IntelTensorflow API Installation

In [1]:
```
1 pip install tensorflow-intel
```

```
Requirement already satisfied: h5py>=2.9.0 in c:\users\ashutoshvmadmin\anaconda3\lib
\site-packages (from tensorflow-intel) (3.7.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\ashutoshvmadmin\anacond
a3\lib\site-packages (from tensorflow-intel) (1.6.3)
Requirement already satisfied: libclang>=13.0.0 in c:\users\ashutoshvmadmin\anaconda
3\lib\site-packages (from tensorflow-intel) (15.0.6.1)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in c:\users\ashutoshvmadmin\anaco
nda3\lib\site-packages (from tensorflow-intel) (0.4.0)
Requirement already satisfied: packaging in c:\users\ashutoshvmadmin\anaconda3\lib\s
ite-packages (from tensorflow-intel) (21.3)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\ashu
toshvmadmin\anaconda3\lib\site-packages (from tensorflow-intel) (0.29.0)
Requirement already satisfied: numpy>=1.20 in c:\users\ashutoshvmadmin\anaconda3\lib
\site-packages (from tensorflow-intel) (1.21.0)
Collecting tensorboard<2.12,>=2.11
  Using cached tensorboard-2.11.2-py3-none-any.whl (6.0 MB)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\ashutoshvmadmin
\anaconda3\lib\site-packages (from tensorflow-intel) (4.3.0)
Collecting tensorflow-estimator<2.12,>=2.11.0
  Using cached tensorflow_estimator-2.11.0-py2.py3-none-any.whl (439 kB)
```

In [1]:
```
1 pip install --upgrade intel-extension-for-tensorflow[cpu]
```

```
Requirement already satisfied: intel-extension-for-tensorflow[cpu] in c:\users\ashutos
hvmadmin\anaconda3\lib\site-packages (0.0.0.dev1)
Note: you may need to restart the kernel to use updated packages.

WARNING: intel-extension-for-tensorflow 0.0.0.dev1 does not provide the extra 'cpu'
```

In [2]:
```
1 import tensorflow as tf
2 print(tf.__version__)
```

```
2.11.0
```

intel/intel-extension-for-tensorflow

Intel® Extension for TensorFlow*

Importing all the libraries

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import shutil
import glob
import cv2
import numpy as np
import math
import datetime
import time
import random
import gc
from tqdm import tqdm
```

In [4]:
```
import keras
import tensorflow as tf
from keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout , Activation, Flatten,Conv2D,Max
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from sklearn.metrics import classification_report,confusion_matrix
from tensorflow.keras import optimizers
from tensorflow.keras import applications
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
```

In [5]:
```
print(tf.__version__)
print(tf.config.list_physical_devices())
```

```
2.11.0
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
```

- Dividing the datset based on labels. train_0 contains all the images which are not infected with weed and train_1 contains all the images which are infected with weed and likewise for test data also.

In [6]:
```
TRAIN_DIR = 'data/train' #Training Dataset Path
TEST_DIR = 'data/test'   #Testing Dataset Path

train_0 = [TRAIN_DIR + '/0' + '/' + '{}'.format(i)
              for i in os.listdir(TRAIN_DIR + '/0')]
train_1 = [TRAIN_DIR + '/1' + '/' + '{}'.format(i)
              for i in os.listdir(TRAIN_DIR + '/1')]


train_imgs = train_0 + train_1
#random.shuffle(train_imgs)  # shuffle it randomly
```

In [7]:
```
#Getting Test Data -
test_0 = [TEST_DIR + '/0' + '/' + '{}'.format(i)
              for i in os.listdir(TEST_DIR + '/0')]
test_1 = [TEST_DIR + '/1' + '/' + '{}'.format(i)
              for i in os.listdir(TEST_DIR + '/1')]


test_imgs = test_0 + test_1
random.shuffle(test_imgs)
```

- function to read and process the images to an acceptable format for our model

In [8]:

```
nrows = 224
ncolumns = 224

#A function to read and process the images to an acceptable format for our model
def read_and_process_image(list_of_images):
    """
    Returns two arrays:
        X is an array of resized images
        y is an array of labels
    """

    X = [] # images
    y = [] # labels

    for image in tqdm(list_of_images):
        X.append(cv2.resize(cv2.imread(image, cv2.IMREAD_COLOR),
                            (nrows,ncolumns), interpolation=cv2.INTER_CUBIC))  #Rea
        #get the labels
        if '0' in image[:12]:
            y.append(0)
        elif '1' in image[:12]:
            y.append(1)

    return X, y
```

Reading images and labels from the training dataset

In [9]:

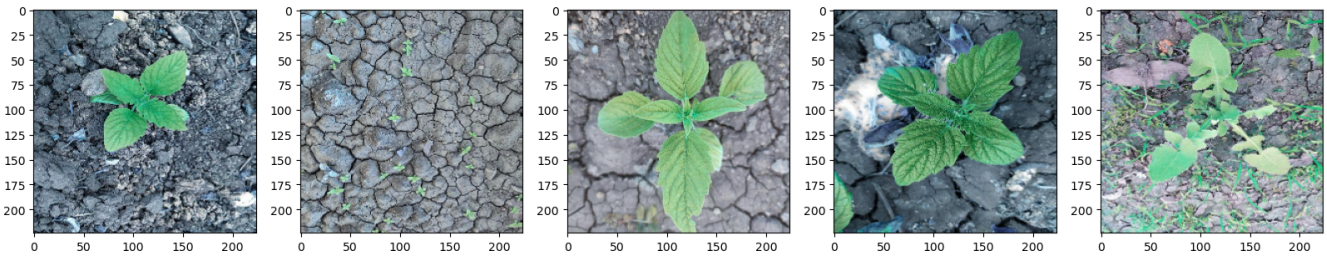
```
X, y = read_and_process_image(train_imgs)
```

```
100%|██████████████████████████████████████████████████████████████████████| 10
39/1039 [00:07<00:00, 131.79it/s]
```

In [10]:

```
plt.figure(figsize=(20,10))
columns = 5
for i in tqdm(range(columns)):
    plt.subplot(5 // columns + 1, columns, i + 1)
    plt.imshow(X[i])
```

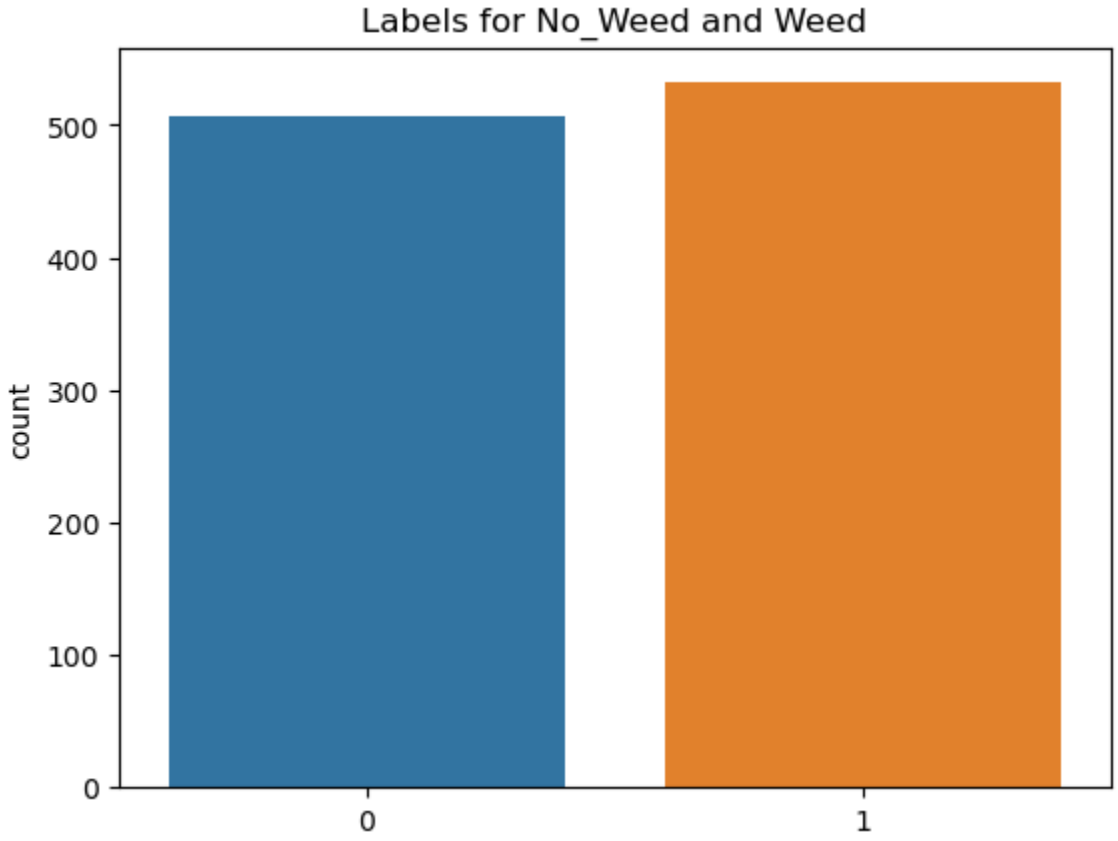
```
100%|████████████████████████████████████████████████████████████████████████
███| 5/5 [00:00<00:00, 46.76it/s]
```



In [11]:
```
del train_imgs
gc.collect()

#Convert list to numpy array
X = np.array(X)
y = np.array(y)


#Lets plot the label to be sure we just have two class
sns.countplot(y)
plt.title('Labels for No_Weed and Weed')
```

```
C:\Users\ashutoshvmadmin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an expli
cit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[11]: Text(0.5, 1.0, 'Labels for No_Weed and Weed')

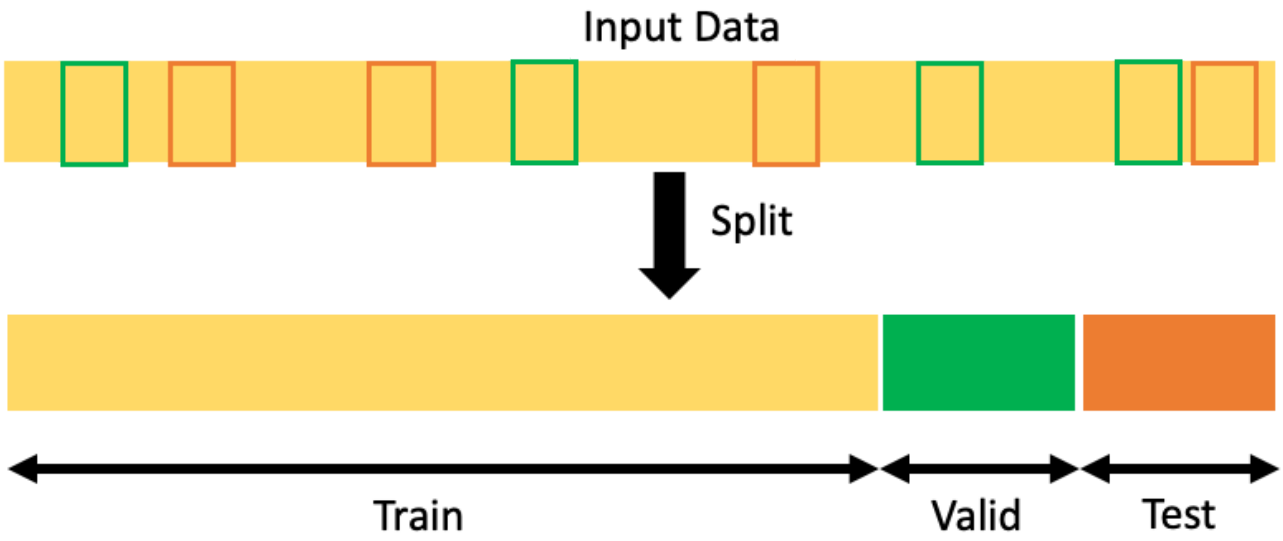



```
<hr style = "border-top: 3px solid black" >
```

Creating Training and Testing Data

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.20,
random_state=2)

print("Shape of train images is:", X_train.shape)
print("Shape of validation images is:", X_val.shape)
print("Shape of labels is:", y_train.shape)
```

```
6 print("Shape of labels is:", y_val.shape)
```



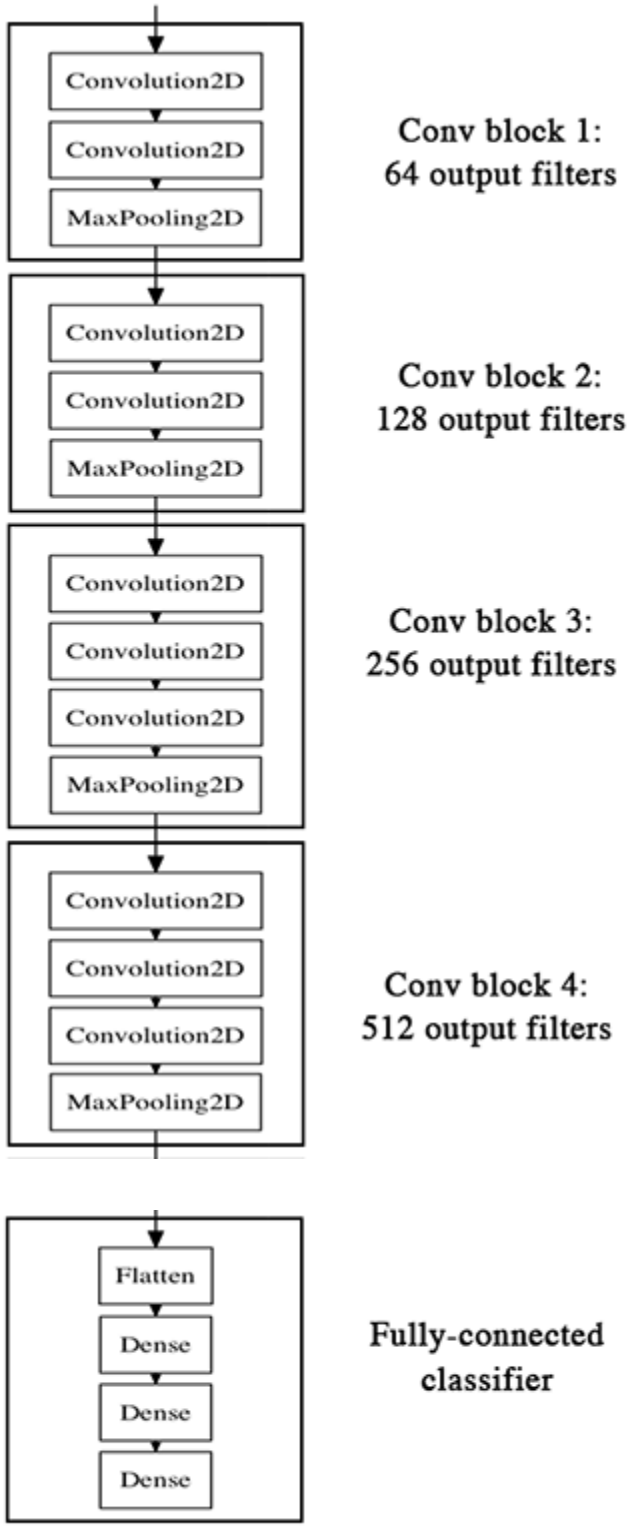
In [13]:
```
1 print("Shape of train images is:", X.shape)
2 print("Shape of labels is:", y.shape)
```

```
Shape of train images is: (1039, 224, 224, 3)
Shape of labels is: (1039,)
```

In [14]:
```
1  del X
2  del y
3  gc.collect()
4
5  #get the length of the train and validation data
6  ntrain = len(X_train)
7  nval = len(X_val)
8  print(ntrain)
9  print(nval)
10
11 #We will use a batch size of 32. Note: batch size should be a factor of 2.***4,8,16,
12 batch_size = 32
```

```
831
208
```

Preparing our Initial Model for Training



In [35]:
```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu',input_shape=(224, 224, 3)))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dropout(0.5))  #Dropout for regularization
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [16]:
```
model.compile(loss='binary_crossentropy', optimizer=optimizers.RMSprop(lr=1e-4), me
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy
optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

In [17]:
```
train_datagen = ImageDataGenerator(rescale=1./255,   #Scale the image between 0 and
                                   rotation_range=50,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,)

#added
train_datagen = ImageDataGenerator(rescale=1./255   #Scale the image between 0 and
                                   )


val_datagen = ImageDataGenerator(rescale=1./255)  #We do not augment validation dat
```

In [18]:
```
train_generator = train_datagen.flow(X_train, y_train, batch_size=batch_size)
val_generator = val_datagen.flow(X_val, y_val, batch_size=batch_size)
```

Training our Initial Model

In [19]:
```
history = model.fit_generator(train_generator,
                              steps_per_epoch=ntrain // batch_size,
                              epochs=50,
                              validation_data=val_generator,
                              validation_steps=nval // batch_size)
```

```
C:\Users\ashutoshvmadmin\AppData\Local\Temp\2\ipykernel_1980\1008497584.py:1: UserWarn
ing: `Model.fit_generator` is deprecated and will be removed in a future version. Plea
se use `Model.fit`, which supports generators.
  history = model.fit_generator(train_generator,
```

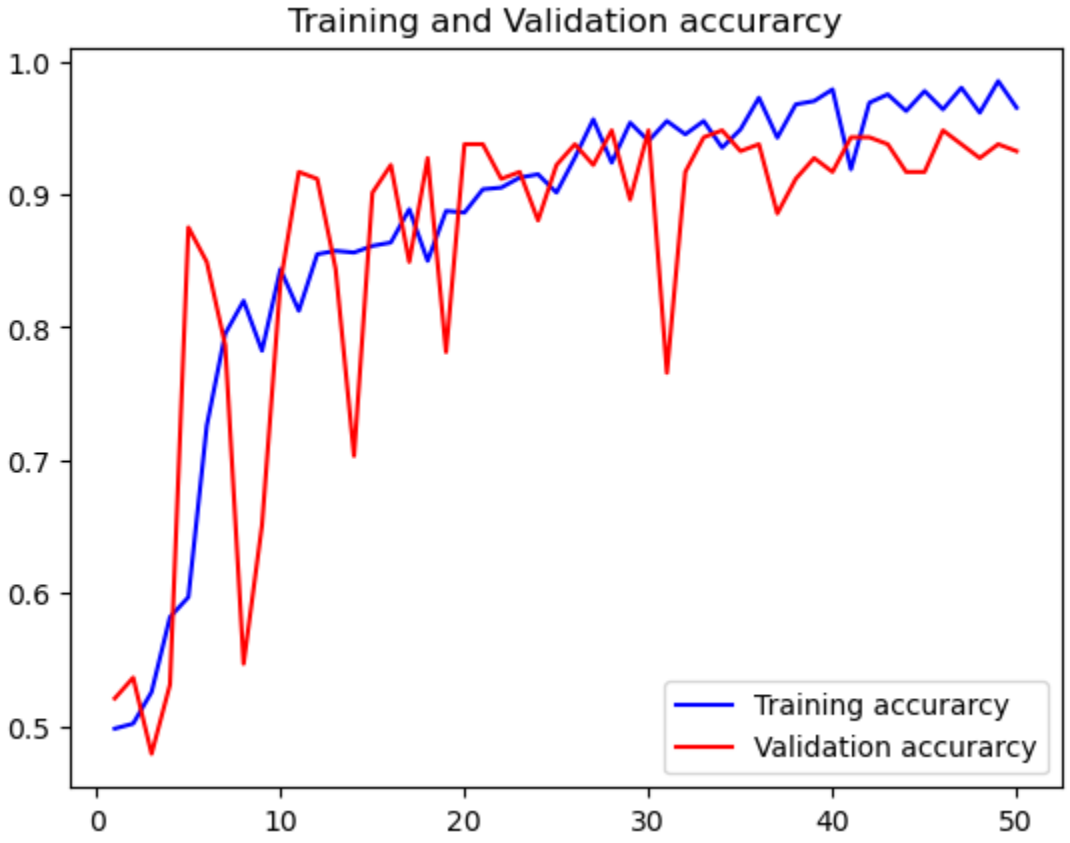
```
Epoch 1/50
25/25 [==============================] - 24s 894ms/step - loss: 1.1702 - acc: 0.4981 -
val_loss: 0.6895 - val_acc: 0.5208
Epoch 2/50
25/25 [==============================] - 22s 895ms/step - loss: 0.6942 - acc: 0.5019 -
val_loss: 0.6872 - val_acc: 0.5365
Epoch 3/50
25/25 [==============================] - 23s 905ms/step - loss: 0.7580 - acc: 0.5257 -
val_loss: 0.6935 - val_acc: 0.4792
Epoch 4/50
25/25 [==============================] - 23s 938ms/step - loss: 0.6936 - acc: 0.5820 -
val_loss: 0.7331 - val_acc: 0.5312
Epoch 5/50
25/25 [==============================] - 23s 938ms/step - loss: 0.6754 - acc: 0.5970 -
val_loss: 0.5516 - val_acc: 0.8750
Epoch 6/50
25/25 [==============================] - 23s 915ms/step - loss: 0.6581 - acc: 0.7259 -
val_loss: 0.4477 - val_acc: 0.8490
Epoch 7/50
25/25 [==============================] - 22s 896ms/step - loss: 0.4954 - acc: 0.7950 -
val_loss: 0.5159 - val_acc: 0.7865
Epoch 8/50
25/25 [==============================] - 23s 910ms/step - loss: 0.4413 - acc: 0.8198 -
val_loss: 0.9165 - val_acc: 0.5469
Epoch 9/50
25/25 [==============================] - 23s 922ms/step - loss: 0.4941 - acc: 0.7822 -
val_loss: 1.0834 - val_acc: 0.6510
Epoch 10/50
25/25 [==============================] - 22s 901ms/step - loss: 0.4391 - acc: 0.8436 -
val_loss: 0.4376 - val_acc: 0.8333
Epoch 11/50
25/25 [==============================] - 22s 889ms/step - loss: 0.4226 - acc: 0.8123 -
val_loss: 0.4079 - val_acc: 0.9167
Epoch 12/50
25/25 [==============================] - 23s 912ms/step - loss: 0.3645 - acc: 0.8548 -
val_loss: 0.3276 - val_acc: 0.9115
Epoch 13/50
25/25 [==============================] - 22s 893ms/step - loss: 0.3661 - acc: 0.8573 -
val_loss: 0.4272 - val_acc: 0.8438
Epoch 14/50
25/25 [==============================] - 22s 900ms/step - loss: 0.3355 - acc: 0.8561 -
val_loss: 0.5437 - val_acc: 0.7031
Epoch 15/50
25/25 [==============================] - 22s 897ms/step - loss: 0.3314 - acc: 0.8611 -
val_loss: 0.3131 - val_acc: 0.9010
Epoch 16/50
25/25 [==============================] - 22s 898ms/step - loss: 0.3315 - acc: 0.8636 -
val_loss: 0.2692 - val_acc: 0.9219
Epoch 17/50
25/25 [==============================] - 23s 901ms/step - loss: 0.2728 - acc: 0.8886 -
val_loss: 0.4138 - val_acc: 0.8490
Epoch 18/50
25/25 [==============================] - 22s 897ms/step - loss: 0.3651 - acc: 0.8498 -
val_loss: 0.2828 - val_acc: 0.9271
Epoch 19/50
25/25 [==============================] - 23s 903ms/step - loss: 0.2738 - acc: 0.8874 -
val_loss: 0.7057 - val_acc: 0.7812
Epoch 20/50
25/25 [==============================] - 23s 907ms/step - loss: 0.2545 - acc: 0.8861 -
```

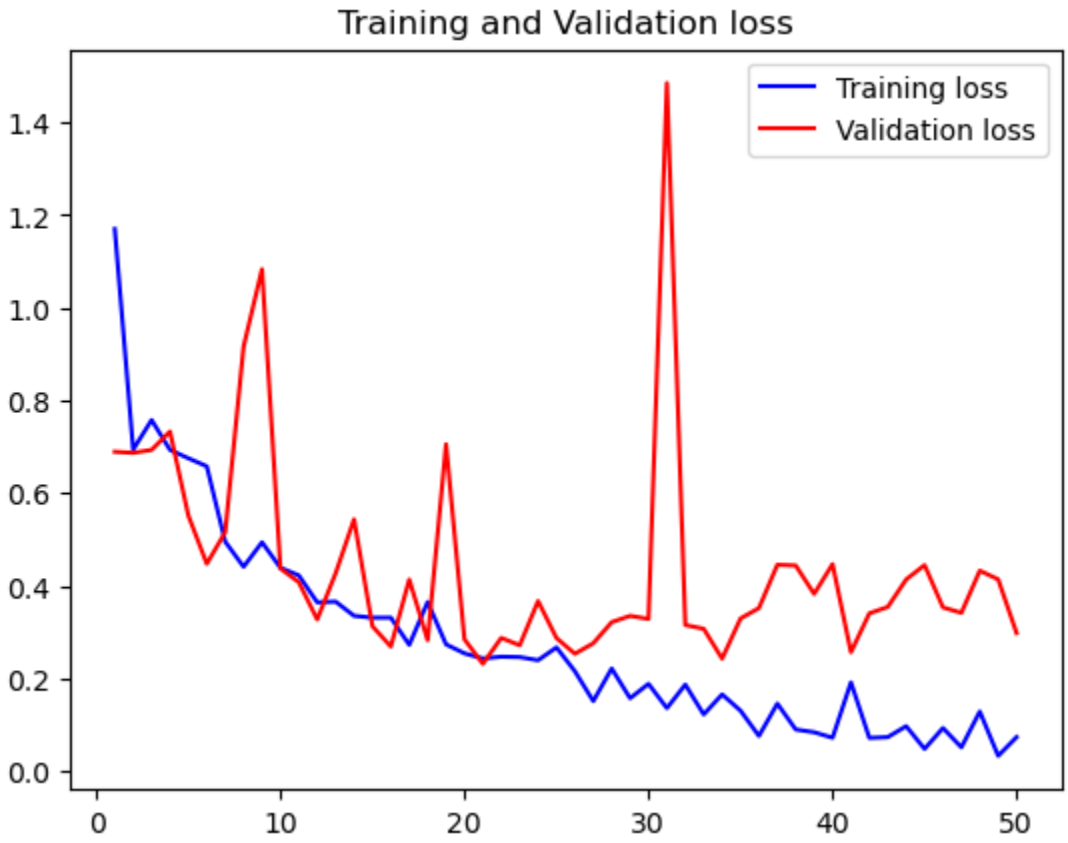
```
val_loss: 0.2847 - val_acc: 0.9375
Epoch 21/50
25/25 [==============================] - 22s 888ms/step - loss: 0.2434 - acc: 0.9036 -
val_loss: 0.2319 - val_acc: 0.9375
Epoch 22/50
25/25 [==============================] - 22s 894ms/step - loss: 0.2473 - acc: 0.9049 -
val_loss: 0.2878 - val_acc: 0.9115
Epoch 23/50
25/25 [==============================] - 22s 890ms/step - loss: 0.2464 - acc: 0.9124 -
val_loss: 0.2719 - val_acc: 0.9167
Epoch 24/50
25/25 [==============================] - 22s 898ms/step - loss: 0.2397 - acc: 0.9149 -
val_loss: 0.3673 - val_acc: 0.8802
Epoch 25/50
25/25 [==============================] - 22s 896ms/step - loss: 0.2672 - acc: 0.9011 -
val_loss: 0.2875 - val_acc: 0.9219
Epoch 26/50
25/25 [==============================] - 22s 895ms/step - loss: 0.2159 - acc: 0.9274 -
val_loss: 0.2535 - val_acc: 0.9375
Epoch 27/50
25/25 [==============================] - 22s 880ms/step - loss: 0.1515 - acc: 0.9562 -
val_loss: 0.2762 - val_acc: 0.9219
Epoch 28/50
25/25 [==============================] - 22s 888ms/step - loss: 0.2218 - acc: 0.9237 -
val_loss: 0.3218 - val_acc: 0.9479
Epoch 29/50
25/25 [==============================] - 22s 889ms/step - loss: 0.1576 - acc: 0.9537 -
val_loss: 0.3354 - val_acc: 0.8958
Epoch 30/50
25/25 [==============================] - 22s 890ms/step - loss: 0.1887 - acc: 0.9399 -
val_loss: 0.3288 - val_acc: 0.9479
Epoch 31/50
25/25 [==============================] - 22s 886ms/step - loss: 0.1362 - acc: 0.9549 -
val_loss: 1.4850 - val_acc: 0.7656
Epoch 32/50
25/25 [==============================] - 22s 872ms/step - loss: 0.1870 - acc: 0.9449 -
val_loss: 0.3157 - val_acc: 0.9167
Epoch 33/50
25/25 [==============================] - 22s 885ms/step - loss: 0.1226 - acc: 0.9549 -
val_loss: 0.3075 - val_acc: 0.9427
Epoch 34/50
25/25 [==============================] - 22s 883ms/step - loss: 0.1659 - acc: 0.9349 -
val_loss: 0.2430 - val_acc: 0.9479
Epoch 35/50
25/25 [==============================] - 22s 895ms/step - loss: 0.1308 - acc: 0.9487 -
val_loss: 0.3297 - val_acc: 0.9323
Epoch 36/50
25/25 [==============================] - 22s 888ms/step - loss: 0.0764 - acc: 0.9725 -
val_loss: 0.3521 - val_acc: 0.9375
Epoch 37/50
25/25 [==============================] - 22s 884ms/step - loss: 0.1460 - acc: 0.9424 -
val_loss: 0.4456 - val_acc: 0.8854
Epoch 38/50
25/25 [==============================] - 23s 912ms/step - loss: 0.0902 - acc: 0.9675 -
val_loss: 0.4441 - val_acc: 0.9115
Epoch 39/50
25/25 [==============================] - 22s 899ms/step - loss: 0.0843 - acc: 0.9700 -
val_loss: 0.3827 - val_acc: 0.9271
Epoch 40/50
```

```
25/25 [==============================] - 22s 889ms/step - loss: 0.0724 - acc: 0.9787 -
val_loss: 0.4470 - val_acc: 0.9167
Epoch 41/50
25/25 [==============================] - 22s 894ms/step - loss: 0.1918 - acc: 0.9186 -
val_loss: 0.2569 - val_acc: 0.9427
Epoch 42/50
25/25 [==============================] - 22s 880ms/step - loss: 0.0720 - acc: 0.9687 -
val_loss: 0.3408 - val_acc: 0.9427
Epoch 43/50
25/25 [==============================] - 22s 892ms/step - loss: 0.0737 - acc: 0.9750 -
val_loss: 0.3546 - val_acc: 0.9375
Epoch 44/50
25/25 [==============================] - 22s 896ms/step - loss: 0.0974 - acc: 0.9625 -
val_loss: 0.4139 - val_acc: 0.9167
Epoch 45/50
25/25 [==============================] - 22s 892ms/step - loss: 0.0483 - acc: 0.9775 -
val_loss: 0.4449 - val_acc: 0.9167
Epoch 46/50
25/25 [==============================] - 23s 939ms/step - loss: 0.0934 - acc: 0.9637 -
val_loss: 0.3536 - val_acc: 0.9479
Epoch 47/50
25/25 [==============================] - 23s 904ms/step - loss: 0.0517 - acc: 0.9800 -
val_loss: 0.3420 - val_acc: 0.9375
Epoch 48/50
25/25 [==============================] - 22s 892ms/step - loss: 0.1289 - acc: 0.9612 -
val_loss: 0.4328 - val_acc: 0.9271
Epoch 49/50
25/25 [==============================] - 22s 890ms/step - loss: 0.0334 - acc: 0.9850 -
val_loss: 0.4142 - val_acc: 0.9375
Epoch 50/50
25/25 [==============================] - 22s 890ms/step - loss: 0.0738 - acc: 0.9650 -
val_loss: 0.2982 - val_acc: 0.9323
```

In [21]:

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

#Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accurarcy')
plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()



plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()

plt.show()
```





In [22]:
```
X_test, y_test = read_and_process_image(test_imgs) #Y_test in this case will be emp
x = np.array(X_test)
test_datagen = ImageDataGenerator(rescale=1./255)
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
261/261 [00:03<00:00, 76.21it/s]
```

Validating our Initial Model

In [23]:
```
out = model.evaluate_generator(val_generator)
print(out)
```

```
C:\Users\ashutoshvmadmin\AppData\Local\Temp\2\ipykernel_1980\4193496103.py:1: UserWarn
ing: `Model.evaluate_generator` is deprecated and will be removed in a future version.
Please use `Model.evaluate`, which supports generators.
  out = model.evaluate_generator(val_generator)

[0.32266777753829956, 0.932692289352417]
```

In [24]:

```
Y_pred = model.predict(X_val)
print(Y_pred.shape)
#y_pred = np.argmax(Y_pred, axis=1) - used for multiclass
y_pred = (Y_pred > 0.5) * 1.0
y_pred = y_pred.reshape(y_val.shape)
y_pred.sum()
```

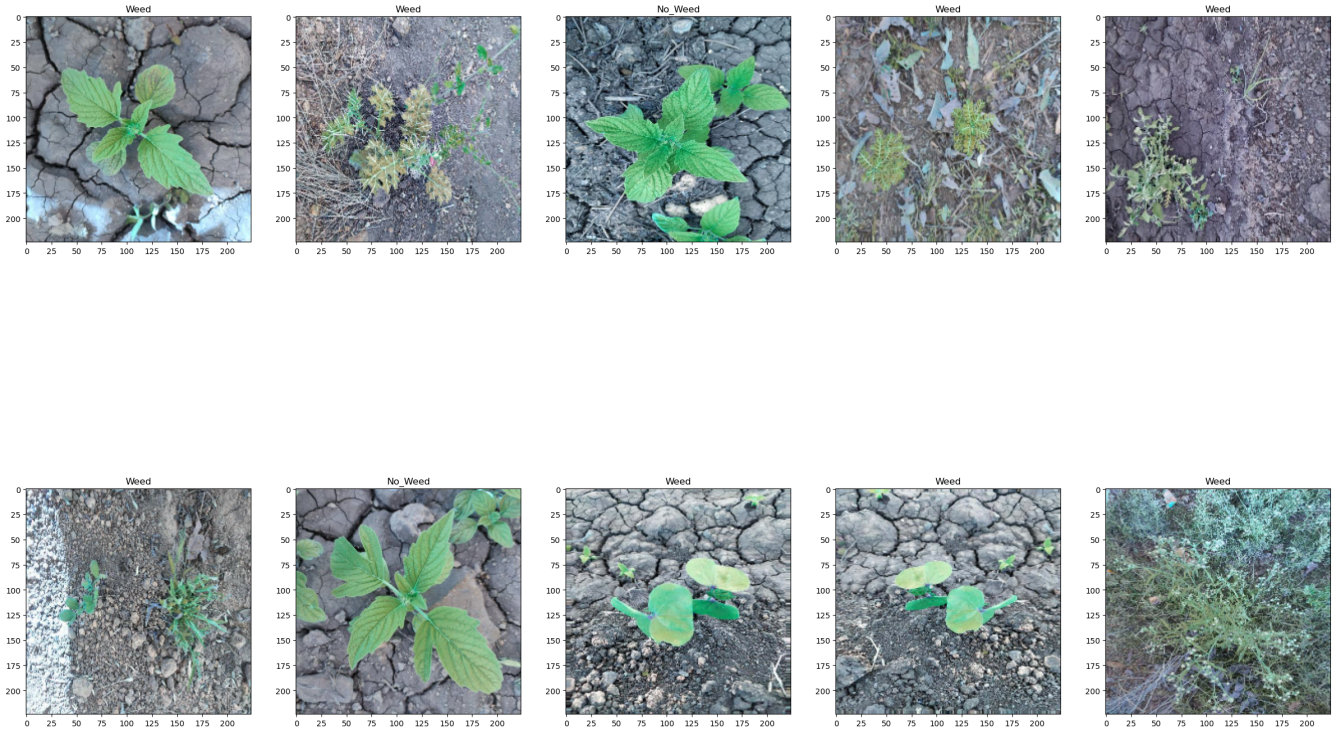
```
7/7 [==============================] - 2s 194ms/step
(208, 1)
```

Out[24]: 104.0

In [25]:

```
i = 0
text_labels = []
plt.figure(figsize=(30,20))

for batch in test_datagen.flow(x, batch_size=1):
    pred = model.predict(batch)
    if pred > 0.5:
        text_labels.append('Weed')
    else:
        text_labels.append('No_Weed')
    plt.subplot(5 // columns + 1, columns, i + 1)
    plt.title(text_labels[i])
    #print(batch[0])
    imgplot = plt.imshow(batch[0])
    i += 1
    if i % 10 == 0:
        break
plt.show()
```

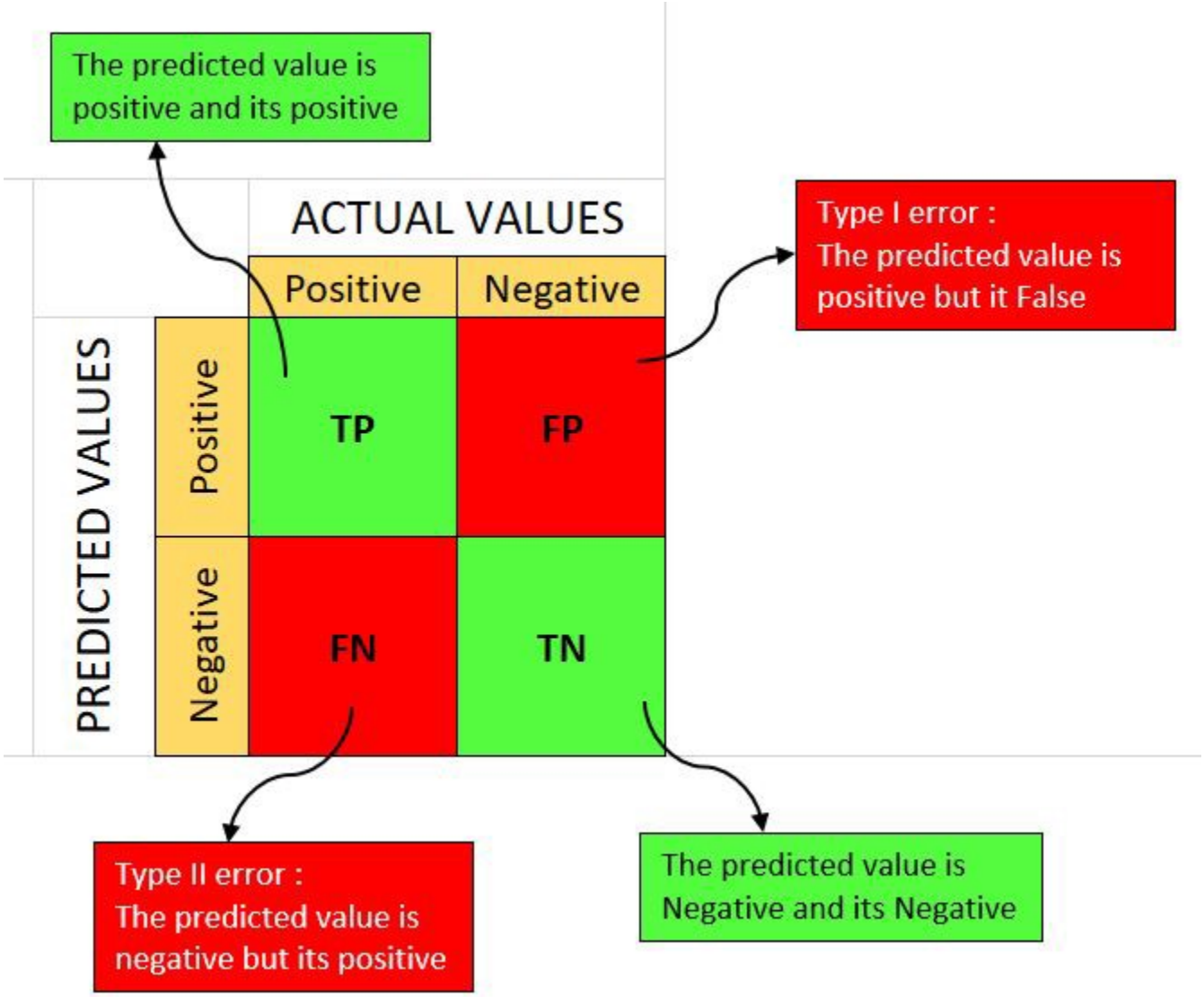
```
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 50ms/step
```



In [26]:

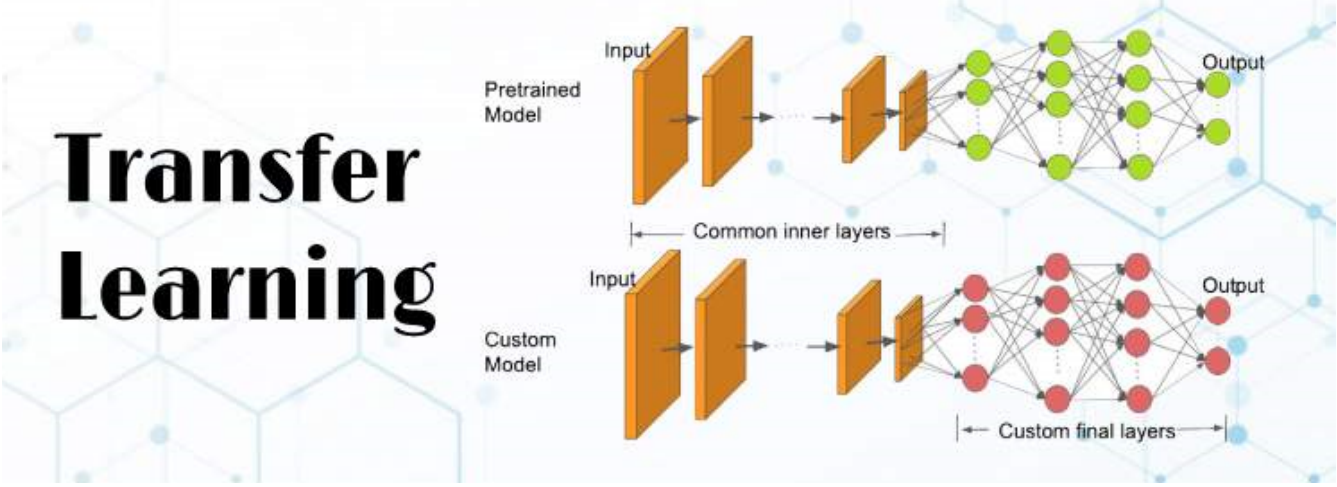
```
print('Confusion Matrix')
print(confusion_matrix(y_val, y_pred))


print('Classification Report')
target_names = ['Weed', 'No Weed']
print(classification_report(y_val, y_pred,target_names=target_names))

# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_val, y_pred)
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(y_val, y_pred)
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(y_val, y_pred)
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_val, y_pred)
print('F1 score: %f' % f1)
```

```
Confusion Matrix
[[ 96   3]
 [  8 101]]
Classification Report
              precision    recall  f1-score   support

        Weed       0.92      0.97      0.95        99
     No Weed       0.97      0.93      0.95       109

    accuracy                           0.95       208
   macro avg       0.95      0.95      0.95       208
weighted avg       0.95      0.95      0.95       208

Accuracy: 0.947115
Precision: 0.971154
Recall: 0.926606
F1 score: 0.948357
```



Transfer Learning

- Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. Transfer learning can be used when the dataset is small, by using a pre-trained model on similar images we can easily achieve high performance.



In [27]:
```
1 base_model = tf.keras.applications.MobileNetV2(input_shape = (224, 224, 3), include
```

In [28]:
```
1 base_model.trainable = False
```

In [29]:
```
1 model = tf.keras.Sequential([base_model,
2                              tf.keras.layers.GlobalAveragePooling2D(),
3                              tf.keras.layers.Dropout(0.2),
4                              tf.keras.layers.Dense(1, activation="sigmoid")
5                             ])
```

Training our Final Model

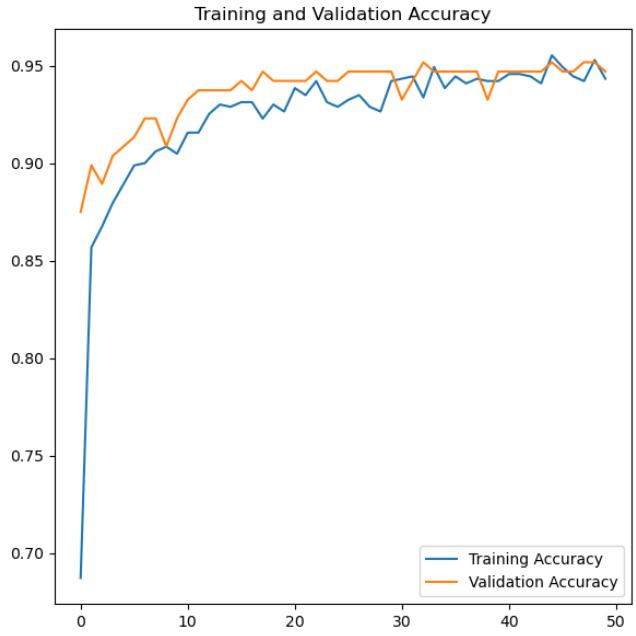
In [30]:

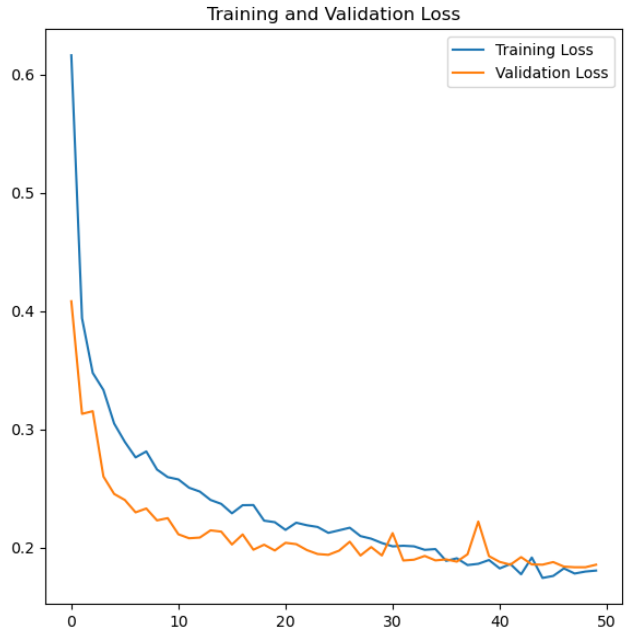
```
base_learning_rate = 0.00001
model.compile(optimizer=tf.keras.optimizers.Adam(lr=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
              metrics=['accuracy'])

history = model.fit(X_train,y_train,epochs = 50 , validation_data = (X_val, y_val))
```

```
0.9302 - val_loss: 0.2024 - val_accuracy: 0.9423
Epoch 20/50
26/26 [==============================] - 13s 489ms/step - loss: 0.2215 - accuracy:
0.9266 - val_loss: 0.1975 - val_accuracy: 0.9423
Epoch 21/50
26/26 [==============================] - 13s 487ms/step - loss: 0.2150 - accuracy:
0.9386 - val_loss: 0.2040 - val_accuracy: 0.9423
Epoch 22/50
26/26 [==============================] - 13s 493ms/step - loss: 0.2210 - accuracy:
0.9350 - val_loss: 0.2029 - val_accuracy: 0.9423
Epoch 23/50
26/26 [==============================] - 13s 488ms/step - loss: 0.2189 - accuracy:
0.9422 - val_loss: 0.1979 - val_accuracy: 0.9471
Epoch 24/50
26/26 [==============================] - 13s 493ms/step - loss: 0.2174 - accuracy:
0.9314 - val_loss: 0.1945 - val_accuracy: 0.9423
Epoch 25/50
26/26 [==============================] - 13s 488ms/step - loss: 0.2124 - accuracy:
0.9290 - val_loss: 0.1939 - val_accuracy: 0.9423
Epoch 26/50
```

In [31]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(50)

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
#Train and validation accuracy
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

#Train and validation loss
plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```





Calculating all the Evaluation Metrics

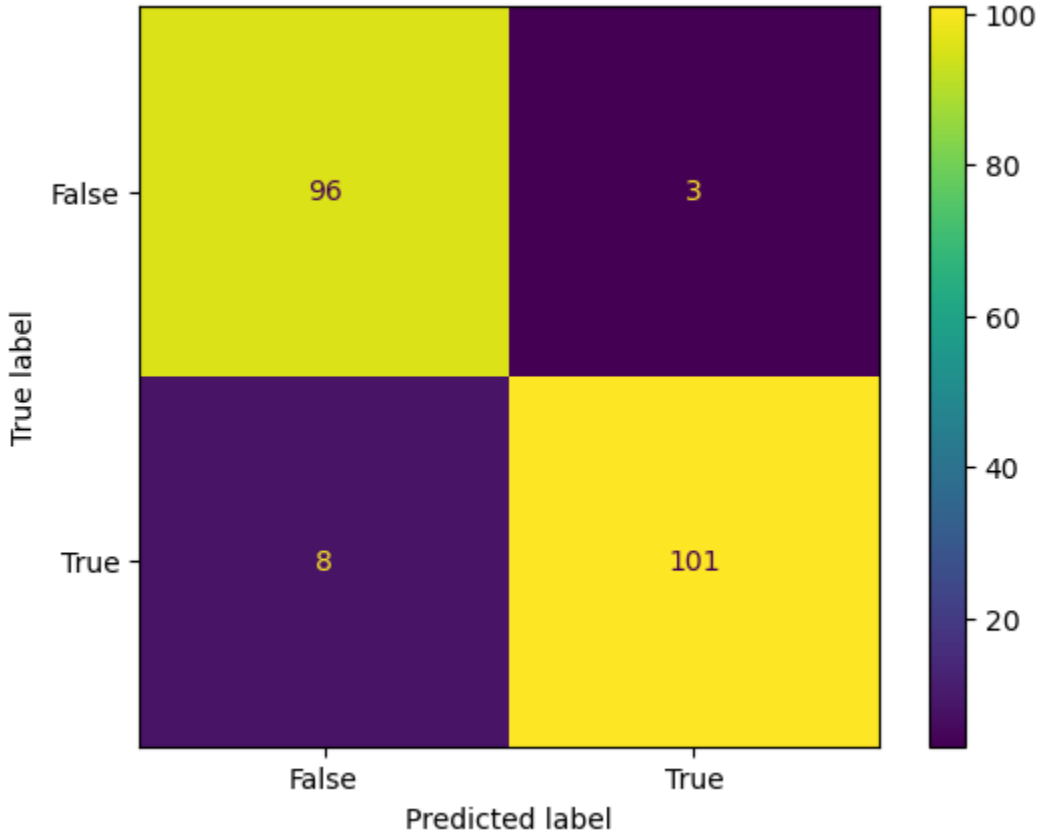
In [32]:
```
predictions = model.predict(X_val)
predictions = predictions.reshape(-1)

for i in range(len(predictions)):
    if predictions[i] > 0.5:
        predictions[i] = 1
    else:
        predictions[i] = 0

print(classification_report(y_val, predictions, target_names = ['No Weed (Class 0)'
```

```
7/7 [==============================] - 3s 363ms/step
                  precision    recall  f1-score   support

No Weed (Class 0)       0.92      0.97      0.95        99
   Weed (Class 1)       0.97      0.93      0.95       109

         accuracy                           0.95       208
        macro avg       0.95      0.95      0.95       208
     weighted avg       0.95      0.95      0.95       208
```

Confusion Metrics

In [42]:
```
from sklearn import metrics

confusion_matrix = metrics.confusion_matrix(y_val, predictions)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, di
```

In [43]:
```
cm_display.plot()
plt.show()
```



Weed Detection Assignment Summary

We are able to achieve F1 Score of 0.95 with this Model

In [44]:
```
f1 = f1_score(y_val, predictions)
```

In [45]:
```
from IPython.display import Markdown, display

f1_text = f"**F1 Score:** {f1:.2f}"
display(Markdown(f1_text))
```

F1 Score: 0.95