plant-disease-cnn-lab7

May 8, 2023

[1]:
```
# Import TensorFlow into collab
import tensorflow as tf
print(f"Tensorflow version: {tf.__version__}")
```

```
Tensorflow version: 2.4.0
```

[2]:
```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

0.1 2. Initialisation code

[3]:
```
# Import required packages
import os
import tensorflow as tf
import pandas as pd
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
%matplotlib inline
```

[4]:
```
# Get Path to Train and Valid folders
train_path = '../input/new-plant-diseases-dataset/New Plant Diseases␣
↪Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/train'
valid_path = '../input/new-plant-diseases-dataset/New Plant Diseases␣
↪Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/valid'

# Get list of all subfolders for each Subset
train_dir = os.listdir(train_path)
valid_dir = os.listdir(valid_path)
# Check length of subfolders
len(train_dir), len(valid_dir)
```

[4]:
```
(38, 38)
```

[5]:
```python
data_dir = "../input/new-plant-diseases-dataset/New Plant Diseases␣
↪Dataset(Augmented)/New Plant Diseases Dataset(Augmented)"
train_dir = data_dir + "/train"
valid_dir = data_dir + "/valid"
diseases = os.listdir(train_dir)
```

[6]:
```python
# Number of images for each disease
nums = {}
for disease in diseases:
    nums[disease] = len(os.listdir(train_dir + '/' + disease))

# converting the nums dictionary to pandas dataframe passing index as plant␣
↪name and number of images as column

img_per_class = pd.DataFrame(nums.values(), index=nums.keys(), columns=["no. of␣
↪images"])
img_per_class
```

[6]:

	no. of images
Tomato___Late_blight	1851
Tomato___healthy	1926
Grape___healthy	1692
Orange___Haunglongbing_(Citrus_greening)	2010
Soybean___healthy	2022
Squash___Powdery_mildew	1736
Potato___healthy	1824
Corn_(maize)___Northern_Leaf_Blight	1908
Tomato___Early_blight	1920
Tomato___Septoria_leaf_spot	1745
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	1642
Strawberry___Leaf_scorch	1774
Peach___healthy	1728
Apple___Apple_scab	2016
Tomato___Tomato_Yellow_Leaf_Curl_Virus	1961
Tomato___Bacterial_spot	1702
Apple___Black_rot	1987
Blueberry___healthy	1816
Cherry_(including_sour)___Powdery_mildew	1683
Peach___Bacterial_spot	1838
Apple___Cedar_apple_rust	1760
Tomato___Target_Spot	1827
Pepper,_bell___healthy	1988
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	1722
Potato___Late_blight	1939
Tomato___Tomato_mosaic_virus	1790
Strawberry___healthy	1824
Apple___healthy	2008

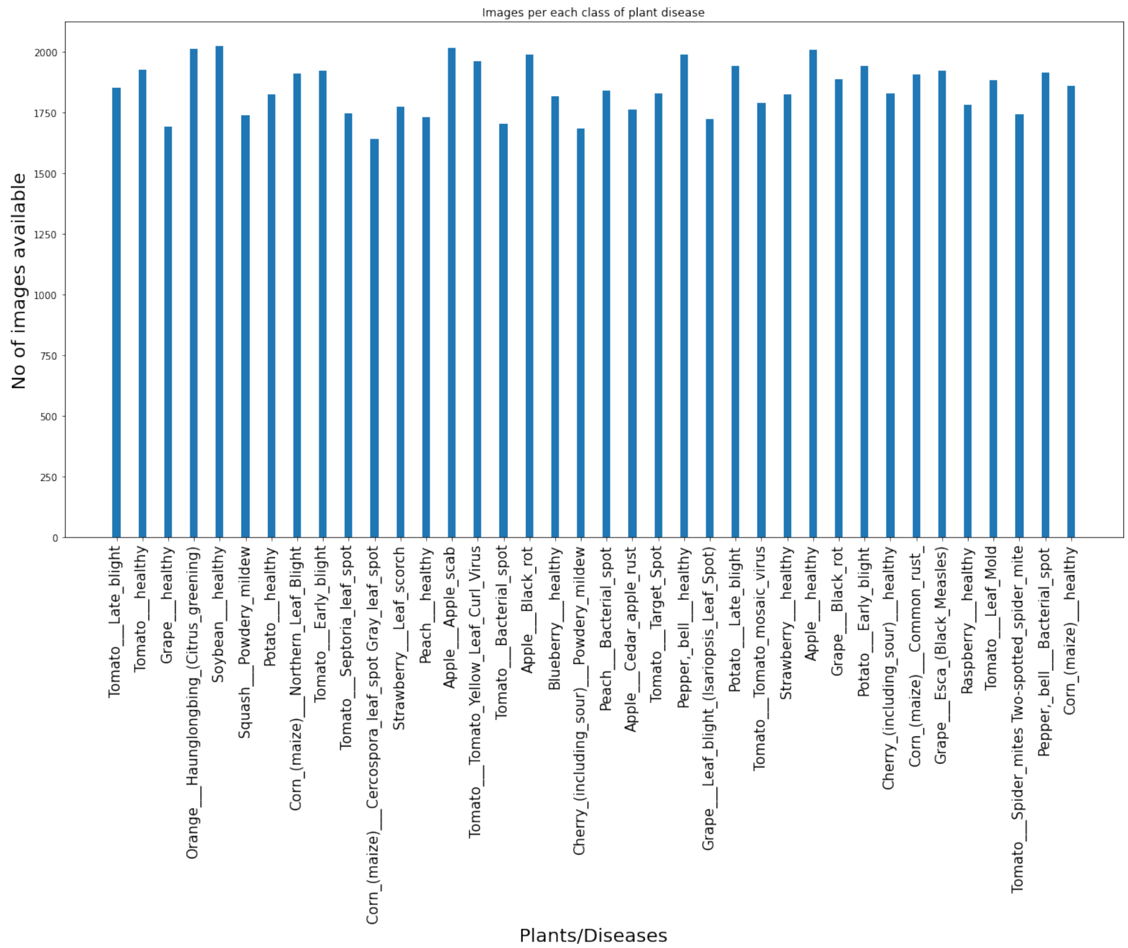
```
Grape___Black_rot                                          1888
Potato___Early_blight                                      1939
Cherry_(including_sour)___healthy                          1826
Corn_(maize)___Common_rust_                                1907
Grape___Esca_(Black_Measles)                               1920
Raspberry___healthy                                        1781
Tomato___Leaf_Mold                                         1882
Tomato___Spider_mites Two-spotted_spider_mite              1741
Pepper,_bell___Bacterial_spot                              1913
Corn_(maize)___healthy                                     1859
```

[7]:

```python
# plotting number of images available for each disease
index = [n for n in range(38)]
plt.figure(figsize=(20, 10))
plt.bar(index, [n for n in nums.values()], width=0.3)
plt.xlabel('Plants/Diseases', fontsize=20)
plt.ylabel('No of images available', fontsize=20)
plt.xticks(index, diseases, fontsize=15, rotation=90)
plt.title('Images per each class of plant disease')
```

[7]:

```
Text(0.5, 1.0, 'Images per each class of plant disease')
```



[8]:
```python
# Create Dataframe

def create_info_df(path):
    """
    input: `path` - folder path
    From folder path, create a Dataframe with columns:
    Plant | Category | Path | Plant___Category | Disease
    return DataFrame
    """

    list_plants = []
    list_dir = os.listdir(path) # Get list direcotry
    # Go through each folder to create url and get required information
    for plant in list_dir:
        url = path +'/'+plant
        for img in os.listdir(url):
            list_plants.append([*plant.split('___'), url+'/'+img, plant])
```

```
    # Create DataFrame
    df = pd.DataFrame(list_plants, columns=['Plant', 'Category',␣
↪'Path','Plant___Category'])
    # Add `Disease` column - if folder name is not Healthy then plant is␣
↪diseased
    df['Disease'] = df.Category.apply(lambda x: 0 if x=='healthy' else 1)

    return df

# Get Validation and Training DF
train_info = create_info_df(train_path)
valid_info = create_info_df(valid_path)

print(train_info.shape, valid_info.shape)

#Unique label list:
unique_plant_cat = np.unique(train_info['Plant___Category'].to_numpy())
print("Number of Categories to predict: ", len(unique_plant_cat))
```

```
(70295, 5) (17572, 5)
Number of Categories to predict:  38
```

[9]:
```
# Creation of constants
IMG_SIZE = 64
IMG_SHAPE = (IMG_SIZE, IMG_SIZE)
batch_size = 32
AUTOTUNE = tf.data.experimental.AUTOTUNE
OUTPUT_SHAPE = 38
NUM_EPOCHS = 20
```

[10]:
```
## FUNCTION UTILS - Prepare Data and Dataset ##

def create_img_df(df_info, frac=0.1, random_state=42):
    return df_info.sample(frac=frac, random_state=random_state).reset_index()

def create_train_val_df(valid_info, train_info, frac=0.1, random_state=42):
    """
    Create Train and validation dataframe
    Return:
      - train dataframe
      - validation dataframe
    """
    valid_df = create_img_df(valid_info, frac, random_state)
    train_df = create_img_df(train_info, frac, random_state)

    # Get information shape
    valid_img_cnt,train_img_count = valid_df.shape[0], train_df.shape[0]
```

```
    total = valid_img_cnt + train_img_count
    # Print information
    print(f'Total images (frac={frac}): ', total)
    print(f"Training ({train_img_count}): {train_img_count/total*100:.2f}% -␣
 ↪Validation ({valid_img_cnt}): {valid_img_cnt/total*100:.2f}%")

    return train_df, valid_df


def get_bool_label(labels):
    # Create a variable of all Labels
    plant_cat_labels = labels.to_numpy()
    # Create Boolean label list
    bool_plant_cat = [unique_plant_cat == plant_cat for plant_cat in␣
 ↪plant_cat_labels]
    # return array
    return bool_plant_cat

# Prepare Data
def prepare_data(train_df, valid_df):
    """
    Get Train and Validation Data Frame and return X_train, X_val, y_train,␣
 ↪y_val
    """
    # create images (X) arrays
    X_train = train_df['Path']
    X_val = valid_df['Path']

    # create labels (y) arrays
    y_train = get_bool_label(train_df['Plant___Category'])
    y_val = get_bool_label(valid_df['Plant___Category'])

    print('Shape: ',X_train.shape, X_val.shape, len(y_train), len(y_val))

    return X_train, X_val, y_train, y_val



# Dataset function utils

# Decode and load image
def decode_img(path, img_shape=IMG_SHAPE):
    """
    Read image from `path`, and convert the image to a 3D tensor
    return resized image.
    input: `path`: Path to an image
    return: resized tensor image
```

```
    """
    print('Image size: ({})'.format(img_shape))
    # Read the image file
    img = tf.io.read_file(path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.cast(img, tf.float32)/255
    # Resize image to our desired size
    img = tf.image.resize(img, img_shape)
    return img

# Configure dataset for performance
def configure_for_performance(ds):
    #ds = ds.cache()
    ds = ds.batch(batch_size)
    #ds = ds.prefetch(buffer_size=AUTOTUNE)
    return ds

# Create a function to get Dataset
def create_dataset(X, y=None, valid_data=False, test_data=False,␣
↪img_shape=IMG_SHAPE):
    """
    Create Dataset from Images (X) and Labels (y)
    Shuffles the data if it's training data but doesn't shuffle if it␣
↪validation data.
    Also accepts test data as input (no labels).
    Return Dataset
    """
    print("Creating data set...")
    # If test data, there is no labels
    if test_data:
        print("Creating test data batches...")
        dataset = tf.data.Dataset.from_tensor_slices((X))
        dataset = dataset.map(lambda x: decode_img(x, img_shape),␣
↪num_parallel_calls=AUTOTUNE)
        dataset = configure_for_performance(dataset)
    # If Valid_data - we don't need to shuffle
    elif valid_data:
        print("Creating Valid data batches...")
        dataset = tf.data.Dataset.from_tensor_slices((X, y))
        dataset = dataset.map(lambda x, y: [decode_img(x, img_shape), y],␣
↪num_parallel_calls=AUTOTUNE)
        dataset = configure_for_performance(dataset)
    else:
        print("Creating Training data batches...")
        dataset = tf.data.Dataset.from_tensor_slices((X, y))
        dataset = dataset.map(lambda x, y: [decode_img(x, img_shape), y],␣
↪num_parallel_calls=AUTOTUNE)
```

```
        dataset = dataset.shuffle(buffer_size=len(X))
        dataset = configure_for_performance(dataset)

    print(dataset.element_spec)

    return dataset


# Create Models function utils #
################################

# Callbacks
# Early stopping Callbacks
early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_accuracy",␣
 ↪patience=5)

# Reduce Learning rate Callbacks
lr_callback = tf.keras.callbacks.ReduceLROnPlateau(monitor = 'val_loss',
                                                   patience=3,
                                                   factor=0.2,
                                                   verbose=2,
                                                   mode='min')


# Useful Functions for Model training, saving and loading

def train_model(transfer_model, epochs = NUM_EPOCHS):
    """
    Trains a given model and returns the trained version.
     Input: model, number of Epochs (default = NUM_EPOCHS)
     Output: model
    """
    # create model
    model = create_model(transfer_model)
    # Create TensorBoard session
    tensorboard = create_tensorboard_callback()

    model.summary()
    print(f"Information: epochs = {epochs} and number of images = {NUM_IMAGES}")

    # Fit model
    model.fit(x=dataset_train,
              epochs=epochs,
              validation_data=dataset_val,
              callbacks=[early_stopping, lr_callback])
    return model
```

```
import datetime
# Save and load model
# Create a function to save a model
def save_model(model, suffix=None):
    """
    Saves a given model in ad models directory and appends a suffix (string).
    """
    # Create a model directory pathname with current time
    modeldir = os.path.join("../output/kaggle/working/saved_models",
                          datetime.datetime.now().strftime("%Y%m%d-%H%M%s"))
    model_path = modeldir + "-" + suffix + ".h5" #save format of model
    print(f"Save model to: {model_path}...")
    model.save(model_path)
    return model_path

# Create a function to load a model
def load_model(model_path):
    """
    Load a saved model from a specify path
    """
    print(f"Loading saved model from: {model_path}...")
    model = tf.keras.models.load_model(model_path)
    return model
```

1 MODELS

We retreived our data from the Dataset folder.

We created multiple methods and functions that would help us to prepare our data into batches and create our model.

We can now start the Modeling phase.

1.1 1. Create own CNN

[11]:
```
#Create and get dataset
FRAC = 1
IMG_SIZE = 64
IMG_SHAPE = (IMG_SIZE, IMG_SIZE)
OUTPUT_SHAPE = len(unique_plant_cat)

train_df, valid_df = create_train_val_df(valid_info, train_info, frac=FRAC)
# Get data ready
X_train, X_val, y_train, y_val = prepare_data(train_df, valid_df)

# Create Dataset #
##################
# Train dataset - shuffle
```

```python
dataset_train = create_dataset(X_train, y_train, img_shape=IMG_SHAPE)
# Validation Dataset - not shuffle
dataset_val = create_dataset(X_val, y_val, valid_data=True, img_shape=IMG_SHAPE)
# Verify length of both datasets
len(dataset_train), len(dataset_val)

NUM_IMAGES = len(y_train) + len(y_val)
```

```
Total images (frac=1):  87867
Training (70295): 80.00% - Validation (17572): 20.00%
Shape:  (70295,) (17572,) 70295 17572
Creating data set…
Creating Training data batches…
Image size: ((64, 64))
(TensorSpec(shape=(None, 64, 64, 3), dtype=tf.float32, name=None),
TensorSpec(shape=(None, 38), dtype=tf.bool, name=None))
Creating data set…
Creating Valid data batches…
Image size: ((64, 64))
(TensorSpec(shape=(None, 64, 64, 3), dtype=tf.float32, name=None),
TensorSpec(shape=(None, 38), dtype=tf.bool, name=None))
```

[12]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout,␣
↪Conv2D, MaxPooling2D, Flatten, BatchNormalization, Activation

NUM_EPOCHS = 20
INPUT_SHAPE = (IMG_SIZE, IMG_SIZE, 3)

# Create model
# 4 conv2D layers
# Batch Normalisation and MaxPooling
def get_model():
    """
    Create a 4 Conv2D layers with
    - Batch Normalisation
    - MaxPooling
    - ReLU activation
    And 2 Dense layers reLU activation (and Dropout)

    Return 38 probabilities (= number of plants we want to predict) -␣
↪activation Softmax
    """

    model_v2 = Sequential([
        # First CNN
        Conv2D(128, kernel_size=3, input_shape=INPUT_SHAPE, activation='relu'),
```

```python
        MaxPooling2D(),
        BatchNormalization(),
        # Second CNN
        Conv2D(256, kernel_size=3, activation='relu'),
        MaxPooling2D(),
        BatchNormalization(),
        # Third CNN
        Conv2D(512, kernel_size=3, activation='relu'),
        MaxPooling2D(),
        BatchNormalization(),
        # Flatten last CNN output for Dense layers
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.2),
        Dense(256, activation='relu'),
        Dropout(0.2),
        # Return 38 probabilities (= number of plants we want to predict)
        Dense(OUTPUT_SHAPE, activation= 'softmax')
    ])

    return model_v2

# To Do: modify Adam optimizer and add a specific Learning rate
model = get_model()
model.compile(optimizer=tf.optimizers.Adam(learning_rate=0.0005),
            loss='categorical_crossentropy',
            metrics=['accuracy']
            )
# Show Summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 128)       3584
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 128)       0
_________________________________________________________________
batch_normalization (BatchNo (None, 31, 31, 128)       512
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 256)       295168
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 256)       0
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 256)       1024
_________________________________________________________________
```

```
conv2d_2 (Conv2D)            (None, 12, 12, 512)       1180160
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 512)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 6, 6, 512)         2048
_________________________________________________________________
flatten (Flatten)            (None, 18432)             0
_________________________________________________________________
dense (Dense)                (None, 512)               9437696
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 38)                9766
=================================================================
Total params: 11,061,286
Trainable params: 11,059,494
Non-trainable params: 1,792
_________________________________________________________________
```

[13]:
```python
# Train Model
history = model.fit(x=dataset_train,
                    epochs=NUM_EPOCHS,
                    validation_data=dataset_val,
                    callbacks=[early_stopping, lr_callback])

# Get Validation Loss and Accuracy
val_loss, val_acc = model.evaluate(dataset_val)
val_acc = round(val_acc, 3)

# Save model
suffix =␣
 ↪'tf_ep-'+str(NUM_EPOCHS)+'_img-'+str(NUM_IMAGES)+'_acc_'+str(val_acc)+'-model_cnn_'+str(FRA
save_model(model, suffix=suffix)
```

```
Epoch 1/20
2197/2197 [==============================] - 258s 33ms/step - loss: 1.8284 -
accuracy: 0.5333 - val_loss: 0.9911 - val_accuracy: 0.7176
Epoch 2/20
2197/2197 [==============================] - 110s 19ms/step - loss: 0.4725 -
accuracy: 0.8578 - val_loss: 0.4172 - val_accuracy: 0.8687
Epoch 3/20
2197/2197 [==============================] - 107s 20ms/step - loss: 0.2554 -
accuracy: 0.9228 - val_loss: 0.3709 - val_accuracy: 0.8936
Epoch 4/20
```

```
2197/2197 [==============================] - 100s 20ms/step - loss: 0.1815 -
accuracy: 0.9439 - val_loss: 0.7657 - val_accuracy: 0.7845
Epoch 5/20
2197/2197 [==============================] - 99s 19ms/step - loss: 0.1387 -
accuracy: 0.9589 - val_loss: 0.4086 - val_accuracy: 0.8930
Epoch 6/20
2197/2197 [==============================] - 99s 20ms/step - loss: 0.1122 -
accuracy: 0.9673 - val_loss: 0.1759 - val_accuracy: 0.9499
Epoch 7/20
2197/2197 [==============================] - 102s 20ms/step - loss: 0.0996 -
accuracy: 0.9711 - val_loss: 0.2914 - val_accuracy: 0.9316
Epoch 8/20
2197/2197 [==============================] - 104s 20ms/step - loss: 0.0852 -
accuracy: 0.9759 - val_loss: 0.4927 - val_accuracy: 0.8941
Epoch 9/20
2197/2197 [==============================] - 102s 20ms/step - loss: 0.0779 -
accuracy: 0.9778 - val_loss: 4.1011 - val_accuracy: 0.6887

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
Epoch 10/20
2197/2197 [==============================] - 100s 20ms/step - loss: 0.0352 -
accuracy: 0.9888 - val_loss: 0.0672 - val_accuracy: 0.9832
Epoch 11/20
2197/2197 [==============================] - 101s 20ms/step - loss: 0.0151 -
accuracy: 0.9953 - val_loss: 0.0849 - val_accuracy: 0.9803
Epoch 12/20
2197/2197 [==============================] - 100s 20ms/step - loss: 0.0116 -
accuracy: 0.9966 - val_loss: 0.0655 - val_accuracy: 0.9856
Epoch 13/20
2197/2197 [==============================] - 101s 20ms/step - loss: 0.0098 -
accuracy: 0.9971 - val_loss: 0.0633 - val_accuracy: 0.9862
Epoch 14/20
2197/2197 [==============================] - 99s 20ms/step - loss: 0.0076 -
accuracy: 0.9975 - val_loss: 0.0631 - val_accuracy: 0.9863
Epoch 15/20
2197/2197 [==============================] - 99s 20ms/step - loss: 0.0080 -
accuracy: 0.9977 - val_loss: 0.0881 - val_accuracy: 0.9808
Epoch 16/20
2197/2197 [==============================] - 100s 20ms/step - loss: 0.0073 -
accuracy: 0.9980 - val_loss: 0.0671 - val_accuracy: 0.9867
Epoch 17/20
2197/2197 [==============================] - 100s 20ms/step - loss: 0.0057 -
accuracy: 0.9980 - val_loss: 0.1179 - val_accuracy: 0.9772

Epoch 00017: ReduceLROnPlateau reducing learning rate to 2.0000000949949027e-05.
Epoch 18/20
2197/2197 [==============================] - 99s 20ms/step - loss: 0.0068 -
accuracy: 0.9980 - val_loss: 0.0633 - val_accuracy: 0.9878
```

```
Epoch 19/20
2197/2197 [==============================] - 99s 20ms/step - loss: 0.0036 -
accuracy: 0.9990 - val_loss: 0.0632 - val_accuracy: 0.9882
Epoch 20/20
2197/2197 [==============================] - 100s 20ms/step - loss: 0.0028 -
accuracy: 0.9991 - val_loss: 0.0657 - val_accuracy: 0.9884

Epoch 00020: ReduceLROnPlateau reducing learning rate to 4.000000262749381e-06.
550/550 [==============================] - 16s 28ms/step - loss: 0.0657 -
accuracy: 0.9884
Save model to: ../output/kaggle/working/saved_models/20230508-17201683566455-tf_
ep-20_img-87867_acc_0.988-model_cnn_1.h5…
```

[13]:
```
'../output/kaggle/working/saved_models/20230508-17201683566455-tf_ep-20_img-8786
7_acc_0.988-model_cnn_1.h5'
```

[14]:
```python
# convert the history.history dict to a pandas DataFrame:
hist_df = pd.DataFrame(history.history)

date = datetime.datetime.now().strftime("%Y%m%d")
hist_csv_file = '../output/kaggle/working/
 ↪history_full_'+str(val_acc)+'_'+str(date)+'.csv'
with open(hist_csv_file, mode='w') as f:
    hist_df.to_csv(f)
```

1.2 2. Evaluation of model

The model have been trained, we can now evaluate it to conclude about the good performance.

[15]:
```python
# Useful functions to evaluate model

# Turn prediction probabilities into their respective label (easier to␣
 ↪understand)
def get_pred_label(prediction_probabilities):
    """
    Turns an array of prediction probabilities into a label.
    """
    return unique_plant_cat[np.argmax(prediction_probabilities)]

# Create a function to unbatch a batch dataset
def unbatchify(batch_data):
    """
    Take batch data and return unbatch data (separate arrays of images and␣
 ↪labels) in a form of a tuple of lists
    """
    img = []
    lbl = []
```

```python
    for image, label in batch_data.unbatch().as_numpy_iterator():
        img.append(image*255)
        lbl.append(get_pred_label(label))

    return img,lbl

# Show images and prediction rate
def show_img_and_prediction(model, nb_img=9):
    # Get predictions
    predictions = model.predict(dataset_val)
    # Get Validation datset images and true labels
    imgs, labels = unbatchify(dataset_val)
    # Get 10 random images in the validation dataset
    img_rdm = np.random.randint(0, len(imgs), nb_img)

    plt.figure(figsize=(20,12))
    for idx, i in enumerate(img_rdm):
        color = 'red'

        plt.subplot(3,3,idx+1)
        plt.imshow(imgs[i].astype('uint8'))
        plt.xticks([])
        plt.yticks([])

        if get_pred_label(predictions[i]) == labels[i]:
            color = 'green'

        plt.title('Pred({}) : {} - {:2.0f}%'.format(i,␣
↪get_pred_label(predictions[i]), np.max(predictions[i])*100), color=color)
        plt.xlabel('Real: {}'.format(labels[i]));


def plot_acc_and_loss(history):
    """
    From Model History, plot two Graphs:
    - Accuracy Train + Validation
    - Loss Train + Validation

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

    plt.figure(figsize=(16,10))
```

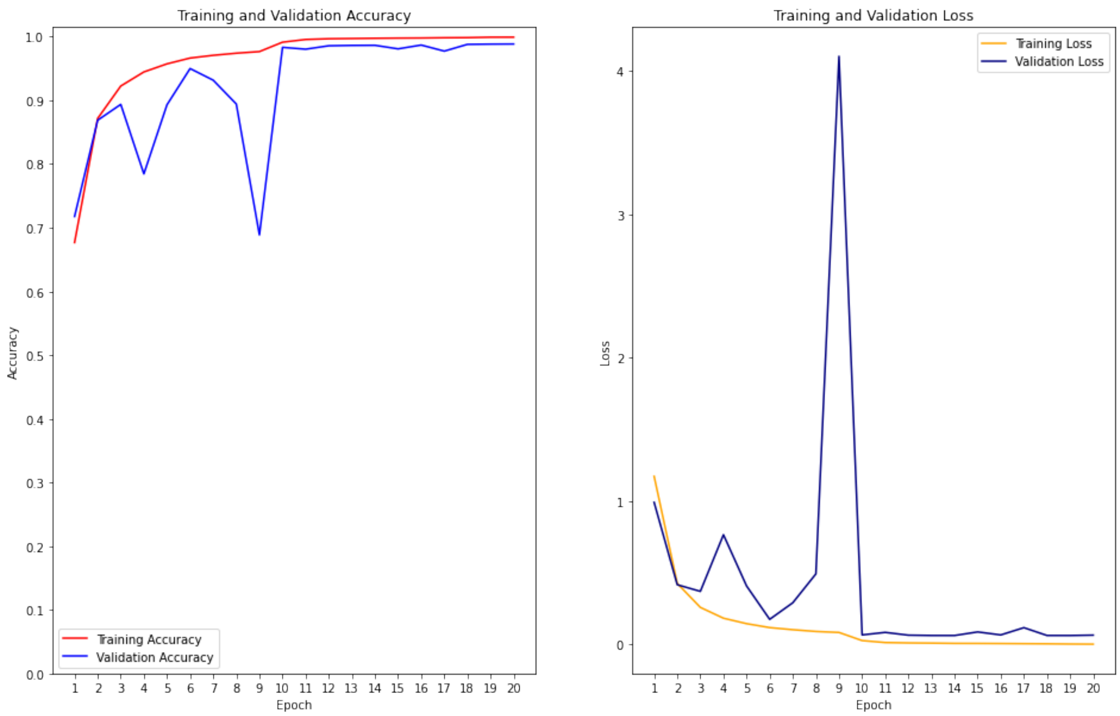
```
    plt.subplot(121)
    plt.plot(epochs, acc, color='red', label='Training Accuracy')
    plt.plot(epochs, val_acc, color='blue', label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.xticks(epochs)
    plt.yticks(np.arange(0,1.1,0.1))
    plt.legend()

    plt.subplot(122)
    plt.plot(epochs, loss, color='orange', label='Training Loss')
    plt.plot(epochs, val_loss, color='navy', label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.xticks(epochs)
    plt.legend()


def plot_pred_prob(predictions, labels, n=1):
    """
    Show the top 3 highest prediction confidences along with the truth label␣
 ↪for sample n.

    Inputs:
      - predictions array
      - labels array
      - n id of sample to check
    """
    pred_prob, true_label = predictions[n], labels[n]

    # Get predicted label
    pred_label = get_pred_label(pred_prob)

    # Get top 3 prediction confidence indexes
    top_3_pred_indexes = pred_prob.argsort()[-3:][::-1]
    # Find the top 3 prediction confidence values
    top_3_pred_values = pred_prob[top_3_pred_indexes]
    # Find the top 3 prediction labels
    top_3_pred_labels = unique_plant_cat[top_3_pred_indexes]

    # Setup plot

    top_plot = plt.barh(np.arange(len(top_3_pred_labels)),
                     top_3_pred_values,
```

```
                    color="grey")
    plt.yticks(np.arange(len(top_3_pred_labels)),
              labels=top_3_pred_labels,
              rotation="horizontal")
    plt.xlabel('Probability')

    # Change color of true label
    if np.isin(true_label, top_3_pred_labels):
        top_plot[np.argmax(top_3_pred_labels == true_label)].set_color("green")
        if top_3_pred_labels[0] != true_label:
            top_plot[0].set_color("red")
        else:
            pass

def get_wrong_preds(predictions, labels, n=9):

    pred_idx = []

    for i, pred_prob in enumerate(predictions):
        pred_label = get_pred_label(pred_prob)
        if pred_label != labels[i]:
            pred_idx.append(i)

        if len(pred_idx) >= n:
            return pred_idx

    return pred_idx
```

[16]:
```
# Get predictions
predictions = model.predict(dataset_val)
# Get Validation datset images and true labels
imgs, labels = unbatchify(dataset_val)
```

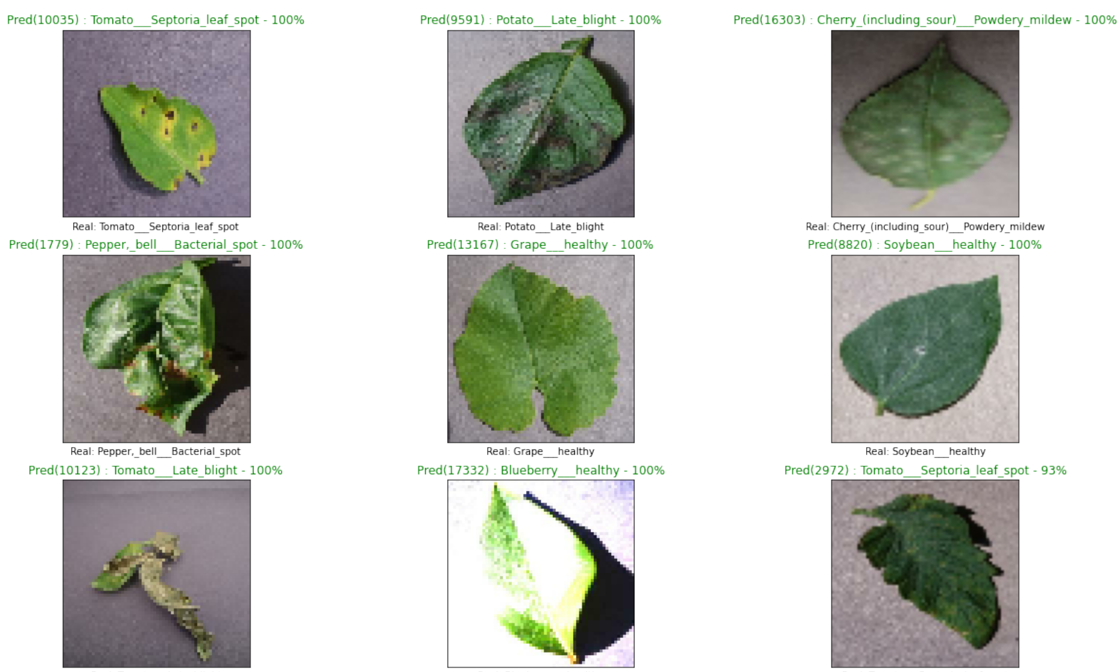
1.2.1 1. Plot Accuracy & Loss

[17]:
```
# Plot Accuracy & Loss
plot_acc_and_loss(history)
```



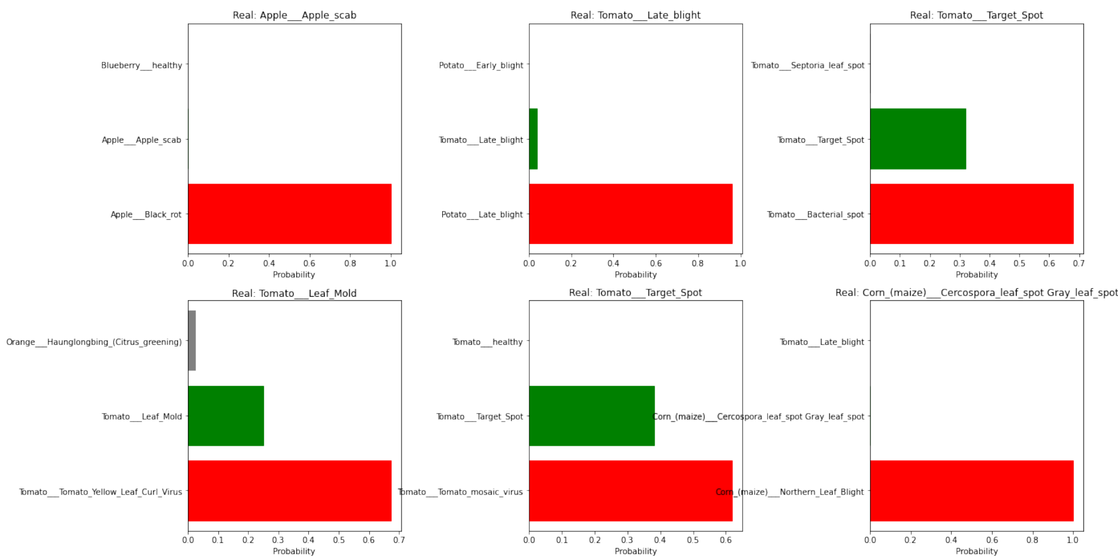
1.2.2 2. Plot Predictions

[18]:
```
show_img_and_prediction(model)
```



1.2.3 3. Plot wrong predictions

[19]:
```
#wrong_pred_idx
wrong_pred_idx = get_wrong_preds(predictions, labels, n=6)

plt.figure(figsize=(20,18))
plt.subplots_adjust(wspace = 0.6)
for i, pred_idx in enumerate(wrong_pred_idx):
    plt.subplot(3,3,i+1)
    plot_pred_prob(predictions, labels, n=pred_idx)
    plt.title('Real: {}'.format(labels[pred_idx]))
```



1.3 Evaluate model with Test dataset

[20]:
```
# Get test data
import os
import pandas as pd


test_path = '../input/new-plant-diseases-dataset/test/test'

test_imgs = [os.path.join(test_path,img) for img in os.listdir(test_path)]
df_test = pd.DataFrame(test_imgs, columns=['Path'])

df_test.head()
```

[20]:
```
                                                Path
0  ../input/new-plant-diseases-dataset/test/test/…
1  ../input/new-plant-diseases-dataset/test/test/…
2  ../input/new-plant-diseases-dataset/test/test/…
3  ../input/new-plant-diseases-dataset/test/test/…
4  ../input/new-plant-diseases-dataset/test/test/…
```

[21]:
```python
#Create Test Dataset
IMG_SIZE = 64
IMG_SHAPE = (IMG_SIZE, IMG_SIZE)

test_dataset = create_dataset(df_test['Path'], test_data=True,␣
 ↪img_shape=IMG_SHAPE)
```

```
Creating data set…
Creating test data batches…
Image size: ((64, 64))
TensorSpec(shape=(None, 64, 64, 3), dtype=tf.float32, name=None)
```

Now that we get our predictions from the Test data we can: - Create a Dataframe with the Test images and the prediciton probabilities for each categories - Show each Test images with Prediction label and Real label

[22]:
```python
# Create a DF with Predictions
preds_df = pd.DataFrame(columns=["id"] + list(unique_plant_cat))
# Append test image ID's to prediction DataFrame
test_ids = [os.path.splitext(path)[0] for path in os.listdir(test_path)]
preds_df["id"] = test_ids
```

[23]:
```python
test_preds = model.predict(test_dataset)
```

[24]:
```python
# Add the prediction probabilities to each plants category columns
preds_df[list(unique_plant_cat)] = test_preds
preds_df.head()
```

[24]:
```
                        id Apple___Apple_scab Apple___Black_rot  \
0      TomatoEarlyBlight6        8.11146e-08       1.65479e-09
1  TomatoYellowCurlVirus4        1.23866e-27                 0
2  TomatoYellowCurlVirus6                  0                 0
3           PotatoHealthy2        9.61807e-17       1.13682e-14
4  TomatoYellowCurlVirus5                  0                 0

  Apple___Cedar_apple_rust Apple___healthy Blueberry___healthy  \
0              7.46612e-08     2.43856e-07          5.0817e-13
1              1.30377e-24     3.28195e-32         3.52738e-36
2                        0               0                   0
3              1.44046e-19     8.95504e-13         5.07863e-16
4              3.15711e-37               0                   0
```

```
  Cherry_(including_sour)___Powdery_mildew Cherry_(including_sour)___healthy  \
0                              3.14819e-11                       4.60061e-15
1                              3.51539e-24                                 0
2                                        0                                 0
3                              9.56348e-19                       2.24454e-13
4                              1.98093e-38                                 0

  Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot  \
0                                        4.07219e-11
1                                                  0
2                                                  0
3                                        3.42566e-26
4                                                  0

  Corn_(maize)___Common_rust_  … Tomato___Bacterial_spot  \
0                 5.54079e-12  …              0.00196833
1                 6.49805e-36  …             1.54836e-17
2                           0  …             8.26207e-31
3                 4.04132e-26  …             2.02316e-26
4                           0  …             1.47491e-19

  Tomato___Early_blight Tomato___Late_blight Tomato___Leaf_Mold  \
0              0.986008          1.23644e-05        0.000105004
1           3.07727e-21          4.57756e-19        1.26733e-22
2                     0                    0                  0
3           6.62256e-20          9.83205e-17        1.40395e-25
4           5.55259e-30          5.39098e-29                  0

  Tomato___Septoria_leaf_spot Tomato___Spider_mites Two-spotted_spider_mite  \
0                 1.58595e-05                                   0.000203834
1                 3.34079e-25                                   4.02564e-25
2                           0                                             0
3                 3.79474e-18                                   6.00498e-19
4                  6.9749e-38                                   3.64209e-35

  Tomato___Target_Spot Tomato___Tomato_Yellow_Leaf_Curl_Virus  \
0            0.0115186                            0.000160498
1          8.89052e-31                                      1
2                    0                                      1
3          1.36065e-16                            6.21335e-26
4                    0                                      1

  Tomato___Tomato_mosaic_virus Tomato___healthy
0                  2.40418e-08      2.47971e-07
1                  4.47605e-22      8.38863e-37
2                            0                0
```

```
3                  2.46068e-21      2.72624e-26
4                            0                0

[5 rows x 39 columns]
```

[ ]:

[ ]: