ETC3250/5350: Project report

Ha Lan Nhi (Chelsea) Le - 30100259 | Malo Hamon - 28754131 | Cecilia Li - 31882560

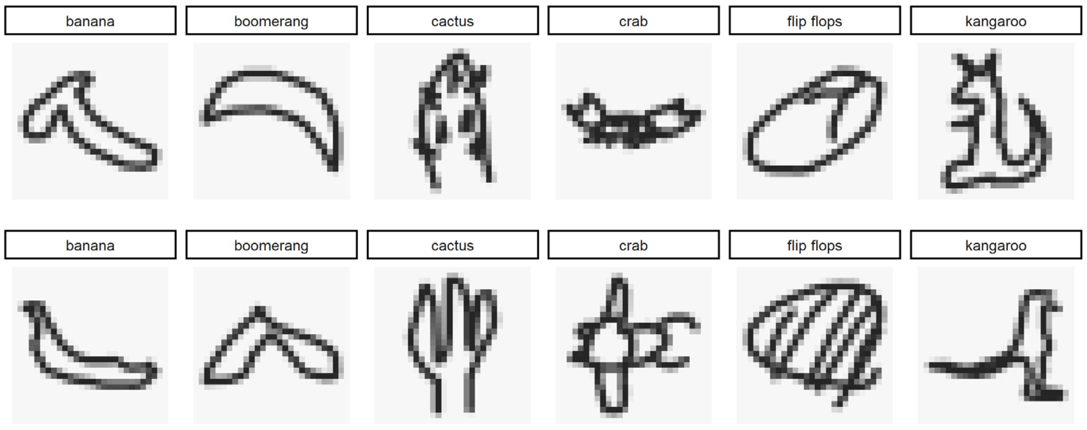
7 June 2020

Introduction

The aim of this project is to build a model that would best classify images of handwritten drawings into one of 6 items: banana, boomerang, cactus, crab, flip flops and kangaroo. The training data provided consist of 6,000 28 by 28 pixel greyscale images with the brightness values for each pixel ranging from 0 to 255 where black is 0 and 255 is white. There is a balanced number of images for each of the 6 items in the data set.

The data is in a wide format where each column represent a pixel of the image (i.e. 28 x 28 = 784 columns) and a column with the label of the item (e.g. banana).

Here is a sample of the images, e.g.



Methodology

We have considered a range of models including Random Forest, XGBoost and Neural Network. Our best model was a Convolutional Neural Network model that was then further improved through bagging and boosting.

Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of Neural Network that specialises in image recognition. Instead of applying a simple pixel to pixel comparison, it applies a series of learned filters to create feature maps e.g. lines and shapes. There are 3 main types of layers in a CNN:

- Convolution layer: Applies as series of filters over the image to create a stack of filtered images of the data (the feature maps). This allows certain features of the image to be recognised such as lines and shapes [3].
- Pooling layer: As the convolution layer records the precise location of each feature, this means that a slight misalignment will result in a different feature map. The pooling layer

addresses this limitation by summarising the presence of a feature in each small areas of the feature maps. We have used max pooling which summarises the most activated presence of a feature. This means that our model is less sensitive to the precise position of the features. Another purpose of this layer is dimension reduction to limit memory and improve speed of the model. [3]

- Fully connected layer: This is connected after a series of convolution and pooling layers to support the image classification using the features identified. [3]

We have chosen the hyperparameters for the convolution and pooling layer based on research conducted, focusing on the best practices for image recognition with neural networks. Most of the sources we researched agreed that the following arrangements are optimal for 28x28 pixel images [1] [2] [3]:

- A 2D Convolution layer with 32 3x3 filters
- A Max Pooling layer with 2x2 pooling size
- A second 2D Convolution layer with 64 2x2 filters
- A final Max Pooling layer with 2x2 pooling size
- A Flattening layer that converts the 3 dimensional data to a 3136x1 matrix

This flattened layer is then the input layer for our fully connected layers, which will perform the pattern recognition.

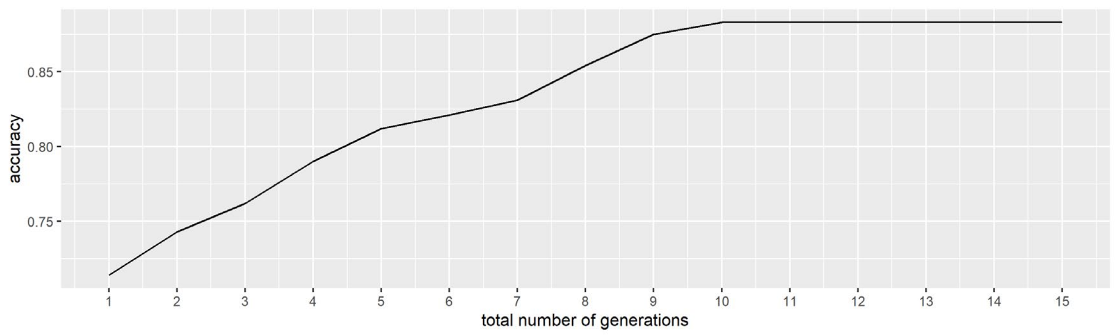
Genetic algorithm

To explore the optimal configuration for the hyperparameters of our fully connected layers, we have used a genetic algorithm to find the optimal combination of the loss function, optimiser, as well as the number of layers and the number of nodes and activation function of each layer.

The genetic algorithm runs through the selection and scoring of models for a number of iterations (i.e. generations). We have then used the best model for the fully connected layer of our CNN.

The algorithm involves first creating an initial population of models with randomised hyperparameters. The training data is passed through and each model is then scored. A subset of the best models (parents) is then included in the next generation, and the rest of the population for the next generation (children) is created by taking properties randomly from either 2 parents (chosen randomly from the best) or a mutant (a model with fully randomised hyperparameters). Each child can inherit a mixture of properties from any one of the chosen parent or mutant [7].

Here is a chart of the accuracy of the best model vs the number of generations we have ran the genetic algorithms for:



As we can see, after 9 generations, the models stop getting better. We have determined that the number of generations to use was 9.

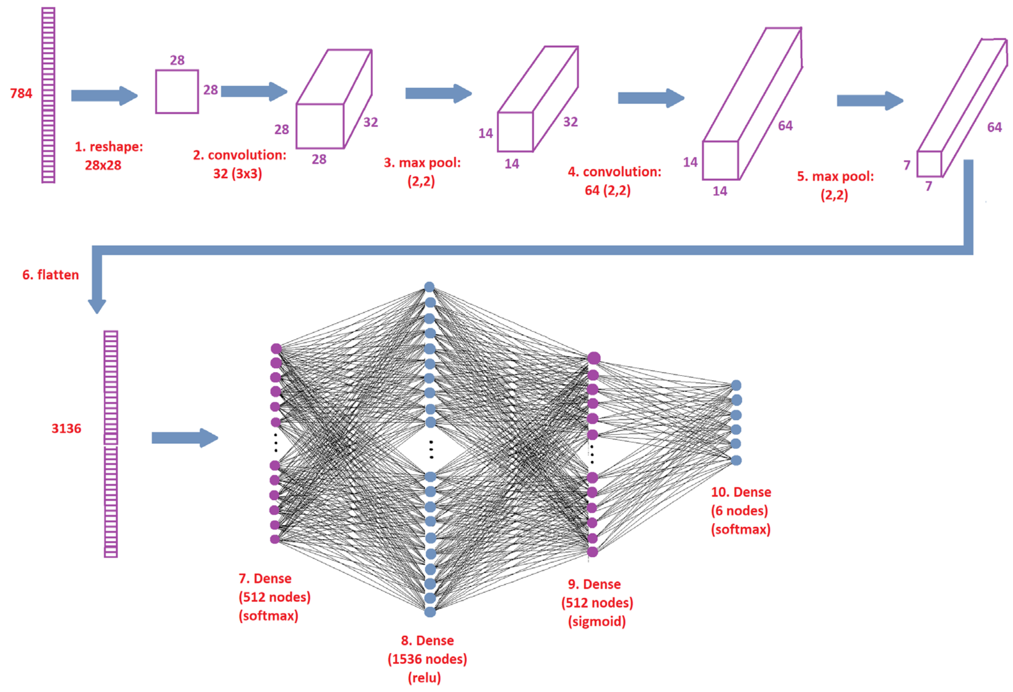
We then used the hyperparameters from the best model after 9 generations for our fully connected layer which is as follows:

- A Dense layer with 512 nodes and softmax activation
- A Dense layer with 1536 nodes and relu activation
- A Dense layer with 512 nodes and sigmoid activation
- A Dense output layer with 6 nodes and softmax activation.

The genetic algorithm also concluded that our model should use the `Adam optimiser`, and the `kullback leibler divergence loss function`.

We have also included dropouts into our CNN as a regulisation method to reduce overfitting and decrease bias. Dropout layers ignore a certain proportion of the weakest neurons in each layer (generally between 30% and 50%), this prevents introducing bias to the model. [4]

The picture below shows the full architecture of our model:



Bagging and Boosting

To further improve this model, we utilised a combination of ensemble methods such as bagging and boosting [6].

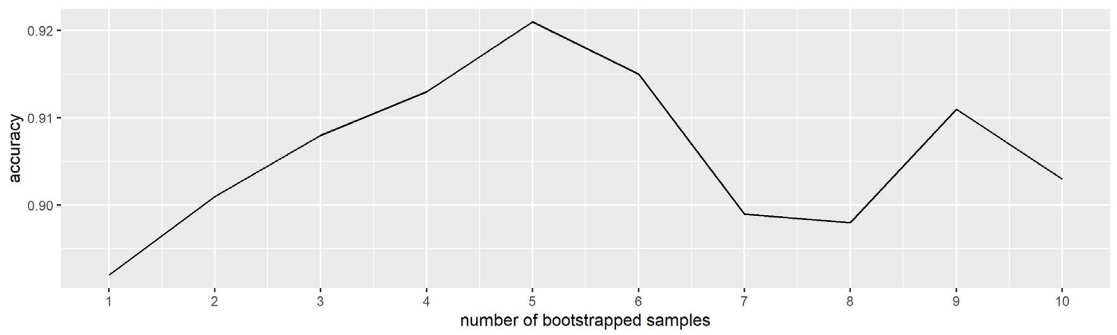
This involves train multiple models on different bootstrapped samples of the training dataset. We then calculate the average value for the responses from each model to obtain the final prediction. This helps minimise the bias of the overall model because we are not relying on a single dataset. [5]

We apply the following “boosting” algorithm to the CNN model on each bootstrapped sample.

1. Split the bootstrapped data into 2 test sets and 1 training set
2. Train the CNN model on the training set
3. Calculate the accuracy of the model using the first test set
4. Determine which observations the model is struggling to predict
5. Add these observations into the training set
6. Repeat steps 2 - 5 on the modified training set
7. Train the CNN model on the one more time on the modified training set
8. Predict the words in the second test set using the trained model

To perform this algorithm, we must split the data into 2 test sets and 1 training set because we are adding observations from the test set back into the training set and therefore the test observations are not new to the model. By having a second test set we can measure the performance improvements of boosting and select the best model.

Here is the accuracy and the corresponding number of bootstrapped samples we average across:

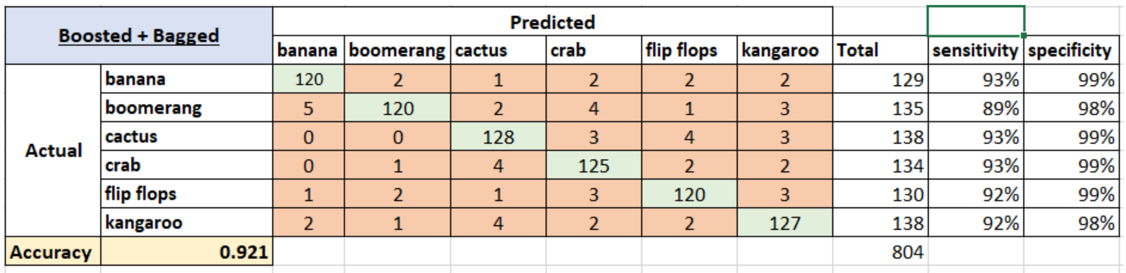


For our final model, we have the results based on the average of 5 boosted models.

Results and Discussion

Our best model using a CNN with bagging and boosting obtains an accuracy of 92%. This compares to the error rate of our best CNN model without bagging and boosting of 88.3%, XGBoost of 81% and Random Forest of 80%.

Here is the confusion table for our best model:

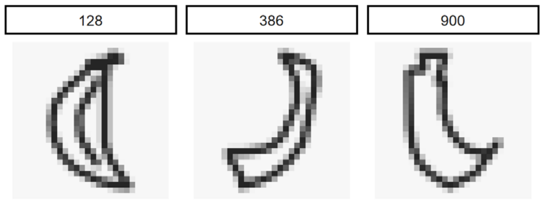


Boosted + Bagged		Predicted								
		banana	boomerang	cactus	crab	flip flops	kangaroo	Total	sensitivity	specificity
Actual	banana	120	2	1	2	2	2	129	93%	99%
	boomerang	5	120	2	4	1	3	135	89%	98%
	cactus	0	0	128	3	4	3	138	93%	99%
	crab	0	1	4	125	2	2	134	93%	99%
	flip flops	1	2	1	3	120	3	130	92%	99%
	kangaroo	2	1	4	2	2	127	138	92%	98%
Accuracy	0.921							804		

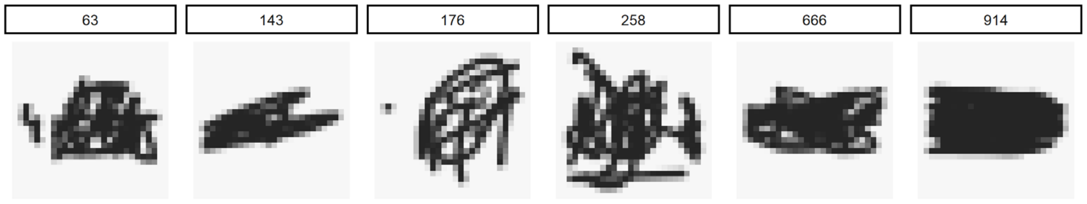
Boomerang is the hardest to classify with an accuracy of 89%. Specificity is high at 98% or 99% for all the words which is pleasing to see.

Overall, the three models decided on the same prediction for 1067 observations in the prediction set and gave different predictions for the remaining 133 observations.

The two most often confused classes are bananas and boomerangs, which were constantly misclassified by both Random Forest and XGBoost but surprisingly posed little difficulty for CNN. When looking at the 133 observations where the 3 models could not reach a single prediction, we found out that the CNN did well on sketches that we think are very hard for a model to correctly distinguish between bananas and boomerangs. Below are some examples where only CNN correctly recognised to be bananas, whereas Random Forest and XGBoost labeled as boomerangs:



When looking at the observations that were consistently misclassified, we discovered sketches where it looks like the player just gave up on the game and decided to make random scribbles. These sketches are indecipherable even with human eyes and all three models could not decide on a unanimous answer.



Conclusion

Through this project, we have seen the powers of a CNN in image classification over other methods such as Random Forest and XGBoost. CNN builds feature maps such as lines and shapes to improve it's classification power and pooling allows our model to be less sensitive to the precise position of the features. However, there is many hyperparameters to specify for the model and we have both relied on our research and the genetic algorithm to determine the combination to use. Despite the variations in the images for the same word (and in some cases, the images are indecipherable even with human eyes), our model performed reasonably well for a modest amount of training data.

References

[1] Ballantyne, A. (2020). Deep Gradient Boosted Learning. https://alan.do/deep-gradient-boosted-learning-4e33adaf2969

[2] Chakraborty, M., Biswas, S., & Purkayastha, B. (2019). A novel ensembling method to boost performance of neural networks. Journal Of Experimental & Theoretical Artificial Intelligence, 32(1), 17-29. doi: 10.1080/0952813x.2019.1610799

[3] Chollet, F. (2020). Building powerful image classification models using very little data. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

[4] Networks, B., & blue, s. (2020). Boosting neural networks. https://stats.stackexchange.com/questions/185616/boosting-neural-networks

[5] Bailly, K., & Milgram, M. (2009). Boosting feature selection for Neural Network based regression. Neural Networks, 22(5-6), 748-756. doi: 10.1016/j.neunet.2009.06.039

[6] Illanko, K. (2020). How do you improve the accuracy of a neural network? https://www.quora.com/How-do-you-improve-the-accuracy-of-a-neural-network

[7] S., S. (2020). Genetic Algorithms + Neural Networks = Best of Both Worlds. https://towardsdatascience.com/gas-and-nns-6a41f1e8146d

[8] Zheng, Zhuo. (2008). Boosting and Bagging of Neural Networks with Applications to Financial Time Series.

[9] Carson Sievert (2018) plotly for R. https://plotly-book.cpsievert.me

[10] Hadley Wickham, Dianne Cook, Heike Hofmann, Andreas Buja (2011). tourr: An R Package for Exploring Multivariate Data with Projections. Journal of Statistical Software, 40(2), 1-18. URL http://www.jstatsoft.org/v40/i02/.

[11] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, Mu Li, Junyuan

[12] Xie, Min Lin, Yifeng Geng and Yutian Li (2020). xgboost: Extreme Gradient Boosting. R package version 1.0.0.2. https://CRAN.R-project.org/package=xgboost

[13] JJ Allaire and Yuan Tang (2020). tensorflow: R Interface to 'TensorFlow'. R package version 2.2.0. https://CRAN.R-project.org/package=tensorflow

[14] JJ Allaire and Fran?ois Chollet (2020). keras: R Interface to 'Keras'. R package version 2.3.0.0. https://CRAN.R-project.org/package=keras

[15] Baptiste Auguie (2017). gridExtra: Miscellaneous Functions for "Grid" Graphics. R package version 2.3. https://CRAN.R-project.org/package=gridExtra

[16] Hadley Wickham (2017). tidyverse: Easily Install and Load the 'Tidyverse'. R package version 1.2.1. https://CRAN.R-project.org/package=tidyverse

[17] Alboukadel Kassambara (2020). ggpubr: 'ggplot2' Based Publication Ready Plots. R package version 0.2.5. https://CRAN.R-project.org/package=ggpubr

[18] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2013). An Introduction to Statistical Learning: with Applications in R. New York :Springer.

[19] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, Mu Li, Junyuan Xie, Min Lin, Yifeng Geng and Yutian Li (2019). xgboost: Extreme Gradient Boosting. R package version 0.90.0.2. https://CRAN.R-project.org/package=xgboost

[20] A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

[21] Max Kuhn and Hadley Wickham (2020). tidymodels: Easily Install and Load the 'Tidymodels' Packages. R package version 0.1.0. https://CRAN.R-project.org/package=tidymodels

[22] Max Kuhn (2020). caret: Classification and Regression Training. R package version 6.0-86. https://CRAN.R-project.org/package=caret

[23] Hadley Wickham, Romain François, Lionel Henry and Kirill Müller (2020). dplyr: A Grammar of Data Manipulation. R package version 0.8.5. https://CRAN.R-project.org/package=dplyr

[24] Bradley Boehmke & Brandon Greenwell (2020) Hands on Machine Learning with R, Taylor & Francis Group https://bradleyboehmke.github.io/HOML/

[25] ETC3250 Introduction to machine learning instructor notes week 6, 7 and 8

Appendix

More details on other models we fitted

Apart from CNN, we also tried out Random Forest and Extreme Gradient Boosting (XGBoost) models.

Random Forests (RF)

This model provides an improvement over bagged trees by breaking the correlation between decision trees built on bootstrapped training samples. The decorrelating process is done as follows: when considering a split in each of the trees, a random sample of m predictors is chosen as split candidates from the totality of predictors. The split is then allowed to use only one from these m predictors. This prevents all the trees from using just the few strongest predictors in their splitting process, which result in highly correlated predictions.

Surprisingly, tuning the hyperparameters for the Random Forests model does not improve its prediction accuracy by much from a model that uses the default hyperparameter values.

Evaluating our best RF model on a pre-labeled test set containing 1500 observations yields 80% accuracy.

Extreme Gradient Boosting (XGBoost)

This is an optimized distributed gradient boosting method that uses the same boosting procedure and tree-based hyperparameters as gradient boosting machines (GBMs) like:

Boosting parameters:

- number of trees
- learning rate

Tree parameters:

- tree depth
- minimum number of observations in terminal nodes

XGBoost improves upon the traditional GBMs by providing:

- Regularization hyperparameters that prevent overfitting,
- Early stopping which stops model assessment when additional trees offer no improvement,
- Loss functions which allow users to define and optimize GBMs by custom evaluation criteria,

Tuning the hyperparameters for XGBoost is computationally heavy yet provides little improvement in prediction accuracy from a simple RF model.

Evaluating our best XGB model on a pre-labeled test set containing 1500 observations yields 81% accuracy.