CS5014 Machine Learning: Practical 2
Learning Visual Attributes

170004680

University of St. Andrews

April 2021

Contents

A How did you process the data, including any splitting, scaling and selection? Justify any such decisions.

A.1 Splitting And Data Leakage

The training data set (‘`data_train.csv`’) was split into a training and evaluation (i.e., testing) set for each of the outputs (‘*color*’ and ‘*texture*’), since separate models are created for predicting both outputs. The training sets use 80% of the original data, with the remaining 20% being allocated to the evaluation sets. This 4:1 split is common in practice and provides sufficient data for both training and evaluating the model. A seed was specified to randomly sample the original data set with reproducible results and to avoid oscillating blocks of samples somewhat sorted by output. Furthermore, the split into the training and evaluation sets is stratified by the outputs, which ensures these sets contain approximately the same percentage of samples for each output class as the original (complete) data set. This avoids bias on the assumption that the original data set is representative of the general distributions of the output classes.

Data leakage is prevented as no data dependant actions are completed before the data set is split into the training and testing sets; the split occurs before any pre-processing. Also, the existence of duplicate rows in the original training data set was checked to ensure that equivalent data could not accidentally end up in both the training and evaluation sets. Thus, none of the data in the training sets have ‘knowledge’ of the data in their corresponding evaluation sets.

A.2 Feature Selection

Basic feature selection has been implemented which chooses the top 80% of features according to an analysis of variance (i.e., ANOVA f-test) scoring; features that are statistically ‘less important’ to the output are removed from the data set. The proportion of features kept (80%) was empirically found to give the best cross validation performance when training the neural networks. Multivariate feature selection techniques should be investigated in future work to further improve general performance.

Further explicit feature selection was experimented with. Naively, the colour histogram features were removed for the texture predicting model, and the ‘histogram of gradients’ features were removed for the colour predicting model. However, this worsened performance, which is expected with the domain knowledge that colour and texture information in images are not so mutually exclusive. Thus, this form of feature selection was not used for the final models.

A.3 Data Imputation

The presence of missing feature values in the provided data sets means that data imputation is required, which replaces the missing data with substituted values.

The implementation replaces missing values with the feature mean, taking advantage of the fact that all input features are numerical. This is a simple approach that improves upon simply dropping samples with missing values, which can introduce bias in the training data, affect the representativeness of the results, and prevent predictions for samples when missing data is present. However, replacing missing values with the feature mean does not factor in correlations between features; it only works at the column level. Better performance may be attained via further investigation of multivariate imputation, but this was not investigated here as the performance benefit for the provided test file is low (there is a single row with missing data), even though the model would be improved in general.

A.4 Data Re-sampling

There is a severe class imbalance in both ‘*color*’ and ‘*texture*’ outputs (see table 1). For example, in the ‘*texture*’ output the ‘fluffy’ class is 38.2% of the distribution, whilst the ‘barbed’ and ‘coarse’ classes are just 0.5% and 0.4% of the distribution, respectively. For severely imbalanced training data, minority classes are harder to predict because there are few examples of these classes; the model becomes bias to the majority classes. When updating the model using gradient descent to minimise the loss function, most epochs change the parameter values in the direction which allows for correct classification of the majority classes. Thus, there is a frequency bias where more emphasis is placed on learning from data

observations that occur more commonly.

Table 1: Tables showing the class distributions for the classification outputs. In both outputs, there is a severe imbalance between the classes.

Texture Class	Frequency
fluffy	510
grassy	272
blurry	171
smooth	161
fuzzy	106
rippled	62
gravel	31
crusty	10
barbed	7
coarse	5

Color Class	Frequency
white	476
green	297
brown	218
gray	101
blue	87
black	73
yellow	20
orange	18
pink	17
red	16
beige	6
purple	6

There are many techniques for handling class imbalance; the implementation uses random over-sampling. Random over-sampling duplicates minority class samples at random until the distribution of the output classes are equal, alleviating the frequency bias of the majority classes. Samples are chosen at random because it cannot be discerned whether one sample is more representative than another and would therefore be better suited for duplication. Also, over-sampling is used because the training set is not sufficiently large enough for under-sampling; the training set resulting from under-sampling would have very few samples for each output class. Note that whilst frequency bias is removed by the over-sampling, no new data for the minority classes is introduced, so resulting models are not likely to become significantly better at predicting the minority classes correctly in general.

An investigation of more sophisticated techniques, such as the 'Synthetic Minority Oversampling Technique' (SMOTE), could further improve the approach. To this end, data augmentation of the image-based features could also provide better results in general; by extending the data set, there will be more samples to train on and variances between the added samples can assist with preventing over-fitting.

A.5 Data Scaling

The training/evaluation data sets contain feature values that are not well-spaced out and do not have well-defined limits, so scaling of the data is required. Scaling the data is also recommended by most of the relevant machine learning algorithms that use some form of gradient descent to improve model performance.

The existence of outliers was investigated to determine the type of scaling needed. To identify outliers, the Z-scores of the feature values were calculated. The Z-score is the signed number of standard deviations that values are from the mean. Values that were more than 3 standard deviations from the mean were considered as outliers. Most features were found to have significant amounts of outliers by the given definition. This meant that scaling via normalisation was unsuitable, so standardisation was used instead. `RobustScaler` was empirically found to provide the best performance and is recommended by `sklearn` for scaling when lots of outliers exist in the data.

B Which machine learning algorithm did you use and why is it suitable for the task at hand?

The machine learning algorithm/model used is artificial neural networks, specifically, feed-forward multi-class neural networks as implemented in `MLPClassifier` by `sklearn`. There are several reasons why the use of neural networks is suitable for the task at hand:

- Neural networks can learn non-linear and complex relationships in the data. For instance, by assigning more hidden layers to the network topology, a neural network can perform a series of feature mappings to discover any complex relationships. However, efforts must be made to prevent over-fitting of the data, a common crux for neural networks hindering general performance.
- By the universal approximation theorem, the neural network models can be made to approximate the functions that predict colour and texture of an object arbitrarily well. How well the neural network approximates the functions is a factor to configure as good general performance is desired.
- Neural networks use back-propagation for learning, which is uniform and highly efficient. The efficiency and uniformity affords for many learning/optimisation strategies to be trialled (back-propagation works for them all so this is made easier) to determine which gives optimal performance for the task at hand.
- Adapting the machine learning model for multi-class classification is trivial for neural networks; a neuron per output class, using soft-max activation, is simply added to the output layer.

C What are the hyper-parameters of your model, what do they represent, and how did you set them?

Hyper-parameters are parameters used for controlling learning which are not derived by the learning algorithm. Unless specified, hyper-parameters have been set automatically via hyper-parameter optimisation (figure 1). Grid search has been used to find the neural network configuration which maximises balanced accuracy (the metric used by the practical leader-board). Whilst grid search is much slower than random search, it ensures the chosen parameter combination is best amongst the searched parameter space. Furthermore, the grid search uses cross-validation so the maximised balanced accuracy reported is based on multiple unseen validation sets, which boosts general model performance. Cross validation with 5 folds has been used, which compromises between keeping the size of the training sets sufficiently large, whilst averaging over enough balanced accuracy scores to ensure better general performance.

- Hidden Layer Depth: The number of hidden layers in the neural network topology. By including hidden layers, non-linearly separable functions and complex relationships as the neural network gains flexibility.
- Hidden Layer Width: The number of neurons present in each hidden layer, which represents a feature mapping (an expansion or reduction of the feature space from the previous layer).
- Neuron Activation Function: The function used to convert neuron inputs to its outputs. For example, rectified linear units ('*relu*') is used by the neural networks (determined by grid search) and helps rectify the vanishing gradient problem for faster learning. For the output layer, the soft-max activation function is used to enable multi-class classification.
- Alpha: The co-efficient multiplied by the L2 penalty term. Thus, ridge regression is being used, which assists with preventing over-fitting.
- Learning Rate Initial: The initial learning rate used. It controls the step-size when updating the network weights.
- Batch Size: The size of mini-batches for stochastic optimisation algorithms. Smaller batch-sizes can converge on 'good' models faster at the expense of not guaranteeing the model is at the global minimum, and vice versa.
- Maximum Iterations: The maximum number of epochs the solver can perform to *converge* before training is aborted. This was increased to 1000 to help ensure convergence for all network configurations considered by the grid search.
- Number Of Iterations Without Change: The maximum number of epochs (number of times each sample is used) without sufficient improvement (given by the tolerance) before training is said to have *converged*. This was set at 100 epochs, which, retrospectively, is high considering that this parameter is also used by early stopping for *patience*.

Figure 1: The screenshot shows example program output when using grid search for neural network hyper-parameter optimisation. The grid search finds the network configurations that maximise the balanced accuracy score when predicting '*color*' and '*texture*'. However, it is seen that the maximised balanced accuracy scores are very high, so the models are likely to have been over-fit to the cross validation sets. A higher L2 penalty co-efficient than reported is recommended to reduce over-fitting.

```
Texture - Training Neural Network Model (Using Grid Search And Cross Validation).
Texture - Best Parameters Found:   {'activation': 'relu', 'alpha': 0.01, 'batch_size': 50, 'hidden_layer_sizes': (100,), 'learning_rate_init': 0.01}
Texture - Best Cross Validation Balanced Accuracy Score:  0.9384703402589581
Color - Training Neural Network Model (Using Grid Search And Cross Validation).
Color - Best Parameters Found:   {'activation': 'relu', 'alpha': 0.001, 'batch_size': 500, 'hidden_layer_sizes': (100, 100), 'learning_rate_init': 0.01}
Color - Best Cross Validation Balanced Accuracy Score:  0.9446627933470039
```

D What type of regularisation is employed by your model?

The neural networks for predicting '*color*' and '*texture*' use two forms of regularisation to prevent over-fitting: weight decay and early stopping.

Weight decay penalises the model during training; a penalty is added to the loss function in proportion to the size of the weights in the model. This encourages the model to map the inputs of the training data set to the outputs in such a way that the weights of the model are kept small. Small weights suggest a less complex and more stable model that is less sensitive to statistical fluctuations in the input data. The `MLPClassifier` uses L2 regularisation, with the degree of the penalty being controlled by the 'alpha' hyper-parameter (set with grid search).

Early stopping monitors model performance on a validation set and stops training when performance degrades. In the implementation, the neural networks use early stopping with 10% of the available training data being reserved as a validation set for early stopping. Whilst early stopping is beneficial for preventing over-fit, a concern is raised by the fact that the training data is limited: there are only 1335 training samples. By reserving an evaluation set, validation sets for the cross validation folds, and validation sets for early stopping, there may not be a desirable number of samples remaining to allow the networks to capture trends in the data and perform well in general (the very aspect regularisation aims to improve). This is an extra concern for the minority classes with fewer training samples to begin with.

Another regularisation option is available for neural network models but was not investigated in the implementation: dropout. A single neural network can simulate having several different network topologies by randomly dropping out nodes during training. This is computationally cheap and an effective regularisation method to improve generalisation error. Future work with neural networks should investigate the use of drop-out to prevent over-fitting, which is a prevalent issue with neural networks due to their high parameterisation.

E Which optimisation strategy did you use and how does it work? What are its strength and weaknesses and why did you choose it over some other approach?

The optimisation strategy used is back-propagation with the adaptive moment estimation ('Adam') solver. All of the optimisation strategies available to `MLPClassifier` in `sklearn` were trialled using grid search, with 'Adam' providing the best performance for the data set over 'lbfgs' and stochastic gradient descent (SGD).

Back-propagation is an algorithm for efficiently calculating gradients and adjusting weights/biases throughout the network topology (in reverse topological order) using chain rule. 'Adam' is an adaptive learning rate optimisation algorithm extending SGD. Specifically, 'Adam' combines two SGD improvements: adaptive gradient (*AdaGrad*) with momentum, and root mean square propagation (*RMSProp*). A learning rate is maintained for each network weight (i.e., parameter). The learning rates of each weight are separately adapted as learning unfolds using estimations of first and second moments of gradient. This differs from classical SGD where a single learning rate (α) is maintained throughout training.

The learning rate for SGD affects the step size but not the ratio between different gradient sizes across different steps (i.e., SGD considers only the direction of gradients and not magnitude). Thus, skewed gradient sizes across multiple steps can cause the network to end up in local minima and saddle points. 'Adam' is, therefore, advantaged by normalising gradients to maintain a relatively constant order of magnitude of gradient sizes. Furthermore, 'Adam' considers not just the direction of steepest decent, but also directions of descent for previous steps. If the gradient suddenly takes a sharp turn in the 'loss landscape', then 'Adam' recognises it should not forget the descent path taken thus far and head in some erratic direction because of a single iteration. Similarly, if the new direction is in line with previous steps, then 'Adam' pushes the gradient even further along said direction to converge to the minima faster. These considerations by 'Adam' allow for good performance with relatively fast convergence, the traversal of more complex 'loss landscapes' (often encountered with neural networks), and with little tuning.

F Which evaluation metrics did you use and why are they suitable for the task at hand? How well did your algorithm perform? How can you be sure that the performance you report is a repeatable result and not a statistical fluke?

The metrics used for evaluating the multi-class neural networks are macro precision, macro recall (i.e., balanced accuracy score in `sklearn`), and macro F1 score.

Macro metrics average performance over the individual output classes, as opposed to averaging over the individual observations (i.e., micro metrics). If a classifier has a high micro-averaged metric, then it performs well overall for that metric; however, the micro-average is not sensitive to the predictive performance for individual classes. Consequently, the micro-average can be misleading when the distribution of the output classes is imbalanced. If macro-averaging has a large value, this indicates that a classifier performs well for each individual class. Macro-average is therefore more suitable as the provided training data has a highly imbalanced class distribution.

There is a need for evaluation metrics beyond accuracy as accuracy is an inadequate metric for the severely imbalanced data; a model can attain a very high accuracy by simply predicting the majority classes most of the time. However, this does not inform on the poor performance for the minority output classes. Precision and recall are more suitable for the severely imbalanced data sets.

Precision concerns what proportion of samples predicted as being of a given output class are actually of that output class. Recall concerns what proportion of actual output samples are classified as such. Without domain knowledge, it is unclear whether minimising false positives (i.e., maximising precision) or minimising false negatives (i.e., maximising recall) should be prioritised. The F1 score, defined as the harmonic mean of precision and recall, provides a way to combine both precision and recall into a single measure, expressing both concerns with a single score. A disadvantage of using precision, recall, and F1 score is that none of these metrics take into account the true negatives. This is not a concern; in this multi-class output scenario, a true negative in one class necessarily affects the precision and recall of its actual output class, which is reflected in the macro metrics.

The performance of the model has been summarised in figure 2. For '*texture*', $\sim 26\%$ of the predictions are correct on average across the texture classes and $\sim 29\%$ of the classes are recalled. For '*color*', $\sim 25\%$ of the predictions are correct on average across the colour classes and $\sim 22\%$ of the classes are recalled. For both the '*texture*' and '*color*' neural networks, a factor affecting performance is that roughly a third of the classes for both outputs are not being predicted, which are the minority classes with a lack of data examples. Implementing more advanced re-sampling techniques, such as SMOTE, and implementing data augmentation of the image-based features can help improve performance by enriching the input training data.

The reported results are repeatable as the same randomly chosen integer has been used to seed all pseudo-random aspects of the machine learning process. The default seed used is '1010'. To test whether

the produced results were a statistical fluke, several seeds were used to generate different instances of the chosen machine learning algorithms. The different model instances were evaluated on their respective evaluation sets, the results of which are in figure 3. The figure shows that none of the results for any model instance was an outlier (outlier is defined as a Z-score greater than 3) so the reported results are not a statistical fluke.

Figure 2: The top screenshot shows the macro metrics for both the '*texture*' and '*colour*' neural network models; the '*texture*' model is better at predicting its output classes. The bottom-left and bottom-right screenshots show the non-macro metrics for the '*texture*' and '*colour*' models respectively, providing greater insight into the performance.

	Neural Network Model	
	Texture	Color
Macro Precision Score	0.261	0.253
Macro Recall Score	0.286	0.224
Macro F1 Score	0.269	0.234

	precision	recall	f1-score	support
barbed	0.0000	0.0000	0.0000	2
blurry	0.2647	0.2647	0.2647	34
coarse	0.0000	0.0000	0.0000	1
crusty	0.0000	0.0000	0.0000	2
fluffy	0.6129	0.5588	0.5846	102
fuzzy	0.2778	0.2381	0.2564	21
grassy	0.5588	0.6909	0.6179	55
gravel	0.3333	0.3333	0.3333	6
rippled	0.3500	0.5833	0.4375	12
smooth	0.2143	0.1875	0.2000	32

	precision	recall	f1-score	support
beige	0.0000	0.0000	0.0000	1
black	0.5556	0.3333	0.4167	15
blue	0.3846	0.2778	0.3226	18
brown	0.2750	0.2500	0.2619	44
gray	0.2000	0.2000	0.2000	20
green	0.4605	0.5932	0.5185	59
orange	0.0000	0.0000	0.0000	4
pink	0.0000	0.0000	0.0000	3
purple	0.0000	0.0000	0.0000	1
red	0.5000	0.3333	0.4000	3
white	0.6700	0.7053	0.6872	95
yellow	0.0000	0.0000	0.0000	4

Figure 3: Table showing the results of executing the machine learning process using different seeds. This creates different models that are evaluated on different evaluation sets but use the same machine learning algorithms. If any of the reported results are an outlier, then said results are a statistical fluke. None of the seeded models produce results that are outliers. Note that an outlier is defined here as a a value which is 3 standard deviations away from the mean.

	Texture			Color		
Seed	Macro Precision	Macro Recall	Macro F1 Score	Macro Precision	Macro Recall	Macro F1 Score
1010 (Default)	0.261	0.286	0.269	0.253	0.224	0.234
1011	0.240	0.244	0.241	0.170	0.167	0.167
1012	0.209	0.213	0.209	0.218	0.207	0.208
1013	0.343	0.299	0.311	0.187	0.187	0.182
1014	0.220	0.225	0.221	0.209	0.165	0.172
1015	0.207	0.214	0.208	0.145	0.146	0.144
1016	0.211	0.210	0.209	0.129	0.137	0.132
1017	0.309	0.264	0.277	0.158	0.147	0.149
1018	0.291	0.296	0.270	0.173	0.167	0.168
1019	0.200	0.218	0.206	0.143	0.151	0.146
Mean	0.249	0.247	0.242	0.179	0.170	0.170
St Dev	0.050	0.036	0.037	0.039	0.028	0.031

Z-Score > 3	Z-Score !> 3

G Are some classes easier than others [to predict] and why do you think that is?

Empirically, it is seen that the texture of an object appears easier to predict than its colour. However, this may be a consequence of the machine learning process used instead of some general assertion about the data. One explanation for this comes from the pre-processing steps splitting the input data in equal proportions for both neural network models. The '*color*' output has more classes and will therefore have

less samples per output class when training compared to the '*texture*' model, so the developed model for predicting '*color*' is disadvantaged.

Within the outputs, the minority classes are harder to predict than the majority classes. This is expected; even though random over-sampling has been used to balance the classes and prevent frequency bias, minority class samples are simply duplicated. Thus, no new data for the minority classes is introduced; the model is unable to capture the general trends of the minority classes using the limited examples. This issue is emphasised by the use of cross validation and early stopping, which creates stratified validation sets, further reducing the amount of available training samples for the minority output classes.

The deficit in ease of prediction associated with the minority classes can be reduced in several ways. More sophisticated re-sampling techniques can be implemented, such as SMOTE, which over-samples the minority classes whilst introducing variance in the added samples. Also, the training data set can be increased with data augmentation. The features are image-based, and objects within images have the same colour and texture when transformations, such as rotation, are applied to the images. Such transformations can be applied to the training features to enrich the data set. This would also help to avoid over-fit, a major issue seen with neural networks, by providing more representative samples of objects in images. For example, applying rotations to the feature samples to create new samples can help prevent the neural network from using factors like position to determine object colour (not significantly correlated in general).

H Compare your solution to a simple logistic regression classifier from sklearn and explain any differences.

A logistic regression model has been trained for each output as a base-line comparison for the neural network models. Both model types use the same pre-processing steps on the training data and both use L2 regularisation. As a result, comparisons between both models mainly illustrate core differences in the models and their optimisation strategies, though the neural network models are advantaged by the use of early stopping to prevent over-fitting (not available for `LogisticRegression` in `sklearn`).

Figure 4 shows the macro metrics evaluated for both model types on the same unseen evaluation data set, given by the default seed. Neural networks are significantly better than logistic regression at predicting the colour and texture of image objects in this context; the F1 score, which balances precision and recall, increased by $\sim 33\%$ when using neural networks for the predictions.

Explanations on why neural networks achieve better performance than logistic regression are aided by a simple fact: logistic regression can be represented as a neural network without hidden layers and which uses gradient descent as a solver. Thus, logistic regression is a subset of neural network classification.

The neural networks have hidden layers in their topology, whilst an equivalent logistic regression network does not. The use of hidden layers gives the neural networks additional flexibility by allowing dimension transformations on the features. The neural network models are therefore more capable at modelling complex trends in the input data to predict the colour and texture of an object. This added flexibility also allows for neural networks to fit data sets that are not linearly separable, unlike logistic regression, which is one reason why neural networks perform better in general.

Figure 4: Table comparing macro metric results between the neural network and base-line logistic regression models for both outputs. The neural network models perform better than the logistic regression models across all considered metrics. The logistic regression models used the same pre-processed training data and employed L2 regularisation. It is seen that the neural networks perform better on average for precision, recall, and therefore F1 score.

	Neural Network Model		Logistic Regression Model	
	Texture	Color	Texture	Color
Macro Precision Score	0.261 (+27.9%)	0.253 (+40.6%)	0.204	0.180
Macro Recall Score	0.286 (+36.8%)	0.224 (+28.7%)	0.209	0.174
Macro F1 Score	0.269 (+33.8%)	0.234 (+33.0%)	0.201	0.176

I Advanced Tasks

I.1 Comparing Neural Networks To Another Model - Random Forests

I.1.1 How Random Forests Work

In summary, a random forest classifier consists of a number of individual decision trees that act as an ensemble. Each individual decision tree in the random forest produces a class prediction given input features. The class predicted by a majority of the trees becomes the prediction of the overall random forest classifier.

Random forests work on the idea that a large number of relatively uncorrelated trees operating as an ensemble will outperform any of the individual constituent trees because the trees protect each other from their individual errors.

Therefore, the pre-requisites for a random forest to perform well are: there exists some actual signal in the input features so the trees are better than random predictors (i.e., garbage in, garbage out), and the decision tree predictions (and therefore their errors) have low correlations with each other. To ensure the second condition, random forest classifiers use two main techniques.

The first technique is bootstrap aggregation (i.e., bagging). Decision trees are sensitive to the data they are trained on, so small changes to their training set can result in significantly different tree structures. Random forests take advantage of this fact by allowing each individual tree to randomly sample from the whole training set with replacement, producing different trees (i.e., trees may get duplicate samples so that their training data is equal in size but different to the training set sampled from).

The second technique is feature randomness. In a normal decision tree, every possible feature is considered and the feature that produces the most separation in the observations (using 'gini impurity') is chosen when a node in the tree is to be split. Splitting occurs until either a maximum depth of the tree is reached or until each node contains only samples from a single class. However, each tree in a random forest may choose only from a random subset of its available features when splitting. Generally, the size of the subset of features is equivalent to the square root of the number of available features, though other configurations are possible. This forces greater variation amongst the trees of the model and results in a lower correlation across trees.

Thus, in a random forest, decision trees are trained not only on different sets of data via bagging, but also use different features to make predictions.

I.1.2 Comparisons And Differences Of Neural Networks And Random Forests

There are differences between the two algorithms in terms of the data pre-processing that is required. For neural networks, different feature ranges result in features with larger values being considered as more important when training, which is undesirable. Therefore, neural networks require the features to be scaled, unlike random forests.

For model training, random forests are significantly advantaged by the ease with which they can be trained. The main hyper-parameter for random forests is the number of decision trees used in the ensemble, which is easily configurable as the more trees the better and a training time trade-off need only be considered. Other hyper-parameters exist with random forests, though good performance is generally achieved with a default configuration. However, neural networks require expert knowledge and/or substantial hyper-parameter optimisation to form a good configuration of the network topology, learning algorithm, optimisation strategy, etc. These hyper-parameters are critical to the model and setting them naively can drastically affect performance.

An additional comparison with regards to the training of the algorithms is that the training process of neural networks is sequential, whilst training random forests is more parallel. In neural networks, training occurs by repeatedly forward propagating on samples and back propagating adjustments to the network. However, a random forest classifier trains by individually training many decision trees. The training of a single decision tree is independent of the other decision trees (i.e., can be trained in parallel).

In terms of structure, both algorithms are similar in the regard that each model is a collection of lesser units; neural networks are collections of artificial neurons, and random forests are collections of decision trees. However, both algorithms use their collections in opposing ways. Neurons within a neural network are aggregated into inter-connected layers, with neuron outputs being fed-forward into subsequent layers to form a classification. The neurons in a neural network act as one. However, each decision tree of a random forest is a classifier on its own and acts independently. It is only with majority voting that a single classification is produced by the random forest.

Decision trees, much like neural networks, are prone to over-fitting because they have a high flexibility. For example, a decision tree with unbounded depth or a neural network with many layers can fit the training data perfectly, but this also models noise within the training data, inhibiting general performance. However, many decision trees are used as an ensemble in random forests which greatly inhibits the over-fitting given by any of the trees individually. As a result, random forests do not suffer from over-fitting as inherently as neural networks.

I.1.3 Explanation Of Neural Network And Random Forest Performance Differences

Random forest classifiers have been trained to predict the colour and texture of objects using the same pre-processing steps as used for the neural network training. Grid search and cross validation was used to find the best hyper-parameter configurations for the random-forest classifiers (see figure 5), also as performed with the neural network models. Thus, different results achieved by random forests and neural networks mainly arise from the differences in the training algorithms used by these classifiers.

Figure 5: The screenshot shows example program output when using grid search for random forest hyper-parameter optimisation. The grid search finds the forest configurations that maximise the balanced accuracy score when predicting 'color' and 'texture', as was the case when optimising the neural networks. It is seen that the maximised balanced accuracy scores are very high, so the models are likely to have been over-fit to the cross validation sets.

```
Training Random Forest Model For Predicting Texture...
Texture - Training Model (Using Grid Search And Cross Validation).
Texture - Best Parameters Found:   {'max_depth': 50, 'max_features': 'log2', 'n_estimators': 5000}
Texture - Best Cross Validation Balanced Accuracy Score:  0.9715176151761516
Training Random Forest Model For Predicting Color...
Color - Training Model (Using Grid Search And Cross Validation).
Color - Best Parameters Found:   {'max_depth': 50, 'max_features': 'log2', 'n_estimators': 2500}
Color - Best Cross Validation Balanced Accuracy Score:  0.9667321713374346
```

It is shown in figure 6 that the results produced by the random forest models for the default seed ('1010') are not statistical flukes (see figure 7) and are therefore valid for comparisons to the neural network results. For all except one metric (macro precision when predicting object texture) the neural networks give better performance with the default seed. Specifically, the F1 score, which balanced precision and recall, is $\sim 9\%$ worse for the random forest predicting texture, and is $\sim 23\%$ worse for the random forest predicting colour. Note that these results seem to support earlier assertions about colour being the harder output to predict.

A more robust analysis of general performance differences between neural networks and random forests is achieved by considering the macro metrics of the models averaged across separate seeds (see figure 8). Random forests have a better macro precision score for both outputs on average. This result is expected. The ensemble usage of individual classifiers by a random forest dampens prediction errors given by any single tree classifier, which results in fewer false positives for the output classes and a greater macro precision. This is especially the case for the '*texture*' output, which seems easier of the two outputs to predict, explaining its significant increase in macro precision with random forests. The random forests have slightly worse macro recall when predicting texture and comparable macro recall when predicting colour compared to the corresponding neural networks.

Figure 6: Table showing the results of running the machine learning process using different seeds to train the random forest classifiers. This creates different models that are evaluated on different evaluation sets but use the same machine learning algorithms and pre-processing. The first result from this table is that the results reported in figure 7 are not a statistical fluke; thus, comparisons of performance between neural networks and random forests for the default seed used by the machine learning program are valid. The second result from this table is that on average, random forests have better macro precision for predicting both the texture and colour of an object and are roughly as good as neural networks in terms of recall.

	Random Forest Model					
	Texture			Color		
Seed	Macro Precision	Macro Recall	Macro F1 Score	Macro Precision	Macro Recall	Macro F1 Score
1010 (Default)	0.399	0.256	0.246	0.226	0.184	0.180
1011	0.258	0.221	0.206	0.225	0.180	0.172
1012	0.305	0.219	0.199	0.209	0.168	0.155
1013	0.279	0.218	0.200	0.241	0.170	0.157
1014	0.217	0.229	0.207	0.190	0.168	0.153
1015	0.299	0.209	0.190	0.183	0.179	0.168
1016	0.300	0.191	0.172	0.226	0.191	0.187
1017	0.361	0.211	0.196	0.219	0.174	0.171
1018	0.340	0.226	0.214	0.209	0.187	0.180
1019	0.306	0.248	0.239	0.173	0.174	0.164
Mean	0.306	0.223	0.207	0.210	0.178	0.169
St Dev	0.049	0.018	0.021	0.021	0.008	0.011
			Z-Score > 3	Z-Score !> 3		

Figure 7: Table comparing macro metric results between the random forest models and the main neural network models for both outputs. The neural network models perform better in general for all metrics, except for macro precision when predicting 'texture', which increased significantly when using random forests.

	Random Forest Model		Neural Network Model	
	Texture	Color	Texture	Color
Macro Precision Score	0.399 (+52.9%)	0.226 (-10.7%)	0.261	0.253
Macro Recall Score	0.256 (-10.5%)	0.184 (-17.9%)	0.286	0.224
Macro F1 Score	0.246 (-8.6%)	0.180 (-23.1%)	0.269	0.234

Figure 8: Chart showing a performance comparison between the neural network and random forest classifiers for predicting object colour and texture. The results are averages over 10 different seeds to understand the general performance of the classifiers better. The random forests have better precision for the output classes of both outputs, and perform comparably (or slightly worse) in terms of recall and therefore F1-score.