```
import numpy as np
import pickle
import cv2
from os import listdir
from sklearn.preprocessing import LabelBinarizer
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation, Flatten, Dropout, Dense
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.preprocessing import image
from keras.preprocessing.image import img_to_array
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

In [2]:

```
EPOCHS = 25
INIT_LR = 1e-3
BS = 32
default_image_size = tuple((256, 256))
image_size = 0
directory_root = '../input/plantvillage/'
width=256
height=256
depth=3
```

Function to convert images to array

In [3]:

```
def convert_image_to_array(image_dir):
    try:
        image = cv2.imread(image_dir)
        if image is not None :
            image = cv2.resize(image, default_image_size)
            return img_to_array(image)
        else :
            return np.array([])
    except Exception as e:
        print(f"Error : {e}")
        return None
```

Fetch images from directory

In [4]:

```
image_list, label_list = [], []
try:
    print("[INFO] Loading images ...")
    root_dir = listdir(directory_root)
    for directory in root_dir :
        # remove .DS_Store from list
        if directory == ".DS_Store" :
            root_dir.remove(directory)

    for plant_folder in root_dir :
        plant_disease_folder_list = listdir(f"{directory_root}/{plant_folder}")

        for disease_folder in plant_disease_folder_list :
            # remove .DS_Store from list
            if disease_folder == ".DS_Store" :
                plant_disease_folder_list.remove(disease_folder)

        for plant_disease_folder in plant_disease_folder_list:
            print(f"[INFO] Processing {plant_disease_folder} ...")
            plant_disease_image_list = listdir(f"{directory_root}/{plant_folder}/{plant_disease_folder}/")

            for single_plant_disease_image in plant_disease_image_list :
                if single_plant_disease_image == ".DS_Store" :
                    plant_disease_image_list.remove(single_plant_disease_image)

            for image in plant_disease_image_list[:200]:
                image_directory = f"{directory_root}/{plant_folder}/{plant_disease_folder}/{image}"
                if image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") == True:
                    image_list.append(convert_image_to_array(image_directory))
                    label_list.append(plant_disease_folder)
    print("[INFO] Image loading completed")
except Exception as e:
    print(f"Error : {e}")
```

```
[INFO] Loading images ...
[INFO] Processing Tomato_Septoria_leaf_spot ...
[INFO] Processing Tomato__Tomato_mosaic_virus ...
[INFO] Processing Tomato_Late_blight ...
[INFO] Processing Tomato_Spider_mites_Two_spotted_spider_mite ...
[INFO] Processing Tomato__Tomato_YellowLeaf__Curl_Virus ...
[INFO] Processing Tomato_healthy ...
[INFO] Processing Pepper__bell___Bacterial_spot ...
[INFO] Processing Potato___healthy ...
[INFO] Processing Tomato__Target_Spot ...
[INFO] Processing Tomato_Leaf_Mold ...
[INFO] Processing Tomato_Bacterial_spot ...
[INFO] Processing Tomato_Early_blight ...
[INFO] Processing Potato___Late_blight ...
[INFO] Processing Potato___Early_blight ...
[INFO] Processing Pepper__bell___healthy ...
[INFO] Image loading completed
```

Get Size of Processed Image

In [5]:

```
image_size = len(image_list)
```

Transform Image Labels uisng Scikit Learn's LabelBinarizer

In [6]:

```
label_binarizer = LabelBinarizer()
image_labels = label_binarizer.fit_transform(label_list)
pickle.dump(label_binarizer,open('label_transform.pkl', 'wb'))
n_classes = len(label_binarizer.classes_)
```

Print the classes

In [7]:

```
print(label_binarizer.classes_)
```

```
['Pepper__bell___Bacterial_spot' 'Pepper__bell___healthy'
 'Potato___Early_blight' 'Potato___Late_blight' 'Potato___healthy'
 'Tomato_Bacterial_spot' 'Tomato_Early_blight' 'Tomato_Late_blight'
 'Tomato_Leaf_Mold' 'Tomato_Septoria_leaf_spot'
 'Tomato_Spider_mites_Two_spotted_spider_mite' 'Tomato__Target_Spot'
 'Tomato__Tomato_YellowLeaf__Curl_Virus' 'Tomato__Tomato_mosaic_virus'
 'Tomato_healthy']
```

In [8]:

```
np_image_list = np.array(image_list, dtype=np.float16) / 225.0
```

In [9]:

```
print("[INFO] Spliting data to train, test")
x_train, x_test, y_train, y_test = train_test_split(np_image_list, image_labels, test_size=0.2, random_state = 42)
```

```
[INFO] Spliting data to train, test
```

In [10]:

```
aug = ImageDataGenerator(
    rotation_range=25, width_shift_range=0.1,
    height_shift_range=0.1, shear_range=0.2,
    zoom_range=0.2,horizontal_flip=True,
    fill_mode="nearest")
```

In [11]:

```
model = Sequential()
inputShape = (height, width, depth)
chanDim = -1
if K.image_data_format() == "channels_first":
    inputShape = (depth, height, width)
    chanDim = 1
model.add(Conv2D(32, (3, 3), padding="same",input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(n_classes))
model.add(Activation("softmax"))
```

Model Summary

In [12]:

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 256, 256, 32)      896       
_________________________________________________________________
activation_1 (Activation)    (None, 256, 256, 32)      0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 256, 256, 32)      128       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 85, 85, 32)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 85, 85, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 85, 85, 64)        18496     
_________________________________________________________________
activation_2 (Activation)    (None, 85, 85, 64)        0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 85, 85, 64)        256       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 85, 85, 64)        36928     
_________________________________________________________________
activation_3 (Activation)    (None, 85, 85, 64)        0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 85, 85, 64)        256       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 42, 42, 64)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 42, 42, 64)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 42, 42, 128)       73856     
_________________________________________________________________
activation_4 (Activation)    (None, 42, 42, 128)       0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 42, 42, 128)       512       
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 42, 42, 128)       147584    
_________________________________________________________________
activation_5 (Activation)    (None, 42, 42, 128)       0         
_________________________________________________________________
batch_normalization_5 (Batch (None, 42, 42, 128)       512       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 21, 21, 128)       0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 21, 21, 128)       0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 56448)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              57803776  
_________________________________________________________________
activation_6 (Activation)    (None, 1024)              0         
_________________________________________________________________
batch_normalization_6 (Batch (None, 1024)              4096      
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 15)                15375     
_________________________________________________________________
activation_7 (Activation)    (None, 15)                0         
=================================================================
Total params: 58,102,671
Trainable params: 58,099,791
Non-trainable params: 2,880
_________________________________________________________________
```

In [13]:

```
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# distribution
model.compile(loss="binary_crossentropy", optimizer=opt,metrics=["accuracy"])
# train the network
print("[INFO] training network...")
```

```
[INFO] training network...
```

In [14]:

```
history = model.fit_generator(
    aug.flow(x_train, y_train, batch_size=BS),
    validation_data=(x_test, y_test),
    steps_per_epoch=len(x_train) // BS,
    epochs=EPOCHS, verbose=1
    )
```

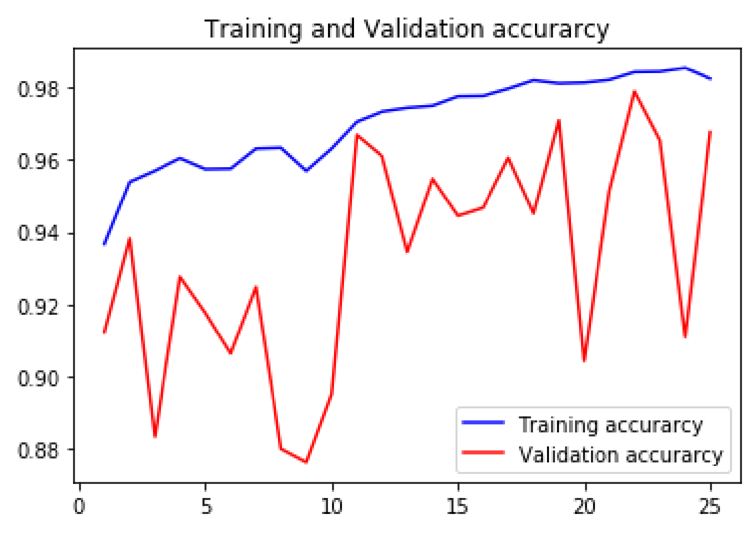
```
Epoch 1/25
73/73 [==============================] - 47s 644ms/step - loss: 0.1989 - acc: 0.9367 - val_loss: 0.4419 - val_acc: 0.9122
Epoch 2/25
73/73 [==============================] - 39s 532ms/step - loss: 0.1316 - acc: 0.9538 - val_loss: 0.2397 - val_acc: 0.9382
Epoch 3/25
73/73 [==============================] - 41s 557ms/step - loss: 0.1191 - acc: 0.9570 - val_loss: 1.2767 - val_acc: 0.8832
Epoch 4/25
73/73 [==============================] - 39s 529ms/step - loss: 0.1093 - acc: 0.9605 - val_loss: 0.5535 - val_acc: 0.9276
Epoch 5/25
73/73 [==============================] - 39s 531ms/step - loss: 0.1226 - acc: 0.9575 - val_loss: 0.6102 - val_acc: 0.9175
Epoch 6/25
73/73 [==============================] - 39s 537ms/step - loss: 0.1196 - acc: 0.9577 - val_loss: 0.5770 - val_acc: 0.9063
Epoch 7/25
73/73 [==============================] - 39s 530ms/step - loss: 0.1023 - acc: 0.9632 - val_loss: 0.5210 - val_acc: 0.9248
Epoch 8/25
73/73 [==============================] - 40s 550ms/step - loss: 0.1026 - acc: 0.9633 - val_loss: 1.7190 - val_acc: 0.8799
Epoch 9/25
73/73 [==============================] - 40s 550ms/step - loss: 0.1238 - acc: 0.9570 - val_loss: 1.9751 - val_acc: 0.8761
Epoch 10/25
73/73 [==============================] - 41s 559ms/step - loss: 0.1035 - acc: 0.9632 - val_loss: 1.0540 - val_acc: 0.8950
Epoch 11/25
73/73 [==============================] - 40s 547ms/step - loss: 0.0818 - acc: 0.9706 - val_loss: 0.1059 - val_acc: 0.9669
Epoch 12/25
73/73 [==============================] - 39s 535ms/step - loss: 0.0766 - acc: 0.9734 - val_loss: 0.1692 - val_acc: 0.9611
Epoch 13/25
73/73 [==============================] - 39s 532ms/step - loss: 0.0661 - acc: 0.9745 - val_loss: 0.2866 - val_acc: 0.9345
Epoch 14/25
73/73 [==============================] - 39s 531ms/step - loss: 0.0703 - acc: 0.9750 - val_loss: 0.1917 - val_acc: 0.9548
Epoch 15/25
73/73 [==============================] - 38s 520ms/step - loss: 0.0589 - acc: 0.9777 - val_loss: 0.2264 - val_acc: 0.9445
Epoch 16/25
73/73 [==============================] - 38s 525ms/step - loss: 0.0614 - acc: 0.9778 - val_loss: 0.2053 - val_acc: 0.9468
Epoch 17/25
73/73 [==============================] - 38s 515ms/step - loss: 0.0552 - acc: 0.9798 - val_loss: 0.1171 - val_acc: 0.9606
Epoch 18/25
73/73 [==============================] - 39s 529ms/step - loss: 0.0496 - acc: 0.9821 - val_loss: 0.2475 - val_acc: 0.9452
Epoch 19/25
73/73 [==============================] - 38s 525ms/step - loss: 0.0500 - acc: 0.9812 - val_loss: 0.1003 - val_acc: 0.9710
Epoch 20/25
73/73 [==============================] - 38s 514ms/step - loss: 0.0487 - acc: 0.9815 - val_loss: 0.8733 - val_acc: 0.9042
Epoch 21/25
73/73 [==============================] - 37s 511ms/step - loss: 0.0457 - acc: 0.9822 - val_loss: 0.2438 - val_acc: 0.9513
Epoch 22/25
73/73 [==============================] - 37s 503ms/step - loss: 0.0428 - acc: 0.9845 - val_loss: 0.0650 - val_acc: 0.9790
Epoch 23/25
73/73 [==============================] - 37s 506ms/step - loss: 0.0431 - acc: 0.9845 - val_loss: 0.1222 - val_acc: 0.9655
Epoch 24/25
73/73 [==============================] - 37s 508ms/step - loss: 0.0402 - acc: 0.9856 - val_loss: 0.8248 - val_acc: 0.9109
Epoch 25/25
73/73 [==============================] - 38s 516ms/step - loss: 0.0465 - acc: 0.9826 - val_loss: 0.1148 - val_acc: 0.9677
```

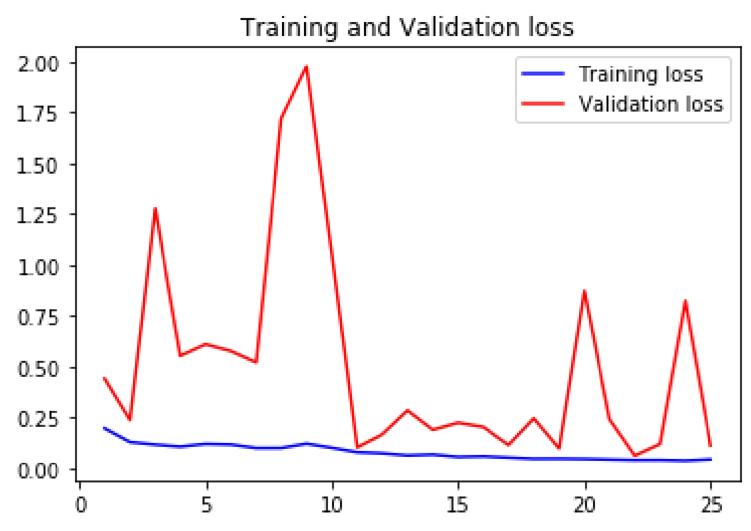
Plot the train and val curve

In [15]:

```
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()
```





Model Accuracy

In [16]:

```
print("[INFO] Calculating model accuracy")
scores = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {scores[1]*100}")
```

```
[INFO] Calculating model accuracy
591/591 [==============================] - 2s 3ms/step
Test Accuracy: 96.77383080755192
```

Save model using Pickle

In [17]:

```
# save the model to disk
print("[INFO] Saving model...")
pickle.dump(model,open('cnn_model.pkl', 'wb'))
```

```
[INFO] Saving model...
```