In [1]:

```
import tensorflow as tf
from tensorflow import keras
%matplotlib inline
import numpy as np
import pickle
import cv2
from os import listdir
from sklearn.preprocessing import LabelBinarizer
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D, BatchNor
from tensorflow.keras import backend as K
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import load_model
```

In [2]:

```
class MyCustomCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('acc') >= 0.96):
            print("Reached 95% accuracy so cancelling training!")
            self.model.stop_training = True
```

In [3]:

```
EPOCHS = 20
INIT_LR = 1e-3
BS = 15
default_image_size = tuple((256, 256))
image_size = 0
width=256
height=256
depth=3
train_dir=r"C:\Users\Glau\Desktop\DP\Pediastrum_cnn\Train"
valid_dir=r"C:\Users\Glau\Desktop\DP\Pediastrum_cnn\Test"
train_folder=listdir(train_dir)
valid_folder=listdir(valid_dir)
```

In [4]:

```
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
```

In [5]:

```
callbacks = MyCustomCallback()
```

In [6]:

```
train_image_list, train_image_label= [], []
for disease_folder in train_folder:
    print(f"processing {disease_folder} ...")
    disease_img_folder= listdir(f"{train_dir}/{disease_folder}")
    #print(disease_img_folder)
    for disease_img in disease_img_folder:
    #for disease_img in disease_img_folder[: : 2]:
        image_directory = f"{train_dir}/{disease_folder}/{disease_img}"
        if image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") ==
            train_image_list.append(convert_image_to_array(image_directory))
            train_image_label.append(disease_folder)
print("[INFO] Image loading completed")
```

```
processing Lacunastrum gracilimum ...
processing Monactinus simplex ...
processing Parapediastrum biradiatum ...
processing Pediastrum angulosum ...
processing Pediastrum duplex ...
processing Pseudopediastrum boryanum ...
processing Stauridium tetras ...
[INFO] Image loading completed
```

In [7]:

```
print(len(train_image_label))
```

```
33600
```

In [8]:

```
valid_image_list, valid_image_label= [], []
for disease_folder in valid_folder:
    print(f"processing {disease_folder} ...")
    disease_img_folder= listdir(f"{valid_dir}/{disease_folder}")

    for disease_img in disease_img_folder:
    #for disease_img in disease_img_folder[: : 2]:
        image_directory = f"{valid_dir}/{disease_folder}/{disease_img}"
        if image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") ==
            valid_image_list.append(convert_image_to_array(image_directory))
            valid_image_label.append(disease_folder)
print("[INFO] Image loading completed")
```

```
processing Lacunastrum gracilimum ...
processing Monactinus simplex ...
processing Parapediastrum biradiatum ...
processing Pediastrum angulosum ...
processing Pediastrum duplex ...
processing Pseudopediastrum boryanum ...
processing Stauridium tetras ...
[INFO] Image loading completed
```

In [9]:

```
print(len(valid_image_label))
```

```
8400
```

In [10]:

```
label_binarizer = LabelBinarizer()
bin_train_image_labels = label_binarizer.fit_transform(train_image_label)
bin_valid_image_labels = label_binarizer.fit_transform(valid_image_label)
pickle.dump(label_binarizer,open('Label_Instance.pkl', 'wb'))
n_classes = len(label_binarizer.classes_)
```

In [11]:

```
print(n_classes)
```

```
7
```

In [12]:

```
np_train_image_list = np.array(train_image_list, dtype=np.float16) / 255.0
np_valid_image_list = np.array(valid_image_list, dtype=np.float16) / 255.0
```

In [13]:

```
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
```

In [17]:

```
# coding: utf8
from keras import layers
from keras import models


#
# image dimensions
#

img_height = 256
img_width = 256
img_channels = 3

#
# network params
#

cardinality = 32


def residual_network(x):
    """
    ResNeXt by default. For ResNet set `cardinality` = 1 above.

    """
    def add_common_layers(y):
        y = layers.BatchNormalization()(y)
        y = layers.LeakyReLU()(y)

        return y

    def grouped_convolution(y, nb_channels, _strides):
        # when `cardinality` == 1 this is just a standard convolution
        if cardinality == 1:
            return layers.Conv2D(nb_channels, kernel_size=(3, 3), strides=_strides, padding

        assert not nb_channels % cardinality
        _d = nb_channels // cardinality

        # in a grouped convolution layer, input and output channels are divided into `cardi
        # and convolutions are separately performed within each group
        groups = []
        for j in range(cardinality):
            group = layers.Lambda(lambda z: z[:, :, :, j * _d:j * _d + _d])(y)
            groups.append(layers.Conv2D(_d, kernel_size=(3, 3), strides=_strides, padding='

        # the grouped convolutional layer concatenates them as the outputs of the layer
        y = layers.concatenate(groups)

        return y

    def residual_block(y, nb_channels_in, nb_channels_out, _strides=(1, 1), _project_shortc
        """
        Our network consists of a stack of residual blocks. These blocks have the same topo
        and are subject to two simple rules:

        - If producing spatial maps of the same size, the blocks share the same hyper-param
        - Each time the spatial map is down-sampled by a factor of 2, the width of the bloc
        """
        shortcut = y

        # we modify the residual building block as a bottleneck design to make the network
        y = layers.Conv2D(nb_channels_in, kernel_size=(1, 1), strides=(1, 1), padding='same
        y = add_common_layers(y)

        # ResNeXt (identical to ResNet when `cardinality` == 1)
        y = grouped_convolution(y, nb_channels_in, _strides=_strides)
        y = add_common_layers(y)

        y = layers.Conv2D(nb_channels_out, kernel_size=(1, 1), strides=(1, 1), padding='sam
        # batch normalization is employed after aggregating the transformations and before
        y = layers.BatchNormalization()(y)

        # identity shortcuts used directly when the input and output are of the same dimens
        if _project_shortcut or _strides != (1, 1):
            # when the dimensions increase projection shortcut is used to match dimensions
            # when the shortcuts go across feature maps of two sizes, they are performed wi
            shortcut = layers.Conv2D(nb_channels_out, kernel_size=(1, 1), strides=_strides,
            shortcut = layers.BatchNormalization()(shortcut)

        y = layers.add([shortcut, y])

        # relu is performed right after each batch normalization,
        # expect for the output of the block where relu is performed after the adding to th
        y = layers.LeakyReLU()(y)

        return y

    # conv1
    x = layers.Conv2D(256, kernel_size=(5, 5), strides=(2, 2), padding='same')(x)
    x = add_common_layers(x)

    # conv2
    x = layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)
    for i in range(3):
        project_shortcut = True if i == 0 else False
        x = residual_block(x, 64, 128, _project_shortcut=project_shortcut)

    # conv3
    for i in range(4):
        # down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2
        strides = (2, 2) if i == 0 else (1, 1)
        x = residual_block(x, 128, 256, _strides=strides)

    # conv4
    for i in range(6):
        strides = (2, 2) if i == 0 else (1, 1)
        x = residual_block(x, 256, 512, _strides=strides)

    # conv5
    for i in range(3):
        strides = (2, 2) if i == 0 else (1, 1)
        x = residual_block(x, 512, 2048, _strides=strides)

    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(7,activation='softmax')(x)

    return x
```

```python
image_tensor = layers.Input(shape=(img_height, img_width, img_channels))
network_output = residual_network(image_tensor)

model = models.Model(inputs=[image_tensor], outputs=[network_output])
print(model.summary())
```

```
__________________________________________________________________________
________________________
lambda_533 (Lambda)             (None, 64, 64, 2)    0          leaky_re_
lu_51[0][0]
__________________________________________________________________________
________________________
lambda_534 (Lambda)             (None, 64, 64, 2)    0          leaky_re_
lu_51[0][0]
__________________________________________________________________________
________________________
lambda_535 (Lambda)             (None, 64, 64, 2)    0          leaky_re_
lu_51[0][0]
__________________________________________________________________________
________________________
lambda_536 (Lambda)             (None, 64, 64, 2)    0          leaky_re_
lu_51[0][0]
__________________________________________________________________________
________________________
lambda_537 (Lambda)             (None, 64, 64, 2)    0          leaky_re_
lu_51[0][0]
```

In [18]:

```python
model.compile(loss="categorical_crossentropy", optimizer=opt,metrics=["accuracy"])

print("[INFO] training network...")
```

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

In [19]:

```
history=model.fit(np_train_image_list,bin_train_image_labels,
                  validation_data=(np_valid_image_list, bin_valid_image_labels),
                  batch_size=BS,
                  epochs=EPOCHS, verbose=1
                  )
```

```
Train on 33600 samples, validate on 8400 samples
Epoch 1/20
33600/33600 [==============================] - 1081s 32ms/step - loss: 0.227
7 - accuracy: 0.9321 - val_loss: 0.0825 - val_accuracy: 0.9813
Epoch 2/20
33600/33600 [==============================] - 1050s 31ms/step - loss: 0.032
7 - accuracy: 0.9906 - val_loss: 5.9535e-06 - val_accuracy: 1.0000
Epoch 3/20
33600/33600 [==============================] - 1056s 31ms/step - loss: 0.035
1 - accuracy: 0.9912 - val_loss: 0.0414 - val_accuracy: 0.9810
Epoch 4/20
33600/33600 [==============================] - 1055s 31ms/step - loss: 0.018
7 - accuracy: 0.9951 - val_loss: 2.4217e-05 - val_accuracy: 1.0000
Epoch 5/20
33600/33600 [==============================] - 1059s 32ms/step - loss: 0.010
4 - accuracy: 0.9973 - val_loss: 3.8912e-05 - val_accuracy: 1.0000
Epoch 6/20
33600/33600 [==============================] - 1056s 31ms/step - loss: 0.005
7 - accuracy: 0.9984 - val_loss: 3.0761e-06 - val_accuracy: 1.0000
Epoch 7/20
33600/33600 [==============================] - 1054s 31ms/step - loss: 0.005
4 - accuracy: 0.9984 - val_loss: 4.0429e-06 - val_accuracy: 1.0000
Epoch 8/20
33600/33600 [==============================] - 1056s 31ms/step - loss: 0.006
9 - accuracy: 0.9982 - val_loss: 2.0887e-05 - val_accuracy: 1.0000
Epoch 9/20
33600/33600 [==============================] - 1054s 31ms/step - loss: 0.004
7 - accuracy: 0.9989 - val_loss: 1.0020e-05 - val_accuracy: 1.0000
Epoch 10/20
33600/33600 [==============================] - 1056s 31ms/step - loss: 0.008
7 - accuracy: 0.9983 - val_loss: 1.1006e-05 - val_accuracy: 1.0000
Epoch 11/20
33600/33600 [==============================] - 1055s 31ms/step - loss: 4.283
7e-05 - accuracy: 1.0000 - val_loss: 1.9384e-06 - val_accuracy: 1.0000
Epoch 12/20
33600/33600 [==============================] - 1056s 31ms/step - loss: 0.003
6 - accuracy: 0.9990 - val_loss: 5.6585e-06 - val_accuracy: 1.0000
Epoch 13/20
33600/33600 [==============================] - 1057s 31ms/step - loss: 4.157
1e-05 - accuracy: 1.0000 - val_loss: 8.0148e-07 - val_accuracy: 1.0000
Epoch 14/20
33600/33600 [==============================] - 1060s 32ms/step - loss: 7.639
3e-06 - accuracy: 1.0000 - val_loss: 2.8479e-07 - val_accuracy: 1.0000
Epoch 15/20
33600/33600 [==============================] - 1059s 32ms/step - loss: 0.004
3 - accuracy: 0.9990 - val_loss: 1.6032e-06 - val_accuracy: 1.0000
Epoch 16/20
33600/33600 [==============================] - 1059s 32ms/step - loss: 9.429
9e-04 - accuracy: 0.9998 - val_loss: 6.0941e-07 - val_accuracy: 1.0000
Epoch 17/20
33600/33600 [==============================] - 1056s 31ms/step - loss: 9.568
5e-06 - accuracy: 1.0000 - val_loss: 1.9594e-07 - val_accuracy: 1.0000
Epoch 18/20
```

```
33600/33600 [==============================] - 1057s 31ms/step - loss: 4.850
1e-06 - accuracy: 1.0000 - val_loss: 7.1667e-08 - val_accuracy: 1.0000
Epoch 19/20
33600/33600 [==============================] - 1061s 32ms/step - loss: 0.002
6 - accuracy: 0.9996 - val_loss: 6.8702e-07 - val_accuracy: 1.0000
Epoch 20/20
33600/33600 [==============================] - 1058s 31ms/step - loss: 3.185
4e-05 - accuracy: 1.0000 - val_loss: 2.3124e-07 - val_accuracy: 1.0000
```

In [20]:

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

```
[INFO] Calculating model accuracy
8400/8400 [==============================] - 60s 7ms/step
Test Accuracy: 100.0
```

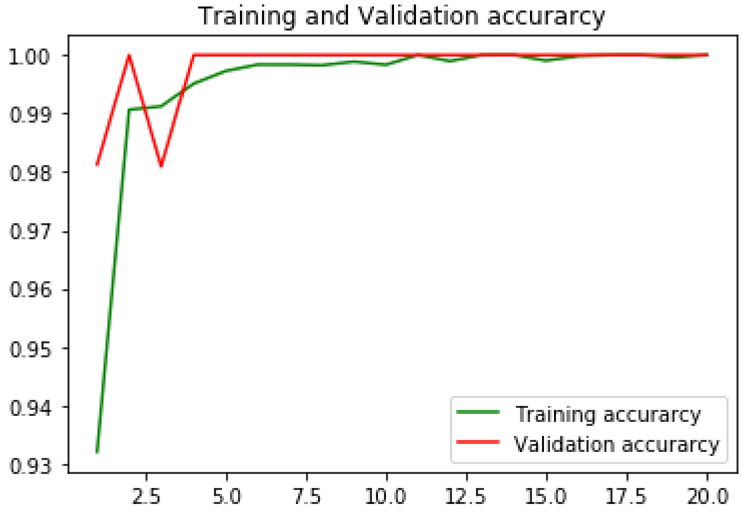
In [ ]:

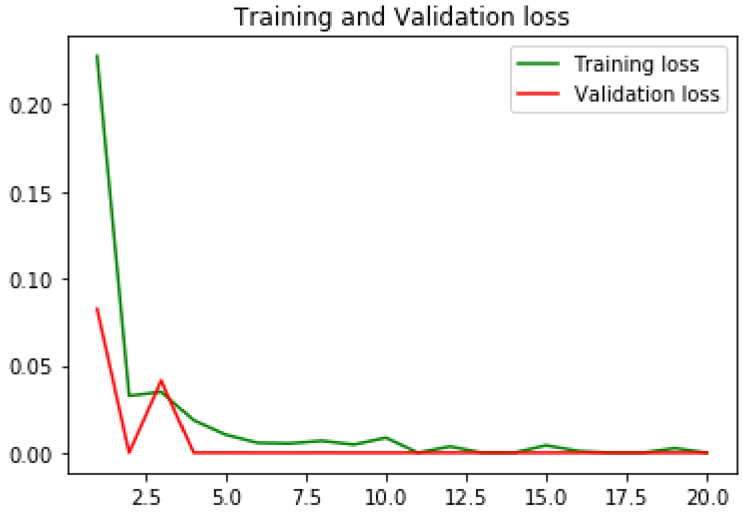
In [21]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

#Train and validation accuracy
plt.plot(epochs, acc, 'g', 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, 'g', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```





In [22]:

```
model.save("model_algae_ped.h5")
```

```
c:\users\glau\.conda\envs\pythongpu\lib\site-packages\keras\engine\saving.p
y:165: UserWarning: TensorFlow optimizers do not make it possible to access
optimizer attributes or optimizer state after instantiation. As a result, we
cannot save the optimizer as part of the model save file.You will have to co
mpile your model again after loading it. Prefer using a Keras optimizer inst
ead (see keras.io/optimizers).
  'TensorFlow optimizers do not '
```

In [ ]:

In [ ]: