Sorghum -100 Cultivar Identification - FGVC 9l

A. Introduction

The Kaggle 'Sorghum Identification' competition is a computer vision challenge to classify various species of Sorghum plants. It is a classification problem. Hosted by the FGVC9 workshop, participants are given a dataset of Sorghum images to train their models on. The objective is to classify the species of Sorghum in new images with high accuracy. The competition is open to all, and the winner will be determined by the accuracy of their model on a test set.

The Sorghum-100 dataset, a subset of RGB imagery created from ABI channels, is labeled by cultivar. This data will be used to develop plant phenotyping techniques to answer questions about the presence or absence of traits, including the cultivar in an image.

Accurate cultivar prediction through machine learning has practical applications in plant breeding. For example, it can detect mislabeled plants or errors in planting, which can be common in large-scale field experiments with hundreds of cultivars. The goal of this project is to solve the identification problem using the data provided.

The identification problem we intend to solve in this project is based on the data found in:

A.1. Challenges

Some of the challenges we have been facing are:

- There are differences in leaf shape, stem color, and seed size of different Sorghums, which can make it challenging to accurately classify Sorghum 100 images.
- The images were taken at different times of the day, resulting in varying levels of brightness across the images.
- Different Sorghum images are coming from different fields, which can add to the variability of the dataset.
- The images need to be resized for ResNet50, which we will be using, and this process has not been optimal so far.
- Since we are using Google Colab Pro Plus, we have only been able to use 10

https://www.kaggle.com/competitions/sorghum-id-fgvc-9/data

A.2. Teams Attending The Competition

This competition is from CVPR 2022. Participants can be part of a team of a maximum of 5. This competition is being sponsored by Saint Louis University.

Participants may use automated machine learning tools such as Google AutoML, and H2O Driverless AI to submit to the competition given that they ensure that they have the license to the used AMLT. The top teams on the score board are:

1. DeepBlueAI(score 0.965, 65 entries)
2. AutoHomeIIT(score 0.959, 371 entries)
3. STAR(score 0.957, 69 entries)

A.3. Possible Approaches

In this problem, one may try at least two different approaches. The first approach is to train different deep-learning models such CNN to detect different plants from the image. The second approach may be used to train data, solve the inference problem, and come up with a class of L-systems describing each category. Then use the grammar to categorize the test set. Given that plants in the same category are not identical, stochastic L-systems could be used to describe each category, then use these grammars on the test set(for example [2], [6], [3]). We intend to put both of these models into action and compare their outcomes and run times. For related work, one could refer to

The data set could be found in the following link: https://www.kaggle.com/competitions/sorghum-id-fgvc-9/data

A.3.1 Explanation Of The Data Set

The 'Sorghum Identification'challenge on Kaggle features a dataset made up of two Excel files and two folders of images, each with a size of around 32 GB. The first Excel file serves as a sample submission, while the second contains the train.cultivar.mapping information. Each image in the dataset is of high quality and has an average size of 1.5 MB.

The dataset consists of 20,000 Sorghum plant images, each labeled with a different species. Participants are tasked with training their models on this dataset and then using their models to classify new, unseen Sorghum images. The competition, which is centered around computer vision, aims to reward the highest classification accuracy on a test set.

A.3.2 Example Of An Image In The Data Set

To view an example from the dataset, refer to Figure 1 (Figure1).

A.4. Link To The Git Repository

Please refer to the following link for access to one of the codes from Kaggle, and to view my first attempt at creating a single neuron from scratch.

Figure 1. Example Of An Image From The Data Set

- https://github.com/alilotfi90/Advanced-Deep-Learning-Project

A.5. GPU

It seems that some contestants used the P100 GPU by NVIDIA, which is a part of the NVIDIA Tesla series. The P100 GPU features NVIDIA's Pascal architecture and is built on a 16nm FinFET manufacturing process, with 3584 CUDA cores and 16GB or 12GB of high-bandwidth memory.

For the challenge, we will be using either the NVIDIA V100 or A100 Tensor Core GPU, depending on which one is available at runtime on Google Colab Pro Plus.

A.6. Related Papers And Resources For Attacking The Problem

We will discuss three new papers in this part. Two([1]) briefly and the third([4]) in details.

A.6.1 First article

The paper titled 'A Survey of Deep Learning Methods for Image Segmentation' (referenced as [1]) provides an overview of the various deep learning techniques used for image segmentation, which involves dividing an image into separate segments or regions. As a beginner in deep learning, particularly Convolutional Neural Networks (CNNs), this paper was very interesting to me since I wanted to learn about CNN before this week's class. The authors study the development of deep learning algorithms, including Fully Convolutional Networks (FCN), U-Net, and SegNet, and compare their efficiency on different datasets. They also discuss the challenges faced in image segmentation using deep learning, such as managing large-scale variations, preserving fine details, and dealing with class imbalance, and highlight recent advancements to overcome these challenges.

A.6.2 Second article

The paper 'Training Large-Scale Structured Support Vector Machines' introduces a technique for training large structured SVM models that are widely used for image classification and object detection. In the third part, the authors present an algorithm which I tried to understand thoroughly. Equation 2 defines the function to be minimized as a function of X, Y, and w. This function refers to Δ, which calculates the overlap between Y and $\bar{Y}$, or in other words, a custom loss function. Initially, it appears that $\bar{Y}_i$ is the solution to Equation 5, and the parameter vector w is updated as outlined in Equation 14.

A.6.3 Third article

Currently working on [4].

B. Codes, Bugs, And Beyond

I wanted to use my own laptop, so I have been fixing several issues. These issues started with my desire to use Jupyter instead of Colab and some problems with pip on Windows. Some packages were also not installed correctly. I tried to solve these problems by searching Google, reading forums, and going through the code block by block. I may currently be experiencing the last issue, which is the code trying to access the GPU but not being able to find it. I haven't tried Anaconda yet, but I might give it a try. I haven't found a solution to this issue yet. The current state of the code (test2.ipynb) can be found on my Github: https://github.com/alilotfi90/Advanced-Deep-Learning-Project. To understand the code, I have been reviewing the tutorials from class and watching videos from Algoexpert-ML. I was able to implement a single neuron from scratch, which can also be found at https://github.com/alilotfi90/Advanced-Deep-Learning-Project.

C. Statistical Analysis

C.0.1 The Code We Implemented For Analyzing The Data

The code which analyzes the data stasticially could be found at https://github.com/alilotfi90/Advanced-Deep-Learning-Project/blob/main/statistics.on.data.ipynb.

- Create a dictionary(dic) of labels, such that dic[label] is a list of image names of the label kind.

- Create dictionary dic_count such that dic_count[label] is the number of image names of the label kind.
- Draw a histogram of frequency of each label.
- Pick a subset of train set.
- Pick a subset of test set.

A more detailed description of the code is as following:

- The code imports two Python modules, pandas and matplotlib.pyplot. pandas is used to work with data in tabular format, and matplotlib.pyplot is used to create plots and visualizations.
- Then we read data from train_cultivar_mapping.csv file using the read_csv() function from the pandas module. This file contains information about the different cultivars of sorghum and the images that correspond to each cultivar.
- Next, we create dictionary dic_count such that dic_count[label] is the number of image names of the label kind.
- The code creates a dictionary called dic, where each key in the dictionary corresponds to a cultivar of sorghum, and the corresponding value is a list of image filenames that are associated with that cultivar. This is done using a for loop that iterates over each row in the CSV file, and checks whether the cultivar column is empty or not, and whether the value is a string or not. If the value is a valid string, it is added to the corresponding list in the dictionary.
- We also determined the size of the smallest class of sorghum, i.e. the class with the fewest number of images. This is done using another for loop that iterates over each key in the dictionary, and keeps track of the size of the smallest class.
- Thereafter, we created a list of labels and a corresponding list of values for each label. These lists are used to create a histogram that shows the distribution of the number of images across the different sorghum classes. This is done using the bar() function from matplotlib.pyplot. We had to do this since the code for different images were long and cause confusion in the reading the histogram.
- The code also creates a table that maps the index of each label to the corresponding sorghum class. This is done using the DataFrame() function from pandas, which creates a new dataframe with the given data. Then, the table() function from matplotlib.pyplot is used to create a table visualization of the dataframe. For part of the table, please refer to Figure 2. For the complete table of correspondence between integers between 0 to 99, please visit https://github.com/alilotfi90/Advanced-Deep-Learning-Project/blob/main/table.jpeg
- This code creates two directories for storing a subset of the train and test images respectively. It first defines the path where the subset images will be saved, and then specifies the number of images to be sampled for each label. It then iterates over the labels in the dic dictionary, and for each label it creates a subdirectory under the corresponding sample directory. It selects a random sample of images for each label using the random.sample method, and removes the sampled images from the dictionary so that they are not included in the test set. Finally, it copies the sampled images to the respective subdirectories of the sample directories for the train and test sets using the shutil.copyfile method.
- In the last part, it picks a subset of train images and test images from the original dataset by randomly selecting a certain number of images from each label. The selected images are then copied to a new directory where they will be saved for later use.

  For the train set, the code defines a directory where the sample images will be saved (sample_dir) and the number of images to be sampled for each label (num_samples). Then, it iterates over each label in the dictionary dic, creates a directory for the label if it does not already exist, and selects a random sample of num_samples images from the corresponding list of images in dic. The selected images are then removed from the list in dic to ensure they are not selected again. Finally, the code copies the sample images to the appropriate label directory in sample_dir.

  The process is then repeated for the test set, with the only differences being the directory where the sample images will be saved and the number of images to be sampled for each label. We selected 20 images and 4 images from each label for the train set and test set, respectively. The subset that we have selected is 3.6 GB in size and is not small enough to be uploaded to GitHub.

C.0.2 Figures

C.0.3 t-SNE Analysis

The code I implemented creates an empty dictionary and assigns each subfolder in the directory an index, so that we can keep track of the different groups of images.

Index	Value
0	PI_257599
1	PI_154987
2	PI_92270
3	PI_152651
4	PI_176766
5	PI_156330
6	PI_329299
7	PI_52606
8	PI_145633
9	PI_273969
10	PI_196586
11	PI_156463
12	PI_22913
13	PI_156393
14	PI_35038
15	PI_152828
16	PI_152694
17	PI_329310
18	PI_251672
19	PI_154750

Figure 2. Table of correspondence between Sorghum id and corresponding integer for the first 20 labels.

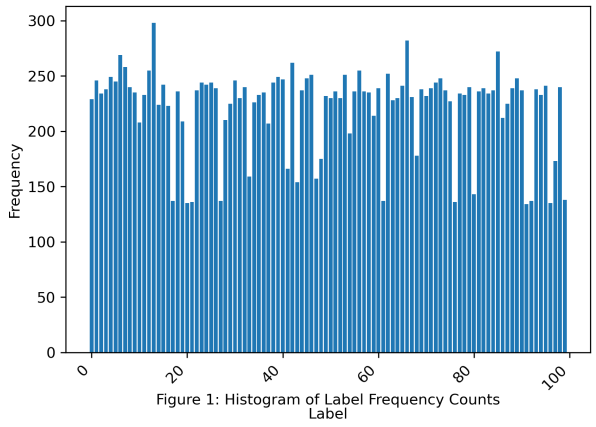

Figure 3. Table of correspondence between Sorghum ID and corresponding integer for the first 20 labels, with integer values used as indices according to the complete version (see Figure 2 on my GitHub for more details).

Next, the code loops through each subfolder and converts the first five images in each subfolder to grayscale, flattens them into vectors, and saves them in a list. The corresponding index of each subfolder (which we obtained in step 2) is saved as the label for each vector.

Then we concatenate all the flattened image vectors into a matrix, and concatenates all the labels into a single array.

Finally, it uses t-SNE (t-distributed stochastic neighbor embedding) to reduce the dimensionality of the data matrix to two dimensions(n_paramenter), and plots the result using Matplotlib. The label array is used to color-code the plot based on the different labels(or subfolder, since we partition our data for better use of Google Colab).

For the plot it generated please refer to Figure 4. To see our implementation, please refer to https://github.com/alilotfi90/Advanced-Deep-Learning-Project/blob/main/tsna_analysis.ipynb

The mentioned link leads to the latest version of my code. There were a few other failed attempts that proved to be quite educational. In one case, I attempted to use Resnet18 to build a feature matrix and then used dimensionality reduction to check how Resnet18 performs on my data. Additionally, I faced challenges because my images were 1024 by 1024, and I had to resize them into smaller images to speed up computation.

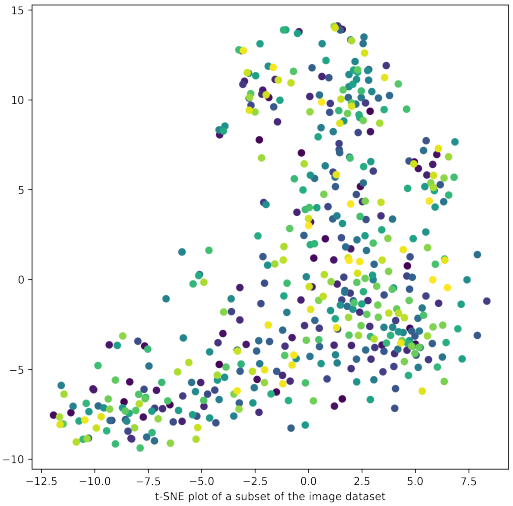

Figure 4. Table of correspondence between Sorghum ID and corresponding integer for the first 20 labels, with integer values used as indices according to the complete version (see Figure 2 on my GitHub for more details).

Overall, the code is performing unsupervised learning on image data using t-SNE to visualize similarities between the different groups of images, based on their pixel values. This is a common technique used in exploratory data analysis for high-dimensional data.

C.0.4 Different Colour Channels and Intensity Frequency

In this section, we will present the three histograms we obtained from our color analysis of the data. For the three color channels, red, green, and blue, we calculated the average intensity among different labels and plotted the frequency histogram. Please refer to Figures 5, 6, and 7 for the average intensity distribution among different labels for the colors blue, red, and green, respectively. It is not surprising that the average intensity of green has a positive shift compared to the other two colors, given the nature of the

pictures of Sorghums. Note that blue intensity around 70 has higher density among different labels (Figure 5), and the red intensity of around 85 has a high frequency among different labels (Figure 6). At first glance, it may seem like a better idea to focus on the green distribution for characterization. This is because the distribution is more evenly spread for the color green among different labels.

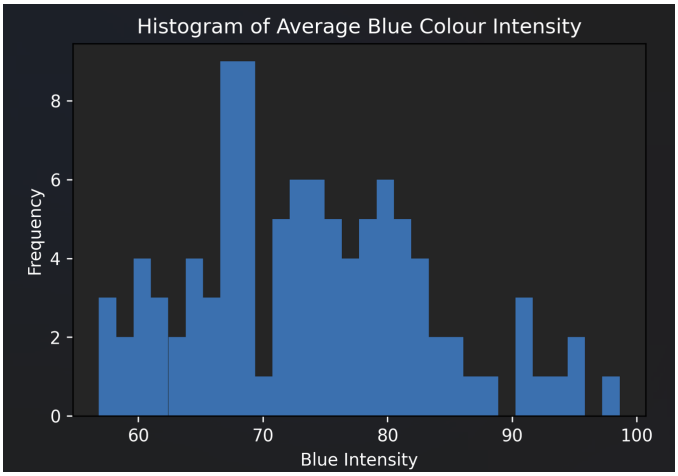

Figure 5. Histogram of Blue channel intensity among 100 different labels.

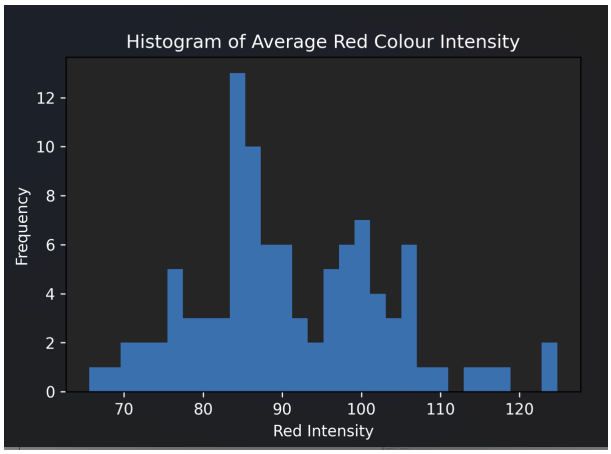

Figure 6. Histogram of Red channel intensity among 100 different labels..

C.0.5 First Few Attempts

Convolutional Neural Networks (CNNs) are a popular type of deep neural network that is commonly used for image classification tasks. CNNs are composed of multiple convolutional layers, which are used to extract features from images, and pooling layers, which reduce the spatial dimensions of the features. One particular CNN architecture that has gained popularity is ResNet50.

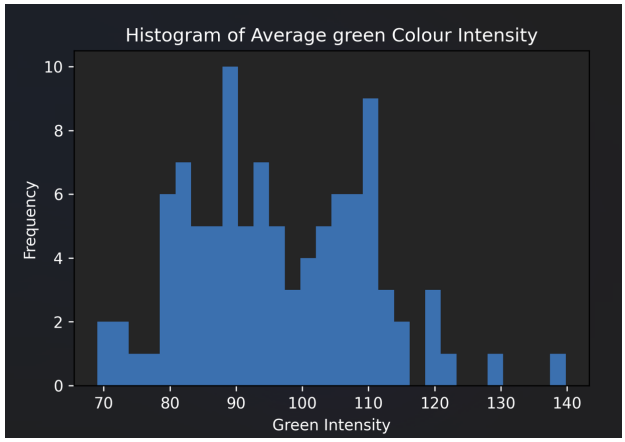

Figure 7. Histogram of Green channel intensity among 100 different labels..

ResNet50 is a CNN architecture that was first introduced in 2015. It is a deep neural network that includes 50 layers and has been shown to be highly effective for image classification tasks. One of the key features of ResNet50 is its use of skip connections. Skip connections allow the gradients to flow directly through the network without being attenuated by multiple layers, making it possible to train much deeper neural networks than was previously possible. This has been shown to improve the accuracy of image classification models.

Our model which is variant of ResNet50 is a Convolutional Neural Network (CNN) that was trained on the Sorghum 100 dataset. The code of the model could be found on https://github.com/alilotfi90/Advanced-Deep-Learning-Project/blob/main/model2.ipynb.

The details of the code is:

- First we load the ResNet50 model from the torchvision.models module, with pre-trained weights on the ImageNet dataset. resnet50 = models.resnet50(pretrained=True).
- Then we added a fully connected layer to output 100 classes. Note that num_ftrs is set to the number of input features to the last fully connected layer as you can see in the code:

```
num\textunderscore ftrs = resnet50.fc
    .in\textunderscore features
resnet50.fc = nn.Linear(num\
    textunderscore ftrs, 100)
```

- In the following we set up the loss function and optimizer. The loss function used is nn.CrossEntropyLoss(), which is commonly used

for multiclass classification tasks. The optimizer we used is stochastic gradient descent (SGD), and with a learning rate of 0.001 and a momentum of 0.9.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(resnet50.
    parameters(), lr=0.001, momentum
    =0.9)
```

My model was trained using Google Colab Plus with a high RAM GPU premium. I trained the model for 10, 20, and 30 epochs, and achieved accuracy of 14%, 39%, and 41%, respectively. The weights of the three trainings are too big. The results show that the model's accuracy improved significantly as we increased the number of epochs. This is expected since the model has more opportunities to learn from the data as the number of epochs increases.

In general, the model architecture seems ok for an image classification task like the one we are working on. However, there may be some room for improvement in the model's hyperparameters, such as the learning rate and the batch size. Also, data augmentation could be added to the training data to increase the model's ability to generalize to new data.

C.0.6 What Could One Do???

So far we have tried to increase the number of epochs. According to papers such as [5], we could also try the following which in the next subsection we will attempt the first item:

1. Increase the number of epochs: The accuracy of the model may continue to improve with more training epochs. We can try increasing the number of training epochs to see if the accuracy improves further.

2. Use a different optimizer: Instead of using SGD, one could try using a different optimizer such as Adam or Adagrad, and experiment with different learning rates and other hyperparameters.

3. Data augmentation: Data augmentation techniques such as rotation, flipping, and scaling can be used to create new training examples and improve the model's ability to generalize to new data.

4. Fine-tuning: We can fine-tune the ResNet50 model by freezing the first few layers and only training the later layers to better fit our specific dataset.

5. Use a different pre-trained model: We can try using a different pre-trained model such as InceptionV3 or EfficientNet, which may be better suited to our dataset.

6. Use an ensemble of models: An ensemble of models can help improve the accuracy by combining the predictions of multiple models.

C.0.7 Changing From SGD to Adam

Next we saved and load our weights from the 40 epochs experiment and changed the optimizer from SGD to Adam. This could be done by changing the following:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(resnet50.parameters(),
    lr=0.001, momentum=0.9)
```

to:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(resnet50.parameters(),
    lr=0.001)
```

and the result of the experiment for 10 epochs is 13% accuracy. By setting the epoch to 30, we achieved an accuracy of 30%.

References

[1] Zeynep Akata, Scott Reed, Daniel Walter, Honglak Lee, and Bernt Schiele. Evaluation of output embeddings for fine-grained image classification. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2927–2936, 2015. 2

[2] Paul Henry Cournède and Cedric Loi. Generating functions of stochastic l-systems and application to models of plant development. *Discrete Mathematics & Theoretical Computer Science*, 2008. 1

[3] FIRIDE YIMENU DAGIM. *IMAGE BASED SORGHUM LEAF DISEASE CLASSIFICATION USING DEEP LEARNING APPROACH*. PhD thesis, 2021. 1

[4] Paul Shekonya Kanda, Kewen Xia, and Olanrewaju Hazzan Sanusi. A deep learning-based recognition technique for plant leaf classification. *IEEE Access*, 9:162590–162613, 2021. 2

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 6

[6] Lonesome Malambo, Sorin Popescu, Nian-Wei Ku, William Rooney, Tan Zhou, and Samuel Moore. A deep learning semantic segmentation-based approach for field-level sorghum panicle counting. *Remote Sensing*, 11(24):2939, 2019. 1