4435 instances and a test group of 2000. Each instance contains data from a 3x3 area with 4 values per pixel giving 36 values along with a single value classifying the central pixel. The 36 values are used as features to predict the central classification. Previous work on this dataset has suggested that a neural network approach gives the best classification results [3].

The work has been coded using python and the PyTorch deep learning library. A basic neural network has been coded and a framework established that allows the network shape and depth and other parameters to be input from a table and multiple runs of the network made to compare the effects of these parameter changes. This is described in detail in the Methodology and Design section.

The groups of parameter changes and the effects these had on the results are presented and discussed in the Experiments, findings and discussion section.

Conclusions are drawn in the final section of this work.

# **Methodology and Design**

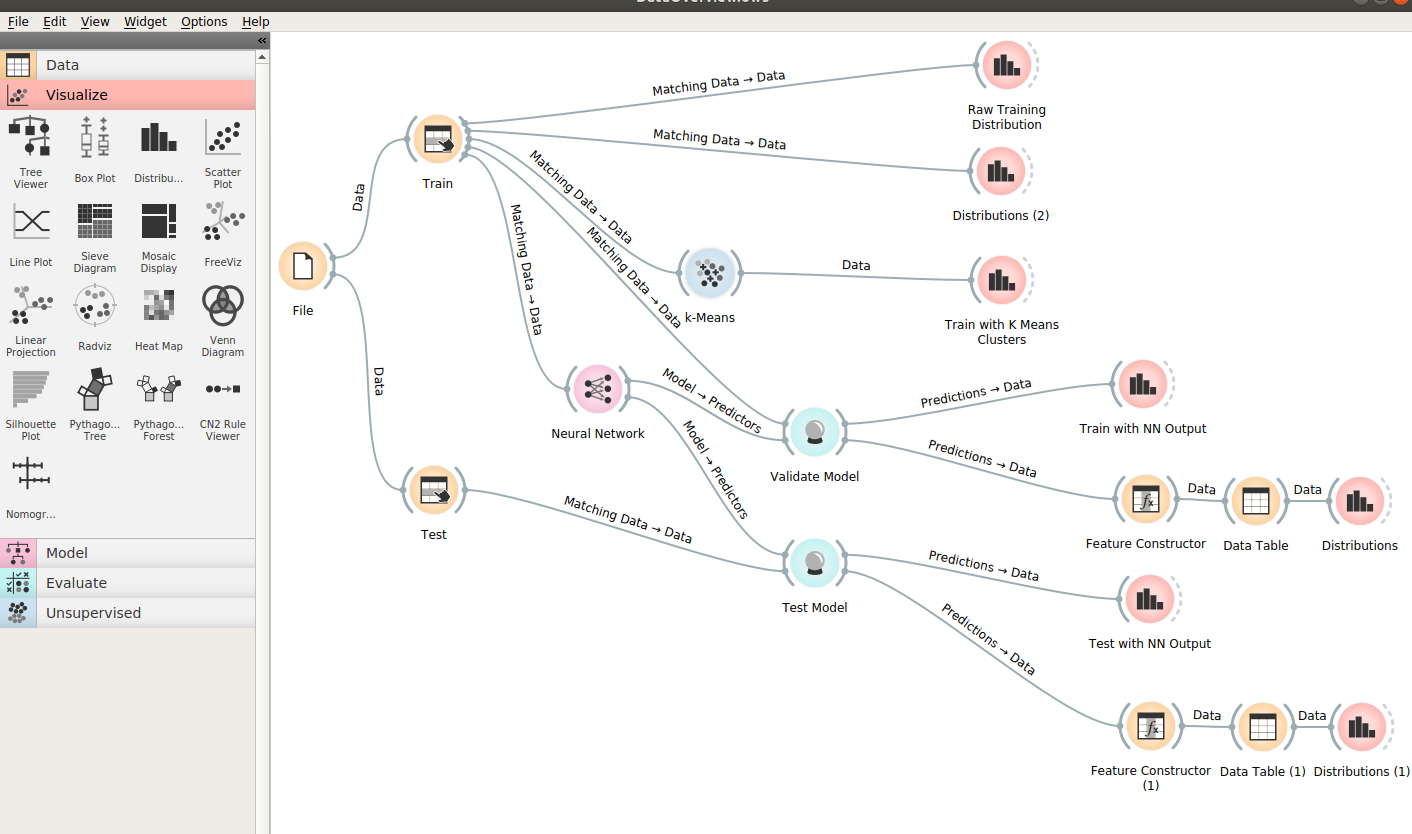
The approach to this work was taken in stages. Firstly the data was reviewed using visualisation tools to assess its distribution and potential problems. The WAME optimiser was built using a standard dataset, then a flexible Neural Network was built to allow parameters to be easily changed. The experiments were then conducted through this framework.

**a) Data Review**

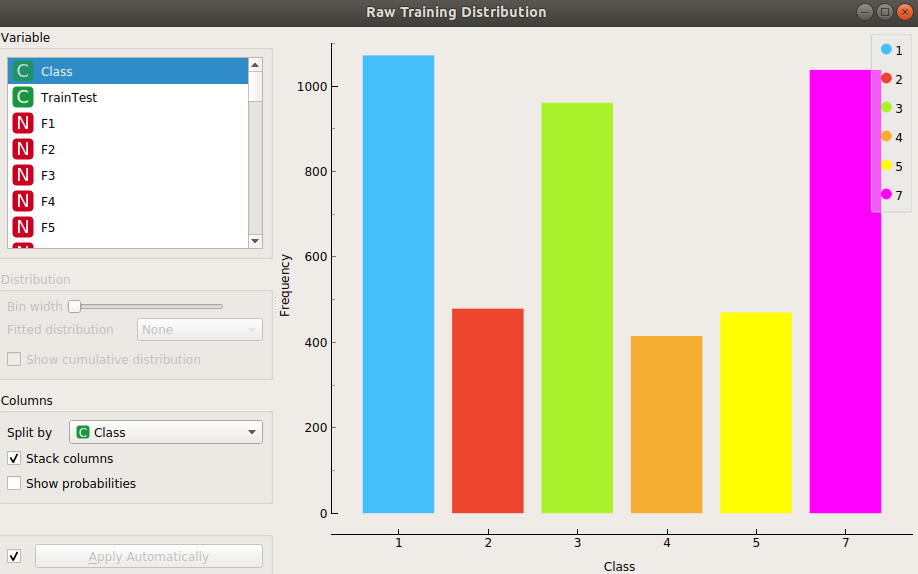
The Landsat data is contained in two files, one with the training dataset and one with the test dataset. To simplify all further processing the two datasets were combined and an additional field added indicating the sorce of the data as ‘train’ or ‘test’. The fields in the combined dataset were named with F1-F36 for the 36 features and Class and TrainTest for the class and origin fields.

The combined data was initially loaded into the Orange framework [4] to allow quick views of the class distribution. This also allows a test Neural Network (NN) to be built in the visual environment to see how succesful the approach might be.

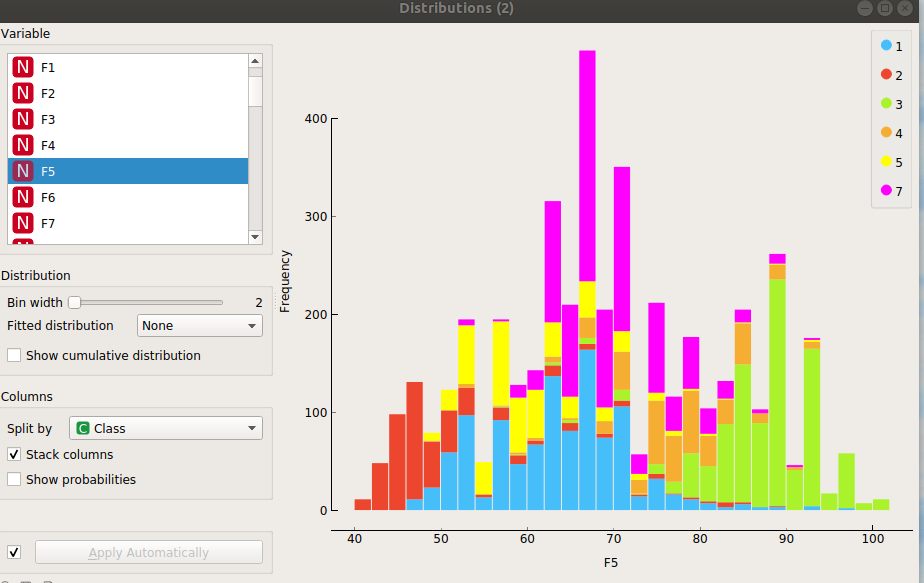
Orange Data Schematic:

  
Illustration 1: Orange - Schematic

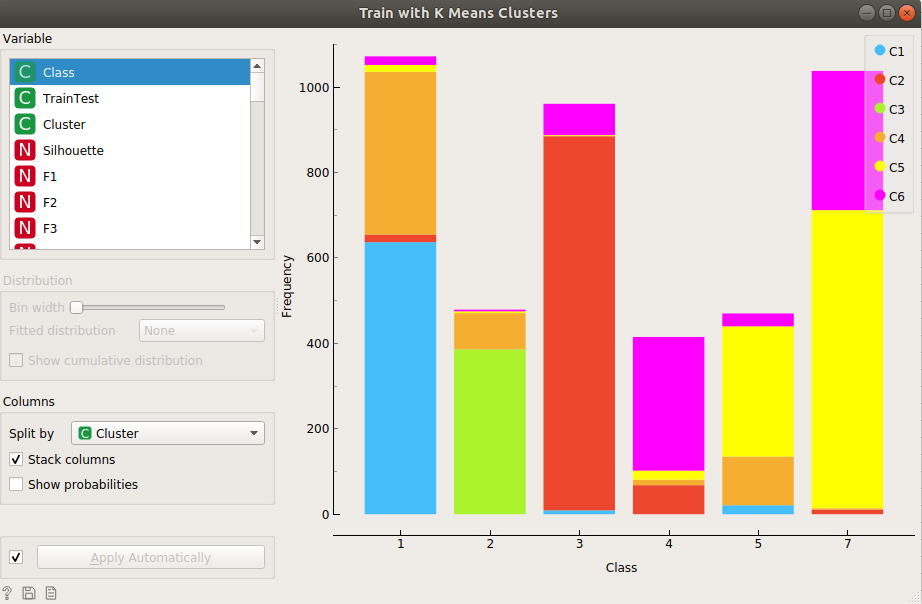
The class distribution within the training data shows no instances of class 6 and an assymetric distribution with classes 1,3 and 7 being 2-3 times more frequent than classes 2,4 and 5:

  
Illustration 2: Orange - Class distribution in training data

The individual features can be quickly viewed and the distribution of classes within each feature easily seen. Looking at Feature 5, the distribution of classes 2 and 3 at the extreme values can be observed

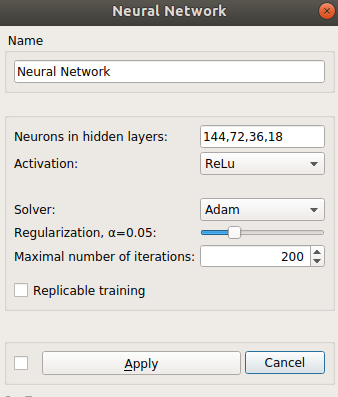
  
Illustration 3: Orange - Single feature distribution with class occurence

Orange allows basic models to be quickly built and tested on the data. A k-Means cluster and simple NN were tried.

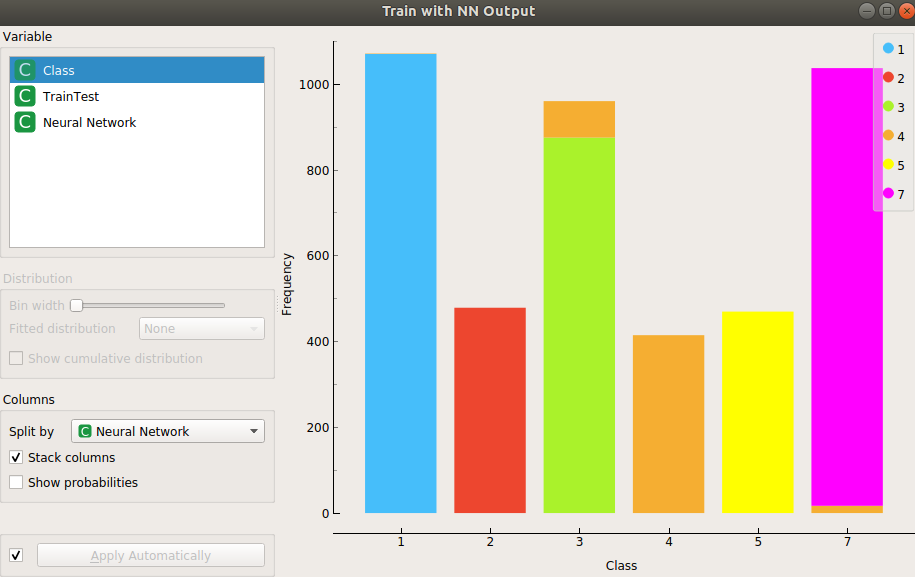
  
Illustration 4: Orange - k-Memas cluster, each bar is one class stacked by the k-Means cluster prediction

The stacked bar chart from the k-Means clustering shows a very poor model. The colours within each class show the cluster allocation, and only class 3 has a single predominant cluster indicating a good recall, all the other classes have very mixed cluster results. It is unlikely that any further refinement would make this a useful algorithm.

The simple NN can be specified using a few parameters:

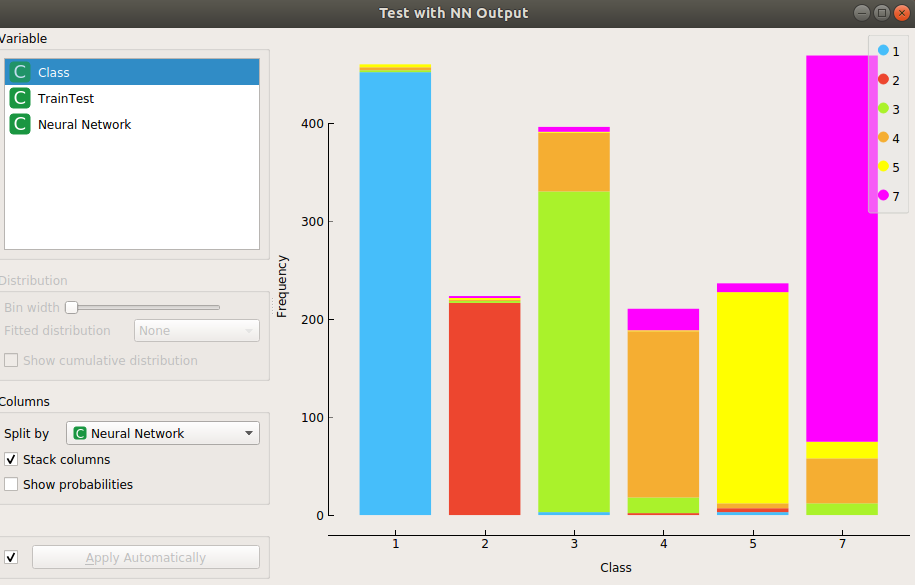
  
Illustration 5: Orange - NN Parameters

The NN wass built from the training data and then applied to that data and the distribution of predicted classes within actual classes can be visualised:

  
Illustration 6: Orange - Class distribution with NN predicted class on TRAINING data

Unlike the k-Means cluster there is excellent recall across all the classes indicating the possibility of building a strong model.

The final step in Orange was then to apply this model to the test data to see how well it worked:

  
Illustration 7: Orange - Class distribution with NN prediction on TEST data

The conclusions from the data review in order of significance are:

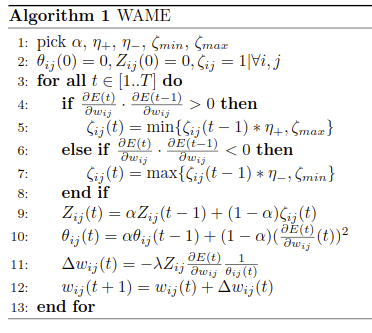
* that the individual features have different distributions and none cover the full range of 1-255 possible for each value. As a result rescaling or normalizing will be needed when building an NN
* The classes are not equally distributed, but with an imbalance of less than 3:1 between the highest and lowest frequently occuring classes this may not be a problem
* The ability of Orange to build a simple NN from the training data that has visually reasonable results on the test data shows the suitability of the NN approach for this dataset.

**b) WAME Optimizer**

It was decided to build the test framework in Python. PyTorch was selected as the learning library due to the ease with which customised models can be specified.

The code for the optimizer was developed based on the Rprop optimizer from the PyTotch source [5], using some guidelines from [6] on the building of a custom optimizer.

The mathematical description of the WAME optimizer in [1] is as follows

  
Illustration 8: WAME Optimizer mathematical specification

The full code code for this work is refenced below. The snippets implementing this algorithm are:

Prior to execution of this loop the values of zeta, Z and Theta are retrieved from the previous cycle.

Get product of current and previous gradients

**gradmult = grad.mul(state['prev'])**

For all instances of zeta where gradmult is greater than zero multiply by etaplus, where it is negative multiply by etaminus. If it is equal to zero this is undefined in the logic but code sets value to 1

**zeta[gradmult.gt(0.)]=zeta[gradmult.gt(0.)].mul(etaplus).clamp(zeta\_min,zeta\_max)**

**zeta[gradmult.lt(0.)] = zeta[gradmult.lt(0.)].mul(etaminus).clamp(zeta\_min, zeta\_max) zeta[gradmult.eq(0.)] = 1**

The gradient updates are calculated using Z and Theta as intermediate values. When dividing by teheta to calculate the change an epsilon value of 1e-10 is added to prevent zero division errors

**Z = Z.mul(alpha).add(zeta.mul(1 - alpha))**

**Theta = Theta.mul(alpha).add(grad.mul(grad).mul(1 - alpha))**

**step\_size = Z.mul(-lr).mul(grad).div(Theta.add(epsilon))**

**p.data = p.data.add(step\_size)**

The functioning of the optimizer was validated using a variation of a workbook taken from Week 9 of the Applied Machine Course [7] with the code adapted to use PyTorch. The PyTorch code was adapted from [8]. The code is in the CustomOptimisers notebook and shows rapid convergence when the WAME optimizer was used. This indicates correct functioning of the code.

**c) Execution Framework**

The full code for the framwork is in the Landsat\_PyT\_kFold notebook. This allows a CSV file to be read with parameters that will control the build of the model. The file may contain multiple lines of parameters, each corresponding to one required model run. A sample one line file is shown:

  
Illustration 9: Sample parameter file entry

The parameters that can be used are:

* **num\_units** – the number of units to be used in the first hidden layer of the NN
* **hidden\_layers** – number of hidden layers in the NN
* **batch\_size** – batch size within each epoch of the model build
* **lr** – learning rate to use for the optimizer
* **epochs** – number of epochs to rum
* **fold\_splits** – number of splits to use in the kFold process
* **optName** – optimiser to be used
* **alpha** – the alpha proportion for Z and Theta updates in Wame optimizer

The routines used to execute the run:

i) Main Block

The main block iterates through the rows of the input file and uses the num\_units and hidden\_layers values to instantiate a model. An optimizer is set up using optName and lr, and a folds object generated using the fold\_splits value. All these items are passed to the RunFolds routine.

The folds are generated using StratifiedKFold to ensure that the dsitribution of classes within each fold is similar to that in the full training dataset.

ii) RunFolds (includes normalixation)

The RunFolds routine runs for each fold in the selection passed to it. For each fold the training and validation datasets are selected from the full dataset supplied.

After this selection the normalize data routine is called. This uses the data distribution within the selected training set to determine the mean and standard deviation to be used for rescaling the data and applies these to both the training and validation sets. Application of the normalization in this fashion prevents leakage of information from the validation dataset into the model build.

A further option is to oversample data from the minor classes within the distribution. This boosts the sample number of the minority classes but can lead to overfitting as the boosting merely replicates random instances of the minority to increase its occurrence. This is controlled manually by commenting / uncommenting code lines and has not been parameterised.

If debug output has been requested the distribution of classes and data value range within the dataset is displayed before and after the normalization and oversampling have been made.

The training data is then passed to the model builder and the results checked by running the generated model against the validation dataset.

Iii) RunModel

RunModel generates a PyTorch DataLoader from the X and Y training sets which supplies data in the requested batch sizes to the model run. For each epoch the DataLoader is iterated and the model forward and backward propagations are run.

At the end of each epoch the accuracy of the model is checked against the full training and validation datasets for reporting purposes only. The scores from the validation dataset are not used in the model build.

iv) Class TestModel

The TestModel class is used to instantiate a model within the PyTorch framework. The number of units in the first hidden layer is taken as a parameter and the number for each subsequent layer is half of that in the prior layer. The total number of hidden layers is a further parameter. This approach allows models of varying complexity to be specified with just 2 parameters. An alternative approach of passing a list with the required number of units for each layer was considered, but this would have required more values to be specified in the controlling input table and made the whole experimental environment significantly more complex.

Once the list of hidden layers has been populated the fully connected layers and following activation layers are specified from the list. The activation function has been selected as ReLu as this is now widely accepted as being computationally efficient and robust.

v) Store

Results are written to a CSV file. This file can be opened and the data copied into the Results workshhet of the ResCalc excel workbook. This allows the whisker plot of the F1 results by fold by run to be seen, as well as detailed breakdown of each run with the data per fold showing the true and false positives for each class. This workbook is available in the “DataForRuns” directory of the project.

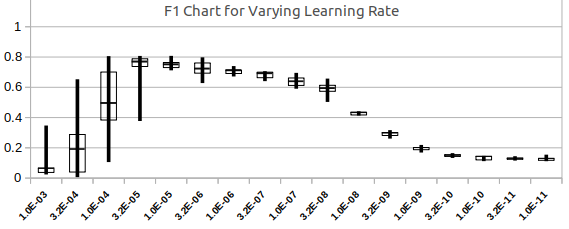
# **Experiments, findings and discussion**

Using the framework described above a number of experiments were undertaken to find the most suitable parameters to use for the model. These were done in sequence to optimize each parameter. The input and output files for each experiment are in the “DataForRuns” folder within the project.

The charts for all results show a solid vertical line with the range of F1 values through the folds of the run with a central line showing the median value and a box outlining the range between the first and third quartile. The F1 score has been used as the metric for selecting runs as it provides a balance between precision and recall. Precision is the ratio of true positives to true and false positives and showing the proportion of cases classified to a given class that actually belong to the class. Recall is the ration of true positives to the class occurrence and identifies the ability to recognise a class. The F1 score is the harmonic average of precision and recall and gives a reasonable measure of the overall model effectiveness.

a) Experiment 1 Learning Rate – TestLR.csv

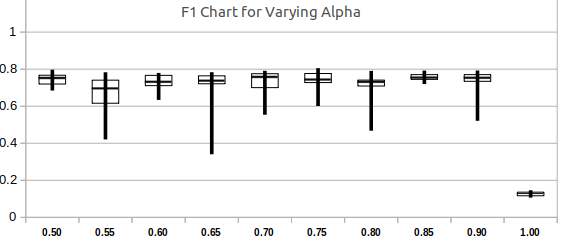
Simple paramers, Model 50 units in first layer, 2 layers, 20 epochs with batch size 1024, wame optimiser vary lr from 10-3 to 10-11 in steps of 100.5.



The result from this experiment shows the F1 being low and volatile with larger learning rates, but stabilising when the learning rate is lower. The optimal level at 10-5 gives a higher F1 with low volatility. For all further experiments the lr will be fixed at this value. The runtime was 8 to 9 seconds per run.

b) Experiment 2 Alpha – TestAlpha.csv

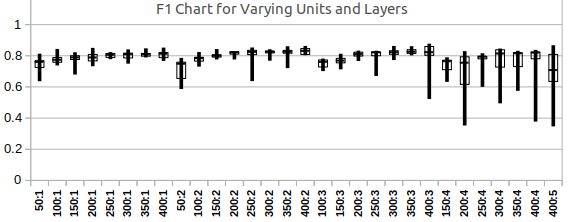
Keeping the other parameters from the previous experiment but now with the learning rate fixed at 10-5. Alpha which controls the update rate for the Z and Theta components of the gradient update within the WAME optimiser will be varied. The value range tested is from 0.5 to 1.0 in steps of 0.05. Runtime still just over 8 seconds per run.



The median F1 value does not appear to change significantly as Alpha is varied, but falls drastically when alphas is 1. The most stable results appear to be at an Alpha of 0.85 and this will be carried forward to the next experiments.

c) Experiment 3 Hidden Layers and Number of Units – TestUnitsLayers.csv

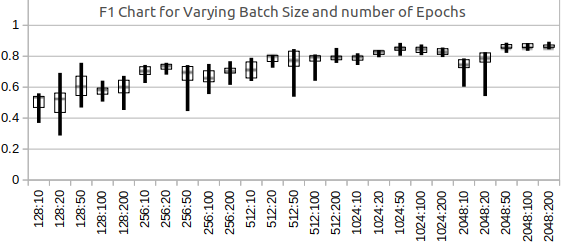
Varying the number of units in the first layer from 50 to 400 in steps of 50 and the number of hidden layers from 1 to 5. Learning rate and Alpha fixed from previous experimental results.



In this run there is a steady increase in F1 as the number of units in the first layer is increased. This applies in each group with a constant number of hidden layers. More complex with 4 or more hidden layers, have high volatility of F1 as indicated by the larger inter-quartile ranges. This will be due to overfitting on the test dataset and consequent poor performance on the validation set. For low volatility and good result 300 initial units with 2 hidden layers is used as the most appropriate model. Execution time was 7 seconds per run on the single hidden layer rising by 1 second for each added hidden layer.

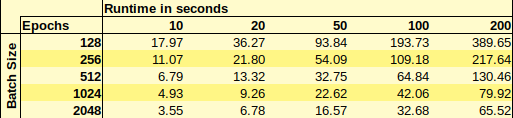
d) Experiment 4 Batch Size and Epochs – TestBatchEpoch.csv

From the previous experiments the lr is fixed at 10-5 and alpha at 0.85. The model has 300 units in the first hidden layer and 2 hidden layers. The batch size and number of epochs are now varied to find the best solving strategy. Batch size is run from 128 to 2048 doubling on each test and epochs of 10 to 200 are tested.



As the batch size is increased the F1 score increases. Larger number of epochs also increase the score. The highest F1 with the least variance is with a batch size of 2048 running for 200 epochs. The improvement over the same batch size and 100 epochs is small, but the total runtime of 65 seconds is acceptable.

In this experiment the runtimes varied significantly with rruntime icreasing linearly with the number of epochs and decreasing with higher batch sizes:

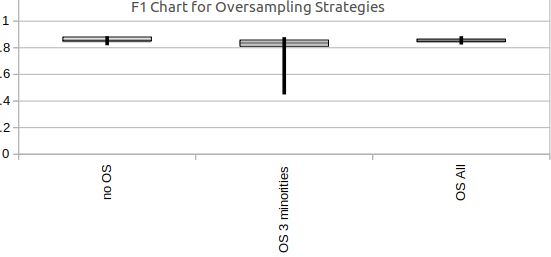


After these experiments the full model parameter set selected is:



e) Experiment 5 – Oversampling, results in ZselectedModelResultsOS.csv

All experiments so far have been conducted with the test data selected with class proportions kept the same as in the full test dataset. This experiment uses random oversampling to boos the number of observations in the test sample for the under-represented classes. The first run oversamples the 3 minority classes (classes 2,4,5 in original data, recoded as 1,3,4 for the runs) and makes their frequency equal to that of the majority class. The second test boosts all the classes to have the same frequency as the majority class.



The results do not show any significant advantage to the oversampling and so it has not been added to the selected model.

f) Experiment 6 – Check Model against Test Data – SelectedModel.csv

The selected model was run by itself without any folds to get the model parameters for the full dataset. This model was then tested against the test dataset.

The results for the model trained on the full training dataset then tested on test dataset show an accuracy of 84% with an F1 score of 0.801

Class Cases TP FP Precision Recall F1

0 461 455 54 0.894 0.987 0.938

1 224 212 6 0.972 0.946 0.959

2 397 387 124 0.757 0.975 0.852

3 211 71 29 0.71 0.336 0.457

4 237 163 26 0.862 0.688 0.765

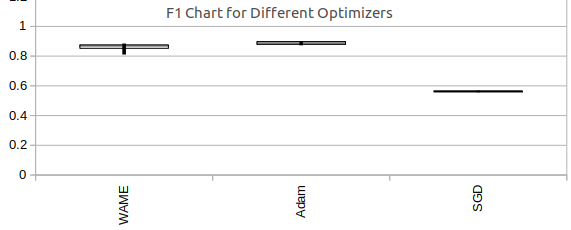
5 470 393 80 0.831 0.836 0.834

Total 2000 1681 319 0.838 0.795 0.801

Test accuracy: 0.868546 validation accuracy: 0.840500

f) Experiment 7 – Different Optimisers – TestOpts.csv

The final experiment was to use the selected model from the previous experiments and try different optimisation modules. The selected ones were Adam and SGD.



The results show a marginal improvement in results for the Adam optimizer over the WAME implementation. It is also worth noting that the execution time for the Adam run was 58 seconds compared to 69 seconds for the WAME optimizer. The conclusion is that on this particular dataset WAME does not improve on exoisting optimization modules.

Running the Adam based model on the Test dataset shows a significant improvement over the WAME results:

Class Cases TP FP Precision Recall F1

0 461 459 7 0.985 0.996 0.99

1 224 218 7 0.969 0.973 0.971

2 397 378 56 0.871 0.952 0.91

3 211 123 32 0.794 0.583 0.672

4 237 216 19 0.919 0.911 0.915

5 470 416 69 0.858 0.885 0.871

Total 2000 1810 190 0.899 0.883 0.888

Test accuracy: 0.950169 validation accuracy: 0.905000

# **Conclusions**

This work has shown that it is possible to implement the WAME optimizer in PyTorch within a framework that allows experimentation on the model details.

Fine tuning the model setup by testing ranges on Learning Rate, Alpha, Model Complexity, Batch Size and Epochs and oversampling techniques led to a significant improvement in the F1 score. The final model achieved 84% accuracy with an F1 score of 0.80 on the test dataset.

Using similar model parameters for an Adam optimiser gave better accuracy of 95% opn the test dataset and an improved F1 score of 0.89

On this particular dataset the WAME optimiser does not offer any significant advantage over Adam and would not be recommended as the optimiser of choice for this dataset.

Follow ups to this work could look in a number of areas:

1) Parameter Selection

Better selection of parameters for the inputs to the runs could be made. In this work each parameter was tested in isolation. Effectively a multi-dimensional problem has been followed one dimension at atime. Using a Monte Carlo based approach the required ranges for each parameter could be given and then multiple runs specified in which all of the parameters are varied. In the framework used the Monte Carlo output could generate a csv file for runs which could be used as the input file in this code.

With 7 parameters to vary and up to 6 values per parameter there would be a universe of over 100,000 parameter sets to choose from and a sampling of many thousand would be required. The significant runtime required prevented this being attempted in this work

2) Fold selection

Within this work all runs used a 10 fold StratifiedKFold model for testing. Other approcahes such as leave-one-out or resampling could be applied.

3) Ensemble Model on Test Data

Once the candidate model was selected a single model was built from the full training set and eveluated on the test data. It would be possible to build an ensemble of models from the folds used in the testing and run each of these on the Test dataset and have a voting approach to the outcime classification.

4) Code Optimization

The WAME optimizer was slower and did not improve on the Adam optimizer results. It is possible that optimization of the code could improve the execution speed.

5) WAME Optimization

This work only looked at the learning rate and alpha as adjustable hyperparameters in the WAME optimizer. Further work could look at the other hyperparameters (eta and zet min and max) to see if tuning them would improve the results.

# **References**

[1] A. Mosca and G. D. Magoulas, ‘Training convolutional networks with weight-wise adaptive learning rates’, in *ESANN*, 2017.

[2] ‘UCI Machine Learning Repository: Statlog (Landsat Satellite) Data Set’. <http://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite)>

[3] ‘A Tutorial on Different Classification Techniques for Remotely Sensed Imagery Datasets’, *ResearchGate*. <https://www.researchgate.net/publication/262917730_A_Tutorial_on_Different_Classification_Techniques_for_Remotely_Sensed_Imagery_Datasets>

[4] B. L. Ljubljana University of, ‘Data Mining’. <https://orange.biolab.si/>

[5] ‘torch.optim.rprop — PyTorch master documentation’. <https://pytorch.org/docs/stable/_modules/torch/optim/rprop.html>

[6] ‘Writing Your Own Optimizers in PyTorch’. <http://mcneela.github.io/machine_learning/2019/09/03/Writing-Your-Own-Optimizers-In-Pytorch.html>

[7] ‘BBK\_BUCI077H7\_1920: Week 9 - Deep Learning 1 (CNN)’. <https://moodle.bbk.ac.uk/mod/folder/view.php?id=644578>

[8] K. Patel, ‘MNIST Handwritten Digits Classification using a Convolutional Neural Network (CNN)’, *Medium*, Dec. 01, 2019. <https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9>

# **Appendix 1- Results (raw data)**

Fold by fold results for selected model run:

# 

The full data from all the runs described in this work can be found in the “DataForRuns” directory within the project folder. Each run has a csv file of results with the name Z[runname]results.csv. This file contains raw output data which can be formatted using the ResCalc.xlsx spreadsheet found in the same directory.

The ResCalc spreadsheet should be opened and the “Results” worksheet selected. All values in this sheet should be selected and deleted. The values from the Zxxxresults.csv file should then be selected and copied and pasted into the Results worksheet of ResCalc.xlsx.

The “SingleRunFullDetails” tab of the spreadsheet can then be selected and in cell Row1Col2 a run number entered. A full table showing each fold of the run and the number of cases in the test set used for the run and the true and false positives for that case will be shown. This is the table shown above for the single run for selected model.

The “F1BoxPlot” worksheet generates the box plots that have ben used in this paper. The title for the plot needs to be manually entered in the cell at Row17Col39. The plot itself has to be manually changed so that the data range used is correct for the number of runs in the particular file./

# **Appendix 2- Code**

The main notebook is “WAME\_framework.ipynb”. The datafile with all the landsat data “sat.all.csv” needs to be in the same directory as the notebook and a subdirectory “DataForRuns” needs to be present and have all the csv files for the various experiments.

In the first cell of the notebook the name of the csv file to be processed can be entered. The DataForRuns directory name is specified here as well.

The second cell contains all the imports for the whole workbook. All the modules refernced will need to be installed for the nortebook to work.

The individual blocks are then executed all the way down to the cell after the Store label which saves the results.

The final cells run the model on the Test data and should only be used when a model has been selected. Over frequent use of the Test dataset will lead to leakage of information from test into the model build and invalidate it.