Overview

Corn is susceptible to many diseases throughout the season; Gray Leaf Spot, Northern Corn Leaf Blight, and Common Rust are three of the most common. As explained by Bayer Crop Science, "Managing these diseases early is essential to keeping your corn crop healthy and protecting your yields."

Corn is an integral part of the agricultural ecosystem and both the American and global economy. According to the United States Grain Council, the US is the largest corn producer in the world, with 96,000,000 acres of land reserved for corn production. Considering how much of an impact corn has on agriculture and economy, it is imperative that farmers be able to avoid the occurrence of leaf diseases affecting their crops. With acres of corn to care for, it can be difficult and time-consuming to identify diseased plants. Since diseased corn plants are not able to be harvested, continuing to water them and care for them is a waste of resources and time

The following project uses deep learning to classify images of healthy and diseased corn leaves in the hopes of helping stakeholders to catch disease early. After identifying a disease, stakeholders can make intelligent decisions to ensure healthy crop in future seasons.

Data Understanding

Our data was sourced from https://www.kaggle.com/smaranjitghose/corn-or-maize-leaf-disease-dataset (https://www.kaggle.com/smaranjitghose/corn-or-maize-leaf-disease-dataset) . This is a dataset for classification of corn/maize plant leaf diseases. The images are in 4 folders, classified as follows:

1. Common Rust - 1306 images
2. Gray Leaf Spot - 574 images
3. Blight - 1146 images
4. Healthy - 1162 images

(The following disease explanations are from the Crop Protection Network, and more information can be found at https://cropprotectionnetwork.org/ (https://cropprotectionnetwork.org/).)

In [1]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.optimizers import RMSprop, Adam
from keras.models import Sequential
from keras.layers import Dense
from keras import models, layers
from keras.utils import plot_model
import numpy as np
import pandas as pd
from PIL import Image
import glob
import os, os.path, shutil
from keras.preprocessing.image import img_to_array, ImageDataGenerator, array_
to_img, load_img
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
import seaborn as sns
import pydot
import random
from dask import bag,  diagnostics
```

Basic EDA

In [2]:
```
#open first image and check size
im = Image.open('data/Blight/Corn_Blight (1).jpg')
arr = np.array(im)
arr.shape
```

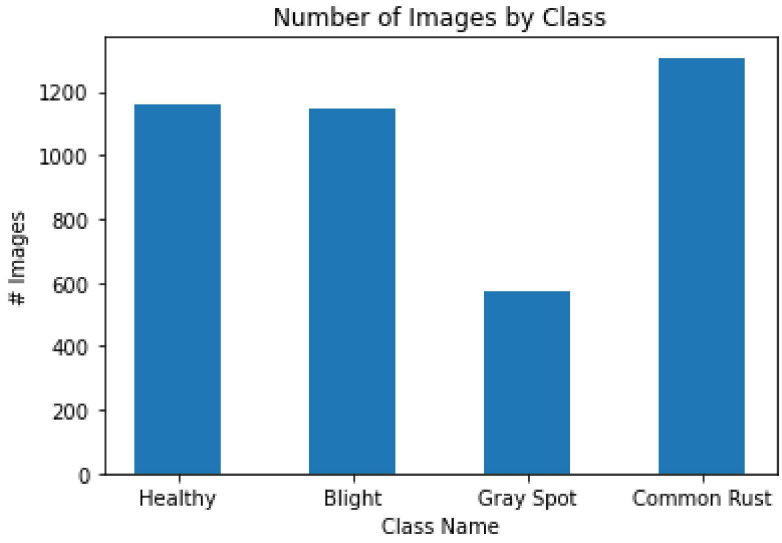
Out[2]: (371, 788, 3)

In [3]:
```
# Directory with Blight pictures
blight_dir = os.path.join('./data/Blight')

# Directory with Common Rust pictures
rust_dir = os.path.join('./data/Common_Rust')

# Directory with Gray Leaf Spot pictures
gray_dir = os.path.join('./data/Gray_Leaf_Spot')

# Directory with Healthy pictures
healthy_dir = os.path.join('./data/Healthy')
```

In [4]:
```
train_blight_names = os.listdir(blight_dir)
no_blight = len(os.listdir(blight_dir))
print('Total Blight Images:', len(os.listdir(blight_dir)))
print(train_blight_names[:5])
print('')

train_rust_names = os.listdir(rust_dir)
no_rust = len(os.listdir(rust_dir))
print('Total Common Rust Images:', len(os.listdir(rust_dir)))
print(train_rust_names[:5])
print('')

train_gray_names = os.listdir(gray_dir)
no_gray = len(os.listdir(gray_dir))
print('Total Gray Leaf Spot Images:', len(os.listdir(gray_dir)))
print(train_gray_names[:5])
print('')

train_healthy_names = os.listdir(healthy_dir)
no_healthy = len(os.listdir(healthy_dir))
print('Total Healthy Images:', len(os.listdir(healthy_dir)))
print(train_healthy_names[:5])
```

```
Total Blight Images: 1146
['Corn_Blight (1).jpg', 'Corn_Blight (1146).jpg', 'Corn_Blight (1147).jpg',
'Corn_Blight (1148).jpg', 'Corn_Blight (1149).jpg']

Total Common Rust Images: 1306
['Corn_Common_Rust (1).jpg', 'Corn_Common_Rust (10).jpg', 'Corn_Common_Rust
(100).JPG', 'Corn_Common_Rust (1000).JPG', 'Corn_Common_Rust (1001).JPG']

Total Gray Leaf Spot Images: 574
['Corn_Gray_Spot (1).jpg', 'Corn_Gray_Spot (10).jpg', 'Corn_Gray_Spot (100).J
PG', 'Corn_Gray_Spot (101).JPG', 'Corn_Gray_Spot (102).JPG']

Total Healthy Images: 1162
['Corn_Health (1).jpg', 'Corn_Health (10).jpg', 'Corn_Health (100).jpg', 'Cor
n_Health (1000).jpg', 'Corn_Health (1001).jpg']
```

In [5]:
```
#plot number of classes to identify possible imbalances
number_classes = {'Healthy': 1162,
                  'Blight': 1146,
                  'Gray Spot': 574,
                  'Common Rust': 1306}

plt.bar(number_classes.keys(), number_classes.values(), width = .5);
plt.title("Number of Images by Class");
plt.xlabel('Class Name');
plt.ylabel('# Images');
```



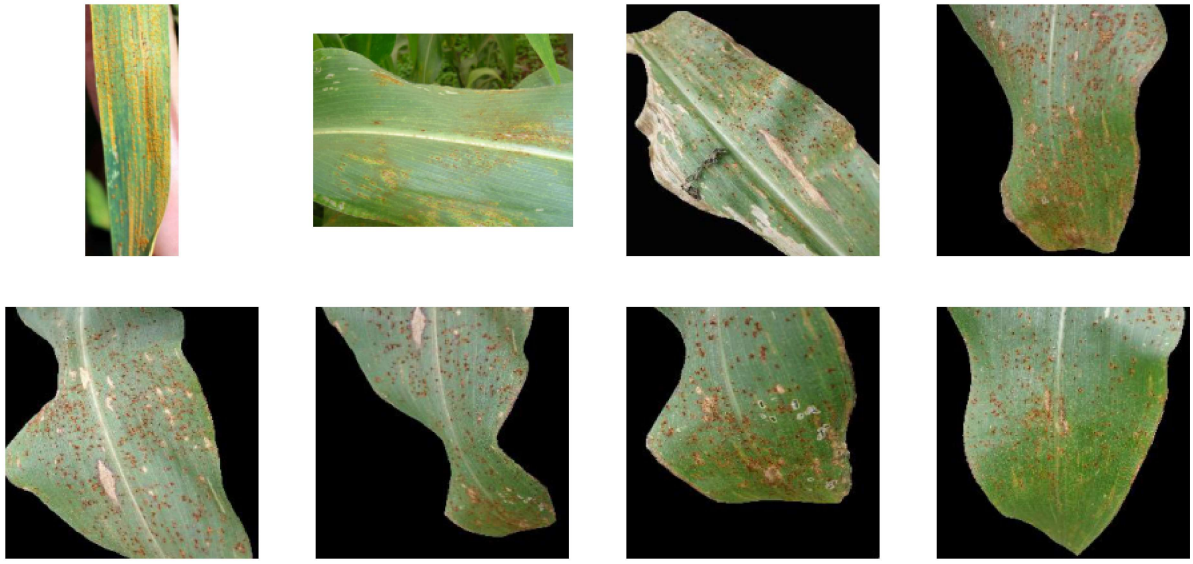
View sample images of each class

In [6]:
```
# Show images displayed 4x4
nrows = 4
ncols = 4

# Index for iterating over images
pic_index = 0

pic_index += 8
next_blight_pix = [os.path.join(blight_dir, fname)
                for fname in train_blight_names[pic_index-8:pic_index]]
next_rust_pix = [os.path.join(rust_dir, fname)
                for fname in train_rust_names[pic_index-8:pic_index]]
next_gray_pix = [os.path.join(gray_dir, fname)
                for fname in train_gray_names[pic_index-8:pic_index]]
next_healthy_pix = [os.path.join(healthy_dir, fname)
                for fname in train_healthy_names[pic_index-8:pic_index]]

def show_image_sample(pic_directory):

    '''Returns 4x2 image samples from given directory'''

    fig = plt.gcf()
    fig.set_size_inches(ncols * 4, nrows * 4)
    for i, img_path in enumerate(pic_directory):
        sp = plt.subplot(nrows, ncols, i + 1)
        sp.axis('Off')
        img = mpimg.imread(img_path)
        plt.imshow(img)
    plt.show()
```

In [7]:
```
# Showing sample of Blight images
show_image_sample(next_blight_pix)
```

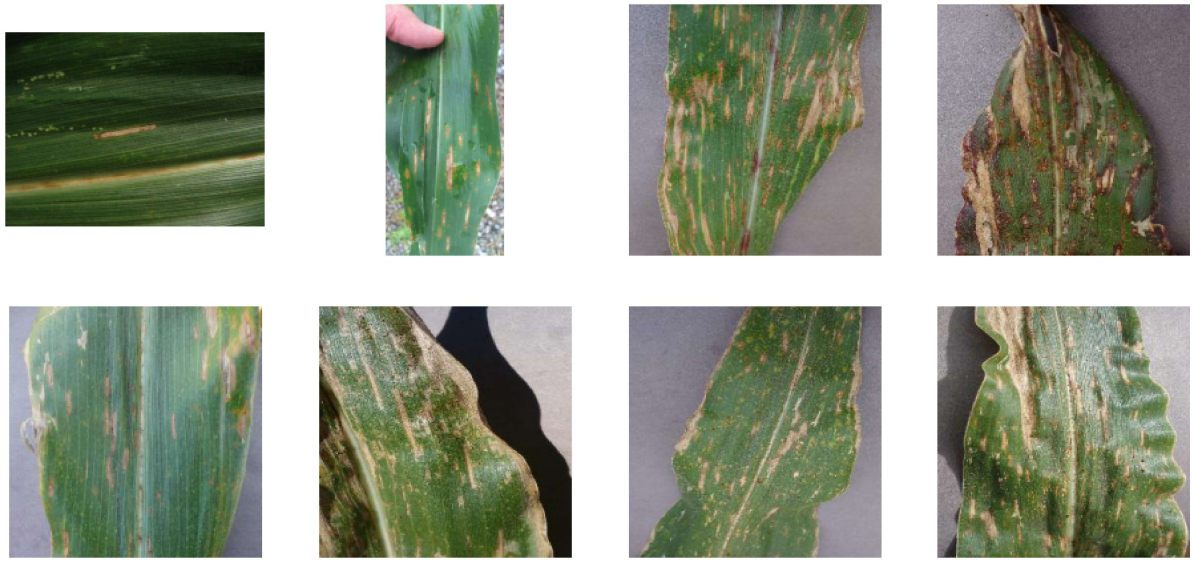
"Northern corn leaf blight (NCLB) is caused by the fungus Setosphaeria turcica. Symptoms usually appear first on the lower leaves. Leaf lesions are long (1 to 6 inches) and elliptical, gray-green at first but then turn pale gray or tan. Under moist conditions, dark gray spores are produced, usually on the lower leaf surface, which give lesions a "dirty" gray appearance. Entire leaves on severely blighted plants can die, so individual lesions are not visible. Lesions may occur on the outer husk of ears, but the kernels are not infected. On hybrids that contain an Ht gene for resistance to the fungus, lesions are smaller, chlorotic, and may develop into linear streaks. These lesions rarely produce spores." https://cropprotectionnetwork.org/ (https://cropprotectionnetwork.org/)

In [8]:
```
# Showing sample of Common Rust images
show_image_sample(next_rust_pix)
```



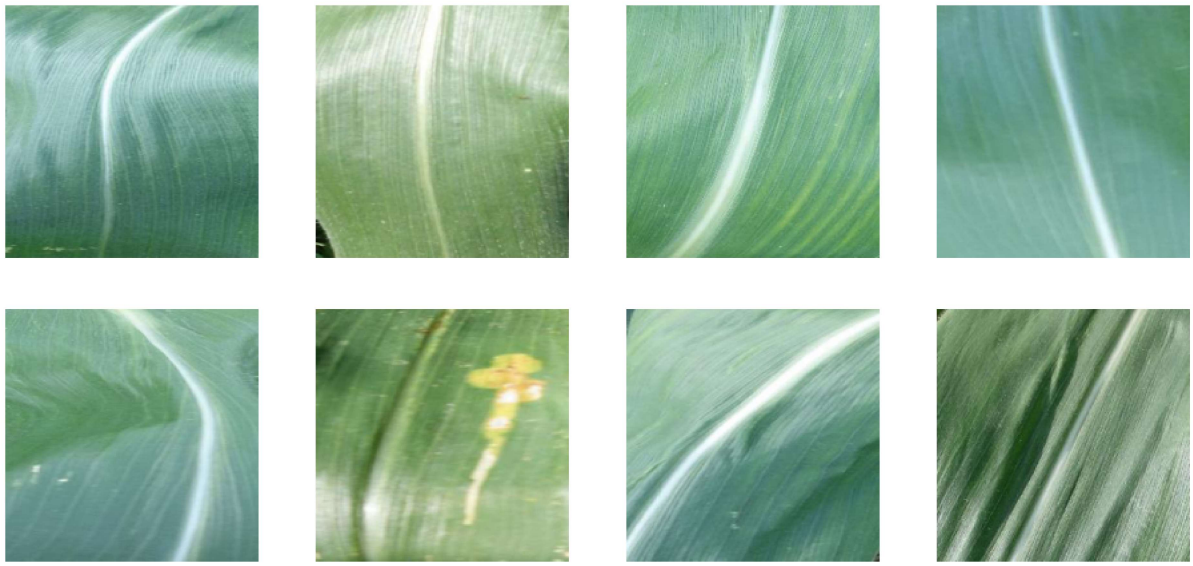
"Common rust is caused by the fungus Puccinia sorghi and occurs every growing season. It is seldom a concern in hybrid corn. Rust pustules usually first appear in late June. Early symptoms of common rust are chlorotic flecks on the leaf surface. These soon develop into powdery, brick-red pustules as the spores break through the leaf surface. Pustules are oval or elongated, about 1/8 inch long, and scattered sparsely or clustered together. The leaf tissue around the pustules may become yellow or die, leaving lesions of dead tissue. The lesions sometimes form a band across the leaf and entire leaves will die if severely infected. As the pustules age, the red spores turn black, so the pustules appear black, and continue to erupt through the leaf surface. Husks, leaf sheaths, and stalks also may be infected." https://cropprotectionnetwork.org/ (https://cropprotectionnetwork.org/)

In [9]:
```
# Showing sample of Gray Leaf Spot images
show_image_sample(next_gray_pix)
```



"Gray leaf spot, caused by the fungus Cercospora zeae-maydis, occurs virtually every growing season. If conditions favor disease development, economic losses can occur. Symptoms first appear on lower leaves about two to three weeks before tasseling. The leaf lesions are long (up to 2 inches), narrow, rectangular, and light tan colored. Later, the lesions can turn gray. They are usually delimited by leaf veins but can join together and kill entire leaves." https://cropprotectionnetwork.org/ (https://cropprotectionnetwork.org/)

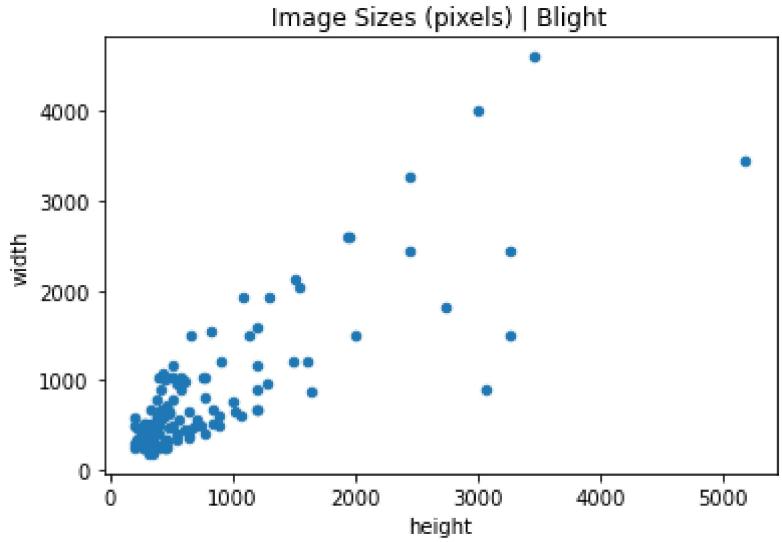
In [10]:
```
# Showing sample of Healthy images
show_image_sample(next_healthy_pix)
```

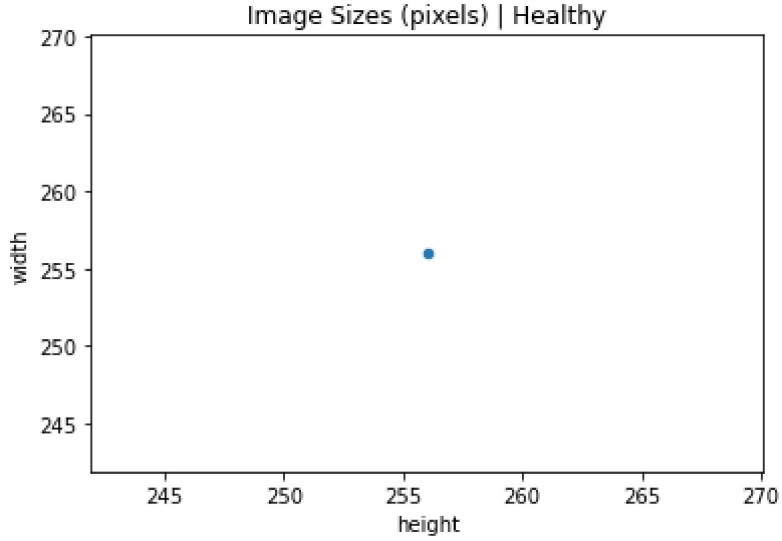


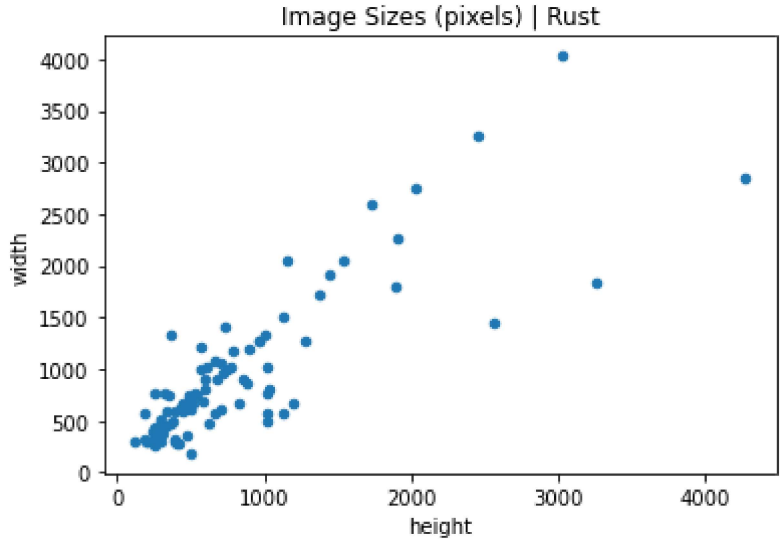
Visualize Image Size before Preprocessing/ Resizing

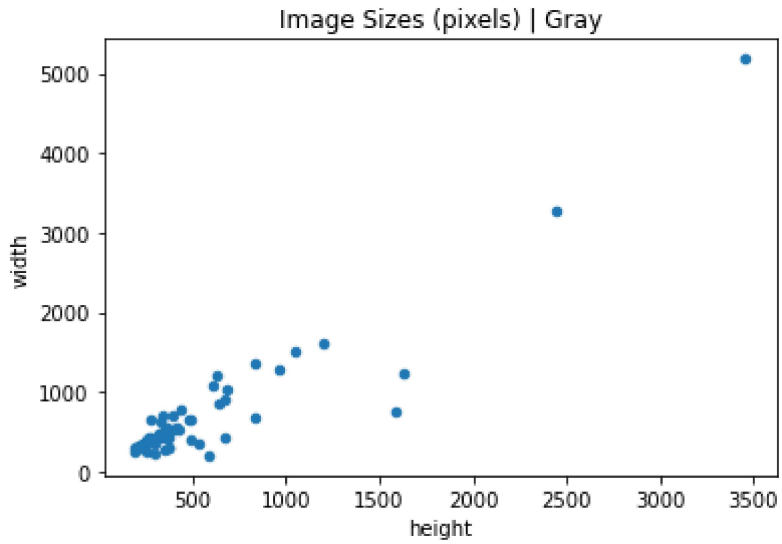
In [11]:
```
directories = {'Blight': 'data/Blight/',
               'Healthy': 'data/Healthy/',
               'Rust': 'data/Common_Rust/',
               'Gray': 'data/Gray_Leaf_Spot/'}

def get_dims(file):

    '''Returns dimensions of an RBG image'''

    im = Image.open(file)
    arr = np.array(im)
    h,w,d = arr.shape
    return h,w

for n,d in directories.items():
    filepath = d
    filelist = [filepath + f for f in os.listdir(filepath)]
    dimsbag = bag.from_sequence(filelist).map(get_dims)
    with diagnostics.ProgressBar():
        dims = dimsbag.compute()

    dim_df = pd.DataFrame(dims, columns=['height', 'width'])
    sizes = dim_df.groupby(['height', 'width']).size().reset_index().rename(co
lumns={0:'count'})
    sizes.plot.scatter(x='height', y='width');
    plt.title('Image Sizes (pixels) | {}'.format(n))
```

```
[########################################] | 100% Completed |  3.4s
[########################################] | 100% Completed |  1.1s
[########################################] | 100% Completed |  2.4s
[########################################] | 100% Completed |  1.9s
```









These images are many different sizes, resizing will need to be applied to all images before modeling.

Splitting Images to Train/ Test Directories

In [12]:
```
#grab image names for each type
imgs_blight = [file for file in os.listdir(blight_dir)]
imgs_gray = [file for file in os.listdir(gray_dir)]
imgs_rust = [file for file in os.listdir(rust_dir)]
imgs_healthy = [file for file in os.listdir(healthy_dir)]
```

In [13]:
```
new_dir = 'split/'
```

In [14]:
```
#Check to ensure all images are included
print('# Blight: ', len(imgs_blight))
print('# Gray: ', len(imgs_gray))
print('# Rust: ', len(imgs_rust))
print('# Healthy: ', len(imgs_healthy))
```

```
# Blight:  1146
# Gray:  574
# Rust:  1306
# Healthy:  1162
```

In [15]:
```
#os.mkdir(new_dir)

#create variables for new split directories for train/ test
train_folder = os.path.join(new_dir, 'train')
train_blight = os.path.join(train_folder, 'blight')
train_gray = os.path.join(train_folder, 'gray')
train_rust = os.path.join(train_folder, 'rust')
train_healthy = os.path.join(train_folder, 'healthy')

test_folder = os.path.join(new_dir, 'test')
test_blight = os.path.join(test_folder, 'blight')
test_gray = os.path.join(test_folder, 'gray')
test_rust = os.path.join(test_folder, 'rust')
test_healthy = os.path.join(test_folder, 'healthy')
```

In [16]:
```
# #create directories for splits
# os.mkdir(train_folder)
# os.mkdir(train_blight)
# os.mkdir(train_gray)
# os.mkdir(train_rust)
# os.mkdir(train_healthy)

# os.mkdir(test_folder)
# os.mkdir(test_blight)
# os.mkdir(test_gray)
# os.mkdir(test_rust)
# os.mkdir(test_healthy)
```

In [17]:
```
#use a 74/13/13 split for train/ test
print('Number of images to test:')
print('# Blight: ', round(len(imgs_blight)*.13))
print('# Gray: ', round(len(imgs_gray)*.13))
print('# Rust: ', round(len(imgs_rust)*.13))
print('# Healthy: ', round(len(imgs_healthy)*.13))
```

```
Number of images to test:
# Blight:  149
# Gray:  75
# Rust:  170
# Healthy:  151
```

In [18]:
```
# #train blight
# imgs = imgs_blight[149:]
# for img in imgs:
#     origin = os.path.join(blight_dir, img)
#     destination = os.path.join(train_blight, img)
#     shutil.copyfile(origin, destination)

# #train grey
# imgs = imgs_gray[75:]
# for img in imgs:
#     origin = os.path.join(gray_dir, img)
#     destination = os.path.join(train_gray, img)
#     shutil.copyfile(origin, destination)

# #train rust
# imgs = imgs_rust[170:]
# for img in imgs:
#     origin = os.path.join(rust_dir, img)
#     destination = os.path.join(train_rust, img)
#     shutil.copyfile(origin, destination)

# #train healthy
# imgs = imgs_healthy[151:]
# for img in imgs:
#     origin = os.path.join(healthy_dir, img)
#     destination = os.path.join(train_healthy, img)
#     shutil.copyfile(origin, destination)
```

In [19]:
```
# # test blight
# imgs = imgs_blight[:149]
# for img in imgs:
#     origin = os.path.join(blight_dir, img)
#     destination = os.path.join(test_blight, img)
#     shutil.copyfile(origin, destination)

# # test grey
# imgs = imgs_gray[:75]
# for img in imgs:
#     origin = os.path.join(gray_dir, img)
#     destination = os.path.join(test_gray, img)
#     shutil.copyfile(origin, destination)

# # test rust
# imgs = imgs_rust[:170]
# for img in imgs:
#     origin = os.path.join(rust_dir, img)
#     destination = os.path.join(test_rust, img)
#     shutil.copyfile(origin, destination)

# # test healthy
# imgs = imgs_healthy[:151]
# for img in imgs:
#     origin = os.path.join(healthy_dir, img)
#     destination = os.path.join(test_healthy, img)
#     shutil.copyfile(origin, destination)
```

In [20]:
```
classes = ['blight','rust','gray','healthy']
num_epochs = 30
batch_size = 32
```

In [21]:
```
# Rescaling images by 1./255
# Splitting data into training and test sets
train_datagen = ImageDataGenerator(rescale = 1/255,
                                   validation_split = 0.2)

# Flow training images in batches
train_generator = train_datagen.flow_from_directory(
        './split/train',
        target_size=(200, 200),
        batch_size=batch_size,
        classes = classes,
        class_mode='categorical',
        subset='training')

# Flow validation images in batches
validation_generator = train_datagen.flow_from_directory(
        './split/train',
        target_size=(200, 200),
        batch_size=batch_size,
        classes = classes,
        class_mode='categorical',
        subset='validation')
```

```
Found 2916 images belonging to 4 classes.
Found 727 images belonging to 4 classes.
```

In [22]:
```
#create images/ labels
train_images, train_labels = next(train_generator)
```

In [23]:
```
train_images.shape[0]
```

Out[23]:
```
32
```

In [24]:
```
# train_img = train_images.reshape(train_images.shape[0], -1)
# test_img = test_images.reshape(test_images.shape[0], -1)
# val_img = val_images.reshape(val_images.shape[0], -1)

# #double check shape to make sure all the images are included
# print(train_img.shape)
# print(test_img.shape)
# print(val_img.shape)
```

In [25]:
```
##create labels for all three splits
#train_y = np.reshape(train_labels[:,0], (3099,1))
```

Modeling with CNN's

Baseline Model: CNN with a Single Convolutional Layer & Softmax Output

In [26]:
```
classes = ['blight','rust','gray','healthy']
num_epochs = 30
batch_size = 32

# Training sample count
training_sample=train_generator.n

# Validation sample count
val_sample=validation_generator.n
```

In [27]:
```
training_sample
```

Out[27]: 2916

In [28]:
```
#first CNN with 2 convolutional layers

input_shape = (64, 64, 3)
x = tf.random.normal(input_shape)

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(200, 200, 3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (5,5), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(4, activation='softmax'))

#Compiling with Adam optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(lr=.003),
                loss="categorical_crossentropy",
                metrics='accuracy')
```

In [29]:
```
history_cnn = model.fit(train_generator,
                    epochs=30,
                    batch_size=batch_size,
                    steps_per_epoch=int((training_sample//batch_size)-1),
                    validation_data= validation_generator)
```

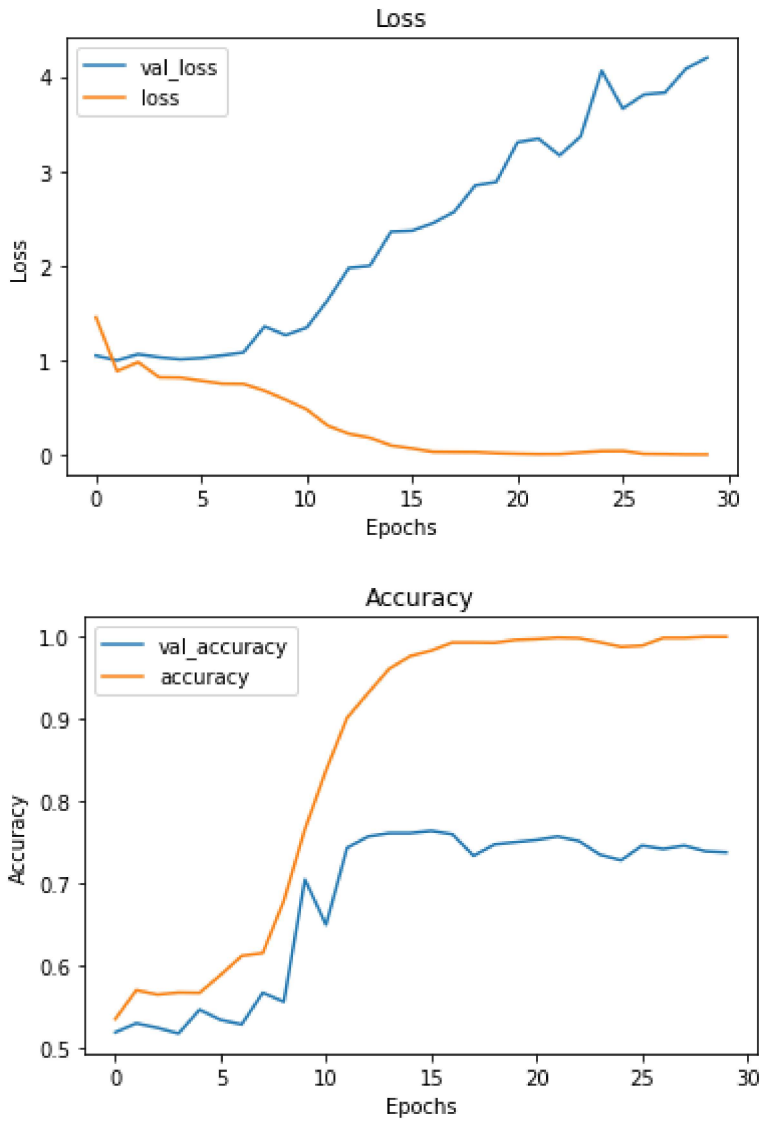
```
Epoch 1/30
90/90 [==============================] - 93s 1s/step - loss: 1.4550 - accurac
y: 0.5347 - val_loss: 1.0489 - val_accuracy: 0.5186
Epoch 2/30
90/90 [==============================] - 91s 1s/step - loss: 0.8857 - accurac
y: 0.5698 - val_loss: 0.9991 - val_accuracy: 0.5296
Epoch 3/30
90/90 [==============================] - 92s 1s/step - loss: 0.9806 - accurac
y: 0.5645 - val_loss: 1.0647 - val_accuracy: 0.5241
Epoch 4/30
90/90 [==============================] - 92s 1s/step - loss: 0.8199 - accurac
y: 0.5670 - val_loss: 1.0316 - val_accuracy: 0.5172
Epoch 5/30
90/90 [==============================] - 92s 1s/step - loss: 0.8164 - accurac
y: 0.5666 - val_loss: 1.0104 - val_accuracy: 0.5461
Epoch 6/30
90/90 [==============================] - 92s 1s/step - loss: 0.7833 - accurac
y: 0.5884 - val_loss: 1.0240 - val_accuracy: 0.5337
Epoch 7/30
90/90 [==============================] - 92s 1s/step - loss: 0.7525 - accurac
y: 0.6115 - val_loss: 1.0517 - val_accuracy: 0.5282
Epoch 8/30
90/90 [==============================] - 92s 1s/step - loss: 0.7498 - accurac
y: 0.6150 - val_loss: 1.0828 - val_accuracy: 0.5667
Epoch 9/30
90/90 [==============================] - 93s 1s/step - loss: 0.6780 - accurac
y: 0.6785 - val_loss: 1.3578 - val_accuracy: 0.5557
Epoch 10/30
90/90 [==============================] - 93s 1s/step - loss: 0.5832 - accurac
y: 0.7654 - val_loss: 1.2656 - val_accuracy: 0.7043
Epoch 11/30
90/90 [==============================] - 92s 1s/step - loss: 0.4803 - accurac
y: 0.8370 - val_loss: 1.3456 - val_accuracy: 0.6492
Epoch 12/30
90/90 [==============================] - 93s 1s/step - loss: 0.3088 - accurac
y: 0.9008 - val_loss: 1.6371 - val_accuracy: 0.7428
Epoch 13/30
90/90 [==============================] - 92s 1s/step - loss: 0.2232 - accurac
y: 0.9309 - val_loss: 1.9755 - val_accuracy: 0.7565
Epoch 14/30
90/90 [==============================] - 93s 1s/step - loss: 0.1807 - accurac
y: 0.9604 - val_loss: 1.9979 - val_accuracy: 0.7607
Epoch 15/30
90/90 [==============================] - 93s 1s/step - loss: 0.0994 - accurac
y: 0.9758 - val_loss: 2.3591 - val_accuracy: 0.7607
Epoch 16/30
90/90 [==============================] - 96s 1s/step - loss: 0.0679 - accurac
y: 0.9825 - val_loss: 2.3692 - val_accuracy: 0.7634
Epoch 17/30
90/90 [==============================] - 95s 1s/step - loss: 0.0317 - accurac
y: 0.9923 - val_loss: 2.4501 - val_accuracy: 0.7593
Epoch 18/30
90/90 [==============================] - 96s 1s/step - loss: 0.0298 - accurac
y: 0.9923 - val_loss: 2.5690 - val_accuracy: 0.7331
Epoch 19/30
90/90 [==============================] - 99s 1s/step - loss: 0.0292 - accurac
y: 0.9919 - val_loss: 2.8482 - val_accuracy: 0.7469
```

```
Epoch 20/30
90/90 [==============================] - 111s 1s/step - loss: 0.0184 - accura
cy: 0.9954 - val_loss: 2.8847 - val_accuracy: 0.7497
Epoch 21/30
90/90 [==============================] - 103s 1s/step - loss: 0.0126 - accura
cy: 0.9965 - val_loss: 3.3036 - val_accuracy: 0.7524
Epoch 22/30
90/90 [==============================] - 101s 1s/step - loss: 0.0084 - accura
cy: 0.9982 - val_loss: 3.3422 - val_accuracy: 0.7565
Epoch 23/30
90/90 [==============================] - 105s 1s/step - loss: 0.0093 - accura
cy: 0.9975 - val_loss: 3.1652 - val_accuracy: 0.7510
Epoch 24/30
90/90 [==============================] - 113s 1s/step - loss: 0.0251 - accura
cy: 0.9926 - val_loss: 3.3656 - val_accuracy: 0.7345
Epoch 25/30
90/90 [==============================] - 117s 1s/step - loss: 0.0400 - accura
cy: 0.9870 - val_loss: 4.0596 - val_accuracy: 0.7276
Epoch 26/30
90/90 [==============================] - 115s 1s/step - loss: 0.0413 - accura
cy: 0.9884 - val_loss: 3.6605 - val_accuracy: 0.7455
Epoch 27/30
90/90 [==============================] - 113s 1s/step - loss: 0.0110 - accura
cy: 0.9979 - val_loss: 3.8088 - val_accuracy: 0.7414
Epoch 28/30
90/90 [==============================] - 100s 1s/step - loss: 0.0089 - accura
cy: 0.9979 - val_loss: 3.8292 - val_accuracy: 0.7455
Epoch 29/30
90/90 [==============================] - 96s 1s/step - loss: 0.0047 - accurac
y: 0.9993 - val_loss: 4.0826 - val_accuracy: 0.7387
Epoch 30/30
90/90 [==============================] - 101s 1s/step - loss: 0.0042 - accura
cy: 0.9993 - val_loss: 4.1975 - val_accuracy: 0.7373
```

In [30]:
```
def viz_train_res(results):

    ''' plots training and validation accuracy and loss using neural network r
esults.
    requires both training and validation accuracy and loss scores. '''

    history = results.history
    plt.figure()
    plt.plot(history['val_loss'])
    plt.plot(history['loss'])
    plt.legend(['val_loss', 'loss'])
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.show()

    plt.figure()
    plt.plot(history['val_accuracy'])
    plt.plot(history['accuracy'])
    plt.legend(['val_accuracy', 'accuracy'])
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.show()
```

In [53]:
```
#adjusting the axis

def viz_train_res2(results):
    history = results.history
    plt.figure()
    plt.axis(ymax = 1)
    plt.plot(history['val_loss'])
    plt.plot(history['loss'])
    plt.legend(['val_loss', 'loss'])
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.show()

    plt.figure()
    plt.axis(ymin = 0)
    plt.plot(history['val_accuracy'])
    plt.plot(history['accuracy'])
    plt.legend(['val_accuracy', 'accuracy'])
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.show()
```

In [31]:
```
viz_train_res(history_cnn)
```



While training accuracy is very high, this model is overfitting and not performing as well on the validation set. Overfitting on training data is common in neural networks, so this issue isn't surpising. However, it poses an issue as the model won't perform as well on data it hasn't seen before - meaning this model would not perform well in the wild. The following models will focus on improving validation performance.

In [32]:
```
#CNN with a single convolutional layer to combat overfitting

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(200, 200
, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    # 4 output neurons for 4 classes
    tf.keras.layers.Dense(4, activation='softmax')
])
```

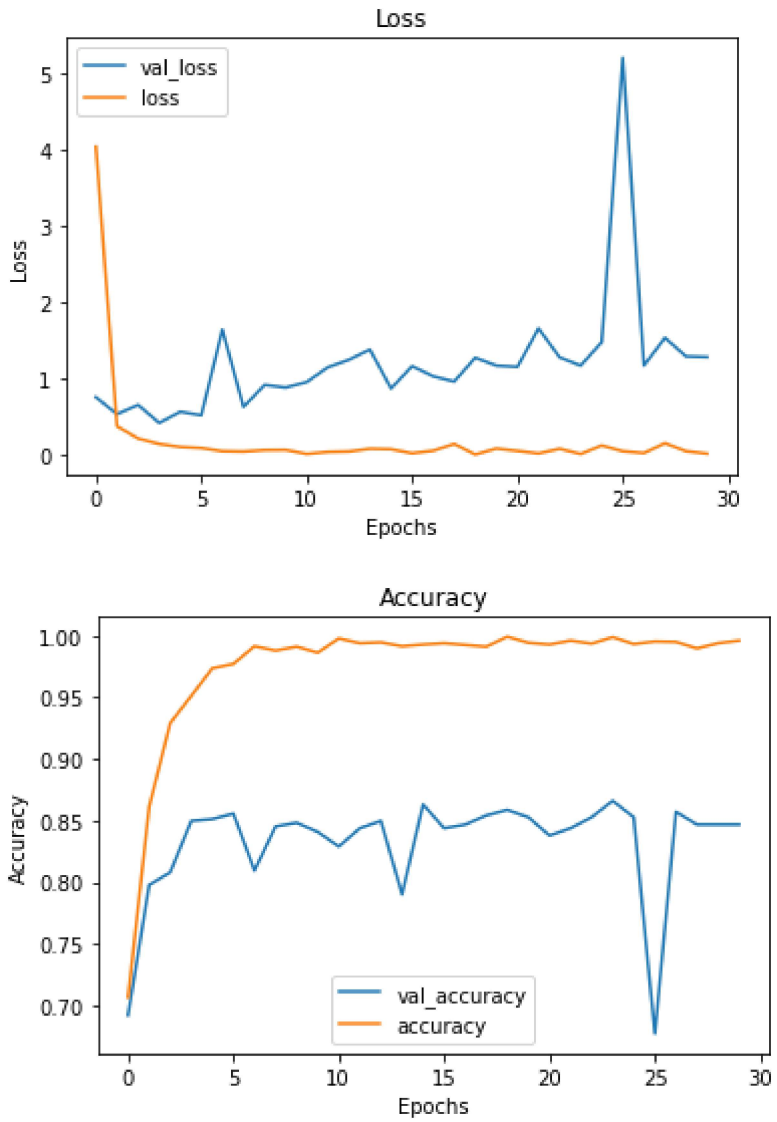
In [33]:
```
# Compiling with RMSprop optimizer
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['accuracy'])
```

In [34]:
```
# Training model
history_cnn2 = model.fit(train_generator,
    steps_per_epoch=int((training_sample//batch_size)-1),
    validation_data = (validation_generator),
    validation_steps = int((val_sample//batch_size)-1),
    epochs=num_epochs,
    verbose=1)
```

```
Epoch 1/30
90/90 [==============================] - 39s 434ms/step - loss: 4.0381 - accu
racy: 0.7065 - val_loss: 0.7566 - val_accuracy: 0.6920
Epoch 2/30
90/90 [==============================] - 39s 428ms/step - loss: 0.3776 - accu
racy: 0.8612 - val_loss: 0.5358 - val_accuracy: 0.7976
Epoch 3/30
90/90 [==============================] - 39s 435ms/step - loss: 0.2154 - accu
racy: 0.9292 - val_loss: 0.6565 - val_accuracy: 0.8080
Epoch 4/30
90/90 [==============================] - 38s 427ms/step - loss: 0.1448 - accu
racy: 0.9513 - val_loss: 0.4201 - val_accuracy: 0.8497
Epoch 5/30
90/90 [==============================] - 38s 424ms/step - loss: 0.1053 - accu
racy: 0.9737 - val_loss: 0.5664 - val_accuracy: 0.8512
Epoch 6/30
90/90 [==============================] - 39s 432ms/step - loss: 0.0914 - accu
racy: 0.9772 - val_loss: 0.5208 - val_accuracy: 0.8557
Epoch 7/30
90/90 [==============================] - 39s 428ms/step - loss: 0.0500 - accu
racy: 0.9916 - val_loss: 1.6417 - val_accuracy: 0.8095
Epoch 8/30
90/90 [==============================] - 38s 424ms/step - loss: 0.0453 - accu
racy: 0.9881 - val_loss: 0.6303 - val_accuracy: 0.8452
Epoch 9/30
90/90 [==============================] - 38s 425ms/step - loss: 0.0645 - accu
racy: 0.9912 - val_loss: 0.9183 - val_accuracy: 0.8482
Epoch 10/30
90/90 [==============================] - 39s 431ms/step - loss: 0.0666 - accu
racy: 0.9863 - val_loss: 0.8845 - val_accuracy: 0.8408
Epoch 11/30
90/90 [==============================] - 39s 438ms/step - loss: 0.0141 - accu
racy: 0.9979 - val_loss: 0.9551 - val_accuracy: 0.8289
Epoch 12/30
90/90 [==============================] - 40s 446ms/step - loss: 0.0399 - accu
racy: 0.9940 - val_loss: 1.1492 - val_accuracy: 0.8438
Epoch 13/30
90/90 [==============================] - 38s 428ms/step - loss: 0.0460 - accu
racy: 0.9947 - val_loss: 1.2467 - val_accuracy: 0.8497
Epoch 14/30
90/90 [==============================] - 39s 429ms/step - loss: 0.0817 - accu
racy: 0.9916 - val_loss: 1.3824 - val_accuracy: 0.7902
Epoch 15/30
90/90 [==============================] - 38s 424ms/step - loss: 0.0757 - accu
racy: 0.9930 - val_loss: 0.8707 - val_accuracy: 0.8631
Epoch 16/30
90/90 [==============================] - 38s 426ms/step - loss: 0.0227 - accu
racy: 0.9940 - val_loss: 1.1641 - val_accuracy: 0.8438
Epoch 17/30
90/90 [==============================] - 38s 426ms/step - loss: 0.0565 - accu
racy: 0.9926 - val_loss: 1.0315 - val_accuracy: 0.8467
Epoch 18/30
90/90 [==============================] - 38s 425ms/step - loss: 0.1442 - accu
racy: 0.9912 - val_loss: 0.9624 - val_accuracy: 0.8542
Epoch 19/30
90/90 [==============================] - 39s 431ms/step - loss: 0.0068 - accu
racy: 0.9993 - val_loss: 1.2746 - val_accuracy: 0.8586
```

```
Epoch 20/30
90/90 [==============================] - 39s 431ms/step - loss: 0.0843 - accu
racy: 0.9944 - val_loss: 1.1717 - val_accuracy: 0.8527
Epoch 21/30
90/90 [==============================] - 38s 426ms/step - loss: 0.0540 - accu
racy: 0.9930 - val_loss: 1.1557 - val_accuracy: 0.8378
Epoch 22/30
90/90 [==============================] - 38s 427ms/step - loss: 0.0205 - accu
racy: 0.9961 - val_loss: 1.6591 - val_accuracy: 0.8438
Epoch 23/30
90/90 [==============================] - 39s 430ms/step - loss: 0.0818 - accu
racy: 0.9937 - val_loss: 1.2796 - val_accuracy: 0.8527
Epoch 24/30
90/90 [==============================] - 39s 431ms/step - loss: 0.0135 - accu
racy: 0.9989 - val_loss: 1.1714 - val_accuracy: 0.8661
Epoch 25/30
90/90 [==============================] - 38s 425ms/step - loss: 0.1248 - accu
racy: 0.9933 - val_loss: 1.4784 - val_accuracy: 0.8527
Epoch 26/30
90/90 [==============================] - 38s 424ms/step - loss: 0.0496 - accu
racy: 0.9954 - val_loss: 5.2042 - val_accuracy: 0.6771
Epoch 27/30
90/90 [==============================] - 38s 417ms/step - loss: 0.0257 - accu
racy: 0.9951 - val_loss: 1.1727 - val_accuracy: 0.8571
Epoch 28/30
90/90 [==============================] - 39s 430ms/step - loss: 0.1540 - accu
racy: 0.9898 - val_loss: 1.5356 - val_accuracy: 0.8467
Epoch 29/30
90/90 [==============================] - 39s 428ms/step - loss: 0.0500 - accu
racy: 0.9940 - val_loss: 1.2914 - val_accuracy: 0.8467
Epoch 30/30
90/90 [==============================] - 38s 425ms/step - loss: 0.0182 - accu
racy: 0.9961 - val_loss: 1.2833 - val_accuracy: 0.8467
```

In [35]:
```
viz_train_res(history_cnn2)
```

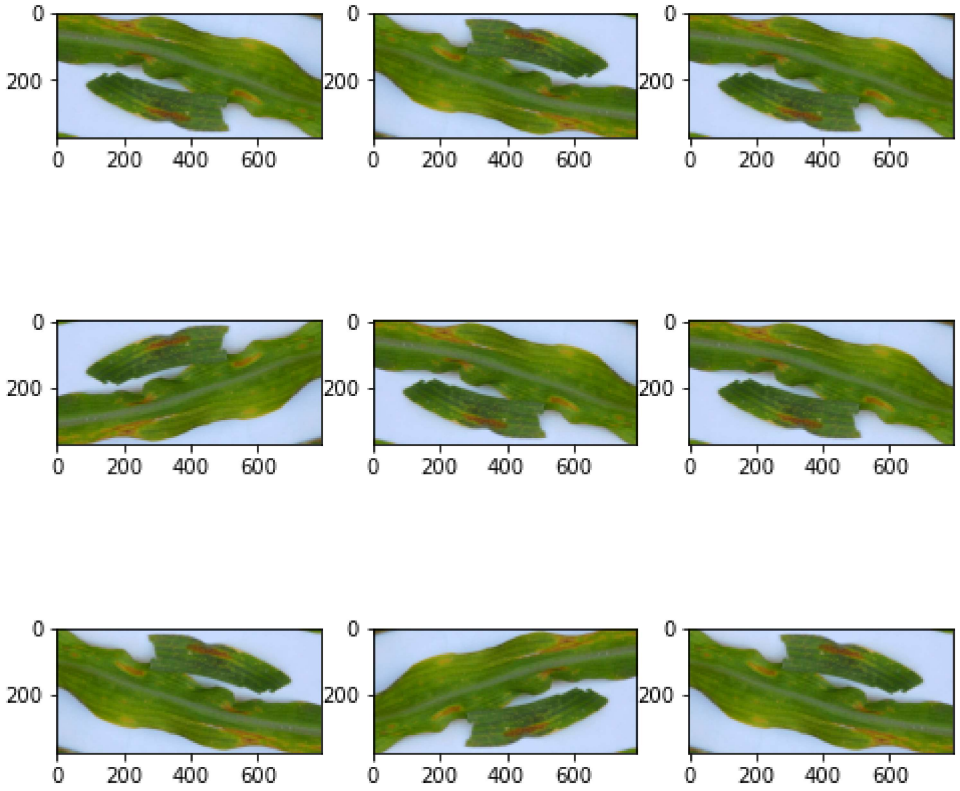


Reducing the number of layers is often an easy way to solve for overfitting. We can see here that while it certainly helped reduce overfitting, there is still more work to be done. Overall, our validation loss isn't as high. Validation accuracy is still suffering from overfitting.

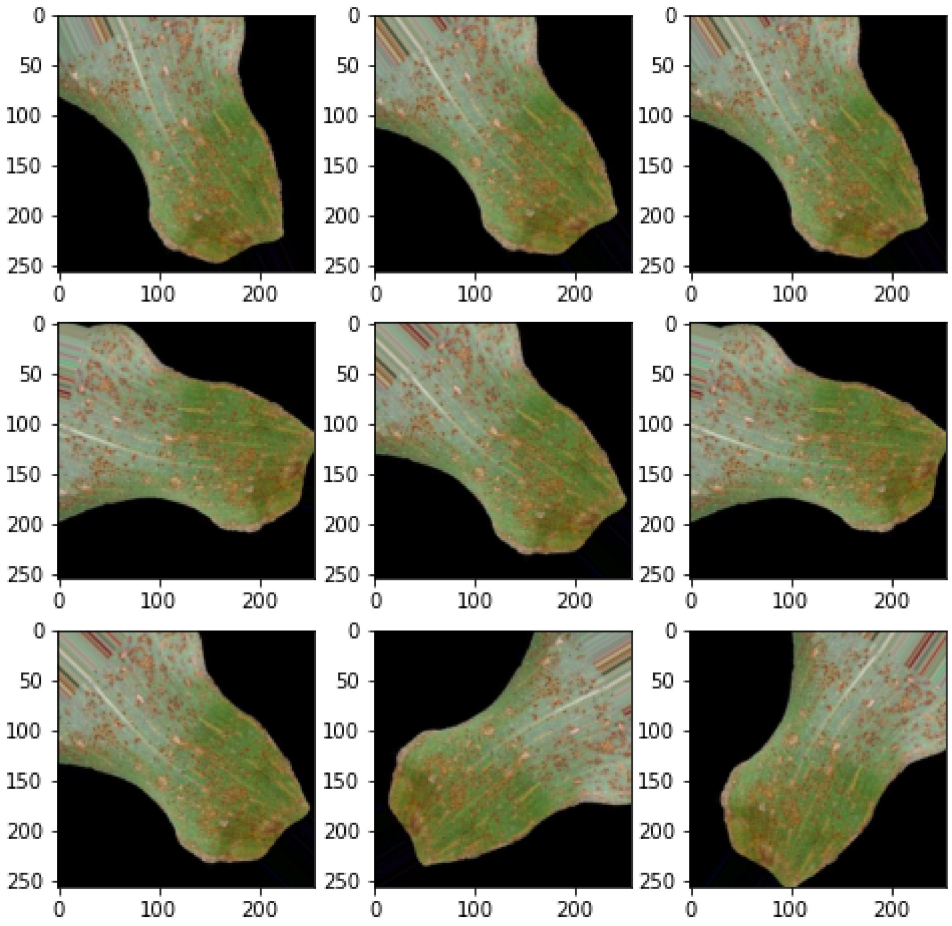
Reloading Images and Applying Data Augmentation

Another way to combat overfitting in image classification is with data augmentation. This process takes an image and changes it in various ways so that the model learns to differentiate to images in many different formats and appearances.

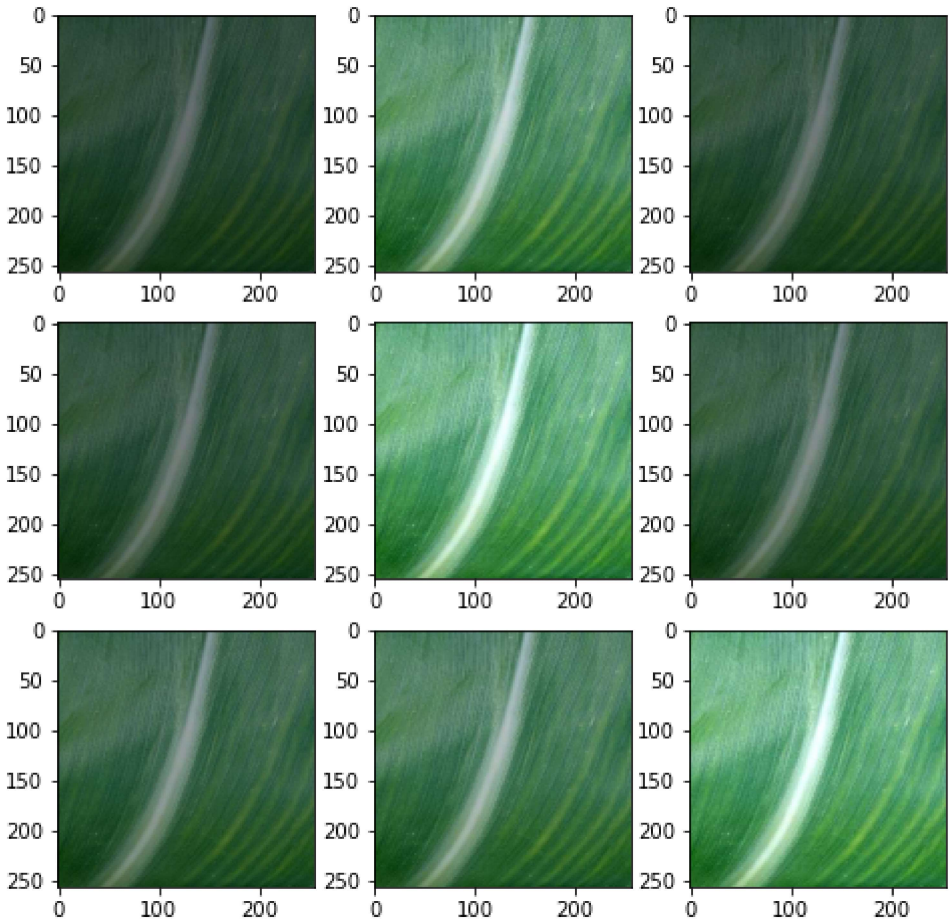
In [36]:
```
# Blight leaf image
img = load_img('data/Blight/Corn_Blight (1).jpg')
data = img_to_array(img)
samples = np.expand_dims(data, 0)
# Demonstrating horizontal and vertical flips
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip=True)
it = datagen.flow(samples, batch_size=1)

# Generate samples and plot
fig=plt.gcf()
fig.set_size_inches(ncols * 2, nrows * 2)
for i in range(9):
    plt.subplot(330 + 1 + i)
    batch = it.next()
    image = batch[0].astype('uint8')
    plt.imshow(image)
plt.show()
```



In [37]:
```
# Data augmentation example
# Common Rust leaf image
img = load_img('./data/Common_Rust/Corn_Common_Rust (1000).JPG')
data = img_to_array(img)
samples = np.expand_dims(data, 0)
# Demonstrating random rotation
datagen = ImageDataGenerator(rotation_range=90, fill_mode='nearest')
it = datagen.flow(samples, batch_size=1)

# Generate samples and plot
fig=plt.gcf()
fig.set_size_inches(ncols * 2, nrows * 2)
for i in range(9):
    plt.subplot(330 + 1 + i)
    batch = it.next()
    image = batch[0].astype('uint8')
    plt.imshow(image)
plt.show()
```



In [38]:
```
# Healthy leaf image
img = load_img('./data/Healthy/Corn_Health (100).jpg')
data = img_to_array(img)
samples = np.expand_dims(data, 0)
# Demonstrating brightness augmentation
datagen = ImageDataGenerator(brightness_range=[0.4,1.5])
it = datagen.flow(samples, batch_size=1)

# Generate samples and plot
fig=plt.gcf()
fig.set_size_inches(ncols * 2, nrows * 2)
for i in range(9):
    plt.subplot(330 + 1 + i)
    batch = it.next()
    image = batch[0].astype('uint8')
    plt.imshow(image)
plt.show()
```



To ensure that our model can generalize to the validation data and not overfit to the training data, we will be randomly augmenting the training images using the methods shown above.

In [39]:
```
# Rescaling images by 1./255
# Splitting data into training and validation sets
# Performing data augmentation on training images
train_datagen = ImageDataGenerator(rescale = 1/255,
                                   horizontal_flip = True,
                                   vertical_flip = True,
                                   rotation_range=90, fill_mode='nearest',
                                   brightness_range=[0.4,1.5],
                                   validation_split = 0.2)

# Flow training images in batches
train_generator = train_datagen.flow_from_directory(
        './split/train',
        target_size=(200, 200),
        batch_size=batch_size,
        classes = classes,
        class_mode='categorical',
        subset='training')

# Flow validation images in batches
validation_generator = train_datagen.flow_from_directory(
        './split/train',
        target_size=(200, 200),
        batch_size=batch_size,
        classes = classes,
        class_mode='categorical',
        subset='validation')
```

```
Found 2916 images belonging to 4 classes.
Found 727 images belonging to 4 classes.
```

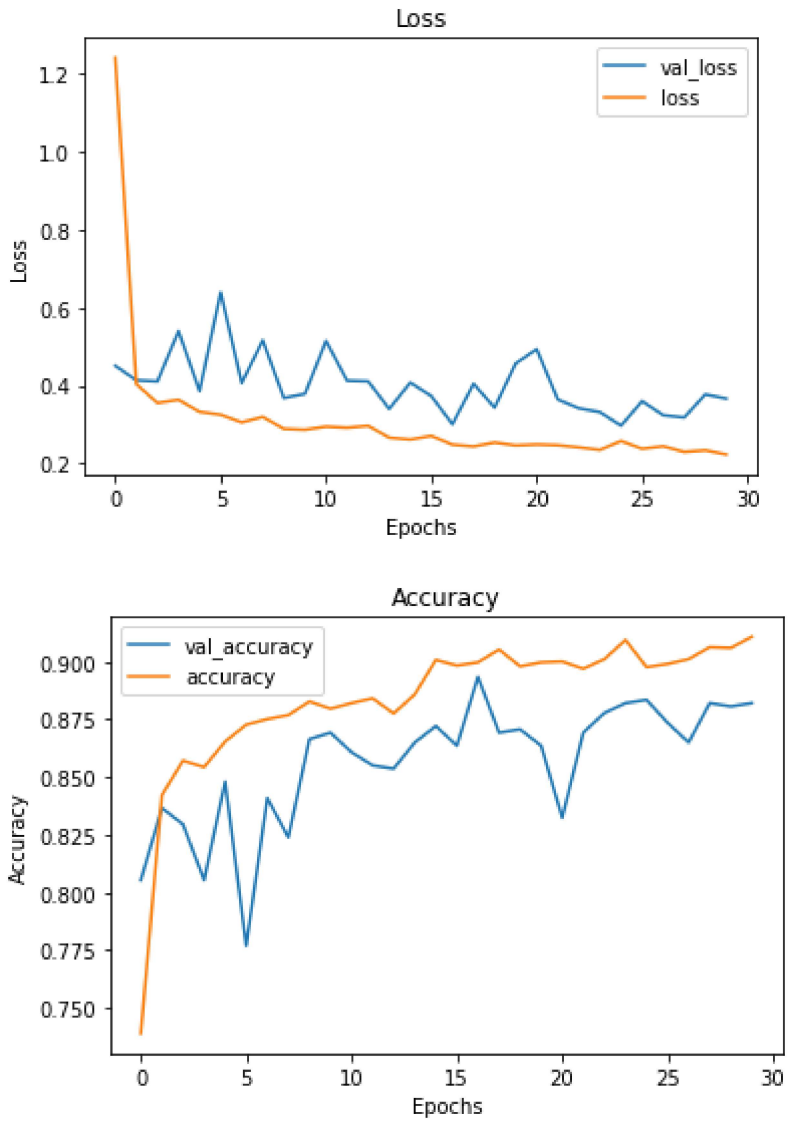
Next CNNs using Data Augmentation & Adam Optimizer

In [40]:
```
model_3 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(200, 200
, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    # 4 output neurons for 4 classes
    tf.keras.layers.Dense(4, activation='softmax')
])

# Compiling with adam optimizer
model_3.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Training
history_cnn3 = model_3.fit(
        train_generator,
        steps_per_epoch=int(training_sample/batch_size),
        validation_data = validation_generator,
        validation_steps = int(val_sample/batch_size),
        epochs=num_epochs,
        verbose=1)
```

```
Epoch 1/30
91/91 [==============================] - 72s 787ms/step - loss: 1.2405 - accu
racy: 0.7389 - val_loss: 0.4506 - val_accuracy: 0.8054
Epoch 2/30
91/91 [==============================] - 71s 785ms/step - loss: 0.4035 - accu
racy: 0.8422 - val_loss: 0.4134 - val_accuracy: 0.8366
Epoch 3/30
91/91 [==============================] - 72s 788ms/step - loss: 0.3553 - accu
racy: 0.8571 - val_loss: 0.4107 - val_accuracy: 0.8295
Epoch 4/30
91/91 [==============================] - 72s 795ms/step - loss: 0.3634 - accu
racy: 0.8544 - val_loss: 0.5396 - val_accuracy: 0.8054
Epoch 5/30
91/91 [==============================] - 72s 794ms/step - loss: 0.3324 - accu
racy: 0.8655 - val_loss: 0.3857 - val_accuracy: 0.8480
Epoch 6/30
91/91 [==============================] - 69s 760ms/step - loss: 0.3251 - accu
racy: 0.8727 - val_loss: 0.6388 - val_accuracy: 0.7770
Epoch 7/30
91/91 [==============================] - 69s 758ms/step - loss: 0.3054 - accu
racy: 0.8752 - val_loss: 0.4063 - val_accuracy: 0.8409
Epoch 8/30
91/91 [==============================] - 70s 764ms/step - loss: 0.3194 - accu
racy: 0.8769 - val_loss: 0.5157 - val_accuracy: 0.8239
Epoch 9/30
91/91 [==============================] - 69s 763ms/step - loss: 0.2888 - accu
racy: 0.8828 - val_loss: 0.3680 - val_accuracy: 0.8665
Epoch 10/30
91/91 [==============================] - 69s 761ms/step - loss: 0.2866 - accu
racy: 0.8797 - val_loss: 0.3786 - val_accuracy: 0.8693
Epoch 11/30
91/91 [==============================] - 70s 772ms/step - loss: 0.2946 - accu
racy: 0.8821 - val_loss: 0.5144 - val_accuracy: 0.8608
Epoch 12/30
91/91 [==============================] - 68s 752ms/step - loss: 0.2918 - accu
racy: 0.8842 - val_loss: 0.4123 - val_accuracy: 0.8551
Epoch 13/30
91/91 [==============================] - 67s 740ms/step - loss: 0.2966 - accu
racy: 0.8776 - val_loss: 0.4109 - val_accuracy: 0.8537
Epoch 14/30
91/91 [==============================] - 68s 753ms/step - loss: 0.2662 - accu
racy: 0.8859 - val_loss: 0.3399 - val_accuracy: 0.8651
Epoch 15/30
91/91 [==============================] - 72s 794ms/step - loss: 0.2614 - accu
racy: 0.9008 - val_loss: 0.4074 - val_accuracy: 0.8722
Epoch 16/30
91/91 [==============================] - 69s 762ms/step - loss: 0.2707 - accu
racy: 0.8984 - val_loss: 0.3730 - val_accuracy: 0.8636
Epoch 17/30
91/91 [==============================] - 70s 770ms/step - loss: 0.2483 - accu
racy: 0.8998 - val_loss: 0.3006 - val_accuracy: 0.8935
Epoch 18/30
91/91 [==============================] - 69s 762ms/step - loss: 0.2437 - accu
racy: 0.9053 - val_loss: 0.4044 - val_accuracy: 0.8693
Epoch 19/30
91/91 [==============================] - 70s 775ms/step - loss: 0.2538 - accu
racy: 0.8981 - val_loss: 0.3432 - val_accuracy: 0.8707
```

```
Epoch 20/30
91/91 [==============================] - 70s 770ms/step - loss: 0.2464 - accu
racy: 0.8998 - val_loss: 0.4565 - val_accuracy: 0.8636
Epoch 21/30
91/91 [==============================] - 72s 792ms/step - loss: 0.2486 - accu
racy: 0.9001 - val_loss: 0.4932 - val_accuracy: 0.8324
Epoch 22/30
91/91 [==============================] - 72s 789ms/step - loss: 0.2469 - accu
racy: 0.8970 - val_loss: 0.3641 - val_accuracy: 0.8693
Epoch 23/30
91/91 [==============================] - 72s 791ms/step - loss: 0.2413 - accu
racy: 0.9012 - val_loss: 0.3416 - val_accuracy: 0.8778
Epoch 24/30
91/91 [==============================] - 72s 791ms/step - loss: 0.2348 - accu
racy: 0.9095 - val_loss: 0.3318 - val_accuracy: 0.8821
Epoch 25/30
91/91 [==============================] - 69s 764ms/step - loss: 0.2576 - accu
racy: 0.8977 - val_loss: 0.2969 - val_accuracy: 0.8835
Epoch 26/30
91/91 [==============================] - 68s 747ms/step - loss: 0.2378 - accu
racy: 0.8991 - val_loss: 0.3598 - val_accuracy: 0.8736
Epoch 27/30
91/91 [==============================] - 69s 760ms/step - loss: 0.2441 - accu
racy: 0.9012 - val_loss: 0.3238 - val_accuracy: 0.8651
Epoch 28/30
91/91 [==============================] - 68s 750ms/step - loss: 0.2297 - accu
racy: 0.9064 - val_loss: 0.3180 - val_accuracy: 0.8821
Epoch 29/30
91/91 [==============================] - 69s 764ms/step - loss: 0.2338 - accu
racy: 0.9060 - val_loss: 0.3771 - val_accuracy: 0.8807
Epoch 30/30
91/91 [==============================] - 68s 750ms/step - loss: 0.2231 - accu
racy: 0.9109 - val_loss: 0.3664 - val_accuracy: 0.8821
```

In [41]:
```
viz_train_res(history_cnn3)
```



Data augmentation greatly improved the overfitting issue, however we're hoping for an even better accuracy score for validation data with lower loss.

CNN With Data Augmentation & Dropout Layer

Dropout layers are another way of combatting overfitting issues. One dropout layer will be added after the last convolutional layer and before the dense layer and output.

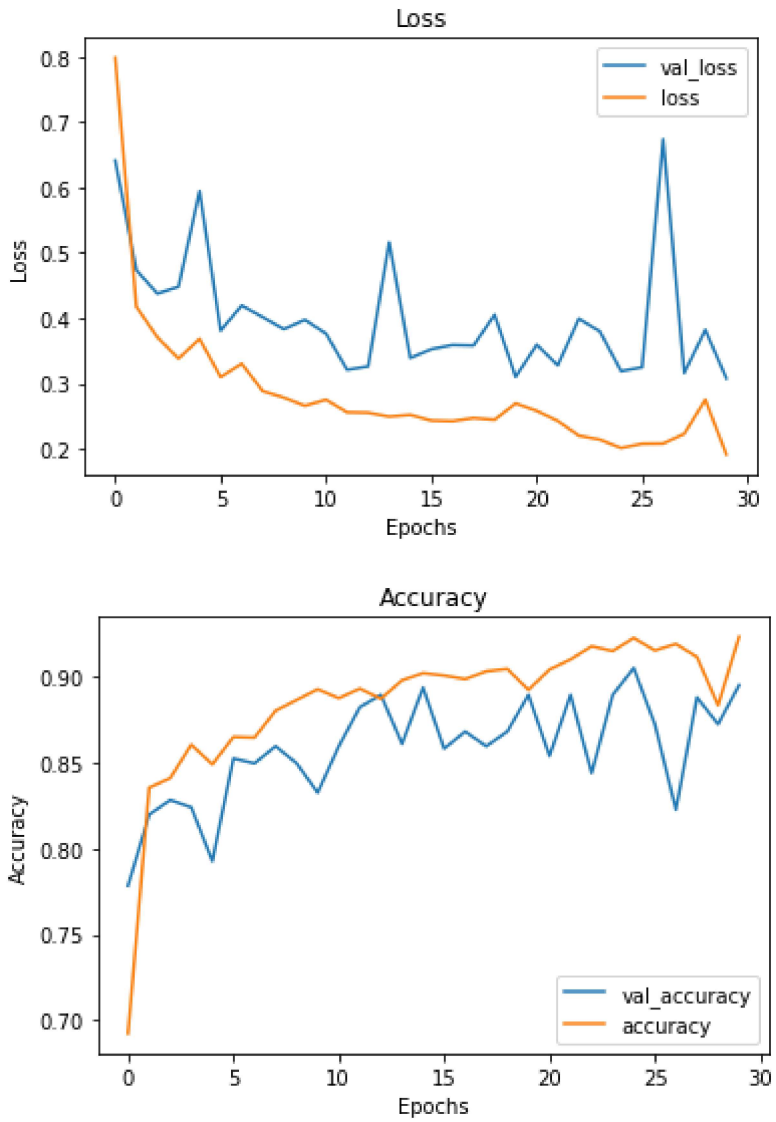
In [42]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(200, 200, 3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (5,5), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(rate = .5))

model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(4, activation='softmax'))

model.compile(optimizer=tf.keras.optimizers.Adam(lr=.003),
                loss="sparse_categorical_crossentropy",
                metrics='accuracy')
```

In [43]:
```
# Compiling with adam optimizer
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Training
history_cnn4 = model.fit(
        train_generator,
        steps_per_epoch=int(training_sample/batch_size),
        validation_data = validation_generator,
        validation_steps = int(val_sample/batch_size),
        epochs=num_epochs,
        verbose=1)
```

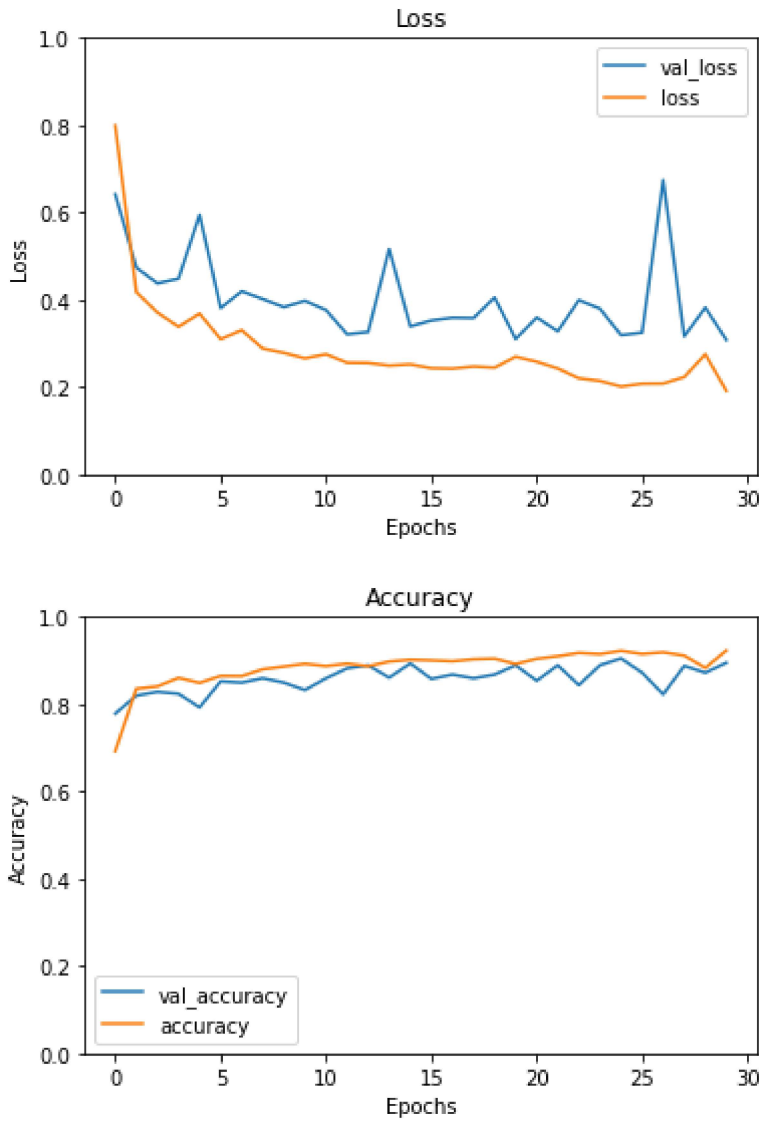
```
Epoch 1/30
91/91 [==============================] - 122s 1s/step - loss: 0.7991 - accura
cy: 0.6924 - val_loss: 0.6416 - val_accuracy: 0.7784
Epoch 2/30
91/91 [==============================] - 135s 1s/step - loss: 0.4177 - accura
cy: 0.8353 - val_loss: 0.4737 - val_accuracy: 0.8196
Epoch 3/30
91/91 [==============================] - 130s 1s/step - loss: 0.3707 - accura
cy: 0.8408 - val_loss: 0.4378 - val_accuracy: 0.8281
Epoch 4/30
91/91 [==============================] - 130s 1s/step - loss: 0.3382 - accura
cy: 0.8603 - val_loss: 0.4483 - val_accuracy: 0.8239
Epoch 5/30
91/91 [==============================] - 132s 1s/step - loss: 0.3683 - accura
cy: 0.8488 - val_loss: 0.5946 - val_accuracy: 0.7926
Epoch 6/30
91/91 [==============================] - 133s 1s/step - loss: 0.3098 - accura
cy: 0.8648 - val_loss: 0.3811 - val_accuracy: 0.8523
Epoch 7/30
91/91 [==============================] - 135s 1s/step - loss: 0.3303 - accura
cy: 0.8644 - val_loss: 0.4195 - val_accuracy: 0.8494
Epoch 8/30
91/91 [==============================] - 132s 1s/step - loss: 0.2883 - accura
cy: 0.8800 - val_loss: 0.4016 - val_accuracy: 0.8594
Epoch 9/30
91/91 [==============================] - 136s 1s/step - loss: 0.2784 - accura
cy: 0.8863 - val_loss: 0.3833 - val_accuracy: 0.8494
Epoch 10/30
91/91 [==============================] - 133s 1s/step - loss: 0.2659 - accura
cy: 0.8925 - val_loss: 0.3975 - val_accuracy: 0.8324
Epoch 11/30
91/91 [==============================] - 133s 1s/step - loss: 0.2751 - accura
cy: 0.8873 - val_loss: 0.3764 - val_accuracy: 0.8594
Epoch 12/30
91/91 [==============================] - 137s 2s/step - loss: 0.2558 - accura
cy: 0.8929 - val_loss: 0.3210 - val_accuracy: 0.8821
Epoch 13/30
91/91 [==============================] - 135s 1s/step - loss: 0.2552 - accura
cy: 0.8870 - val_loss: 0.3262 - val_accuracy: 0.8892
Epoch 14/30
91/91 [==============================] - 148s 2s/step - loss: 0.2492 - accura
cy: 0.8977 - val_loss: 0.5160 - val_accuracy: 0.8608
Epoch 15/30
91/91 [==============================] - 135s 1s/step - loss: 0.2519 - accura
cy: 0.9019 - val_loss: 0.3391 - val_accuracy: 0.8935
Epoch 16/30
91/91 [==============================] - 134s 1s/step - loss: 0.2431 - accura
cy: 0.9005 - val_loss: 0.3525 - val_accuracy: 0.8580
Epoch 17/30
91/91 [==============================] - 134s 1s/step - loss: 0.2423 - accura
cy: 0.8984 - val_loss: 0.3589 - val_accuracy: 0.8679
Epoch 18/30
91/91 [==============================] - 135s 1s/step - loss: 0.2470 - accura
cy: 0.9029 - val_loss: 0.3582 - val_accuracy: 0.8594
Epoch 19/30
91/91 [==============================] - 134s 1s/step - loss: 0.2445 - accura
cy: 0.9043 - val_loss: 0.4053 - val_accuracy: 0.8679
```

```
Epoch 20/30
91/91 [==============================] - 135s 1s/step - loss: 0.2695 - accura
cy: 0.8922 - val_loss: 0.3101 - val_accuracy: 0.8892
Epoch 21/30
91/91 [==============================] - 135s 1s/step - loss: 0.2580 - accura
cy: 0.9040 - val_loss: 0.3594 - val_accuracy: 0.8537
Epoch 22/30
91/91 [==============================] - 134s 1s/step - loss: 0.2426 - accura
cy: 0.9098 - val_loss: 0.3278 - val_accuracy: 0.8892
Epoch 23/30
91/91 [==============================] - 134s 1s/step - loss: 0.2201 - accura
cy: 0.9175 - val_loss: 0.3992 - val_accuracy: 0.8438
Epoch 24/30
91/91 [==============================] - 134s 1s/step - loss: 0.2139 - accura
cy: 0.9147 - val_loss: 0.3797 - val_accuracy: 0.8892
Epoch 25/30
91/91 [==============================] - 135s 1s/step - loss: 0.2011 - accura
cy: 0.9223 - val_loss: 0.3189 - val_accuracy: 0.9048
Epoch 26/30
91/91 [==============================] - 136s 1s/step - loss: 0.2076 - accura
cy: 0.9150 - val_loss: 0.3248 - val_accuracy: 0.8722
Epoch 27/30
91/91 [==============================] - 136s 1s/step - loss: 0.2079 - accura
cy: 0.9189 - val_loss: 0.6741 - val_accuracy: 0.8224
Epoch 28/30
91/91 [==============================] - 136s 1s/step - loss: 0.2228 - accura
cy: 0.9112 - val_loss: 0.3161 - val_accuracy: 0.8878
Epoch 29/30
91/91 [==============================] - 136s 1s/step - loss: 0.2750 - accura
cy: 0.8831 - val_loss: 0.3820 - val_accuracy: 0.8722
Epoch 30/30
91/91 [==============================] - 134s 1s/step - loss: 0.1912 - accura
cy: 0.9230 - val_loss: 0.3078 - val_accuracy: 0.8949
```

In [44]:
```
viz_train_res(history_cnn4)
```



In [54]:
```
viz_train_res2(history_cnn4)
```



Adding the dropout layer actually caused accuracy to decrease and there's concern about the dip in accuracy around epoch 15. This model is actually performing worse than the previous one. This performance could be because this model has 2 convolutional layers, the next model will drop one of those layers in the hopes of solving for this dip and raising the overall accuracy while maintaining a lower level of overfitting.

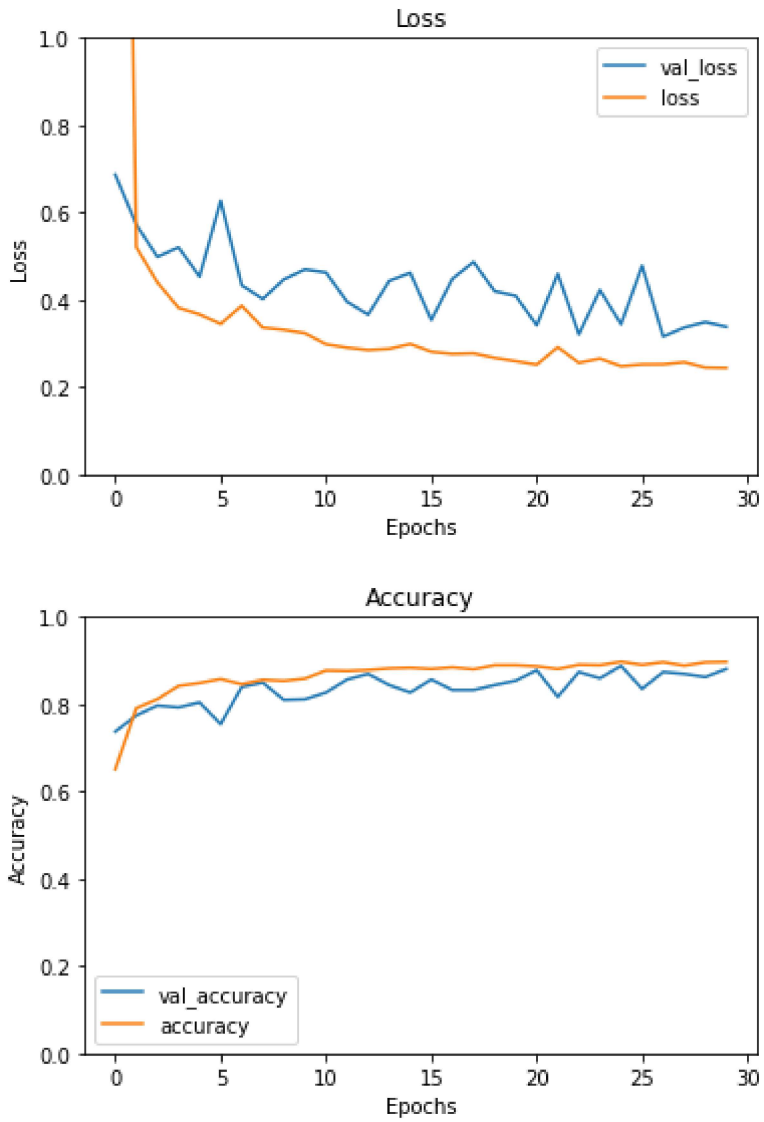
In [47]:
```
#removing a convolutional layer and keeping the dropout layer
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(200, 200, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(rate = .5))

model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(4, activation='softmax'))

model.compile(optimizer=tf.keras.optimizers.Adam(lr=.003),
                loss="sparse_categorical_crossentropy",
                metrics='accuracy')
```

In [48]:
```
# Compiling with adam optimizer
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Training
history_cnn5 = model.fit(
        train_generator,
        steps_per_epoch=int(training_sample/batch_size),
        validation_data = validation_generator,
        validation_steps = int(val_sample/batch_size),
        epochs=num_epochs,
        verbose=1)
```

```
Epoch 1/30
91/91 [==============================] - 85s 937ms/step - loss: 3.5264 - accu
racy: 0.6508 - val_loss: 0.6867 - val_accuracy: 0.7372
Epoch 2/30
91/91 [==============================] - 85s 939ms/step - loss: 0.5203 - accu
racy: 0.7909 - val_loss: 0.5719 - val_accuracy: 0.7741
Epoch 3/30
91/91 [==============================] - 86s 944ms/step - loss: 0.4397 - accu
racy: 0.8114 - val_loss: 0.4982 - val_accuracy: 0.7969
Epoch 4/30
91/91 [==============================] - 87s 952ms/step - loss: 0.3816 - accu
racy: 0.8419 - val_loss: 0.5201 - val_accuracy: 0.7926
Epoch 5/30
91/91 [==============================] - 88s 971ms/step - loss: 0.3665 - accu
racy: 0.8485 - val_loss: 0.4530 - val_accuracy: 0.8040
Epoch 6/30
91/91 [==============================] - 88s 971ms/step - loss: 0.3448 - accu
racy: 0.8575 - val_loss: 0.6268 - val_accuracy: 0.7543
Epoch 7/30
91/91 [==============================] - 88s 963ms/step - loss: 0.3863 - accu
racy: 0.8457 - val_loss: 0.4333 - val_accuracy: 0.8395
Epoch 8/30
91/91 [==============================] - 88s 971ms/step - loss: 0.3363 - accu
racy: 0.8558 - val_loss: 0.4019 - val_accuracy: 0.8494
Epoch 9/30
91/91 [==============================] - 88s 972ms/step - loss: 0.3312 - accu
racy: 0.8533 - val_loss: 0.4467 - val_accuracy: 0.8097
Epoch 10/30
91/91 [==============================] - 88s 972ms/step - loss: 0.3234 - accu
racy: 0.8585 - val_loss: 0.4694 - val_accuracy: 0.8111
Epoch 11/30
91/91 [==============================] - 88s 967ms/step - loss: 0.2981 - accu
racy: 0.8773 - val_loss: 0.4624 - val_accuracy: 0.8267
Epoch 12/30
91/91 [==============================] - 88s 962ms/step - loss: 0.2902 - accu
racy: 0.8762 - val_loss: 0.3954 - val_accuracy: 0.8565
Epoch 13/30
91/91 [==============================] - 87s 953ms/step - loss: 0.2847 - accu
racy: 0.8783 - val_loss: 0.3658 - val_accuracy: 0.8693
Epoch 14/30
91/91 [==============================] - 88s 970ms/step - loss: 0.2874 - accu
racy: 0.8821 - val_loss: 0.4434 - val_accuracy: 0.8438
Epoch 15/30
91/91 [==============================] - 92s 1s/step - loss: 0.2987 - accurac
y: 0.8835 - val_loss: 0.4618 - val_accuracy: 0.8267
Epoch 16/30
91/91 [==============================] - 95s 1s/step - loss: 0.2808 - accurac
y: 0.8807 - val_loss: 0.3535 - val_accuracy: 0.8565
Epoch 17/30
91/91 [==============================] - 88s 970ms/step - loss: 0.2759 - accu
racy: 0.8845 - val_loss: 0.4480 - val_accuracy: 0.8324
Epoch 18/30
91/91 [==============================] - 87s 960ms/step - loss: 0.2771 - accu
racy: 0.8800 - val_loss: 0.4866 - val_accuracy: 0.8324
Epoch 19/30
91/91 [==============================] - 87s 954ms/step - loss: 0.2669 - accu
racy: 0.8894 - val_loss: 0.4197 - val_accuracy: 0.8438
```

```
Epoch 20/30
91/91 [==============================] - 87s 960ms/step - loss: 0.2593 - accu
racy: 0.8894 - val_loss: 0.4091 - val_accuracy: 0.8537
Epoch 21/30
91/91 [==============================] - 85s 937ms/step - loss: 0.2513 - accu
racy: 0.8870 - val_loss: 0.3415 - val_accuracy: 0.8778
Epoch 22/30
91/91 [==============================] - 87s 954ms/step - loss: 0.2913 - accu
racy: 0.8807 - val_loss: 0.4594 - val_accuracy: 0.8168
Epoch 23/30
91/91 [==============================] - 87s 960ms/step - loss: 0.2555 - accu
racy: 0.8904 - val_loss: 0.3210 - val_accuracy: 0.8736
Epoch 24/30
91/91 [==============================] - 88s 967ms/step - loss: 0.2652 - accu
racy: 0.8894 - val_loss: 0.4225 - val_accuracy: 0.8594
Epoch 25/30
91/91 [==============================] - 88s 971ms/step - loss: 0.2478 - accu
racy: 0.8974 - val_loss: 0.3440 - val_accuracy: 0.8878
Epoch 26/30
91/91 [==============================] - 88s 964ms/step - loss: 0.2518 - accu
racy: 0.8901 - val_loss: 0.4777 - val_accuracy: 0.8352
Epoch 27/30
91/91 [==============================] - 87s 957ms/step - loss: 0.2520 - accu
racy: 0.8963 - val_loss: 0.3158 - val_accuracy: 0.8736
Epoch 28/30
91/91 [==============================] - 87s 955ms/step - loss: 0.2570 - accu
racy: 0.8883 - val_loss: 0.3365 - val_accuracy: 0.8693
Epoch 29/30
91/91 [==============================] - 87s 960ms/step - loss: 0.2446 - accu
racy: 0.8960 - val_loss: 0.3489 - val_accuracy: 0.8622
Epoch 30/30
91/91 [==============================] - 87s 951ms/step - loss: 0.2437 - accu
racy: 0.8970 - val_loss: 0.3382 - val_accuracy: 0.8807
```

In [49]:
```
viz_train_res2(history_cnn5)
```



Although this model doesn't provide the highest accuracy, it does much better managing overfitting. Both accuracy and loss scores show less variance between training and validation sets.

Bringing in Test Data

In [50]:
```
# Rescaling images by 1./255
# Performing data augmentation on TESTING images
train_datagen = ImageDataGenerator(rescale = 1/255)

# Flow training images in batches
test_generator = train_datagen.flow_from_directory(
        './split/test',
        target_size=(200, 200),
        batch_size=batch_size,
        classes = classes,
        class_mode='categorical')
```

```
Found 545 images belonging to 4 classes.
```

In [51]:
```
steps = test_generator.n
```

In [52]:
```
test_loss, test_acc = model.evaluate(test_generator)
print('test acc:', test_acc)
```

```
18/18 [==============================] - 11s 587ms/step - loss: 0.6487 - accu
racy: 0.7780
test acc: 0.7779816389083862
```

Conclusion

The final model with a single convolutional layer plus data augmentation and a dropout layer performed only marginally better than the model with a single layer and data augmentation. Loss is slightly lower and overfitting is a bit better. This final model should remain dependable as the dataset grows as the dropout layer will continue to battle overfitting along with data augmentation.

Training accuracy landed around 90% while validation accuracy settles around 85%. This means that Maize Well can be expected to correctly class new images with an 85% accuracy.

Recommendations & Future Work

Maize Well can be trusted to properly class common maize leaf diseases. We recommend to use this product as early as possible to catch disease before it spreads to the rest of the crop.

Our recommendation: **use Maize Well!**

Going forward, we would like to expand the database to include other common diseases as well as take advantage of region data as some diseases are more common in certain climates. Including region information would provide the user with more information on how to prevent and spot disease for their particular crop.