Leaf Classifier

CAPSTONE PROJECT
KAGGLE COMPETITION

Abdulrahman Abuzaid | Machine Learning Udacity Nanodegree | Feb 7, 2019

I. Definition

PROJECT OVERVIEW

Plants are an important aspect of life on the planet, without them, life on earth will go extinct, as they provide the necessary Oxygen for our existence [1]. Since there are around 400,000 types of plants in the planet [2], formulating an automatic classifier and identifier is an important endeavor. Such a classifier will improve humanity's efforts to preserve all plants, conduct scientific floricultural studies, help identify plant diseases, and possibly reduce global warming.

This domain has been an active area of research recently, with work being conducted on various plant species, and using different algorithms and analysis methods. Chinese researchers in 2018 proposed plant species classifiers using deep learning [5], while other researchers proposed different machine learning algorithms such as Support Vector Machines and K-Nearest Neighbors [7, 8]. Different approaches are also found in the literature, with some applying algorithms directly on the images while others approached the challenge by extracting leaf shape features [6]. Moreover, recent research also explored the possibility to identify and classify apple leaf diseases in India [7].

I chose this project since my life partner is a floriculturist and I want to impress her and introduce her to the power of Machine Learning, Deep Learning, and Artificial Intelligence in general.

PROBLEM STATEMENT

The leaf classification project is based on a Kaggle competition [4] where participants are challenged to produce a machine learning model capable of learning to classify 99 different plant species based on leaf images and features extracted from these images [3]. The challenge is to have the minimum classification error possible. The ultimate solution should be able to classify all leaves correctly. The classification challenge can be improved in future extensions by including more plant species.

In this project, I plan to solve this challenge by first loading the image data and the extracted features, followed by data exploration and preprocessing, and finally applying different machine learning algorithms such as decision trees, Adaboost, Support Vector Machines, K-Nearest Neighbors, and Convolutional Neural Networks. I hope that one or a combination of these powerful algorithms will be able to successfully solve this challenge by accurately predicting most or all of the provided test images, which is achieved by minimizing the log loss metric defined in the subsequent section.

EVALUATION METRIC

The Kaggle competition specifies the evaluation metric as the multi-class logarithmic loss as given below.

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\, log(p_{ij})$$

where N is the number of images in the test set, M is the number of species label, log is the natural logarithm, y_{ij} is *one* if observation i is in class j and *zero* otherwise, and finally, p_{ij} is the predicted probability that observation i belongs to class j.

Predicted probabilities are to be replaced by the following values to avoid the extremes of the *log* function.

$$p' = \max(\min(p, 1 - 10^{-15}), 10^{-15})$$

In addition, the accuracy score metric was used to assess the performance of the various classifier algorithms under study. Indeed, the accuracy score is more intuitive as it is the percentage of correct predictions [9]. However, the accuracy score has a drawback as it does not account for the probabilistic nature of the prediction and only provides true/false predictions. Using the log loss is helpful when it is relevant to know the probability of a certain sample to belong to each class under study. This approach enables us to capture the uncertainty in our predictions.

II. Analysis

DATA EXPLORATION

The data for this project has been provided by Kaggle [4] and it is freely available for download. The data consists of two types: image data and tabular data. The image data consists of 16 images for each one of the 99 species under study. The images are in black and white to emphasize the shapes of these leaves rather than their colors. The tabular data consists of three feature vectors of size 64 each which were previously extracted from the images as detailed in this article [3]. The extracted features are for the shape, margin, and texture.

To summarize, the available data is as follows:

Image Data (test and train datasets)

- 16 x 99 = 1584 images
- 991 training images (which was further split into training and validation data (80% training and 20% validation)

- 595 testing images
- Varying pixel dimensions
- Binary pixel data (black and white)

Tabular Training Data

- 991 rows x 194 columns (which was further split into training and validation data (80% training and 20% validation)
- One row for each image with the following columns
  - 64 columns for each extracted feature, margin, shape, and texture, in that order. (64 x 3 = 192)
  - Unique id column for each image
  - Species column

Tabular Testing Data

- 595 rows x 193 columns
- One row for each image with the following columns
  - 64 columns for each extracted feature, margin, shape, and texture, in that order. (64 x 3 = 192)
  - Unique id column for each image

Sample Data and Summary Statistics for Tabular Data

A sample of the training data is provided in Table 1 for the curious reader. Evidently, not much can be gained by assessing these number directly and they look a bit random.

id	species	margin 1	margin 64	shape 1	shape 64	texture 1	texture 64
1	Acer Opalus	0.0078	0.0039	0.0020	0.0006	0.0005	0.0007
2	Pterocarya Stenoptera	0.0059	0.0273	0.0000	0.0007	0.0009	0.0007
3	Quercus Hartwissiana	0.0059	0.0000	0.0078	0.0010	0.0012	0.0010
5	Tilia Tomentosa	0.0000	0.0273	0.0059	0.0005	0.0005	0.0004
6	Quercus Variabilis	0.0059	0.0137	0.0000	0.0007	0.0006	0.0008

Table 1: Sample of training data (showing 6 features out of 192 and the target label)

Statistic	Margin 1	Margin 64	Shape 1	Shape 64	Texture 1	Texture 64
count	990.00000	990.00000	990.00000	990.00000	990.00000	990.00000
mean	0.01741	0.00437	0.00074	0.00073	0.02194	0.01942
std	0.01974	0.00925	0.00027	0.00027	0.04404	0.02277
min	0.00000	0.00000	0.00017	0.00017	0.00000	0.00000
25%	0.00195	0.00000	0.00052	0.00051	0.00000	0.00098
50%	0.00977	0.00195	0.00072	0.00071	0.00684	0.01172
75%	0.02539	0.00391	0.00093	0.00092	0.02051	0.02930
max	0.08789	0.08984	0.00239	0.00243	0.41309	0.14160

Table 2: Summary Statistic for the same subset of 6 features

The summary statistics for a subset of the feature columns is available in Table 2. As can be seen, the feature columns have varying mean and standard deviation values, signifying that scaling of feature values is important. Feature scaling will be discussed in the Data Preprocessing section.

Finally, a sample image is displayed in Fig. 1 for one of the training samples.

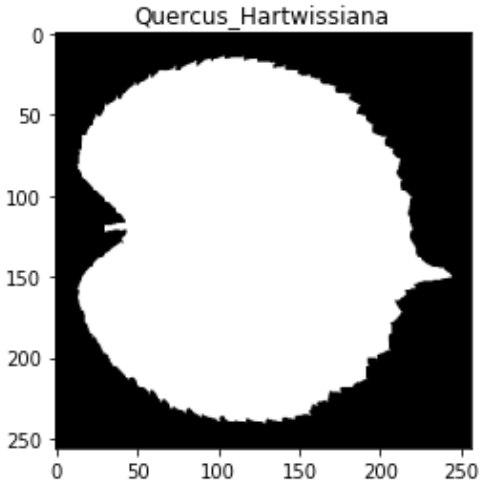

Figure 1: Sample image for training sample #3

EXPLORATORY VISUALIZATION

In order to better understand the data statistics, a scatter plot is shown in Fig. 2. The histograms for the margin and texture features show that most of the values are around zero, while the shape feature values are mostly above zero. Moreover, correlation is seen, especially in the shape features, which hints at the possible benefit of feature selection or dimensionality reduction, which will be assessed next. Principle Component Analysis (PCA) was applied on all of the 192 features in order to assess the cumulative variance explained by adding each additional feature. The first PCA component explains 22% of the variance while the first five components explain 53% of the variance. The first 20 components (10% of the features) explain 80% of the data variance. The full component/variance relation is shown in Fig 3.

Even though PCA showed that 20 features explain 80% of the variation of the data, I chose to train the selected algorithms on all of the data due to two reasons: 1) I am interested to achieve the best performance possible, and 2) computational power is available through cloud GPU computing in Udacity Workspace. In case of running these algorithms on a mobile app or CPU, I would consider utilizing the reduced dimensions to speed up the computations.

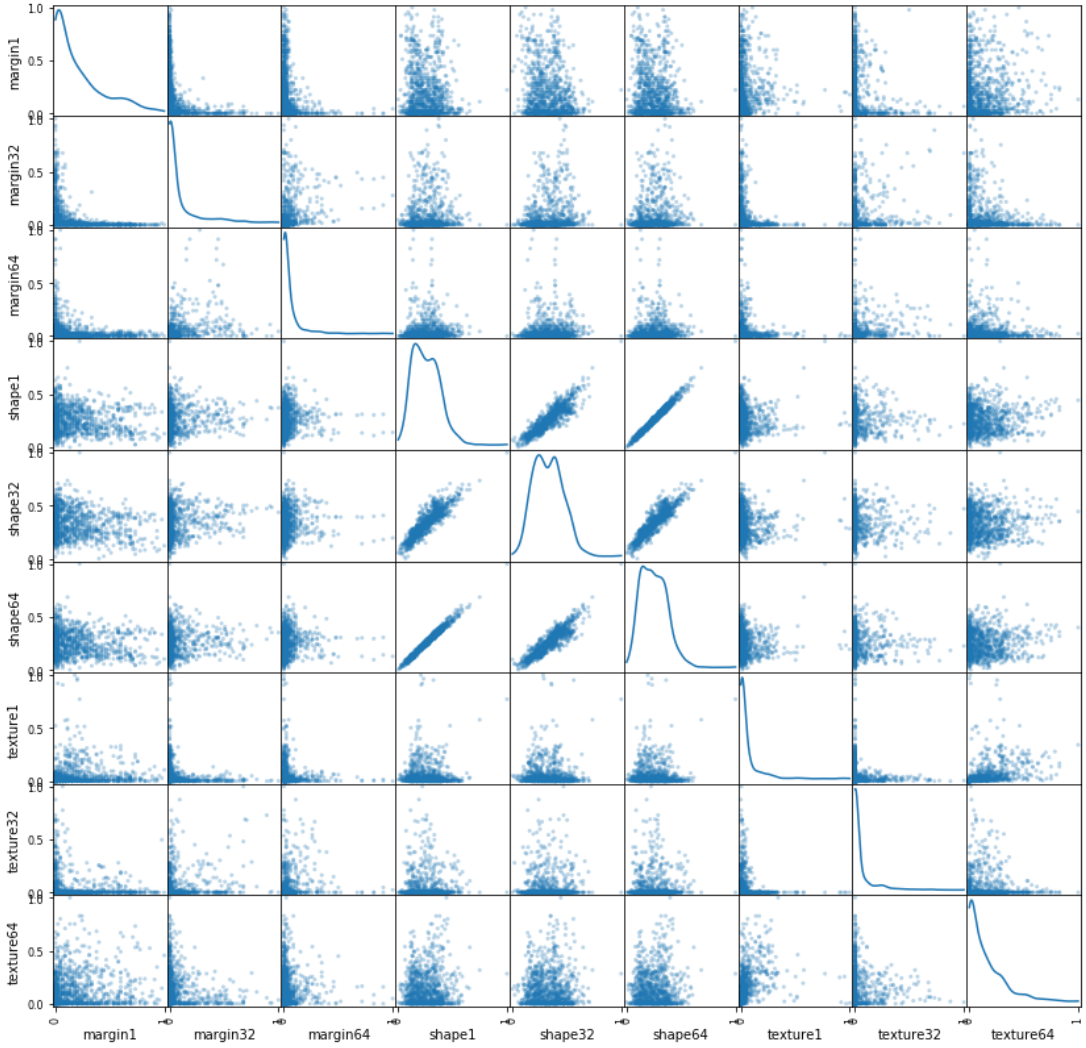

Figure 2: Scatter Plot for 9 features

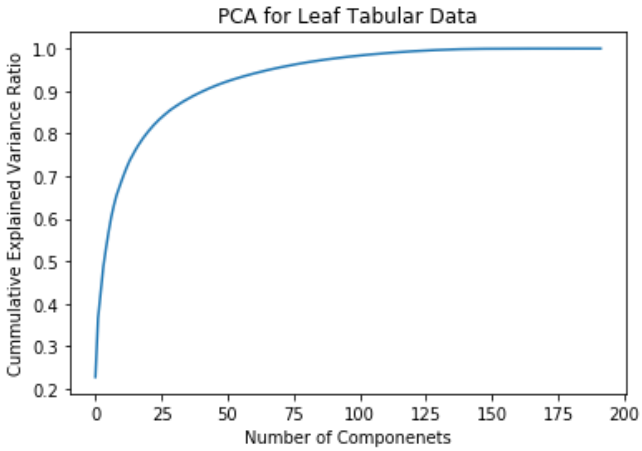

Figure 3: The cumulative explained variance ratio versus the number of components

ALGORITHMS AND TECHNIQUES

This problem is a supervised classification problem and can be solved through applying this class of machine learning algorithms. The problem is considered as supervised since we know the label that we want to predict: the plant species. The problem is a classification problem since we are predicting a categorical label, and not a continuous value.

First, using the tabular data, a number of classification algorithms are tested and compared, such as Decision Trees, AdaBoost, Support Vector Machines, and K-Nearest Neighbors. The GridSearch function is utilized in order to automatically fine tune the hyper parameters.

Next, Convolutional Neural Networks are applied directly on the 2D images to capture the spatial data and extract more features. The final solution proposed is a combined CNN algorithm that trains on both, the tabular and image data.

CNN is a state-of-the-art algorithm for image recognition and classification with immense applications. It requires huge computational capability which makes them more suitable to run on GPUs. CNNs can be considered as an extension to Neural Networks, but working on 2D/3D data instead of 1D vectors. However, both CNNs and NNs consist of layers of neurons with weights and biases that are applied to the input, passed through an activation function and finally fed to the next layer or final output.

CNNs start with the most important layer, which is the convolutional layer. This layer convolutes the input image with a square of a smaller dimension to capture the different features in the image, the more neurons in each convolutional layer the more features that can be extracted. Following are the pooling layers, such as max pooling and global average pooling, which act on the convoluted images to decrease the spatial dimension and reduce the required computations. Applied to the output of the pooling layer is an activation function, such as ReLu or tanh, which serve to capture the non-linearity in the input data. A repeated set of these three layers: convolution, pooling, and activation captures more and more complex features with each added set of layers. At the end of the sets of layers is a fully connected layer, that is connected to all neurons in the previous layer and predicts the final output of the network. These fully connected layers are the same used in regular NNs.

CNNs use the user-defined network architecture to feed the training data and update their randomly initialized weights and biases. The network goes through training iterations, where in each iteration it sees a prespecified batch size of the data, until eventually the algorithm converges. Finally, CNNs run the testing data through the network and the precalculated weights to arrive at a predicted classification. The algorithm assigns a specific probability for each sample and species pair. Meaning, the probability of a specific sample to be belonging to each species.

The following parameters and choices are defined for the CNN algorithm:

- Neural Network Architecture
    - Number of layers
    - Layer type: convolutional, fully-connected, pooling, flattening, dropout, activations
    - Neurons in each layer.
- Number of epochs: training iterations
- Batch size: number of samples fed to the network at each training iteration
- Solver type (Adam, etc.)
- Learning rate for the solver

BENCHMARK MODEL

The first benchmark model I considered was to reshape the image data into a 1d vector and feed it to a Multi-Layer Perceptron Neural Network without any extracted features. However, training for this model was not successful and instead, I decided to try out a set of different algorithms as benchmarks. The following algorithms/techniques will serve as benchmarks for this project:

- Models based on tabular data alone:
    - Decision Trees
    - Adaboost
    - Support Vector Machines
    - KNN
- Model based on image data alone: CNN.

The details of tuning all of the above models is available in the accompanying Jupyter notebook. The results for all benchmark models is shown in Fig 4 and 5 by plotting the accuracy and log-loss metrics for the original training dataset, and the log-loss metric computed through the Kaggle platform for the original test dataset.

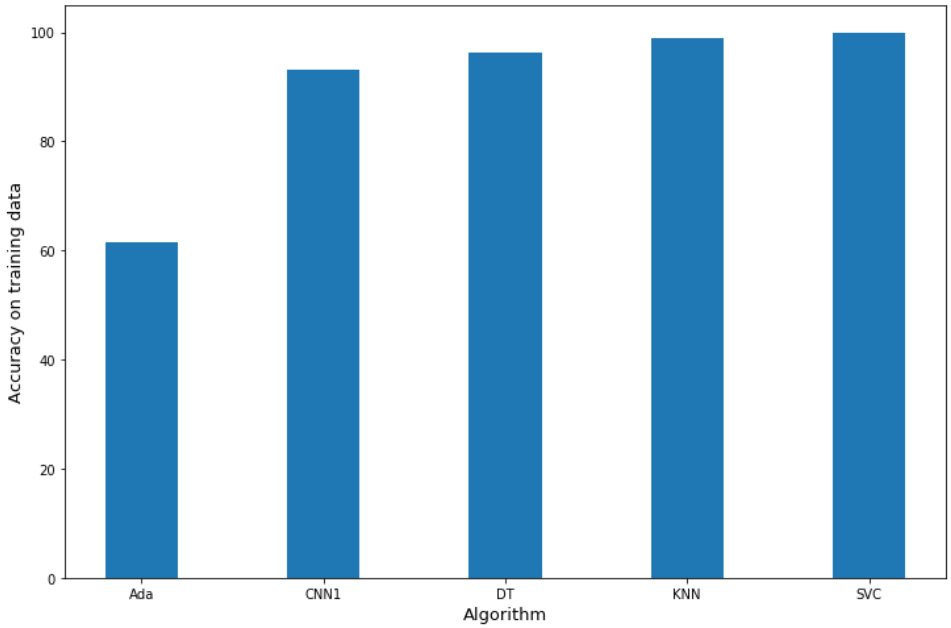

Figure 4: Accuracy for training data for the selected benchmark models

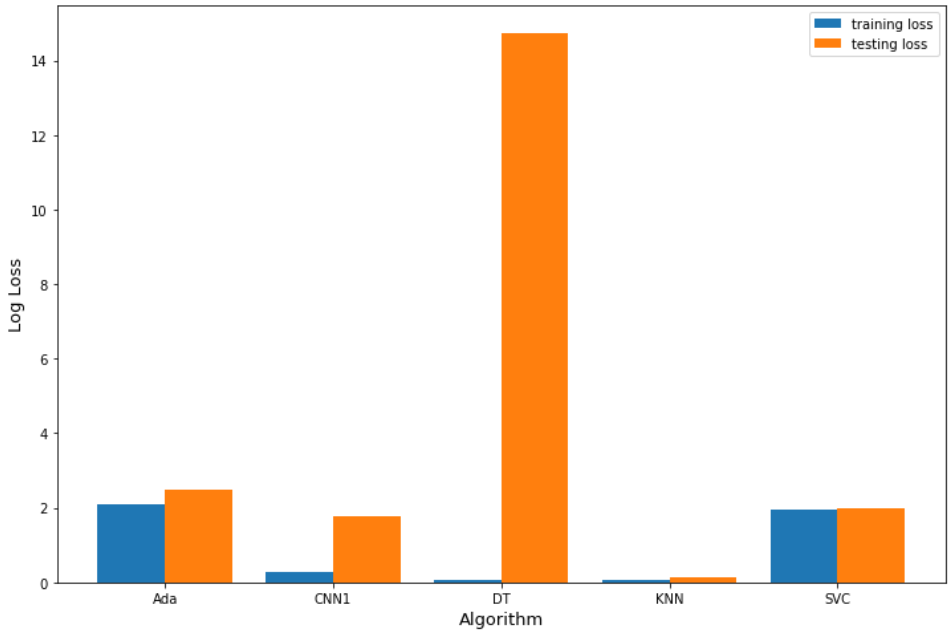

Figure 5: Log loss for five benchmark models for training and testing data

III. Methodology

DATA PREPROCESSING

Data preprocessing was implemented on both the tabular and image data as follows:

- Tabular data:
  - Loading the training and testing datasets.
  - Dropping the species column from training data to split data into X (features) and y (labels).
  - In order to improve algorithm performance in general, features in both training and testing datasets underwent feature transformation to rescale each features between the min and max. Transformed data is between zero and one.
  - Splitting the provided training data further to have training and validation data. Split is 80% training and 20% validation. Final split when combined with all data is 50% training, 12.5% validation, and 37.5% testing.
  - PCA was attempted and applied to the data, but it was not considered to maximize metric performance.
- Image data:
  - Extracting image data from the zip file in the cloud.
  - Loading image training and testing data.
  - Rescaling all image dimensions to be 256 x 256 pixels.
  - Rescaling each pixel value to be between zero to one.
  - Splitting the training images into training and validation sets with the same proportions as the tabular data.
- One-Hot encoding for the label data.

IMPLEMENTATION

The solution implemented is a Convolutional Neural Network classifier that combines both the tabular and image data. The implementation process was carried out as follows:

1- Loading and preprocessing the tabular and image data.
2- Define network architectures for the two proposed models, one for training on the tabular data and another on the image data. The network architectures are displayed in Fig 6 and 7.
3- Combine the two models into one, and define a network architecture to train the combination prior to the final output. The final model is shown in Fig 8.
4- Define the optimizer details and parameters.
5- Compile the model using categorical cross entropy and accuracy as the metric.
6- Provide mechanism to capture and save the best model found so far, based on the lowest validation loss.
7- Fit the model with the training and validation data. Define the batch size.

8- Using the best model found after the last epoch, calculate the accuracy and log loss metrics for the training data.
9- Save a submission file with the predicted probabilities for each sample in the testing dataset.
10- Submit the prediction to Kaggle and report the computed log loss on the testing dataset.
11- Repeat from step 1 if the result is not significantly better than the best benchmark model.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 128, 128, 16)      80
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 64, 64, 16)        0
_________________________________________________________________
dropout_10 (Dropout)         (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 32)        2080
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_11 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 8, 8, 64)          8256
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_12 (Dropout)         (None, 4, 4, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_7 (Dense)              (None, 99)                101475
=================================================================
Total params: 111,891
Trainable params: 111,891
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: Image data model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 200)               38600
_________________________________________________________________
batch_normalization_2 (Batch (None, 200)               800
_________________________________________________________________
activation_3 (Activation)    (None, 200)               0
_________________________________________________________________
dropout_13 (Dropout)         (None, 200)               0
=================================================================
Total params: 39,400
Trainable params: 39,000
Non-trainable params: 400
_________________________________________________________________
```

Figure 7: Tabular data model

```
____________________________________________________________
Layer (type)                 Output Shape              Param #
============================================================
merge_2 (Merge)              (None, 299)               0
____________________________________________________________
dense_9 (Dense)              (None, 200)               60000
____________________________________________________________
activation_4 (Activation)    (None, 200)               0
____________________________________________________________
dropout_14 (Dropout)         (None, 200)               0
____________________________________________________________
dense_10 (Dense)             (None, 99)                19899
============================================================
Total params: 231,190
Trainable params: 230,790
Non-trainable params: 400
____________________________________________________________
```

Figure 8: Final model to combine the tabular and image models

Throughout this project, I faced various challenges such as:

- It took me a while to figure out the way to build a combined model for the tabular and image data. I am glad that I looked into the keras documentation and read about the Merge layer and an example of how to use it. I was skeptical of it at first but then it turned out to be working better than all the other benchmark models.
- Finding a combination of hyperparameters and network architectures that worked, especially that the feedback from the simulation takes time due to the complexity of the network architecture.
- Initially I overlooked the importance of scaling the data. Once I did that, I noticed an improvement in performance by scaling the tabular and image data to be between zero and one.

REFINEMENT

In the process of arriving at the best solution, various points of the implementation were reconsidered such as, preprocessing of data, changing the network architecture, and changing the network and training hyper parameters. Below is a list of the reconsiderations attempted while refining this project:

- Not preprocessing the tabular and image data.
- Preprocessing the tabular and image data to be between zero and one.
- Changing the number of filters in the convolutional layers between 8 and 64.
- Experimenting the number of convolutional layers from one to three.
- Trying Dropout layers from zero to 0.5.
- Trying different activation layers: Relu and Tanh.
- Adding Batch normalization.
- Increasing the number of epochs.
- Increasing the number of fully-connected layers.

- Increasing the number of nodes in a fully-connected layer.
- Experimenting with Flatten layer versus Global Average Pooling layer.
- Decreasing the learning rate for the optimizer from .01 to .0001.
- Different validation and training splits by varying validation between 10-15%

Moreover, the initial solution for the problem I attempted was the CNN1 model, applied only to the image data. I expected this model to perform well, however, it was actually outperformed significantly by the KNN model applied to the tabular data alone. Comparison between the initial and final model solution and the best benchmark model is provided in Table 3.

Model	Training Accuracy	Training Loss	Testing Loss
KNN (best benchmark)	100	0.05	0.14
CNN1 (initial)	91.6	0.35	1.79
CNN2 (final)	100	0.001	.02

Table 3: Results for the initial and final models compared to the best benchmark model

IV. Results

MODEL EVALUATION AND VALIDATION

The final model is a robust model that can be trusted. It performed exceptionally well on the validation dataset and it generalized perfectly on the never-seen-before testing dataset. The final model is providing the expected solution by using the following parameters and architectures:

- Network architecture is as shown in Fig 6-8.
- For the image model, the first, second and third convolutional layers have 16, 32 and 64 layers, respectively. Each with a kernel and stride size of 2. Each followed by a max pooling layer of size and stride of 2.
- Dropout of 0.5.
- Flatten layer before combining with the tabular model
- Tabular mode with a fully connected layer of size 200, followed by batch normalization and a dropout of 0.5.
- Final merged model starts with concatenating the output of the two models, followed by two dense layers of sizes 200 and 99, which is the number of species classes.
- Activation layers: tanh
- Final Activation layer: softmax
- RMSprop optimizer with learning rate of 0.001, categorical cross entropy loss, and accuracy metric
- Epochs = 2000,

JUSTIFICATION

As shown in Table 3, the final model significantly outperformed all the benchmarks and the initial model. A log loss of 0.05 is really close to zero, which would be the perfect solution. I believe that the model can be refined further as evident from the Kaggle leaderboard. However, the current score is still considered as a significant solution to the classification problem.

V. Conclusion

FREE-FORM VISUALIZATION

Since the labels of the testing dataset are hidden and only available in the Kaggle competition in order to calculate the log loss metric. I am going to use the validation dataset to shown that the model is achieving high accuracy score by displaying nine sample images, with the predicted (P) and real (R) plant species. As shown in Fig 9, all of the images were identified correctly.

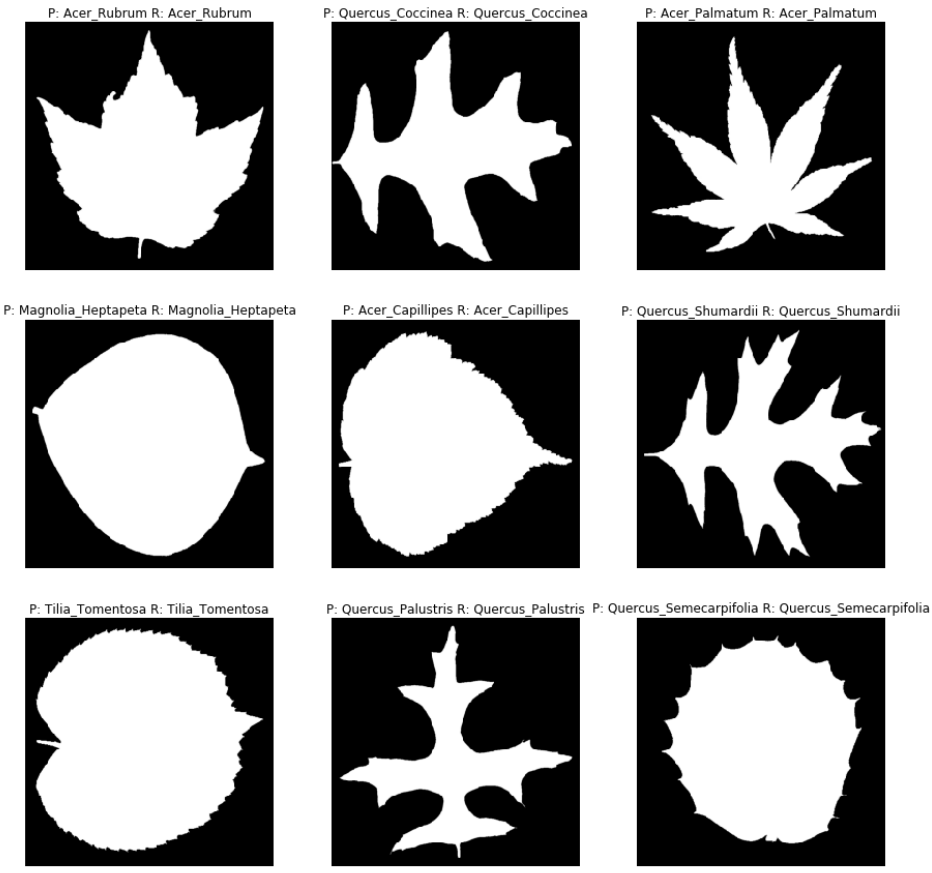

Figure 9: Predicted vs. real plant species for nine sample validation images

REFLECTION

I followed the following procedure when attempting the whole project:

1- Searching for an interesting problem and dataset that caught my attention. I am glad that I found the Leaf Classification challenge on Kaggle as I have a personal and family interest in plants.
2- The data was retrieved and uploaded to Udacity workspace due to availability of GPUs.
3- The data was loaded and preprocessed with scaling and by exploring dimensionality reduction.
4- Benchmark models were identified and assessed.
5- The final classifier model was proposed and enhanced through an iterative and continuous procedure of changing network architecture and tuning hyper parameters.
6- The model with best performance was saved and a submission file was saved and then submitted to the Kaggle competition website.

Moreover, the idea to have two CNN models, one for the tabular data and another for the image data and subsequently merge them was not an easy one to implement, as it was not covered in the Udacity classroom.

I faced challenges in almost all of the above steps, as it is harder to complete an independent project compared to guided ones. However, the experience is highly rewarding and it definitely adds to my confidence as an aspiring machine learning engineer.

IMPROVEMENT

I can point out that the proposed solution in this project can be further improved by various methods, such as:

- Extracting more features from the images other than the shape, texture, and margin proposed in the article [3]. Of course, the CNN algorithm applied on image data extracts spatial features automatically, however, manual addition of features might improve the model further.
- The final model was trained in 30-45 minutes on GPUs on Udacity workspace, which is not bad. However, in future iterations, I plan to explore using PCA to reduce the dimensionality of the problem and with minimal effect on the metric performance. Additionally, reducing the size of the input images is possible to improve the performance.
- Finally, image augmentation can be used to “cheat” and increase the number of our training dataset.

References

[1] https://www.nature.com/scitable/blog/our-science/no_trees_no_humans

[2] https://news.mongabay.com/2016/05/many-plants-world-scientists-may-now-answer/

[3] https://www.researchgate.net/publication/266632357_Plant_Leaf_Classification_using_Probabilistic_Integration_of_Shape_Texture_and_Margin_Features

[4] https://www.kaggle.com/c/leaf-classification

[5] Plant Leaf Classification Based on Deep Learning. *2018 Chinese Automation Congress (CAC), Automation Congress (CAC), 2018 Chinese*. 2018:3165. doi:10.1109/CAC.2018.8623427.

[6] Haque F, Haque S. Plant Recognition System Using Leaf Shape Features and Minimum Euclidean Distance. *ICTACT Journal on Image & Video Processing*. 2018;9(2):1919-1925. doi:10.21917/ijivp.2018.0272.

[7] Sivakamasundari G., Seenivasagam V. Classification of Leaf Diseases in Apple Using Support Vector Machine. *International Journal of Advanced Research in Computer Science*. 2018;9(1):261-265. doi:10.26483/ijarcs.v9i1.5124.

[8] Prasetyo E, Adityo RD, Suciati N, Fatichah C. Multi-class K-support Vector Nearest Neighbor for Mango Leaf Classification. *Telkomnika*. 2018;16(4):1826-1837. doi:10.12928/TELKOMNIKA.v16i4.8482.

[9] http://wiki.fast.ai/index.php/Log_Loss