Implementation

In this section of the paper, we will be discussing about the implementation part of the project. The details will be explained further in small subsections. The first step that we did was to collect some necessary datasets to feed some deep learning models for the training phase. With the collected datasets, we also performed some data pre-processing as to get the data in the correct format for the training phase. At the same time, we build some deep learning models that were available in the fast.ai library to get a comparison of which models is best, by comparing the performance using some useful evaluation metrics, for our scenario and to be used in the competition. Lastly, in this section, we will provide the user the final analysis and comparison made after the training phase and also our final choice of which models to be used. In the next section, an analysis of the evaluation of our simulation part will be presented. To summarise, we developed an activity diagram as a means of framework to standardise our workflow which can be seen the figure)))))).

Before going into the details of our implementation, it is worth spending some time to understand the general framework used for our project. This means getting an understanding of the software that we used for the project. The main library that was advised by the university to use is the fast.ai library. Fast.ai is a deep learning library that gives users with high-level components that can rapidly and easily deliver state-of-the-art results in conventional deep learning domains. This is because it is a high-level API that was built on top of the Pytorch framework. The library provided us with many useful libraries that aided our process. For example, the data loaders function. Data Loaders are used to get the dataset from the filesystem into a usable format and to handle the labels correctly. Since we only did a simulation of the drone on the field, no hardware was involved in the entire process of the project.

1. Dataset

In the very first step of the implementation, we have gathered datasets that we deemed suitable to be trained on the deep learning models. In total we combined three different datasets that we found on the internet. The first dataset was provided by the university which was a plant seedling dataset by COMPUTER VISION IN BIOSYSTEMS Department of Electrical and Computer Engineering – Aarhus University.\ref{ Giselsson2017} the dataset contains 12 different classes that includes varied plant species. Examples include Miaze plant, common wheat plant and the desired sugar beet plant. In total there are x number of images.

The second dataset was from. It contains images of. The total amount of images is x.

Lastly, we used a dataset from Kaggle. It contains images of. The total amount of images is x. In total we used around 16000 images that was for the training of the deep learning models.

1. Data pre-processing

Before we proceed with the training part, we prepared all the data in a standard way. We used the data loaders function to resize the images to 128x128 pixels and the method we used was squish method that is found in the library. For the training-validation split, we split the data to 80-20 split. 80 percent of the data was used for training and 20 percent for validation. The split is crucial because we do not want our model to be biased and have a false impression of better model accuracy. The training set is the set of data that is used to train and make the model learn the hidden features or patterns in the data. The validation set is used to validate the performance of the model during training.

1. Models

While we were preparing the data, we also build some well-developed deep learning models that were found in the fast.ai library. The models that we have build were Resnet18, resnet34, Resnet50, Alexnet, Vgg19, Squeezenet, and Densenet121. All the models mentioned, were trained separately and its performance were recorded. We built the models by using the vision\_learner function. The function is a pretrained model that can be finetuned by transfer learning. For Vison tasks, we usually use a Convolutional Neural Networks (CNN). We included parameters like the data loader, the architecture of the model, and the metrics that we want to use for evaluation purposes like accuracy and error rate.

1. Training

For the training, we used the fit\_one\_cycle() method. Our goal is to get a very low error rate and highest accuracy possible. During this phase we also applied some optimization techniques in order to improve the model’s performance. The first optimization was to use a learning rate finder. We used the learn.lr\_find() function that was included in the library to find the suggested learning rate and we passed the value to the fit\_one\_cycle function as a parameter. The second technique we used was to experiment with different number of epochs to train each model. We observed the error rate for each model we train and stopped them at the point where the error rate stopped decreasing. For example, we used 7 epoch to train vgg19 while only 5 epoch to train alexnet.

1. Comparisons: Analysis of results

After training the models, we proceeded to compare the results of training using the training loss, validation loss and the accuracy for the evaluation metrics. We performed the comparisons of the model in order to find the best model to use for the competition. Based on \ref{}, one can observe that the model vgg19 obtained the highest accuracy possible compared to the other models. Even though vgg19 obtained the highest accuracy, when performing our simulation, we found that the model does not perform the best. Therefore, we compared again under the simulation phase which will be explained in the next section.

|  |  |  |  |
| --- | --- | --- | --- |
| Models | Training Loss | Validation loss | Accuracy |
| Resnet18 | 0.694761 | 0.576014 | 0.840406 |
| Resnet34 | 0.616316 | 0.527605 | 0.858555 |
| Resnet50 | 0.495615 | 0.461564 | 0.874077 |
| Alexnet | 1.199039 | 0.991522 | 0.738265 |
| Vgg19 | 0.052474 | 0.036602 | 0.987851 |
| Squeezenet | 0.866343 | 0.745793 | 0.798598 |
| Densenet | 0.306583 | 0.367283 | 0.903868 |

References

@article{Giselsson2017,

author = {Giselsson, Thomas Mosgaard and Dyrmann, Mads and J{\o}rgensen, Rasmus Nyholm and Jensen, Peter Kryger and Midtiby, Henrik Skov},

journal = {arXiv preprint},

keywords = {benchmark,database,plant seedlings,segmentation,site-specific weed control},

title = {{A Public Image Database for Benchmark of Plant Seedling Classification Algorithms}},

year = {2017}

}