Ruslan Osmanov ID: 327480026

Beni Wolftson ID: 316535798

Import Dependencies

In [1]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report ,ConfusionMatrixDisplay ,confusion_matrix
from keras import layers
from tensorflow.keras.callbacks import TensorBoard
import pandas as pd
import os
import cv2
from tqdm import tqdm
from mpl_toolkits.axes_grid1 import ImageGrid
import random
import matplotlib.colors as mcolors
from numpy import expand_dims
from keras.preprocessing.image import ImageDataGenerator
```

Dinamic memory allocation Only if using gpu

In [2]:
```
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
config = ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)
```

In [3]:
```
print(tf.config.list_physical_devices('GPU'))
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Define classes and matching id for future use in training models

In [4]:
```
class_names= ["Black-grass", "Charlock", "Cleavers", "Common Chickweed", "Common wheat", "Fat Hen",
              "Loose Silky-bent", "Maize", "Scentless Mayweed", "Shepherds Purse", "Small-flowered Cranesbill",
              "Sugar beet"]
dataset = 'C:/Users/rusla/Documents/Data/plant-seedlings-classification/train'
#dataset = 'D:/My Documents/Jupiter NoteBook/Project 3/plant-seedlings-classification/train'
labels_id = {class_name : i for i,class_name in enumerate(class_names)}
```

In [5]:
```
labels_id
```

Out[5]:
```
{'Black-grass': 0,
 'Charlock': 1,
 'Cleavers': 2,
 'Common Chickweed': 3,
 'Common wheat': 4,
 'Fat Hen': 5,
 'Loose Silky-bent': 6,
 'Maize': 7,
 'Scentless Mayweed': 8,
 'Shepherds Purse': 9,
 'Small-flowered Cranesbill': 10,
 'Sugar beet': 11}
```

Load images from training folders ,resize them to smaller size for faster proccesing and rescaling to [0.0 ,1.0]

In [6]:
```
image_size = (128,128) # new dimansions
def load_data():
    images,labels = [],[]
    for folder in os.listdir(dataset): # run over 12 folders
        label = labels_id[folder]
        #load images from each folder with progress bar
        for file in tqdm(os.listdir(os.path.join(dataset,folder)),desc=folder.ljust(25, ' '),bar_format="{l_bar}{bar:20}{r_ba

            img_path = os.path.join(os.path.join(dataset,folder),file)
            img = cv2.imread(img_path) #read images as BGR
            img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB) #convert images to RGB
            img = cv2.resize(img,image_size) #resize images

            images.append(img) # add image to list
            labels.append(label) # add label to list

    images = np.array(images,dtype=np.float32)/255. # convert list to array and normalize
    labels = np.array(labels,dtype=np.uint8) # convert list to vector
    return (images,labels)
```

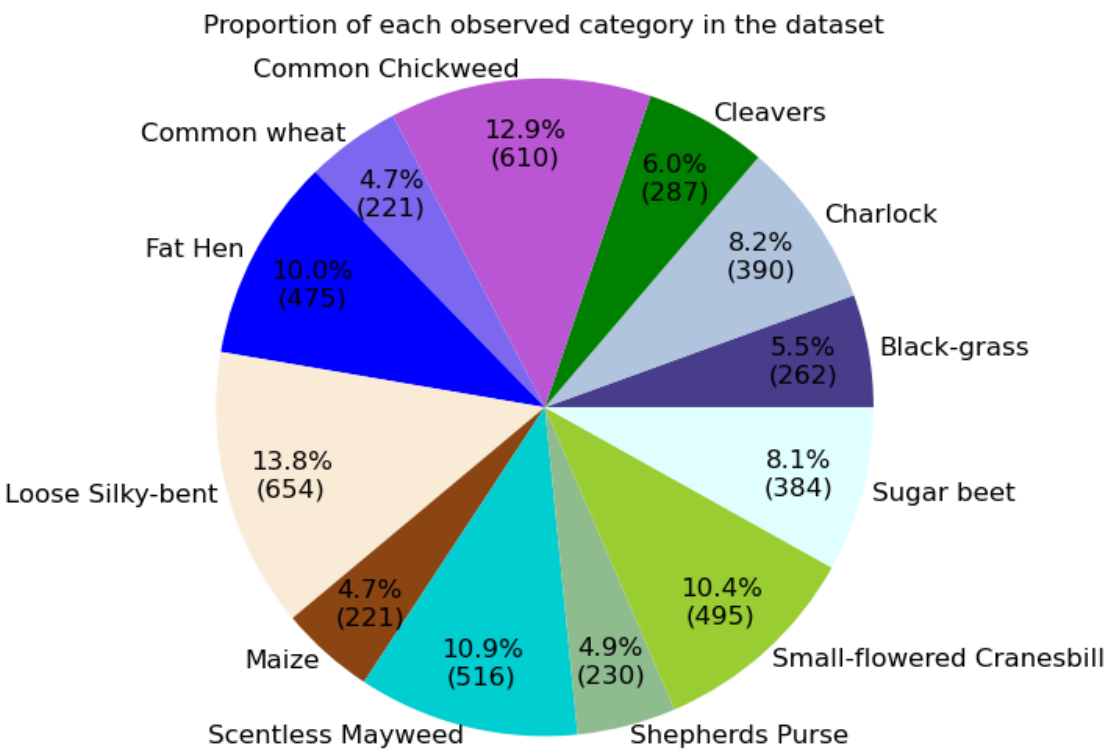
In [7]:
```
def evaluation_plots(classifier,y_lim =[0,2]):
    # function for plot training progress (loss and accuracy over epochs)
    losses = pd.DataFrame(classifier.history.history)
    fig,axs =plt.subplots(1,2,figsize=(12,4))
    axs[0].plot(losses[['loss','val_loss']],label=['Train_loss' ,'Val_loss'])
    axs[1].plot(losses[['accuracy','val_accuracy']],label=['Train_accuracy' ,'Val_accuracy'])
    title = ['loss','accuracy']
    for i in range(2):
        axs[i].grid(visible=True ,which='minor',linestyle='--',alpha=0.3)
        axs[i].minorticks_on()
        axs[i].grid()
        axs[i].set_xlabel('epocs')
        axs[i].set_title(title[i])
        axs[i].legend(loc='upper left')
    axs[0].set_ylim(y_lim)
    plt.show()

class estimator:
    # for purpose of using  metrics from sklern package (confusion matrix )
    _estimator_type = ''
    classes_=[]
    def __init__(self, model, classes):
        self.model = model
        self._estimator_type = 'classifier'
        self.classes_ = classes
    def predict(self, X):
        y_prob= self.model.predict(X) # prediction
        y_pred = y_prob.argmax(axis=1) # take the index of maximum  predicted probability
        return y_pred
```

In [8]:
```
X,y = load_data() # run load data function with progress bar
```

```
Black-grass               : 100%|████████████████████| 263/263 [00:02<00:00, 99.30it/s]
Charlock                  : 100%|████████████████████| 390/390 [00:01<00:00, 249.61it/s]
Cleavers                  : 100%|████████████████████| 287/287 [00:00<00:00, 536.97it/s]
Common Chickweed          : 100%|████████████████████| 611/611 [00:00<00:00, 770.76it/s]
Common wheat              : 100%|████████████████████| 221/221 [00:00<00:00, 234.64it/s]
Fat Hen                   : 100%|████████████████████| 475/475 [00:01<00:00, 466.18it/s]
Loose Silky-bent          : 100%|████████████████████| 654/654 [00:03<00:00, 202.23it/s]
Maize                     : 100%|████████████████████| 221/221 [00:01<00:00, 167.91it/s]
Scentless Mayweed         : 100%|████████████████████| 516/516 [00:00<00:00, 637.24it/s]
Shepherds Purse           : 100%|████████████████████| 231/231 [00:00<00:00, 437.10it/s]
Small-flowered Cranesbill: 100%|████████████████████| 496/496 [00:01<00:00, 409.21it/s]
Sugar beet                : 100%|████████████████████| 385/385 [00:02<00:00, 143.79it/s]
```

Plot proportions of the training dataset it will be splited to train and validation

In [9]:
```
_,count = np.unique(y,return_counts=True) #count samples in each unique label
# Creating autocpt arguments
random.seed(3)
def func(pct, allvalues):
    absolute = int(pct / 100.*np.sum(allvalues))
    return "{:.1f}%\n({:d})".format(pct, absolute)

colors = random.choices(list(mcolors.CSS4_COLORS),k = 15)
plt.rcParams["figure.figsize"] = (6,6)

plt.pie(count,
       labels = class_names,
       autopct = lambda pct: func(pct, count),
       textprops={'fontsize': 12},
       pctdistance=0.8, labeldistance=1.03, colors = colors)

plt.axis('equal')
plt.title("Proportion of each observed category in the dataset")
plt.show();
```



Display flower samples from the Dataset

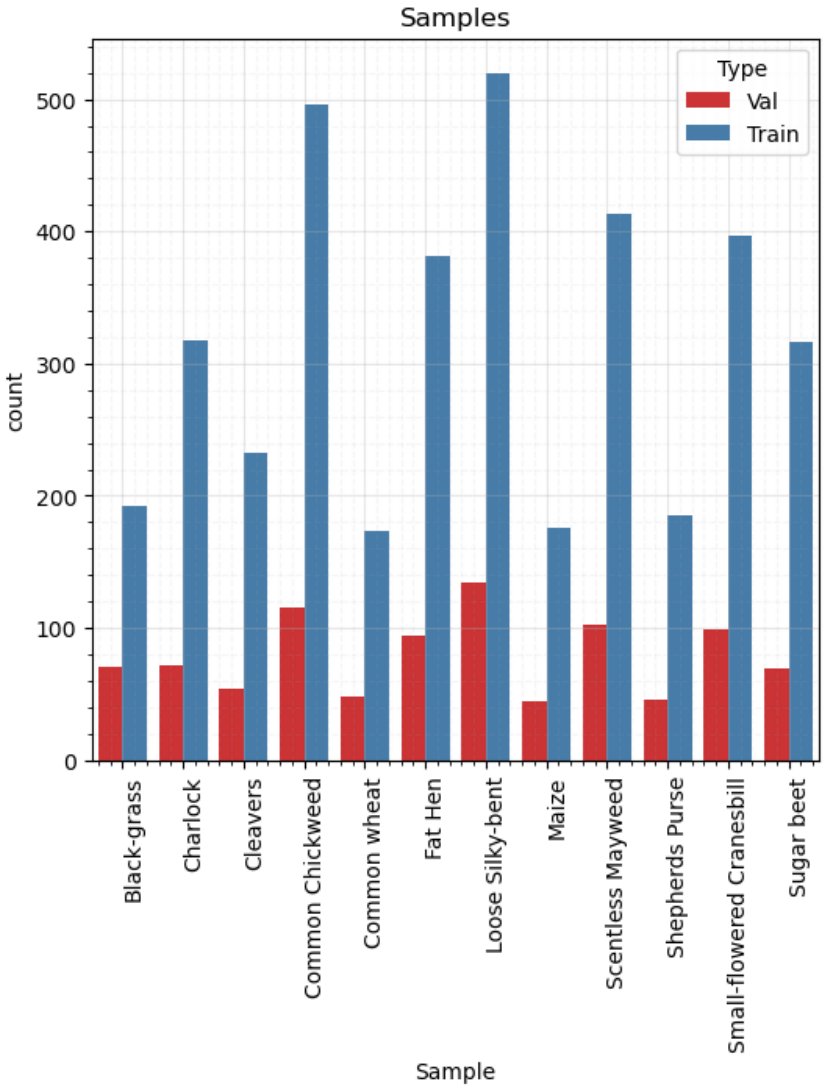
In [10]:
```
# plot first few images in the class
valueList, indicesList = np.unique(y, return_index=True)
fig, ax = plt.subplots(2,6,figsize=(12,5),sharex=True ,sharey=True)
for i,idx in enumerate(indicesList):
    if i <=5:
        ax[0,i].imshow(X[idx])
        ax[0,i].set_title(class_names[i])
        ax[0,i].axis('off')
    else:
        ax[1,i-6].imshow(X[idx])
        ax[1,i-6].set_title(class_names[i])
        ax[1,i-6].axis('off')
plt.tight_layout()
plt.show()
```

Split Dataset to two part for training and debugging

In [11]:
```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=101)
```

In [12]:
```
#plot datasets after splitting
test = pd.DataFrame({'Sample':y_val})
train = pd.DataFrame({'Sample':y_train})
test['Type']='Val'
train['Type']='Train'
df=pd.concat([test,train],axis=0,ignore_index=True)
sns.countplot(data = df,x='Sample',hue='Type',palette='Set1')
plt.xticks(ticks=range(12), labels=class_names, rotation=90, horizontalalignment='left')
plt.grid()
plt.grid(visible=True ,which='major',linestyle='-',alpha=0.3)
plt.grid(visible=True ,which='minor',linestyle='--',alpha=0.1)
plt.minorticks_on()
```

```
plt.title('Samples')
plt.show()
```



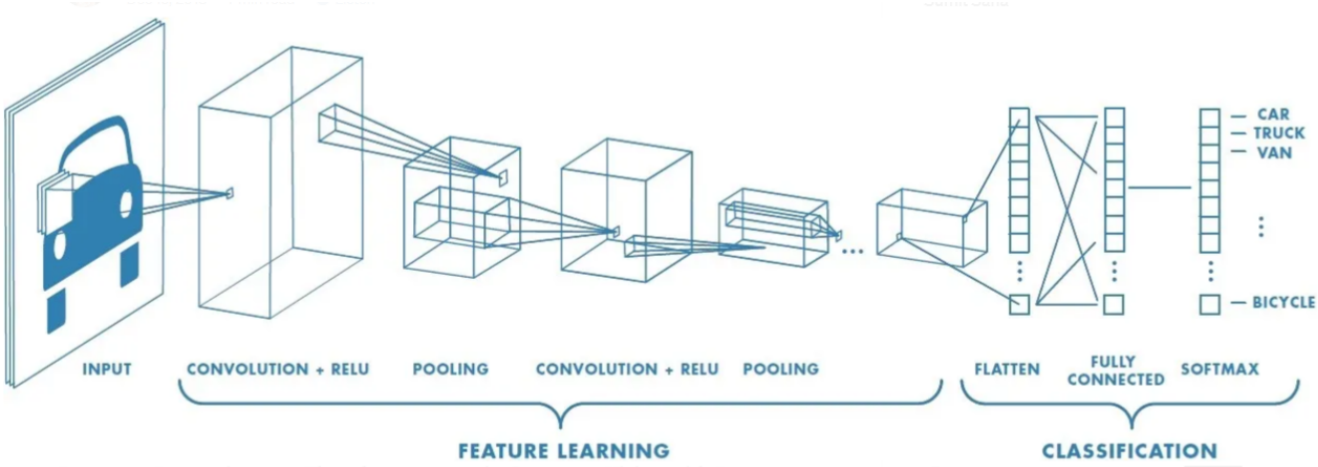
In [13]:
```
print('Training: {} / Val: {} /Total: {}'.format(X_train.shape[0],X_val.shape[0],np.sum([X_train.shape[0],X_val.shape[0]])))
```

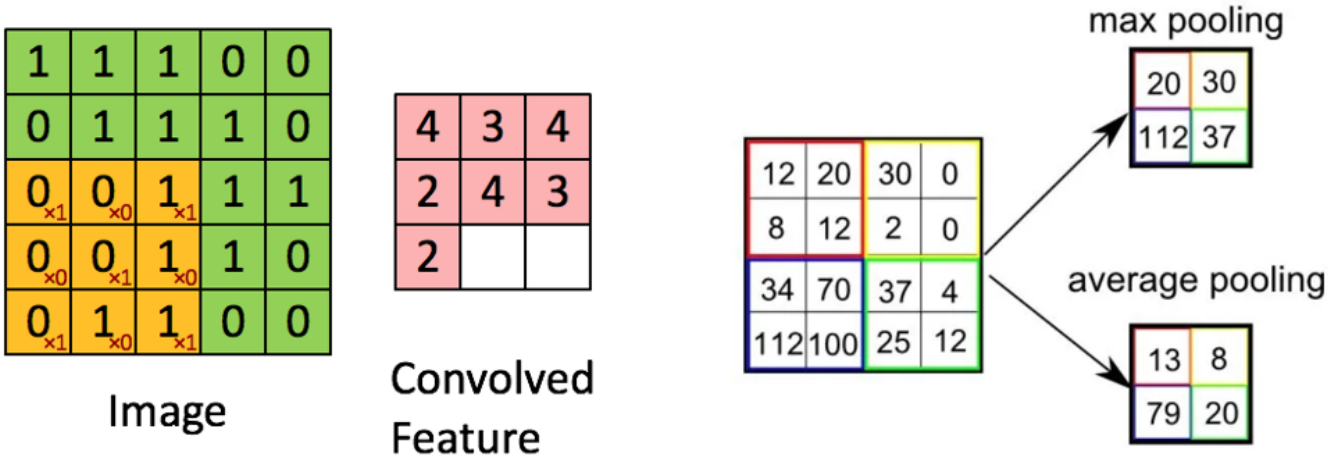
```
Training: 3800 / Val: 950 /Total: 4750
```

Training

Model with 4 layers of convolution and one fully conected layer

For example:





Activation Function - An activation function determines if a neuron should be activated or not activated. This implies that it will use some simple mathematical operations to determine if the neuron's input to the network is relevant or not relevant in the prediction process.

Softmax - An activation function that turns a vector of K real values into a vector of K real values that sum to 1, so that they can be interpreted as probabilities.

In [14]:
```
name='CNN-128-MP-64-MP-64-MP-64-MP-Flaten-FC128'
classifier = keras.Sequential(name=name)
classifier.add(layers.Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu',input_shape=(128,128,3)))
classifier.add(layers.MaxPool2D(pool_size=(2,2))) #Down Sampling
classifier.add(layers.Conv2D(64, kernel_size=(2, 2),padding='same', activation='relu')) # convolution with 64 filters with s
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(2, 2),padding='same', activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(2, 2), activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Flatten())
classifier.add(layers.Dense(128, activation='relu'))
classifier.add(layers.Dense(12, activation='softmax'))
```

In [15]:
```
log_dir = "C:/Users/rusla/Documents/logs/"+ name
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=False)
es_callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
```

Compile and Run the training

We are using **sparse_categorical_crossentropy** instead **categorical_crossentropy** because we didnt use one-hot encoding to target vectore

Adam optimization is an algorithm that used to update network weights iteratively based on training data instead of the traditional stochastic gradient descent method.Adam combines the benefits of two other stochastic gradient descent extensions and the Adaptive Gradient Algorithm (AdaGrad), which retains a learning speed per-parameter that improves performance . Root Mean Square Propagation (RMSProp) also preserves per-parameters learning rates adjusted to the weight based on the average of recent magnitudes.

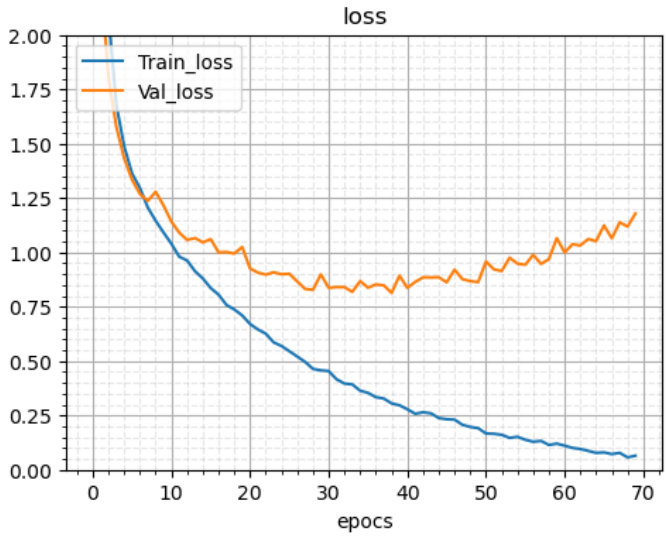
Batch size refers to the number of training examples that are used in one forward/backward pass of the neural network during the training process.

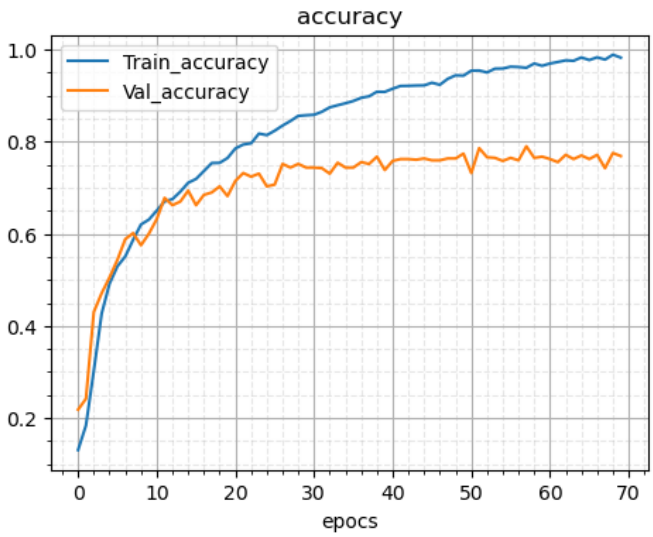
Epoch - One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE

In [ ]:
```
classifier.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.0001),
              metrics=['accuracy'])
history=classifier.fit(X_train, y_train, batch_size=32, epochs=70,validation_split=0.2,callbacks=[tb_callback])
```

Overfitting- occurs when a machine learning model is trained too well on the training data, and as a result, it does not generalize well to new, unseen data. In other words, the model has learned patterns in the training data that do not hold true for the broader population, and as a result, its predictions on new data are less accurate.

In [17]:
```
evaluation_plots(classifier)
```

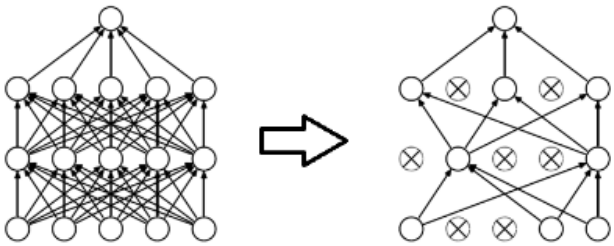




As we can see above ,there is an overfit ,because the model is trained too well on the training data, we could have stopped the model at epoch 30.

Model with 4 layers of convolution, one fully conected layer and dropout

Dropout - is a regularization technique to prevent overfitting. It works by randomly "dropping out" a certain percentage of the nodes in the network during training. This means that the output of these nodes is set to zero and they are not included in the forward or backward pass. The effect of dropout is to effectively train an ensemble of models, with each model seeing a different subset of the nodes in the network.



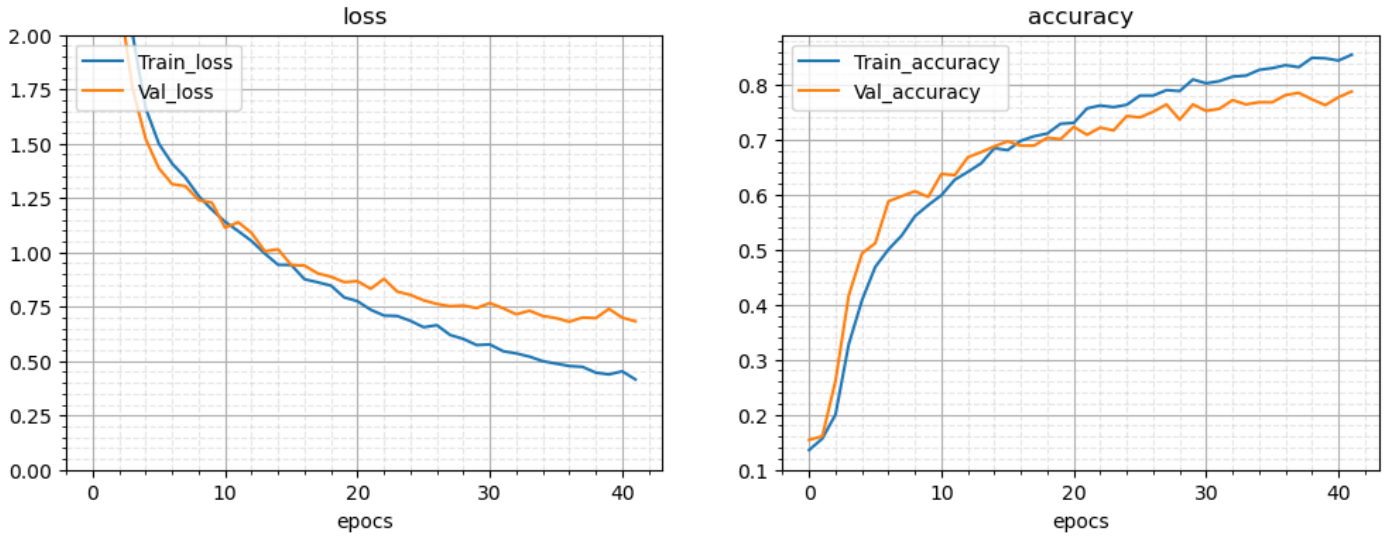
In [18]:
```
name = 'CNN-128-MP-64-MP-64-MP-64-MP-D0.4-Flaten-FC128'
classifier = keras.Sequential(name=name)
classifier.add(layers.Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu',input_shape=(128,128,3)))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(2, 2),padding='same', activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(2, 2),padding='same', activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(2, 2), activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Dropout(0.4)) # probability for dropping the neuron
classifier.add(layers.Flatten())
classifier.add(layers.Dense(128, activation='relu'))
classifier.add(layers.Dense(12, activation='softmax'))
```

In [19]:
```
log_dir = "C:/Users/rusla/Documents/logs/" + name
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=False)
es_callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
```

In [20]:
```
classifier.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.0001),
              metrics=['accuracy'])
```

In [ ]:
```
history=classifier.fit(X_train, y_train, batch_size=32, epochs=100,validation_split=0.2,callbacks=[es_callback,tb_callback])
```

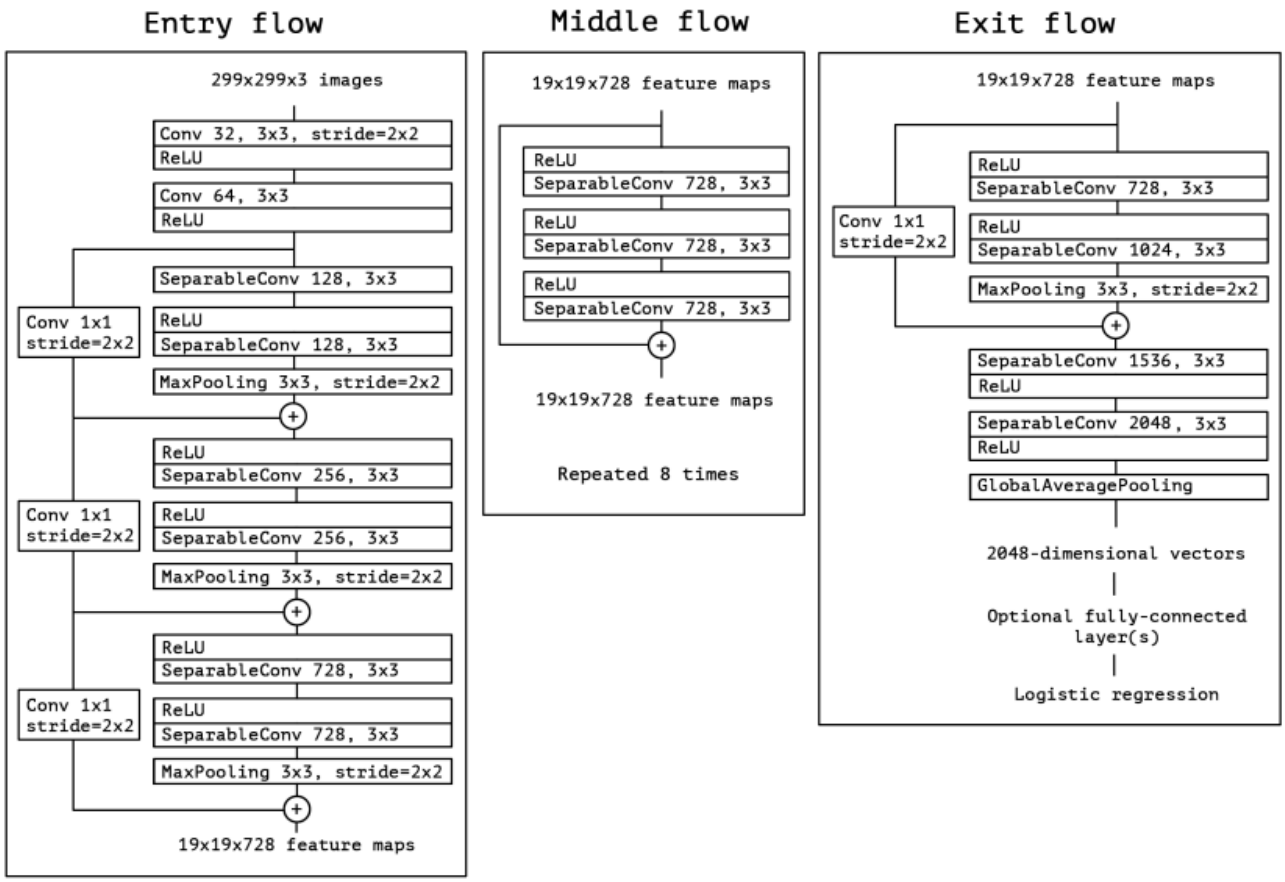
In [22]:
```
evaluation_plots(classifier)
```



Now we used a dropout as a regularization technique to prevent overfitting ,it can be seen from the plot that a gap between training and validation losses are close to each other, and in addition we used early stopping technique that involves interrupting training when the model's performance not improving

Transfer Learning

- Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. It can greatly save on time and resources, as well as improve the performance of the model on the second task.
- There are a few different ways to use transfer learning in deep learning. One approach is to use the weights from a pre-trained model as the starting point for training a new model on a second task. Another approach is to fine-tune a pre-trained model on the second task by continuing to train it with new data.
- Transfer learning is often used in practice because it can be difficult and time-consuming to train a deep learning model from scratch on a large dataset. By using a pre-trained model as a starting point, it is possible to train a good model much more quickly, and with less data.
- In our aproach we will use Xception pretrained models and only re-train the last and first layers



Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6

In [ ]:
```
## Import pretrained model without fullyconected layer and use Average pooling
base_model=keras.applications.Xception(input_shape=(128, 128, 3),include_top=False,weights='imagenet',pooling='avg')
for layer in base_model.layers:
    layer.trainable = False
for layer in base_model.layers[-5:]: # train only last 5 layers
     layer.trainable = True
print(base_model.summary())
```

In [99]:
```
#Build model and added our fully conected layers with dropouts
model = keras.Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dropout(0.4))
model.add(layers.Dense(12, activation='softmax'))
model.summary()
```

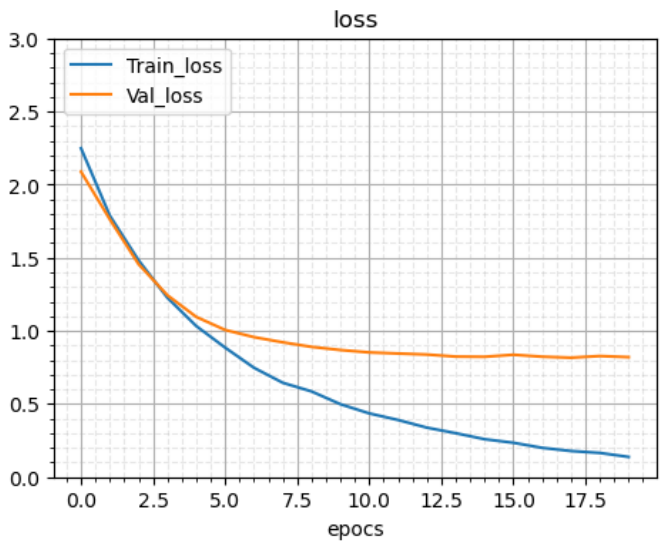
```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 2048)              20861480

 flatten_7 (Flatten)         (None, 2048)              0

 dense_14 (Dense)            (None, 256)               524544

 dropout_2 (Dropout)         (None, 256)               0

 dense_15 (Dense)            (None, 12)                3084

=================================================================
Total params: 21,389,108
Trainable params: 3,691,276
Non-trainable params: 17,697,832
_________________________________________________________________
```

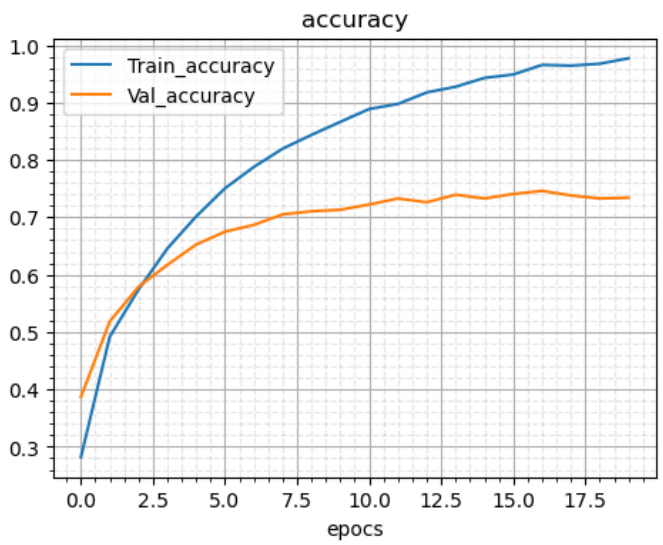
In [100…
```
log_dir = "C:/Users/rusla/Documents/logs/Xception_"+'FC128_D0.4'
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=False)
```

In [101…
```
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.0001),
              metrics=['accuracy'])
```

In [ ]:
```
history=model.fit(X_train, y_train, batch_size=64, epochs=20,validation_split=0.2,callbacks=[tb_callback])
```

In [103…
```
evaluation_plots(model,[0,3])
```





As wee can see hear there is gap between training and validation (loss/accuracy) and validation accuracy low. This problem occures because the training dataset is very small ,to overcome this issue we will be using data augmentation technique to add more data - different configurations of same picture.

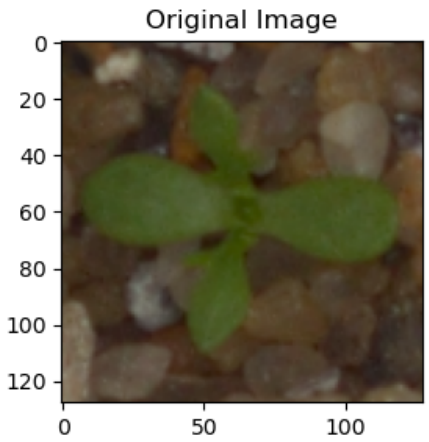
Data Agmentation

- Image augmentation is a technique used to artificially increase the size of a training dataset by creating modified versions of images in the dataset. This is done by applying a set of predefined augmentation techniques such as rotation, cropping, scaling, and flipping to the original images. The goal of image augmentation is to introduce variations in the training dataset so that the model can learn to recognize and classify images better, even if they are slightly different from the images in the original dataset. This can help to reduce overfitting, which occurs when a model performs well on the training data but poorly on new, unseen data.

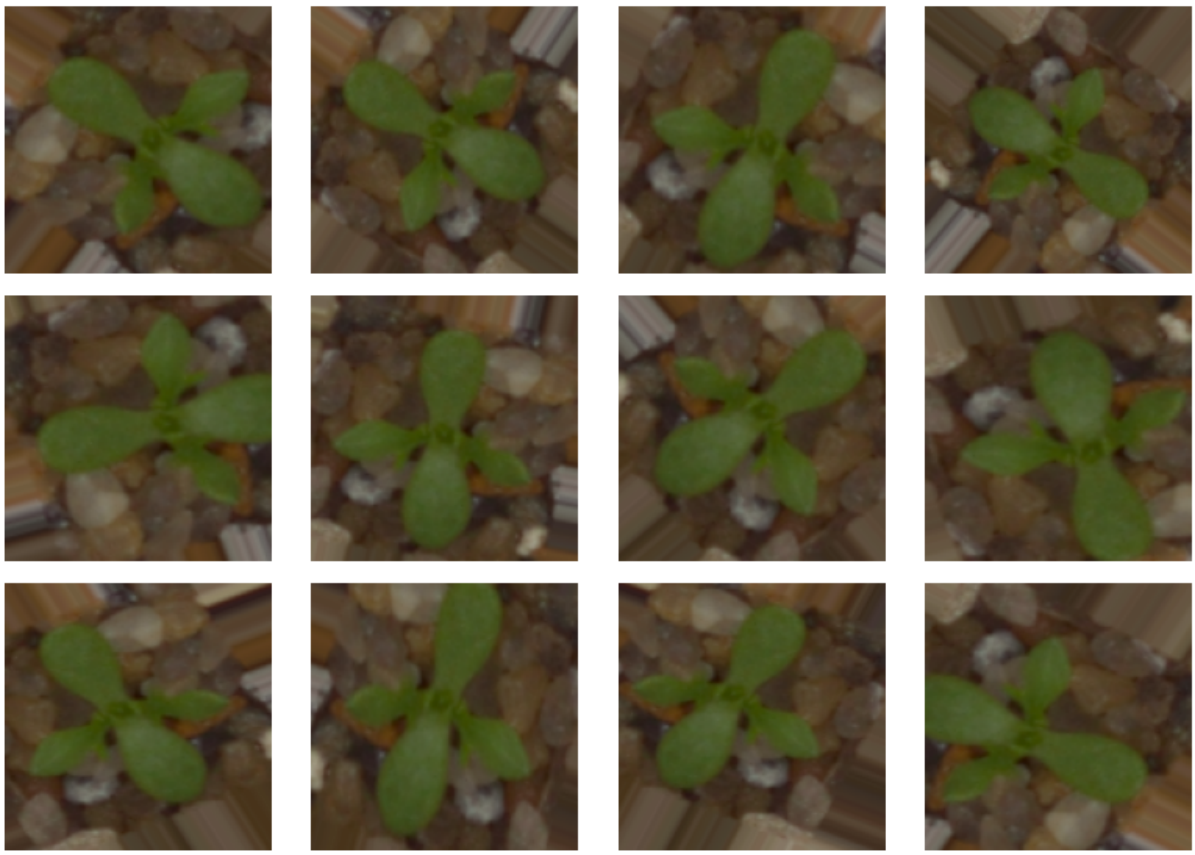
In [105…
```
# Initialize Augmentation generator
datagen = ImageDataGenerator(
        rotation_range= 160,  # randomly rotate images in the range
        zoom_range=0.2,
        width_shift_range=0.1,  # randomly shift images horizontally
        height_shift_range=0.1,  # randomly shift images vertically
        shear_range=0.2,
        vertical_flip=True,  # randomly flip images vertically
        horizontal_flip=True,  # randomly flip images horizontally


        validation_split=0.2,
    )
```

Example of Augmentation generator output

In [106…
```
plt.figure(figsize=(3,3))
plt.imshow(X_train[273])
plt.title('Original Image')
plt.show()
```



In [107…
```
rows = 3
columns = 4
# convert into a 4-D array of 1 element of 3D array representing
# prepare the iterator; flow() takes in a 4D array and returns an iterator containing a batch of images
aug_iter = datagen.flow(expand_dims(X_train[273], 0), batch_size=1)
fig, axes = plt.subplots(rows,columns)
for r in range(rows):
    for c in range(columns):
        image = aug_iter.next()[0]
        axes[r,c].imshow(image.astype(float))
        axes[r,c].axis("off")
fig.set_size_inches(8,6)
plt.suptitle('Augmented Images')
plt.tight_layout()
```



In [ ]:
```
base_model=keras.applications.Xception(input_shape=(128, 128, 3),include_top=False,weights='imagenet',pooling='avg')
for layer in base_model.layers:
    layer.trainable = False
for layer in base_model.layers[-50:]: # ReTrain the last 50 layers
     layer.trainable = True
for layer in base_model.layers[:11]: # ReTrain the first 10 layers
     layer.trainable = True
print(base_model.summary())
print(len(base_model.layers))
```

In [109…
```
model = keras.Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(12, activation='softmax'))
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 2048)              20861480

 flatten_8 (Flatten)         (None, 2048)              0

 dense_16 (Dense)            (None, 256)               524544

 dense_17 (Dense)            (None, 12)                3084

=================================================================
Total params: 21,389,108
Trainable params: 12,741,980
Non-trainable params: 8,647,128
_________________________________________________________________
```

In [110…
```
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.00003),
              metrics=['accuracy'])
log_dir = "C:/Users/rusla/Documents/logs/Xception_"+'FC128_Aug'
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=False)
es_callback = keras.callbacks.EarlyStopping(monitor='val_loss',patience=8)
```

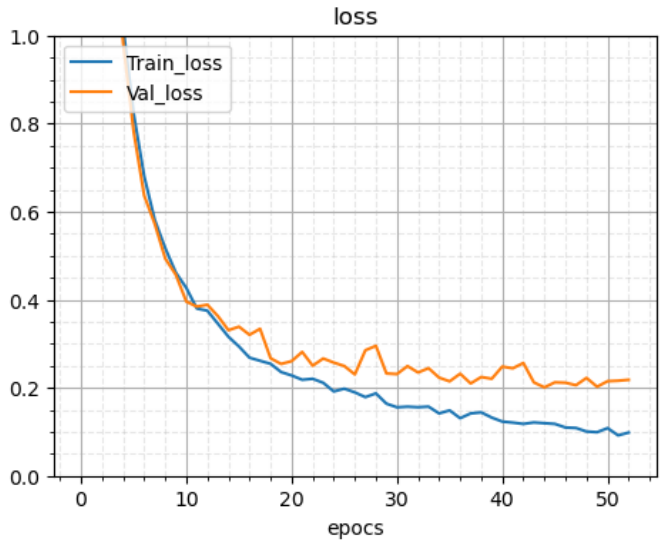
In [111…
```
train_generator=datagen.flow(X_train, y_train, batch_size=64,subset='training')
val_generator =datagen.flow(X_train, y_train,batch_size=32, subset='validation')
step_size_train = train_generator.n // train_generator.batch_size
step_size_val = val_generator.n // val_generator.batch_size
```

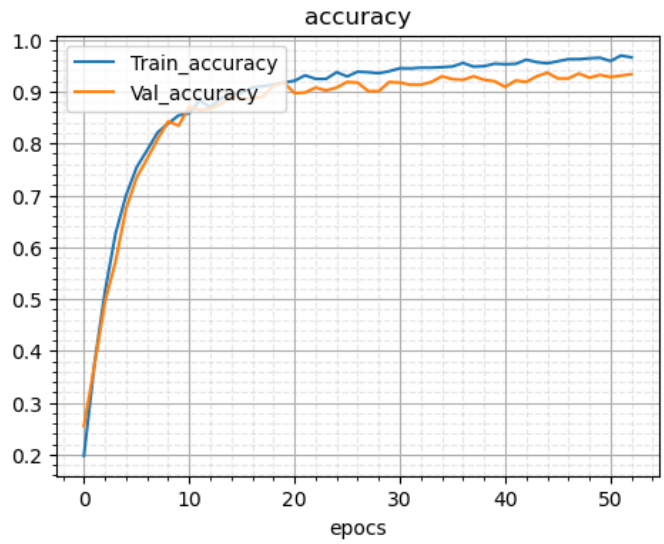
- In the code block above we created augmentation generators for train and validation ,each image pass to the generator and return a batch of 32 augmented images.
- To prevent repetitve training and validation we config steps_per_epoch and validation_steps according to next equation:

$$\text{number of images (train/val) / batch size}$$

In [ ]:
```
history=model.fit(train_generator,steps_per_epoch=step_size_train,
                  validation_data=val_generator,validation_steps=step_size_val, epochs=100 ,callbacks = [tb_callback,es_call
```

In [115…
```
evaluation_plots(model,[0,1])
```





From the plots above can be seen we have high accuracy than previous options.

Evaluate model on Validation Dataset

In [116…
```
est = estimator(model, class_names)
predicted_labels = est.predict(X_val)
```

```
30/30 [==============================] - 1s 31ms/step
```

Classification Report

It is one of the performance evaluation metrics of a classification-based machine learning model. It displays your model's precision, recall, F1 score and support. It provides a better understanding of the overall performance of our trained model. To understand the classification report of a machine learning model, you need to know all of the metrics displayed in the report.

- Precision is defined as the ratio of true positives to the sum of true and false positives.
- Recall is defined as the ratio of true positives to the sum of true positives and false negatives.
- The F1 is the weighted mean of precision and recall. The closer the value of the F1 score is to 1.0, the better the expected performance of the model is.
- Support is the number of actual occurrences of the class in the dataset. It doesn't vary between models, it just diagnoses the performance evaluation process.

In [117…
```
print(classification_report(y_val ,predicted_labels ,digits=4,target_names=class_names))
```

```
                           precision    recall  f1-score   support

              Black-grass     0.7826    0.5070    0.6154        71
                  Charlock     0.9863    1.0000    0.9931        72
                  Cleavers     0.9804    0.9259    0.9524        54
          Common Chickweed     0.9417    0.9826    0.9617       115
              Common wheat     0.9375    0.9375    0.9375        48
                   Fat Hen     0.9579    0.9681    0.9630        94
          Loose Silky-bent     0.7834    0.9179    0.8454       134
                     Maize     0.9375    1.0000    0.9677        45
        Scentless Mayweed      0.9417    0.9417    0.9417       103
          Shepherds Purse      0.9286    0.8478    0.8864        46
Small-flowered Cranesbill     0.9899    0.9899    0.9899        99
               Sugar beet     0.9412    0.9275    0.9343        69

                 accuracy                         0.9189       950
                macro avg     0.9257    0.9122    0.9157       950
             weighted avg     0.9186    0.9189    0.9156       950
```

- As can be seen from the Classification report above,The data is unbalanced (diffirent support samples) ,an avarege accuracy about 0.91 however precision of each class varies from 0.99 to 0.5 ,this phenomenom can also be seen in recall and f1-score

Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification algorithm. It helps you to understand how your classification model is performing by comparing the predicted classes and the actual classes in the data set.

- Expected down the side: Each row of the matrix corresponds to a predicted class.
- Predicted across the top: Each column of the matrix corresponds to an actual class.

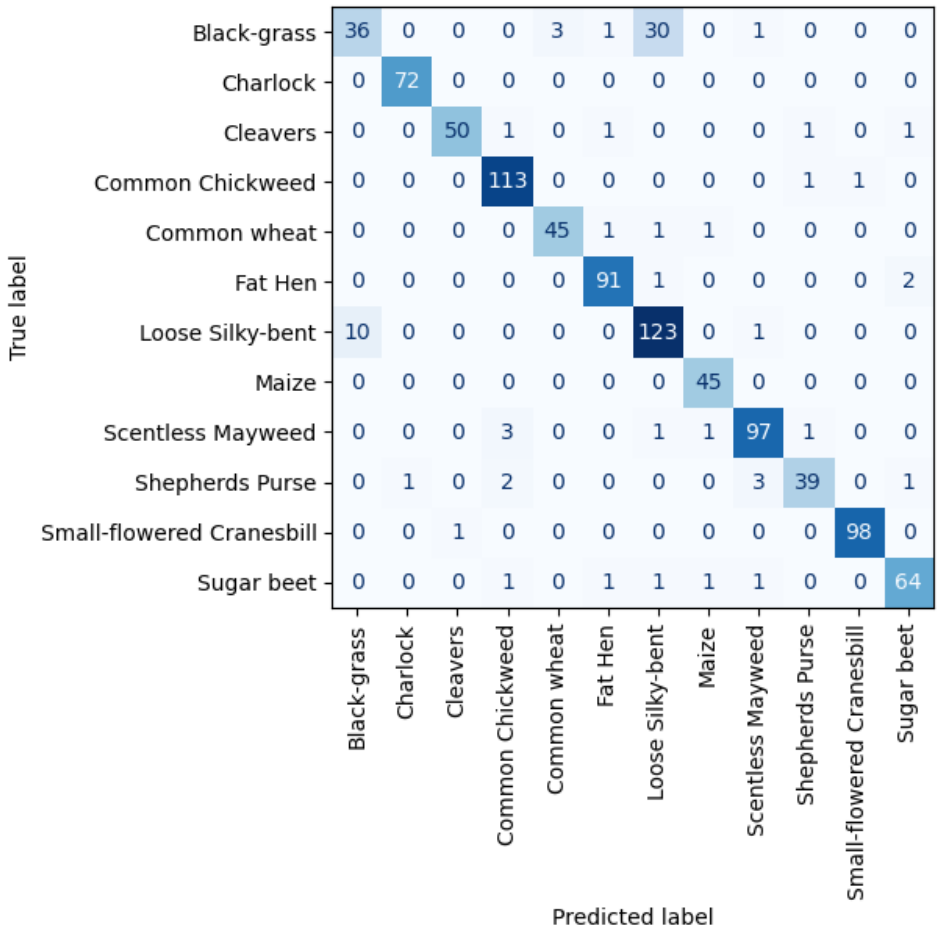
The counts of correct and incorrect classification are then filled into the table.

The total number of correct predictions for a class go into the expected row for that class value and the predicted column for that class value.

In the same way, the total number of incorrect predictions for a class go into the expected row for that class value and the predicted column for that class value.

In [118…
```
fig, ax = plt.subplots(figsize=(5, 5))
ConfusionMatrixDisplay.from_estimator(est,X_val,y_val,display_labels=class_names,cmap='Blues',colorbar=False,xticks_rotation=
plt.show()
```

```
30/30 [==============================] - 1s 32ms/step
```



- From the Confusion matrix below it can be seen that a lot of samples of Black Grass class predicted as Loose Silky-bent

In [119…
```
def plot_image(i, predictions_array, true_label, img,ax):
    # plot image and prediction label
    true_label, img = int(true_label[i]), img[i]
    ax.set_xticks([])
    ax.set_yticks([])
    ax.imshow(img)

    predicted_label = np.argmax(predictions_array) # id of predicted class
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'

    ax.set_xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)
```

```
def plot_value_array(i, predictions_array, true_label,ax):
    # plot probability of prediction according to each class
    true_label = int(true_label[i])
    ax.set_xticks(range(12))
    ax.grid(axis="y")
    thisplot = ax.bar(range(12), predictions_array, color="#777777")
    ax.set_ylim([0, 1])
    predicted_label = np.argmax(predictions_array)
    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')
    if predicted_label == true_label:
         color = 'blue'
    else:
         color = 'red'
    ax.text(0.35, 0.9, 'True :{}'.format(class_names[true_label]), dict(size=10,c='b'))
    ax.text(0.35, 0.8, 'Predicted :{}'.format(class_names[predicted_label]), dict(size=10,c=color))
```

In [120…
```
predictions = model.predict(X_val) #run prediction on validation Dataset
```

```
30/30 [==============================] - 1s 34ms/step
```

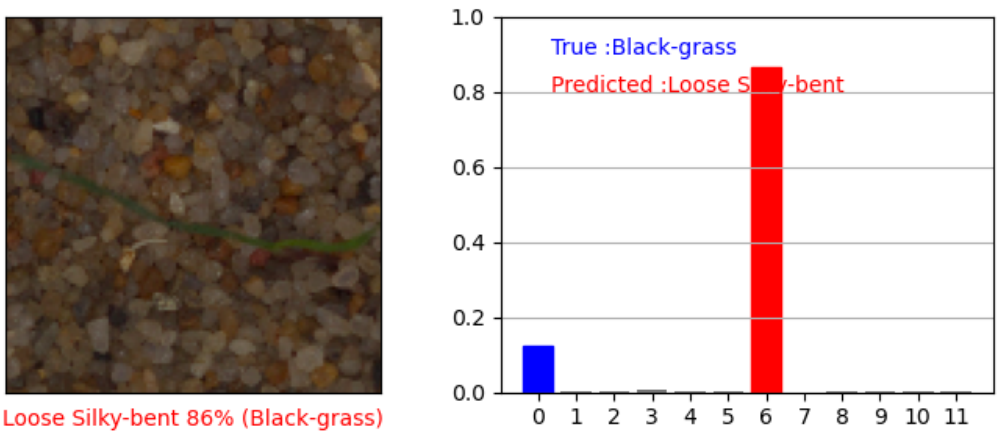
In [121…
```
np.unique(y_val,return_index=True) # find first idx of each unique class
```

Out[121]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11], dtype=uint8),
 array([33,  1, 22,  2, 16,  0,  7, 25, 24, 17,  3, 37], dtype=int64))
```

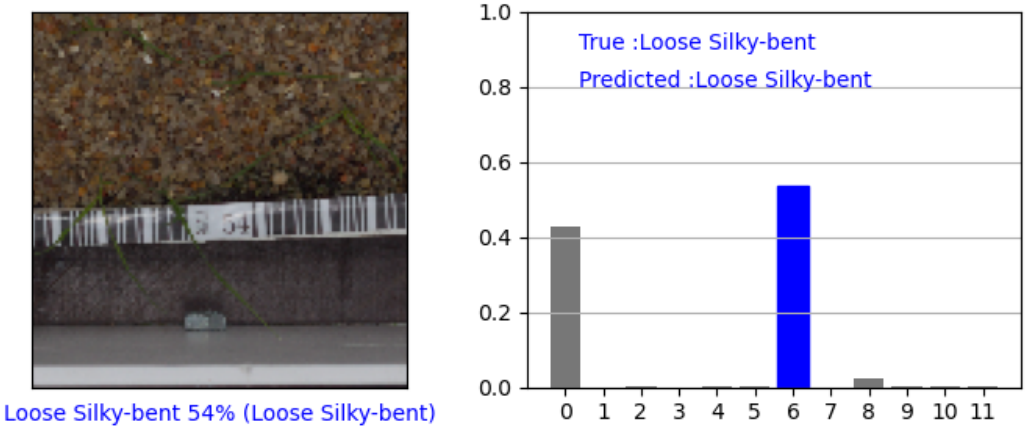
In [122…
```
np.where(y_val==0)
```

Out[122]:
```
(array([ 33,  47,  69,  94,  99, 101, 104, 105, 118, 122, 128, 130, 131,
        137, 148, 158, 182, 183, 184, 204, 206, 224, 233, 242, 250, 255,
        273, 284, 347, 356, 364, 418, 419, 433, 440, 446, 447, 476, 497,
        507, 524, 527, 535, 545, 581, 585, 621, 627, 634, 666, 688, 703,
        716, 744, 763, 782, 784, 787, 802, 813, 833, 845, 848, 868, 869,
        874, 884, 888, 903, 917, 948], dtype=int64),)
```

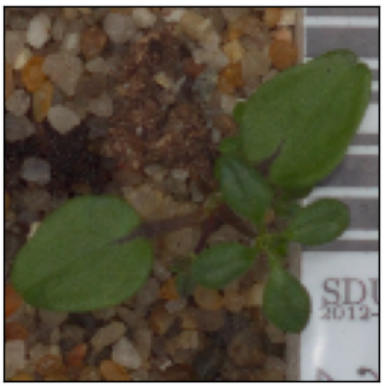
In [123…
```
#InCorrect Prediction
i = 888
fig,ax=plt.subplots(1,2,figsize=(7,3))
plot_image(i, predictions[i], y_val, X_val,ax[0])
plot_value_array(i, predictions[i],  y_val,ax[1])
plt.tight_layout()
plt.show()
```



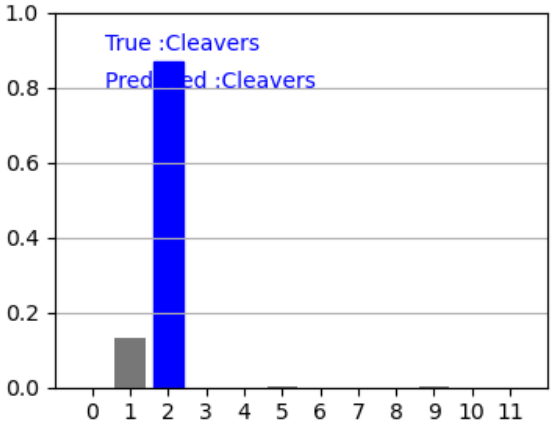
In [126…
```
#Correct Prediction
i = 7
fig,ax=plt.subplots(1,2,figsize=(7,3))
plot_image(i, predictions[i], y_val, X_val,ax[0])
plot_value_array(i, predictions[i],  y_val,ax[1])
plt.tight_layout()
plt.show()
```



In [131…
```
#Correct Prediction
i = 22
fig,ax=plt.subplots(1,2,figsize=(7,3))
plot_image(i, predictions[i], y_val, X_val,ax[0])
plot_value_array(i, predictions[i],  y_val,ax[1])
plt.tight_layout()
plt.show()
```



Cleavers 87% (Cleavers)



Upload and Predicted test images

In [132…
```
df_test = pd.read_csv('C:/Users/rusla/Documents/Data/plant-seedlings-classification/sample_submission.csv')
x_test=[]
for f, species in tqdm(df_test.values, miniters=100):
    img = cv2.imread('C:/Users/rusla/Documents/Data/plant-seedlings-classification/test/{}'.format(f))
    x_test.append(cv2.resize(img, (128,128))) #resize

x_test = np.array(x_test, np.float32)/255. #rescale
```

```
100%|██████████████████████████████████████████████████████████████████████████| 794/794 [00:01<00:00, 516.61it/s]
```

In [133…
```
p_test = model.predict(x_test, verbose=1) #predict classes of test images

preds = []
# create data structure for submission to kaggle
for i in range(len(p_test)):
    pos = np.argmax(p_test[i])
    preds.append(list(labels_id.keys())[list(labels_id.values()).index(pos)])
```

```
25/25 [==============================] - 1s 47ms/step
```

In [134…
```
df_test['species'] = preds
df_test.to_csv('submission.csv', index=False)
```

In [135…
```
df_test
```

Out[135]:

	file	species
0	0021e90e4.png	Small-flowered Cranesbill
1	003d61042.png	Small-flowered Cranesbill
2	007b3da8b.png	Sugar beet
3	0086a6340.png	Common Chickweed
4	00c47e980.png	Loose Silky-bent
...	...	...
789	fea355851.png	Loose Silky-bent
790	fea3da57c.png	Common wheat
791	fef2ade8c.png	Fat Hen
792	ff65bc002.png	Charlock
793	ffc6f8527.png	Loose Silky-bent

794 rows × 2 columns

Conclusions

Submission and Description	Private Score	Public Score	Selected
submission.csv Complete (after deadline) · now	0.6209	0.6209	☐

From Kaggle we get low score ,to increase performance we need:

- Train all layers in the Transfer learning technique
- Use another Model from Transfer learning
- More data augmentation
- Use of L1 and L2 regularization
- Fine-tuning /Hyper parameter tunning
- Use diffirent activation functions
- Additional feature enginering
- Use Ensemble technique (combined models)
- Use Diffirent Optimizarion algorithms or Tuning momentum parameters in Adam optimazer

In [ ]: