Machine Learning Engineer Nanodegree

Capstone Project

Andrew Smith
10th November 2017

Definition

Project Overview

Keeping an accurate inventory of natural resources is a key role of any land management agency. In the 1999 paper *Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables* (Blackard et al, Computers and Electronics in Agriculture 24, 1999) Blackard and Dean compared the multi-class classification accuracy of a feed forward artificial neural network with that of a statistical discriminant analysis model tasked with predicting the forest cover type at 581,012 sites using 12 variables such as soil type, elevation and horizontal distance to nearest water feature. Their stated goal was to provide a predictive model which could prove either more cost effective, or even have more utility than the manual recording or remote sensing methods more commonly used. The goal of this project will be to compare the accuracy of an ANN constructed to have a close architecture and accuracy to that used in the original paper with that of an ANN constructed using techniques unavailable or uncommonly used at the time and at a scale only practical on modern massively parallel GPU hardware, the assumption being that a more accurate predictive model would be of more use to the aforementioned agencies. The dataset used will be the same one as used in the 1999 paper, which was gathered from the US Forest Service and the US Geological Survey. The dataset was obtained from the UCI Machine Learning Repository.

Problem Statement

As stated in the original paper "Accurate natural resource inventory information is vital to any private, state, or federal land management agency. Forest cover type is one of the most basic characteristics recorded in such inventories. Generally, cover type data is either directly recorded by field personnel or estimated from remotely sensed data. Both of these techniques may be prohibitively time consuming and/or costly in some situations." (Blackard et al 1999)
The purpose of this project is to assess the improvement in accuracy in multi-class classification between a ANN based on the one in the original paper and a network

constructed using techniques impractical, unavailable or simply not in common practice at the time. It will do so by constructing a reference network based on the one in the 1999 paper and a series of more complex networks based on the techniques mentioned above, using the same set of numerical and categorical data used in the original study, with the goal of creating a more accurate predictive model.

Metrics

The models in the original paper were assessed using accuracy which is defined as

$$accuracy = \frac{1}{n}\sum_{i=0}^{n-1} 1\left(\hat{y}_i = y_i\right)$$

where n is the number samples and $(\widehat{y_i} = y_i)$ is 1 where the prediction is correct and 0 otherwise.

As the dataset used for this study is heavily skewed towards just two of the available seven classes it is possible for accuracy to give a misleading result since a good performance on the majority classes can hide a very poor performance on the minority classes.
To help mitigate this a weighted accuracy score will also be used. The weights are calculated using the scikit-learn compute_class_weight and compute_sample_weight functions. According to the documentation for the scikit-learn compute_class_weight and compute_sample_weight functions the class weights are calculated as

$$weight_c = \frac{n}{mn_c}$$

where $weight_c$ is the weight of class c, n is the number of samples, m is the number of classes and n_c is the number of samples of class c. The weight of each sample is the weight of its associated class. The weighted accuracy is calculated as

$$weightedAccuracy = \frac{1}{n}\sum_{i=0}^{n-1} w_{y_i}\left(\hat{y}_i = y_i\right)$$

where n is the number of samples, w_{y_i} is the sample weight of y_i and $(\widehat{y_i} = y_i)$ is 1 where the relation is true and 0 otherwise. It is possible for a model to have a good accuracy but a

poor weighted accuracy and for a model to have a good weighted accuracy but a poor accuracy. Ideally the model will achieve similar scores for each metric.

Analysis

Data Exploration

The dataset for this project was acquired from the UCI Machine Learning Repository at http://archive.ics.uci.edu/ml/datasets/Covertype. It was originally gathered for use in the paper *Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables* (Blackard et al, Computers and Electronics in Agriculture 24, 1999). It contains 581,012 instances of data. The forest cover type labels were found by sampling 30*30 meter cells from the US Forest Service Region 2 Resource Information System. The independent variables were collected from both the US Forest Service and the US Geological Survey. All of the data was gathered from the Roosevelt National Forest in northern Colorado.
Each of the 581,012 instances of data has 12 independent variables and one label. Of the 12 independent variables 10 are quantitative and are described in the information accompanying the dataset as:

Elevation	Elevation in meters
Aspect	Aspect in degrees azimuth
Slope	Slope in degrees
Horizontal_Distance_To_Hydrology	Horizontal distance to nearest water feature
Vertical_Distance_To_Hydrology	Vertical distance to nearest water feature
Horizontal_Distance_To_Roadway	Horizontal distance to nearest roadway
Hillshade_9am	0-255 index, Hillshade index at 9am. Summer Solstice
Hillshade_Noon	0-255 index, Hillshade index at noon. Summer Solstice
Hillshade_3pm	0-255 index, Hillshade index at 3pm. Summer Solstice
Horizontal_Distance_To_Firepoints	Horizontal distance to nearest wildfire ignition points.

Two of the variables are qualitative and are described in the information accompanying the dataset as follows:

Wilderness Area	Wilderness Area Designation.
Soil Type	Soil type designation.

The wilderness areas are: Rawah Wilderness Area, Neota Wilderness Area, Comanche Peak Wilderness Area, Cache La Poudre Wilderness Area.
The information regarding the soil type designation is rather extensive and is provided with the dataset accompanying this project.
The qualitative variables are all already one-hot encoded in the provided dataset, giving the dataset a total of 55 columns (10 quantitative values, 44 one-hot encoded classifications and one classification label).

Each instance of the data has one of 7 labels.

Spruce/Fir	211840 instances
Lodgepole Pine	283301 instances
Ponderosa Pine	35754 instances
Cottonwood/Willow	2747 instances
Aspen	9493 instances
Douglas-fir	17367 instances
Krummholz	20510 instances

The data in the dataset is heavily skewed towards Spruce/Fir and Lodgepole Pine, which together make up > 85% of the available data.

Exploratory Visualization

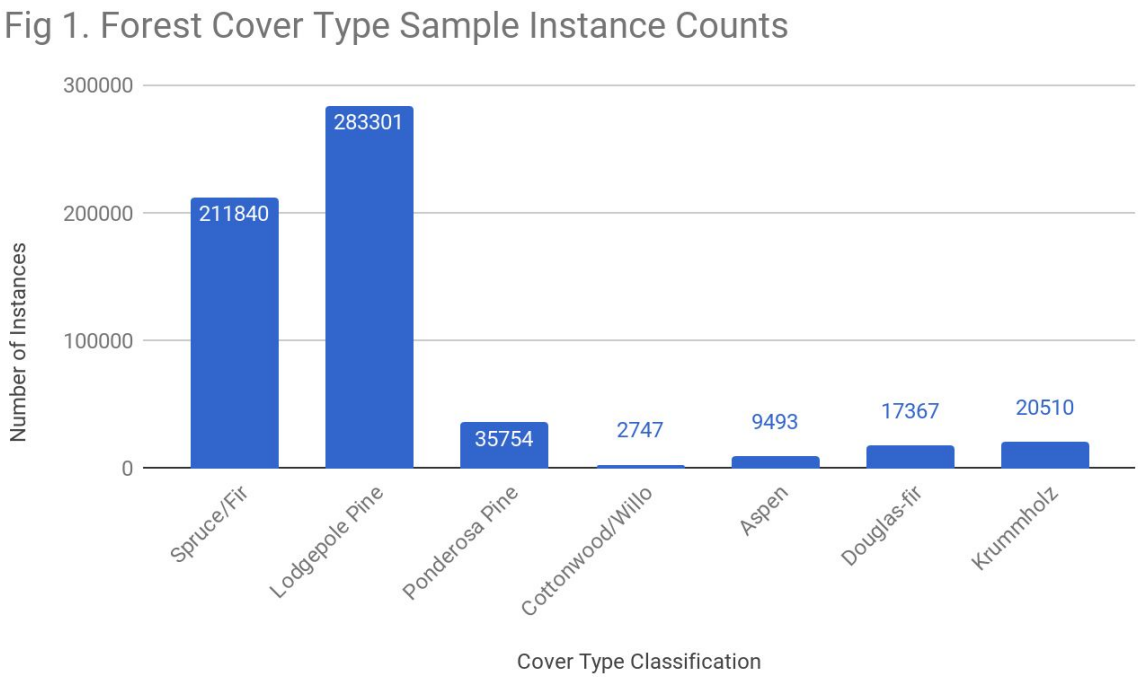


Fig 1 above shows the distribution of the data classifications. The skewed nature of this dataset will require special consideration in the selection of metrics and, as seen below, in the treatment of data during selection and training. Spruce/Fir and Lodgepole Pine together make up over 85% of the available data. A classifier which does well on just those two classes but fails to predict every other class could still show a reasonable accuracy. For example a classifier with an accuracy of 80% on Spruce/Fir and Lodgepole Pine but 0% on each other class would still show an accuracy of ≈ 68%.

Algorithms and Techniques

Artificial Neural Networks (ANN) loosely mimic animal nervous systems. Each network consists of one or more neurons (units) which have one or more inputs, one or more outputs and some – generally simple – function to create an output from the inputs. When connected together and carefully arranged they are capable of modelling very complex functions. The networks are often arranged into layers with different properties for example:

- Linear layers simply output the input values directly. In classification problems these are often seen as the input layer.
- Fully connected layers connect each unit to every unit in the previous layer, so if the previous layer has 54 units and the current layer has 120 then each of the 120 units will have 54 inputs.
- A softmax layer uses the softmax function to ensure that the sum of its outputs is 1. It is useful in networks where the output is a vector of probabilities that the input has a specific classification, such as this one.

Since the purpose of this project is a comparison of the capabilities of the original ANN and one constructed using more modern techniques, both the original paper and this project make use of multi layer or ‘deep’ neural networks. The networks are arranged in layers with an input layer, an output layer and one or more ‘hidden’ layers. These networks have a significant number of tuning parameters including:

- Activation Function such as sigmoid or Rectifier (ReLu)
- Loss function such as Mean Squared Error or Categorical Cross Entropy
- Learning Optimizer, such as Stochastic Gradient Descent.
- Learning Rate
- Momentum
- Number of units per layer
- Connectivity Between Layers
- Type of layer, such as fully connected or convolutional.
- Data handling, such as sample weights
- Weight and bias initialization

Benchmark Model Algorithms

The benchmark model will be constructed to be close to the one used in the original paper, using a 3 layer ANN. It will use a linear input layer with 54 units, a fully connected hidden sigmoid hidden layer with 120 units and a sigmoid output layer with 7 units. The loss function will be Mean Squared Error and the optimizer will be Stochastic Gradient Descent. The learning rate, momentum and weight and bias optimization will be as in the original paper. One of the designs for the baseline model will use sample weights (See Benchmark section below for exact details)

Subject Model Algorithms

The subject model will have between 3 and 6 layers consisting of a linear input layer with 54 units, from 1 to 4 fully connected hidden layers using a ReLu activation function, and 1 output layer of 7 units using a softmax activation function. The hidden layers will each contain between 60 and 2160 units. The loss function will be categorical cross entropy and the optimizer will be Stochastic Gradient Descent. Learning rate, momentum and weight and bias initialization will be left at the defaults for the keras library used to build the model. The initial models will not use sample weights. (See Implementation section below for exact details). The subject model algorithms are each selected to be representative of a modern approach to building an ANN.

Benchmark

The original intention was to construct a benchmark model as described above with an accuracy at least close to that of the network in the original paper of 70.52%, using weights initialized over a truncated normal distribution of 1.0, a learning rate of 0.05 and a momentum of 0.5 over just 100 Epochs. The model would attempt to maximise the available training data, despite the skewed nature of the data by using stratification (via the stratified option of sklearn's train_test_split function) in the selection of the data to ensure that each class is represented in the testing and training data in proportions equal to that in the overall data. This proved to be naive for two reasons. Firstly the original paper was training models over a *minimum* of 1000 epochs, with a cut off at a loss of 0.05 meaning that the model is likely to have had many thousands or even tens of thousands of epochs training. Secondly the skewed nature of the data lead to the model heavily underperforming at predicting the minority classes. A number of approaches were taken to address this including random under sampling, random over sampling, random under sampling with manually set quotas (via the unbalanced learn extension to sklearn), and class weights, calculated via the sklearn function and applied via the class_weight option in the keras model fit function.

Fig 2. Accuracy Per Class By Preprocessing Method at 1000 Epochs

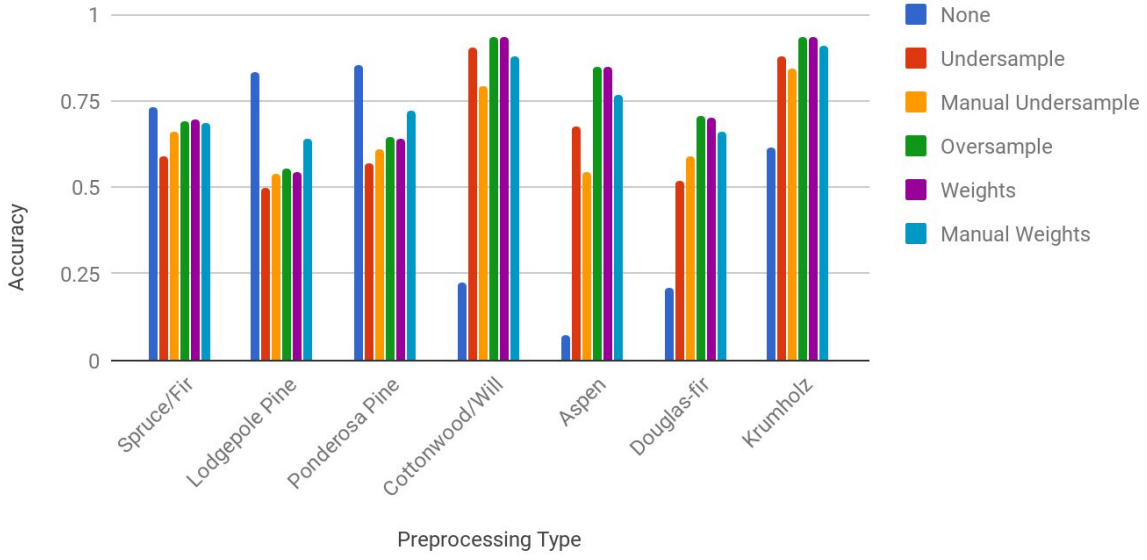
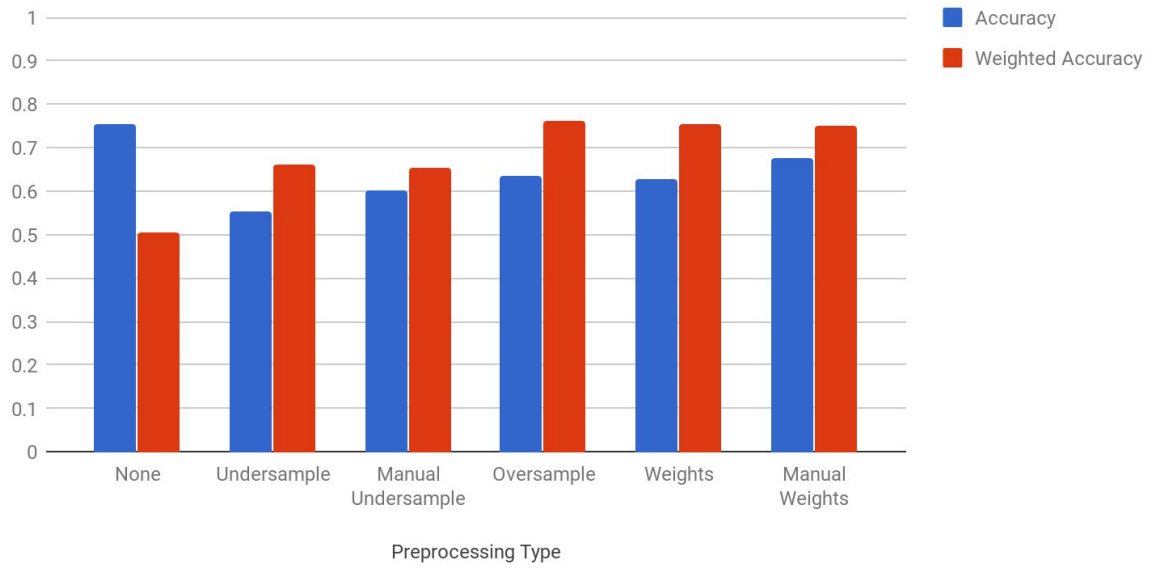


Fig 3. Accuracy and Weighted Accuracy By Preprocessing Type at 1000 Epochs



As can be seen in Fig 2 & 3 above, the model with no preprocessing of the data performed very poorly on the minority classes. It's overall accuracy was ≈ 75.45% but its weighted accuracy was only ≈ 50.53%.
The unmodified undersampling, oversampling and class weight calculations all had similar effects on the results by boosting the performance of the minority classes at the cost of a reduction in the accuracy of the majority classes leading to a reduction in overall accuracy.

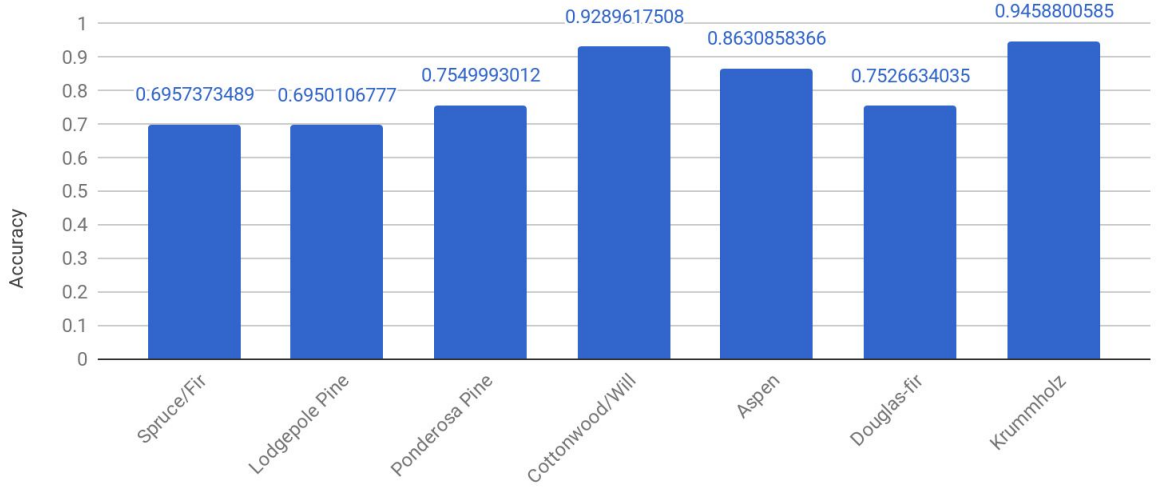
Preprocessing Type	Accuracy	Weighted Accuracy
Undersampling	55.50%	66.23%
Oversampling	63.53%	76%
Weights	62.86%	75.59%

To attempt to address the underperformance on the majority classes two approaches were tried. Firstly the undersampling was re run with manually set quotas for each class. Instead of using 2198 samples per class, the quotas for Spruce/Fir, Lodgepole Pine, Ponderosa Pine and Douglas-fir were set at 3297, 1.5 times the quotas for the other classes.
The second approach was to add code to multiply the class weight for each class by some constant set per class. The class weights for Spruce/Fir and Douglas-fir were multiplied by 1.25 and the class weights for Lodgepole Pine and Ponderosa Pine were multiplied by 1.5. It is worth noting that these modified weights were used in the calculation of the loss function but **not** in the calculation of the final weighted accuracy.

Preprocessing Type	Accuracy	Weighted Accuracy
Manual Undersampling	60.04%	65.38%
Manual Weights	67.56%	75.14%

Finally the training was run using the manual weighting method outlined above but over a total of 10,000 Epochs. This achieved an accuracy of ≈ 71.34% and a weighted accuracy of ≈ 80.52%. The wide discrepancy between the two values is a result of the model still performing relatively poorly on the majority classes, as shown in the following chart. It is chosen as the benchmark because it is the closest to the accuracy achieved in the original paper.

Fig 4. Accuracy Per Class With Manually Weighted Samples After 10,000 Epochs



Methodology

Data Preprocessing

1. The qualitative independent variables were already one hot encoded in the downloaded dataset so they were left as is
2. The quantitative independent variables were all scaled to values between 0 and 1 using the MinMax utility from sklearn.
3. The integers assigned to the class labels were each reduced by one to make them run from 0-6 rather than 1-7 to prevent an index out of range error.
4. The labels and the variables were split.
5. The labels and variables were randomly shuffled and split into test and training sets. The training set contained 80% of the original data. The test set contained the remaining 20%. The data was stratified using the stratify option offered by sklearn to ensure that each class appeared in proportions in the test and training sets equal to their proportions in the overall dataset.
6. The labels were passed through the keras to_categorical utility to transform them from a class label to a sparse matrix.

During the preparation of the data for some of the attempts at creating a benchmark model either random over sampling, random under sampling or class/sample weighting was utilised.

Random Over Sampling selects instances from the minority classes multiple times to balance the number of instances with the majority classes.

Random Under Sampling drops instances of the majority classes to equalise the numbers of each class, or set the numbers of each class to some manually set value.

Class weighting calculates a value based on the frequency each class appears in the dataset (see metrics, above). This value can be used when calculating the loss value during training and can also be used when calculating the weighted accuracy.

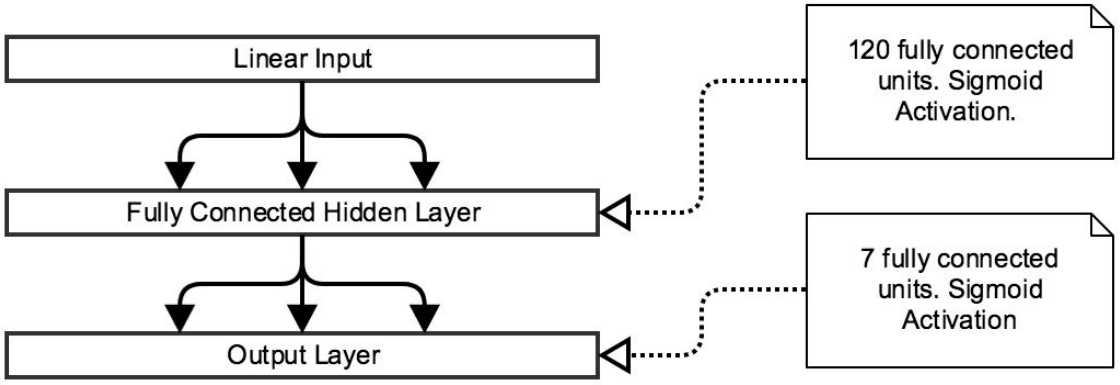
For the final two attempts at creating a benchmark model the class weights were modified by multiplying each calculated class weight by a constant set per class. This modified class weight was used in the calculation of the loss function but not in the calculation of the weighted accuracy.

Implementation

Benchmark Implementation

The final benchmark model was constructed in python using the Keras Framework with a Tensorflow back end. It was constructed using the Keras Sequential model class. It had a linear input layer, a single fully connected hidden layer consisting of 120 units and an output layer of 7 units. Both the hidden and output layer used a sigmoid activation function. The loss function was mean squared error and the optimizer was stochastic gradient descent. The weights were randomly initialized over a truncated normal with a mean of 0.0 and a standard deviation of 1.0. The learning rate was set to 0.05, the decay was set to 1×10^{-6} and the momentum was set to 0.5. A class weighting was used when calculating the loss as described in the Data Processing and Benchmark sections. The model was trained over 10,000 epochs using 80% of the available data as a training set and evaluated after training using the remaining 20%. Data stratification was utilised to ensure that all the classes were represented in both the training and testing data in proportions equal to their proportions in the full dataset. The batch size during training was 128. During evaluation sample weights were supplied calculated using the method described in the metrics section. Both an accuracy and a weighted accuracy were calculated for the model.

Fig. 5. Benchmark Model Architecture.

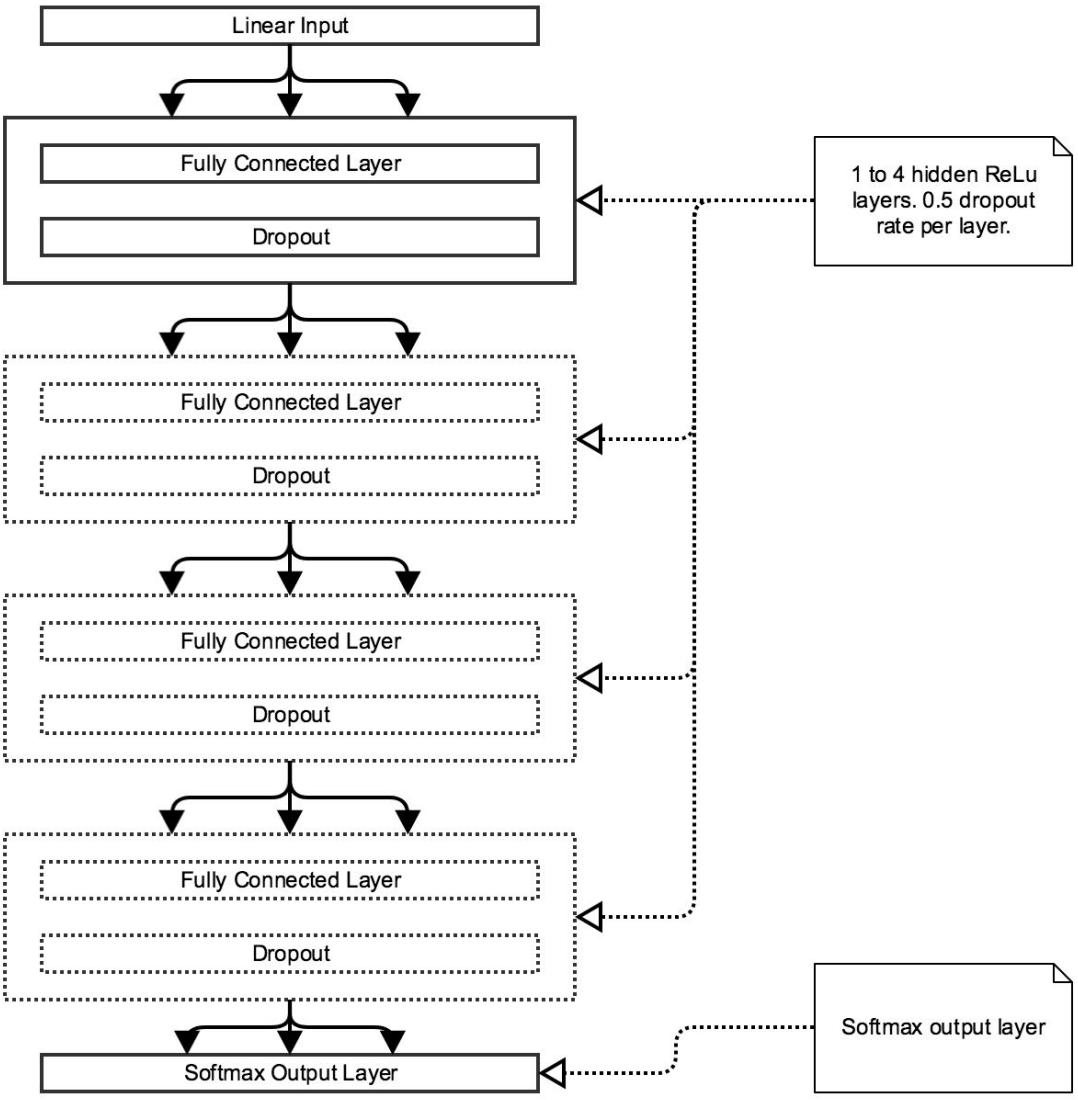


Subject Model Implementation

The subject models were also constructed in python using the Keras Framework with a Tensorflow back end. They were constructed using the Keras Sequential model class. They each had a linear input layer and between 1 and 4 fully connected hidden layers. Each hidden layer had between 60 and 2160 units with a rectifier activation function. Each hidden layer also had a dropout rate of 0.5 applied to it for regularization. Each model had an equal number of units in each hidden layer, so for example a model might have 3 hidden layers of 60 units or 2 hidden layers of 2160 units but no models had a hidden layer of 60 units and a

hidden layer of 2160 units. The output layer had 7 units and a softmax activation function. The loss function was categorical cross entropy and the optimizer was stochastic gradient descent. The weights were initialized over a truncated normal with a mean of 0.0 and a standard deviation of 0.05. The model had a learning rate of 0.01, a decay of 1×10^{-6}, a momentum of 0.9 and used Nesterov momentum. The models were each trained over 1000 epochs using 80% of the available data as a training set and evaluated after training using the remaining 20%. Data stratification was utilised to ensure that all the classes were represented in both the training and testing data in proportions equal to their proportions in the full dataset. The batch size during training was 128.

Fig. 6. Subject Model Architecture.



No serious issues were encountered with the development of the project although a significant refactor became necessary to handle the various options available for training the benchmark model

Refinement

The key refinement technique was to methodically increase the size and depth of the subject model. The smallest and simplest model had a single hidden layer of only 60 units. It had an overall accuracy of ≈ 78.49% but a weighted accuracy of only ≈ 56.89%. The discrepancy is the result of very poor predictive accuracy on the minority classes such as only ≈ 8% on Aspen.

The largest and deepest model had four layers of 2160 units each (40 units per layer per independent variable). It had an overall accuracy of ≈ 96.34% and a weighted accuracy of ≈ 94.84%. It was not the most accurate model. Due to the performance of the larger models even on the minority classes it was not felt necessary to attempt to improve the results by using class weights or random over or under sampling. The most accurate model, using three layers of 2160 units each had an accuracy of ≈ 96.46% and a weighted accuracy of ≈ 95.45%. Ideally it would be run for 10,000 epochs to make it a closer match to the benchmark model but the computing time required would be somewhere in the region of 130 hours which is not practical for this project. See fig 7 below for a comparison of the results of the various architectures explored.

Perhaps the most striking result of the refinement process was the ability of the larger networks to accurately predict all classes with relatively little loss of accuracy on the minority classes.

Fig 8 below compares the accuracy per class of a network with a single hidden layer of 60 units and a network with 3 hidden layers of 2160 units each. The smaller network has a good accuracy on the majority classes but performance drops on the minority classes, and is only ≈ 8% on Aspen. In contrast the larger network achieves an accuracy of > 90% on each class despite the smallest minority class (cottonwood/willow) having fewer than 1% of the number of samples of the largest majority class (lodgepole pine)

Fig 7. Accuracy and Weighted Accuracy By Architecture

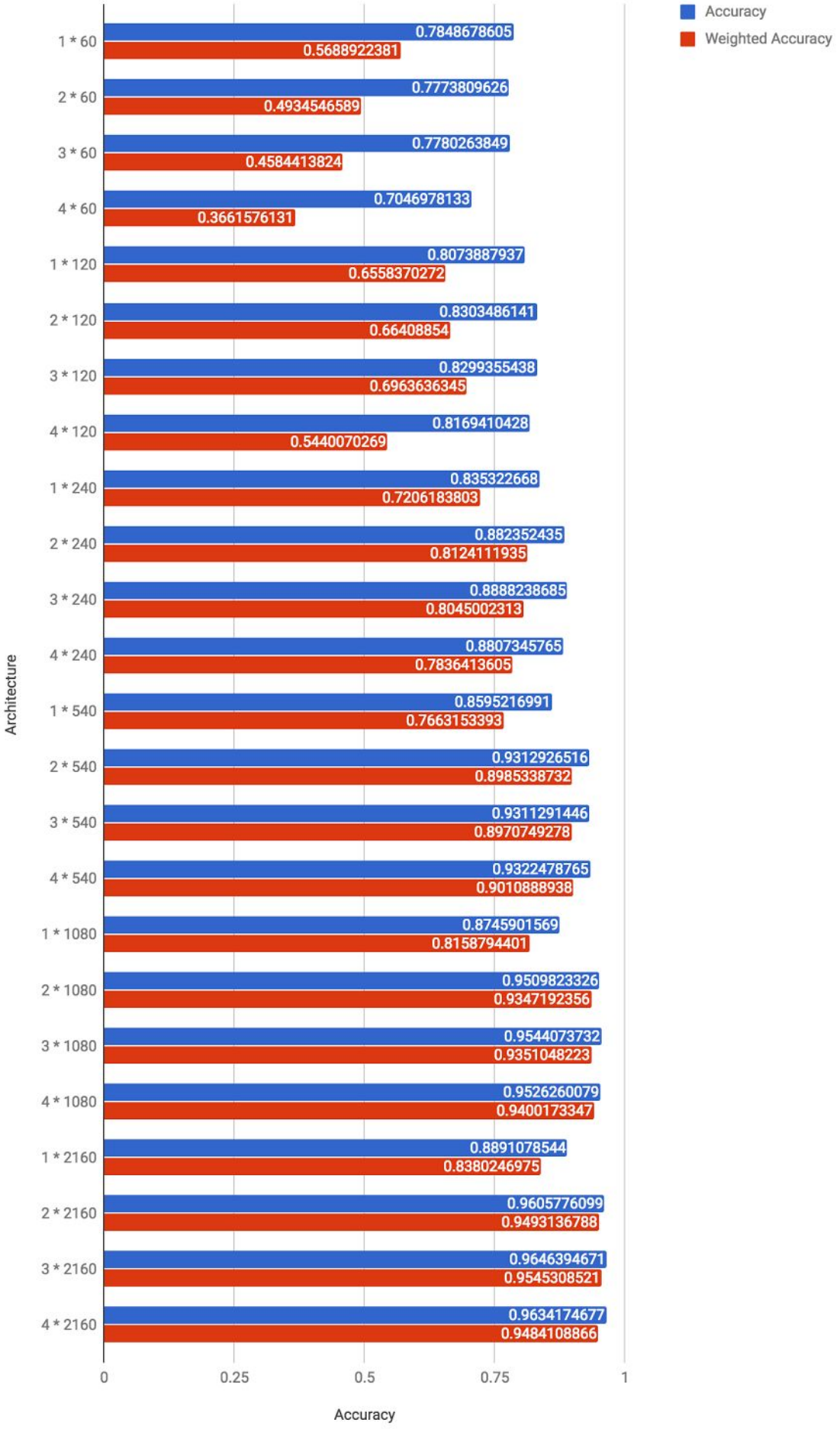
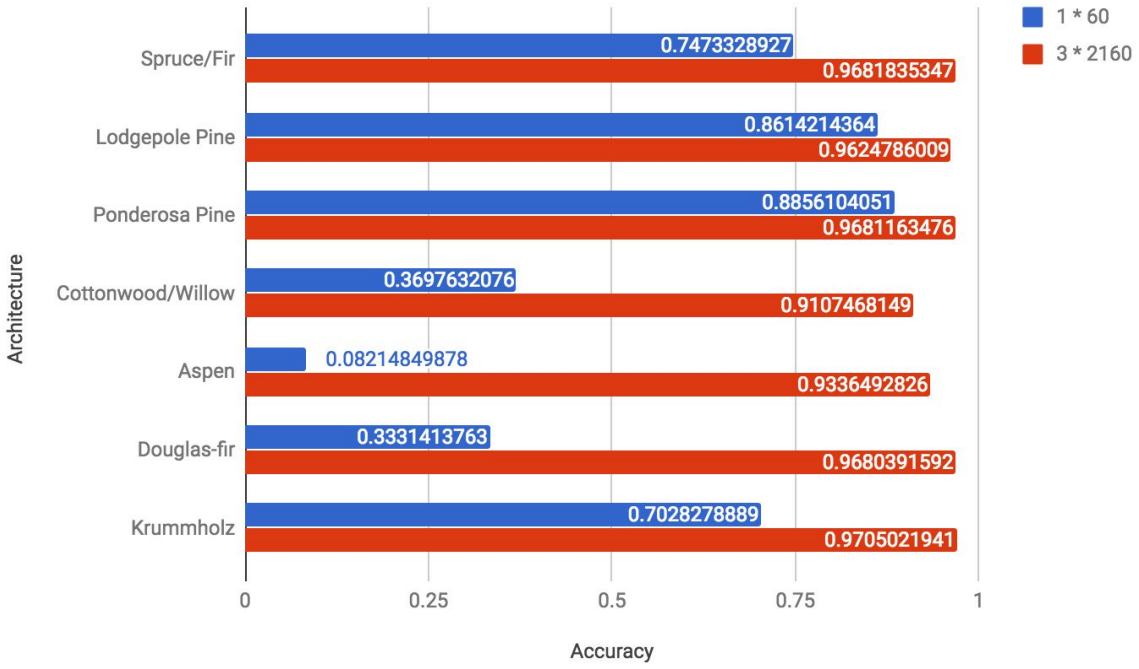


Fig 8. Accuracy Per Class of 1 x 60 and 3 x 2160 Networks

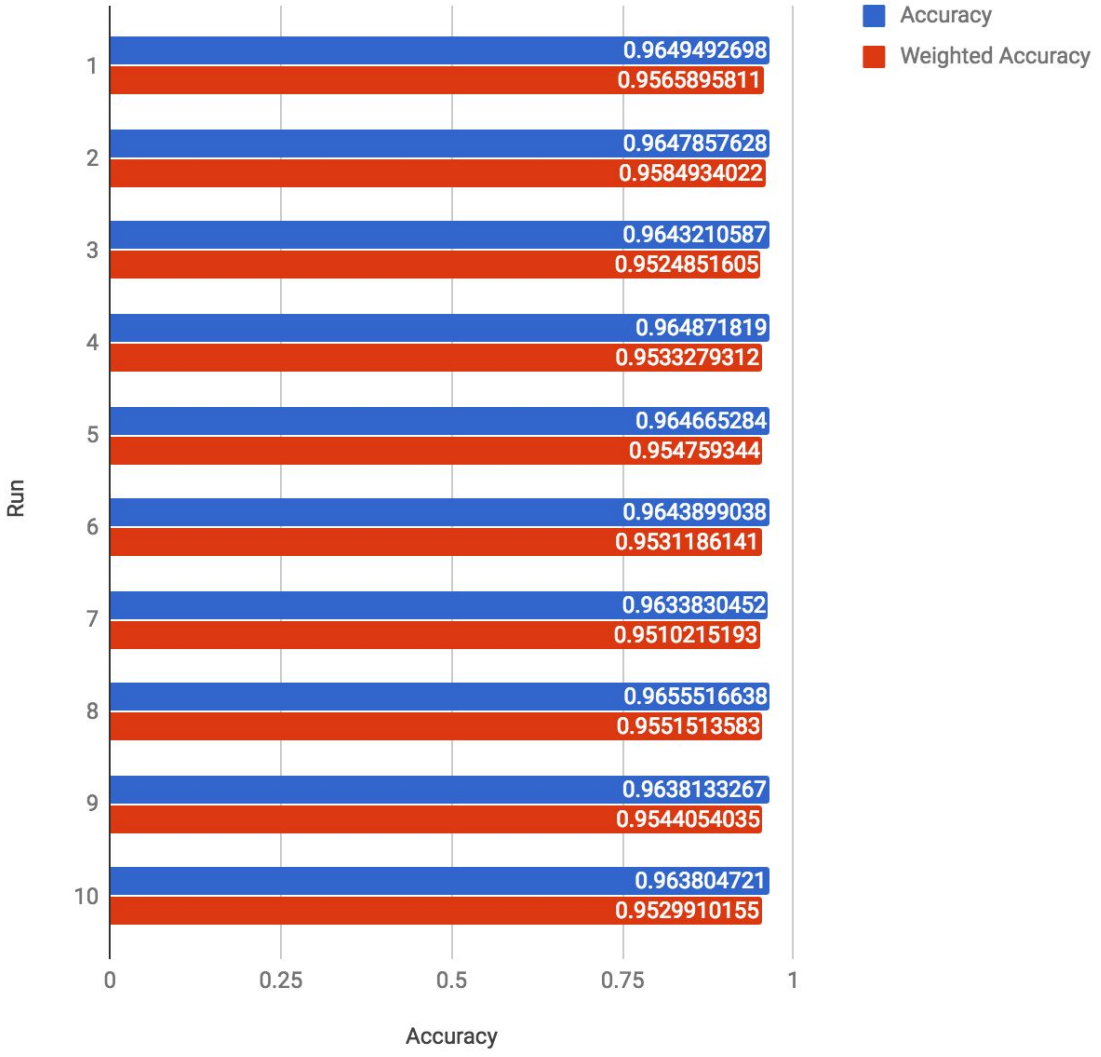


Results

Model Evaluation and Validation

The final version of the model used 3 hidden layers with 2160 units each. It was selected because it had the best accuracy and weighted accuracy of all the architectures evaluated. All hyperparameters were as described in the implementation section above.

Fig 9. Accuracy and Weighted Accuracy of Validation Models



As it was not possible to create or collect new data to test the model for generalization and robustness the experiment was re-run 10 times with different random seeds to ensure that different training and testing samples were selected each time. As can be seen in Fig 9 the accuracy and weighted accuracy of the models vary very little when trained and tested with different data. The range between the maximum and minimum accuracy of the validation

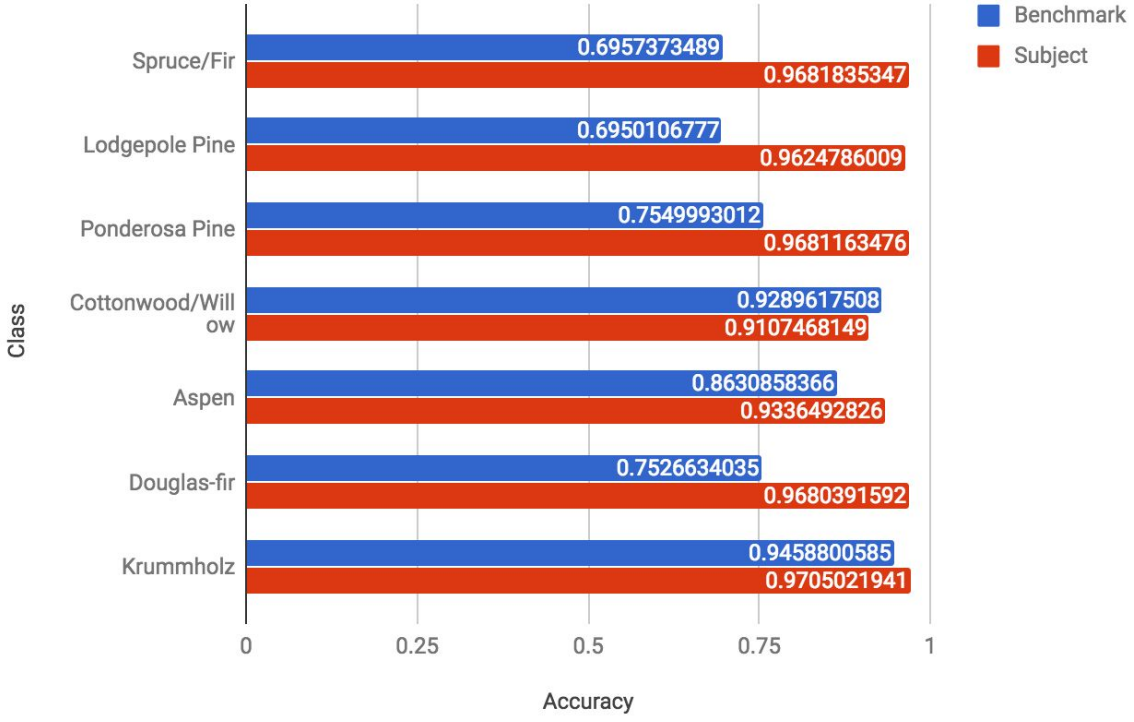
models is ≈ 0.2% and the range between the weighted accuracy of the validation models is ≈ 0.7%

Justification

The selected model had an accuracy of ≈ 96.46% and a weighted accuracy of ≈ 95.45%. The results of the selected model represent an improvement of ≈ 25.12% overall accuracy and ≈ 14.93% weighted accuracy when compared to the benchmark model. It had a better accuracy than the benchmark model in all classes except cottonwood/willow where the benchmark had an accuracy of ≈ 92.90% compared to an accuracy on the selected model of ≈ 91.07%

The overall accuracy of the chosen solution suggests that it could be a useful tool but given the limited nature of the dataset further work would be required to create a tool useful in forest inventory management.

Fig 9. Accuracy Per Class of Benchmark and Subject Models



Conclusion

Freeform Visualization

Fig 10 Accuracy of Cottonwood/Willow, Aspen and Lodgepole Pine. 1 Layer Networks.

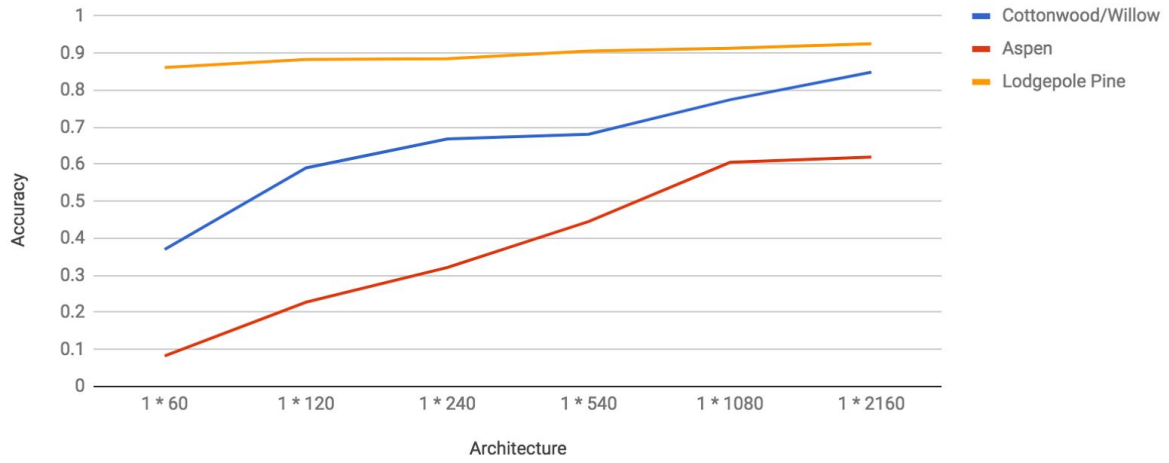


Fig 11 Accuracy of Cottonwood/Willow, Aspen and Lodgepole Pine. 2 Layer Networks.

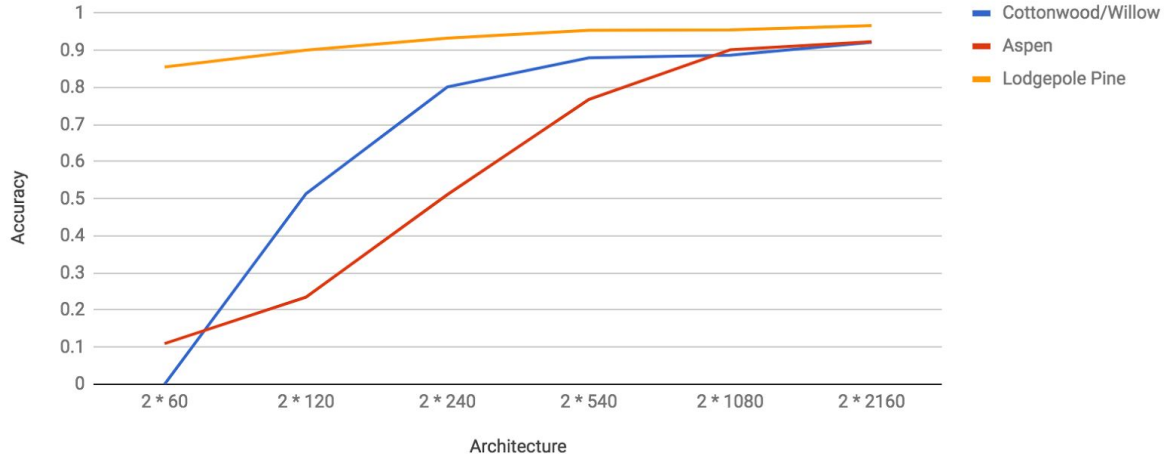


Fig 12 Accuracy of Cottonwood/Willow, Aspen and Lodgepole Pine. 3 Layer Networks.

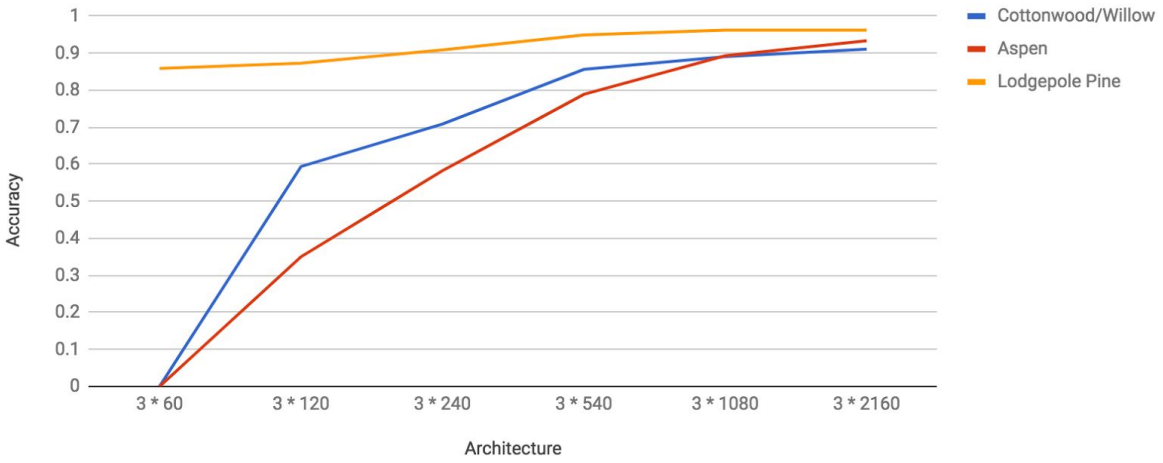
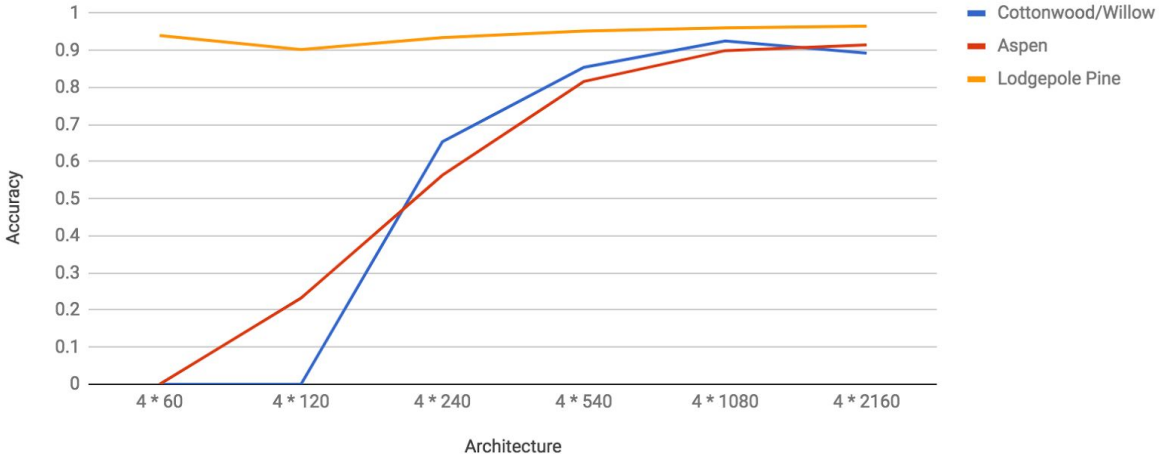


Fig 13 Accuracy of Cottonwood/Willow, Aspen and Lodgepole Pine. 4 Layer Networks.



Figures 10 to 13 above compare the the accuracy of the network on the least represented minority classes (Cottonwood/Willow and Aspen) with the accuracy on the largest majority class (Lodgepole Pine) as the network complexity grows. As can be seen even the simplest networks achieve a high accuracy on the majority class but have an accuracy close or equal to 0% for the minority classes. Their ability to learn and generalize from minority classes with relatively few instances grows as the network complexity increases, although it appears to fall away a little on the very largest network (4 x 2160 units)

Reflection

This project was created by:

1. Browsing the UCI Machine Learning Repository looking for suitable datasets and/or inspiration.
2. Downloading and examining the forest coverage dataset
3. Reading the original paper associated with the project and deciding it might be interesting to train a ‘modern’ neural network on the same dataset to compare the results
4. Carrying out some initial experiments which suggested that the goal was achievable.
5. Spending a great deal of time trying to get a benchmark model returning results similar to those found in the original paper, once I understood the problems associated with a skewed dataset.
6. Running the training sessions on Floydhub.
7. Collecting and presenting the results.

By far the most challenging aspect of the project was creating a benchmark model which could achieve results comparable to those in the original paper. I spent a lot of time experimenting with different pre-processing approaches and experimental parameters for this step. In comparison to this the subject models and final results were relatively easily obtained.
Several aspects of the results were quite surprising to me. I hoped to achieve some improvement but I did not expect to achieve the levels of accuracy seen in the final results. They are so high I do wonder if the model is instead learning to somehow infer the borders of the contiguous areas of coverage, or some other aspect from the variables provided rather than learning to predict the kind of coverage which will grow given the conditions described by the independent variables. I was even more surprised by the stability of the results when presented with differently selected and ordered test and training data. So much so that I even went to the extent of manually examining portions of the test and training data created with different seeds to confirm for myself that I had not made some mistake which was resulting in the same data being presented each time. Another aspect I found surprising was how good the larger networks were at learning to predict the minority classes without requiring any preprocessing such as using class weighs or random under/over sampling.

Improvement

All of the test and training data used in this project came from the US, the Roosevelt National Park. For the tool to be generally applicable it would almost certainly need to be trained on a far wider set of conditions and coverage types. It would then be interesting to present the model with testing data from completely unseen locations to validate its ability to generalise beyond the physical locations it has been trained with.
Even with the data available it would also be interesting to see if improvements could be made to the accuracy on the minority classes by pre-processing the data using one of the methods utilised for the benchmark model or varying some of the other available

hyperparameters. The accuracy on the smallest minority class is good but still 5-6% lower than the accuracy on the majority classes.