MSc in Computer Science - University of Moratuwa
CS5613 - Neural Networks
Group Assignment

Plant Seedlings Images Classification using Convolutional Neural Networks (CNNs)

Team Avengers

D.P. Jayasuriya (209337M)

T.P.N. Silva (209381P)

H.T.C Leelananda (209351B)

Git Repository URL

https://github.com/DulanGit/Plant-Seedling-Classification

Table of Contents

INTRODUCTION

The objective of this assignment is developing a deep CNN model to classify the images of plant seedlings.

System Configuration

- Tensorflow version 2.2.0
- Python 3.7.6
- Ubuntu 20.04
- Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
- Nvidia GeForce GTX 1650 (4GB)
- 16 GB RAM

Data Set

The Data Set used for plant seedling classification consisted of a training data set and a testing data set which contained images of plant seedlings at various growth stages. There were 12 plant species used in the classification.

- Black-grass
- Charlock
- Cleavers
- Common Chickweed
- Common wheat
- Fat Hen
- Loose Silky-bent
- Maize
- Scentless Mayweed
- Shepherds Purse
- Small-flowered Cranesbill
- Sugar bee

Data Preprocessing

Cleaning and preprocessing of data is a crucial step of the project. It consisted of 3 phases.

a. Resize the image
b. Convert RGB images to HSV format.
c. Blur the images to remove the noise.

d. Create a mask to remove the background.

e. Normalize the pixel values

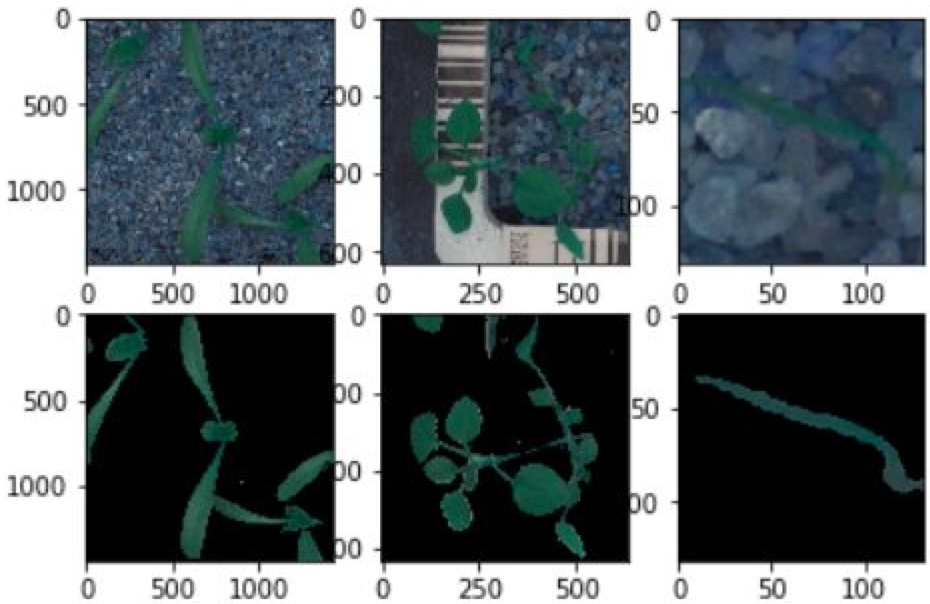
The images were converted to HSV. HSV color space consists of 3 matrices, 'hue', 'saturation' and 'value'. 'Hue' represents the color, 'saturation' represents the amount to which that respective color is mixed with white and 'value' represents the amount to which that respective color is mixed with black.

Then the images were filtered using GaussianBlur technique.

It was followed by creating a mask to separate the background and extract the seedling.

```
def preprocess_image(self, image):
    """
    This method will do the pre-processing for a given image

    """
    # Resize the image
    image = cv.resize(image, Preprocess.input_shape)

    # Use Gaussian blur
    blurred_image = cv.GaussianBlur(image, (5, 5), 0)

    # Convert to HSV image
    HSV_image = cv.cvtColor(blurred_image, cv.COLOR_BGR2HSV)

    # Create the mask based on the green values
    lower_green = (25, 40, 50)
    upper_green = (75, 255, 255)
    mask = cv.inRange(HSV_image, lower_green, upper_green)
    kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (11, 11))
    mask = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)

    # Create bool mask
    binary_mask = mask > 0

    # Apply the mask
    clear = np.zeros_like(image, np.uint8)  # Create empty image
    clear[binary_mask] = image[binary_mask]  # Apply boolean mask to the origin image

    # Normalize pixel values
    normalized_image = clear / 255

    return normalized_image
```

*** The following image shows the original image and its respective pre processed image**



Converting String Labels to Binary Classification

We have converted the labels(String format) to a binary classification. So an array of size 12 was created. When a species was detected it was denoted as 1 and vise versa.

- Example: If Blackgrass is detected, the array will be as follows.

[1,0,0,0,0,0,0,0,0,0,0,0]

Strategy to prevent overfitting

To prevent the overfitting we have created a function that randomly changes the image characteristics during fitting. This is called data augmentation in machine learning. We have created this data generator such that it will randomly rotate, zoom, shift and crop images and make augmented images.

```
datagen = ImageDataGenerator(
    zoom_range=0.2,                  # This will randomly zoom the image
    brightness_range=[0.7, 1.3],     # Randomly change the brightness
    horizontal_flip=True,            # This will randomly flip - horizontally
    vertical_flip=True,              # This will randomly flip - vertically
    rotation_range=180,              # Randomly rotate images 0 to 180
    width_shift_range=0.15,           # Randomly shift the images - horizontally
    height_shift_range=0.15           # Randomly shift the images - vertically
)
datagen.fit(trainX)
```

Model Compile

Since this is a multi class classification, we have chosen **categorical_crossentropy** as the loss function and **adam algorithm** as the optimizer. Initial learning rate was set to **0.001** and momentum to 0.9 inorder to skip the local minimas while training.

```
# Set Optimizer
opt = tf.keras.optimizers.Adam(lr=0.001, momentum=0.9)
# Model compile
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

Model Training

KFold cross validation was used to train the models. Initially no. of splits (n_splits or K) was set to 5. The value 5 is a standard value and the training time will show an exponential increase when increasing this value.

```
from sklearn.model_selection import KFold
cv = KFold(n_splits=5, random_state=42, shuffle=False)
k_fold_count = 0
for train_index, test_index in cv.split(X):
    k_fold_count += 1
    print("K FOLD : {}".format(k_fold_count))
    trainX, testX, trainY, testY = X[train_index], X[test_index], TrainLabel[train_index], TrainLabel[
        test_index]
```

Learning Rate Reduction

We intend to reduce the learning rate as the convergence is faster. If the validation accuracy didn't increase for a set no. of epochs (**patience=4**), then the learning rate is reduced by a factor of **0.5.**

```
# learning rate reduction parameters
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=4, verbose=1,
                                            factor=0.5, min_lr=0.00001)
```

Model Saving

Only the best models were saved using the ModelCheckpoint file. This will check the validation accuracy and save the model only if there is a increase in validation accuracy.

```
filepath2 = "output/weights_best_fold-" + str(k_fold_count) + "_{epoch:02d}-val{val_accuracy:.2f}.hdf5"
checkpoint_all = ModelCheckpoint(filepath2, monitor='val_accuracy',
                                 verbose=1, save_best_only=True, mode='max')
```

Model Evaluation

Confusion matrix was used to evaluate the model in the end. Using this method it is easier to check whether there is a class biased towards another class. If so then we can change the preprocessing/hyperparameters accordingly.

```python
def get_confusion_matrix(self, best_model_path):
    # Load data
    X, y = self.pre_pro.load_data()

    # Encode labels
    encode_labels = self.label_encoder.transform(y)

    # Make labels categorical
    categorical_labels = np_utils.to_categorical(encode_labels)

    # Load model
    model = self.get_model()
    model.load_weights(best_model_path)

    # PREDICTIONS
    y_predictions = model.predict(X)
    y_class = np.argmax(y_predictions, axis=1)
    y_check = np.argmax(categorical_labels, axis=1)

    # Confusion matrix
    conf_matrix = confusion_matrix(y_check, y_class)
    print(conf_matrix)
```

QUESTION1

Defining our Model

Our CNN model consisted of 4 convolution layers and 3 connected layers.

- **BatchNormalization -** used to speed up the learning by normalizing the weights of output of the previous layer.
- **Conv2D -** used to extract localized image features
- **DropOut -** used to randomly update dense layer weights to prevent overfitting
- **MaxPooling2D -** used to reduce the output resolution of the convolution layer.

Metric function is used to evaluate the performance of our model and the compile() method takes a metrics argument which is a list of metrics; optimizer and loss.

```python
class MyModel(Model):
    def __init__(self):
        pass

    @staticmethod
    def get_model(verbose=1):
        model = Sequential()

        model.add(Conv2D(filters=64, kernel_size=(5, 5), input_shape=(80, 80, 3), activation='relu'))
        model.add(BatchNormalization(axis=3))
        model.add(Conv2D(filters=64, kernel_size=(5, 5), activation='relu'))
        model.add(MaxPooling2D((2, 2)))
        model.add(BatchNormalization(axis=3))
        model.add(Dropout(0.1))

        model.add(Conv2D(filters=128, kernel_size=(5, 5), activation='relu'))
        model.add(BatchNormalization(axis=3))
        model.add(Conv2D(filters=128, kernel_size=(5, 5), activation='relu'))
        model.add(MaxPooling2D((2, 2)))
        model.add(BatchNormalization(axis=3))
        model.add(Dropout(0.1))

        model.add(Conv2D(filters=256, kernel_size=(5, 5), activation='relu'))
        model.add(BatchNormalization(axis=3))
        model.add(Conv2D(filters=256, kernel_size=(5, 5), activation='relu'))
        model.add(MaxPooling2D((2, 2)))
        model.add(BatchNormalization(axis=3))
        model.add(Dropout(0.1))

        model.add(Flatten())
```

The summary of the model is as follows.

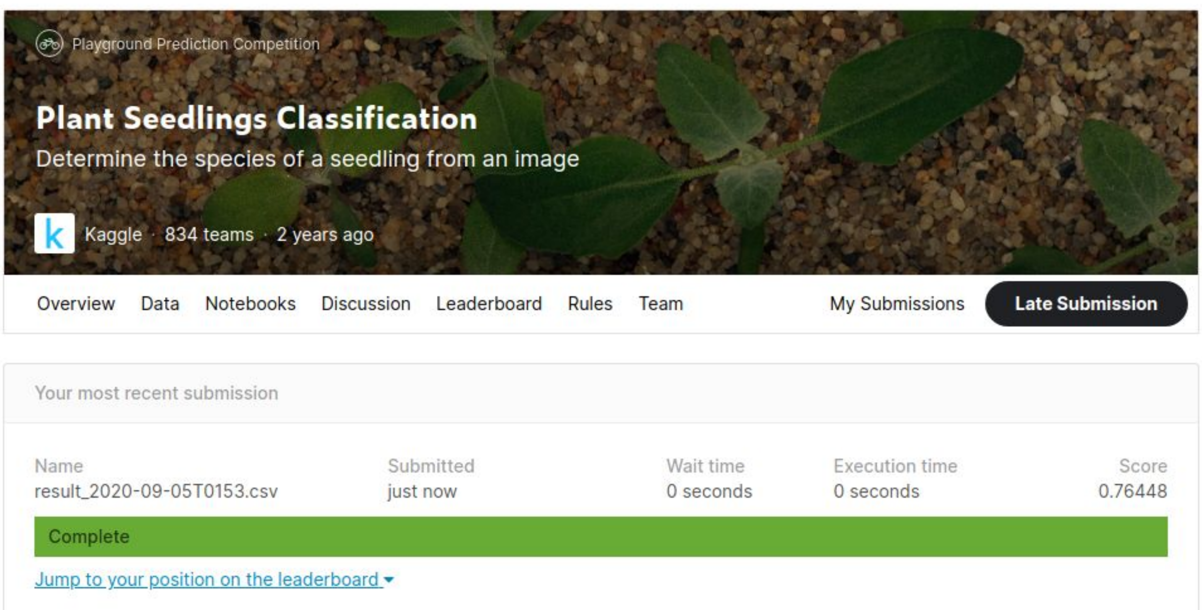
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 76, 76, 64)        4864      
_________________________________________________________________
batch_normalization (BatchNo (None, 76, 76, 64)        256       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        102464    
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 36, 36, 64)        0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 36, 36, 64)        256       
_________________________________________________________________
dropout (Dropout)            (None, 36, 36, 64)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 128)       204928    
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 128)       512       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 128)       409728    
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 128)       0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 14, 14, 128)       512       
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 128)       0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 256)       819456    
_________________________________________________________________
batch_normalization_4 (Batch (None, 10, 10, 256)       1024      
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 6, 256)         1638656   
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 256)         0         
_________________________________________________________________
batch_normalization_5 (Batch (None, 3, 3, 256)         1024      
_________________________________________________________________
dropout_2 (Dropout)          (None, 3, 3, 256)         0         
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0         
_________________________________________________________________
dense (Dense)                (None, 256)               590080    
_________________________________________________________________
batch_normalization_6 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792     
_________________________________________________________________
batch_normalization_7 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 12)                3084      
=================================================================
Total params: 3,844,684
Trainable params: 3,841,868
Non-trainable params: 2,816
_________________________________________________________________
```

Different optimizers vs Test accuracy

Optimizer	Test Accuracy
SGD	85.62
Adam	82.34
RMSprop	72.21
Adagrad	70.96

Kaggle Submission for Our Model

*** We received an accuracy of 0.76448 for this kaggle submission.



Confusion Matrix

The confusion matrix is used to analyze the errors in the model and for evaluation purposes. This diagram depicts the confusion matrix we obtained for 12 classes and it is evident that the first class is biased towards 7th class.

```
[[ 0  0  0  0  9  0 91  0  2  0  0  0]
 [ 0 93  1  0  0  3  0  0  5  0  0  0]
 [ 0 16 69  0  7  1  0  2  7  0  0  0]
 [ 0  0  0 94  0  0  1  1  3  3  0  0]
 [ 0  0  1  1 84  0  3  0 13  0  0  0]
 [ 0  1  4  3  1 87  2  0  1  2  0  1]
 [ 0  0  0  0  1  0 99  0  2  0  0  0]
 [ 0  1  0  1  0  0  3 65 32  0  0  0]
 [ 0  2  2  1  1  0 16 14 65  0  0  1]
 [ 0  4  2  9  0  2  0  4 28 49  2  2]
 [ 0 14  0  1  0  0  0  0  3  4 80  0]
 [ 0  0  1  0  0  8  2  1 23  0 10 57]]
```

This might be the reason for the reduced accuracy. This should be further tuned and trained to get a better accuracy. For uptodate information please refer to theGr github readme page.

https://github.com/DulanGit/Plant-Seedling-Classification/blob/master/README.md

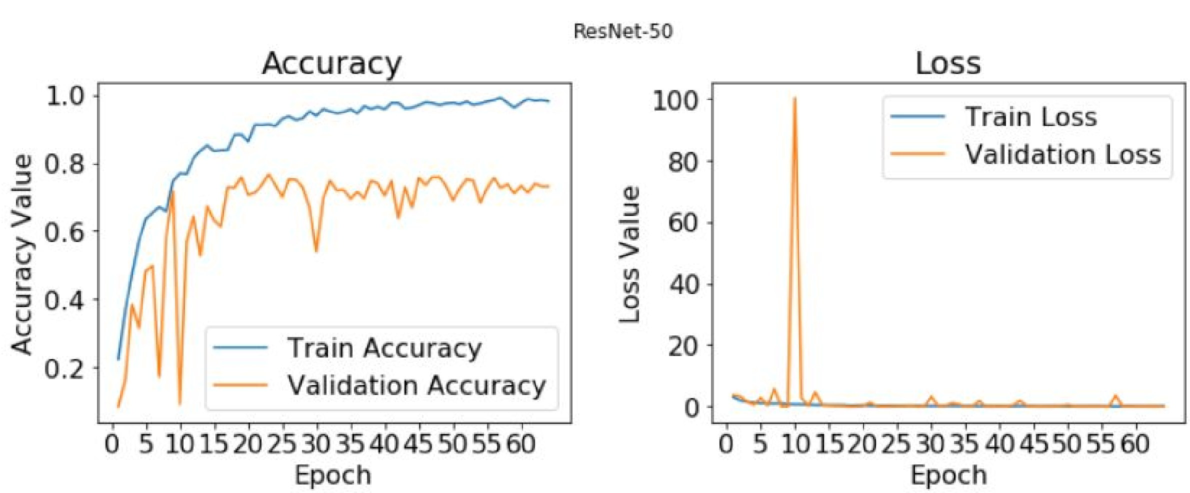
QUESTION 2

CNN ResNet 50 Model

Two dense layers and one dropout layer was added after the Resnet50 base model. This was changed from the original 1000 classes classification to 12 classes. Imagenet weights were loaded as initial weights.

```
def get_model(verbose, type='resnet'):
    # Load base model
    if not type == 'resnet':
        model = tf.keras.applications.ResNet50(include_top=False, weights='imagenet', input_shape=(input_shape[0],
                                                                                                 input_shape[1], input_shape[2]))
    else:
        model = tf.keras.applications.VGG19(include_top=False, weights='imagenet', input_shape=(input_shape[0],
                                                                                              input_shape[1], input_shape[2]))
    input_layer = model.inputs
    x = model.layers[-1].output
    # Add Top model
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.3)(x)
    x = tf.keras.layers.Dense(12, activation='softmax')(x)
    model = tf.keras.Model(input_layer, x)
    if verbose == 1:
        print(model.summary())
```

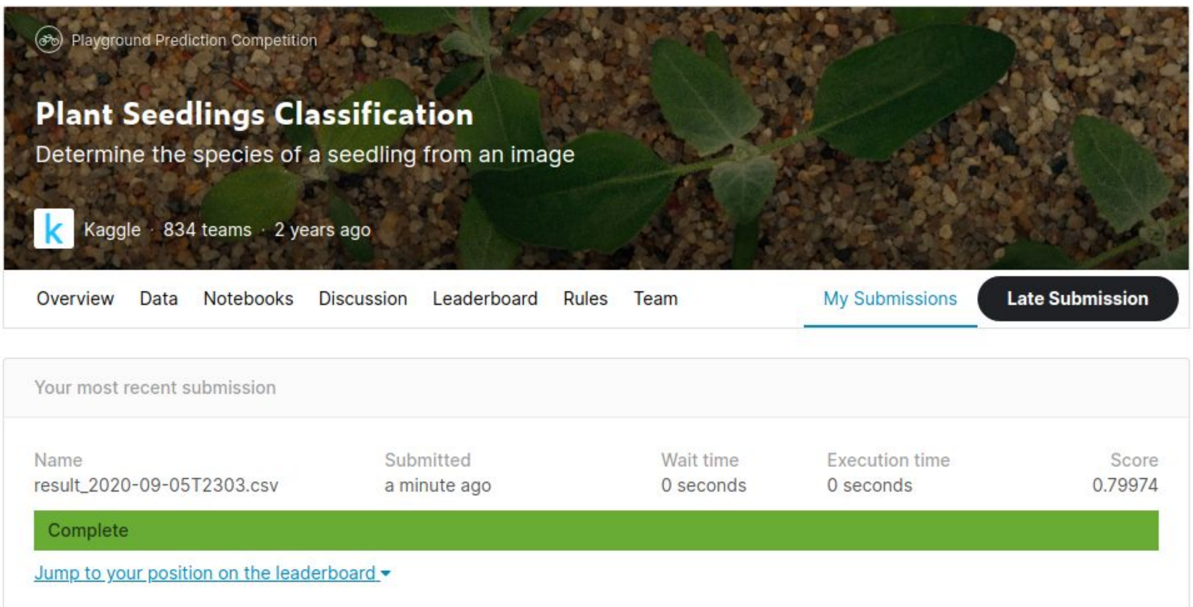
*** The first graph shows the accuracy value against the epoch for the relevant train and validation accuracies while the loss value plotted against the epoch for train and validation losses is shown by the second graph.



QUESTION3

Kaggle submission

The final submission was done by using the **Transfer Learn Model** as we received a higher accuracy than our model.



REFERENCES

https://techtutorialsx.com/2019/11/08/python-opencv-converting-image-to-hsv/

https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html

https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau

https://keras.io/api/layers/pooling_layers/max_pooling2d/

https://scikit-learn.org/stable/user_guide.html

https://machinelearningmastery.com/crash-course-convolutional-neural-networks/

https://www.tensorflow.org/api_docs/python