HW 7

This assignment covers the overall concepts of Neural Networks and Convolutional Neural Networks.

Install Tensorflow

If `import tensorflow as tf` gives Module not found error then it means tensorflow is not installed.

conda: `conda create -n tf tensorflow`
`conda activate tf`

pip: `pip install tensorflow`

Tutorials

- Tensorflow Quickstart
- MNIST Basic Image Classification w Keras

Instructions

Keep the following in mind for all notebooks you develop:

- Structure your notebook.
- Use headings with meaningful levels in Markdown cells, and explain the questions each piece of code is to answer or the reason it is there.
- Make sure your notebook can always be rerun from top to bottom.
- Please start working on this assignment as soon as possible. If you are a beginner in Python this might take a long time. One of the objectives of this assignment is to help you learn python and scikit-learn package.
- See README.md for homework submission instructions

Data Modeling w Tensorflow

DO NOT ERASE MARKDOWN CELLS AND INSTRUCTIONS IN YOUR HW submission

- **Q** - QUESTION
- **A** - Where to input your answer

Data Preparation

We have provided a leaf dataset which contains over 1000 leaf disease images with their ground truth labels in the csv file. You will find the dataset under the data folder. Download the dataset and complete the homework as per below instructions.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
print(tf.__version__)
```

```
2.16.1
```

Tensorflow Dataset

We are using Tensorflow to train an image classifier model. Tensorflow has it's own apis for creating a data pipeline which makes it easier to feed the data into a Tensorflow model.

- To use any suitable api to create the data needs to be loaded into a Tensorflow data object.
- See this API for help: Directory Dataset
- Tensorflow Dataset

Q1 Create Data pipeline using the tensorflow api from the image list and labels

1. Create tensorflow train dataset (75% of the whole data)
2. Create tensorflow validation dataset (25% of the whole data)
3. Preprocess the dataset (Apply Horizontal and vertical flips)
4. Resize all images to 128X128
5. Keep the train batch size 32 and validation batch size 16

A1 Replace ??? with code in the code cell below

Disclaimer

In the below code, I was able to use image_dataset_from_directory() after reading the Tensorflow documentation. The given function seems to accomplish all of the above steps EXCEPT applying Horizontal and Vertical Flips. To accomplish the Horizontal and Vertical flips, I asked ChatGPT for assistance. This increased my accuracy significantly when fitting the model.

In [ ]:
```
#tf_train_data =
#tf_val_data =
from tensorflow.keras.layers import RandomFlip

# Performs Horizontal and Vertical Flips.
flips = tf.keras.Sequential([
    RandomFlip("horizontal_and_vertical")
])

seed=128
```

```
directory = "train/"
tf_train_data = tf.keras.preprocessing.image_dataset_from_directory(
    directory,
    labels='inferred',
    label_mode='int',
    class_names=None,
    color_mode='rgb',
    #batch_size=32, # defaults to 32
    image_size=(128, 128),
    shuffle=True,
    seed=seed,
    validation_split=0.25,
    subset="training",
    #interpolation defaults to linear
)#.map(lambda x, y: (flips(x), y)).prefetch(tf.data.AUTOTUNE)

class_names = tf_train_data.class_names

tf_train_data = tf_train_data.map(lambda x, y: (flips(x), y)).prefetch(tf.data.AUTOTUN

tf_val_data = tf.keras.preprocessing.image_dataset_from_directory(
    directory,
    labels='inferred',
    label_mode='int',
    class_names=None,
    color_mode='rgb',
    batch_size=16,
    image_size=(128, 128),
    shuffle=True,
    seed=seed,
    validation_split=0.25,
    subset="validation",

).map(lambda x, y: (flips(x), y)).prefetch(tf.data.AUTOTUNE)
```

```
Found 1034 files belonging to 3 classes.
Using 776 files for training.
Found 1034 files belonging to 3 classes.
Using 258 files for validation.
```

Image Visualization

Before going on to form the data pipeline, lets have a look at some of the images in the dataset and visualise them with their labels.

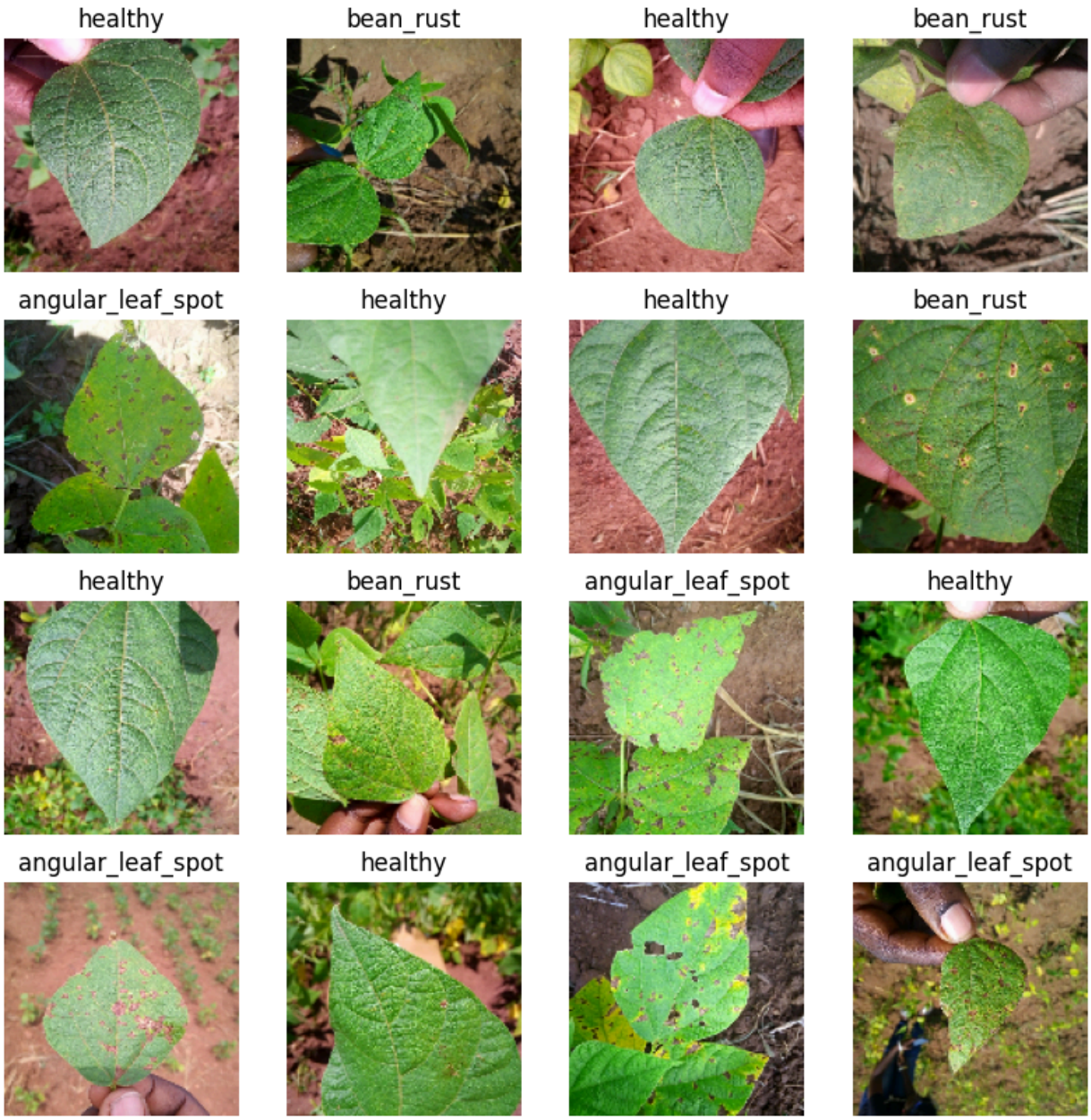
Q2

1. Plot 16 images with their class name from the first batch of the train data (Use 4X4 plot graph)
2. See if images are from different class. If not, modify the train and validation data so that we have different classes in every batch.

A2 Add cells as per your need below.

In [ ]:
```
plt.figure(figsize=(10, 10))
for images, labels in tf_train_data.take(1):
```

```
    for i in range(16):
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
    plt.show()
```



In [ ]:

Model Definition

We use Tensorflow to define the model that is Modified LeNet inspired Keras example.

1. All average pooling layers have been replaced with max pooling layers
2. The input shape is 128X128. The first convolutional layer uses "same" padding to ensure the data is in the same shape as it is in the paper by the time it hits the first pooling layer.

Q3 Create CNN Model on the training set, and evaluate

1. The model will be of 3 ConV net and followed by MaxPooling after each of them

- Layer 1 is a convolutional layer (ConV net): Kernel is 3X3 and number of kernel is 6 and `relu` as activation function
  - Maxpool after 1st ConV net: 2X2
  - Add 30% Dropout Layer
- Layer 2 is a convolutional layer (ConV net): Kernel is 5X5 and number of kernel is 10 and `relu` as activation function
  - Maxpool after 2nd ConV net: 2X2
  - Add 30% Dropout Layer
- Layer 3 is a convolutional layer (ConV net): Kernel is 5X5 and number of kernel is 16 and `relu` as activation function
  - Maxpool after 2nd ConV net: 2X2
  - Add 30% Dropout Layer
  - Flatten this output

1. Layer 4 is a dense layer with 1024 output units using `relu` as activation function
2. Layer 5 is a dense layer with 512 output units using `relu` as activation function
3. Layer 6 is a dense layer with 128 output units using `relu` as activation function
4. Layer 7 (last layer) is a Dense Layer that is using `softmax` as an activation function

A3 Replace ??? with code in the code cell below

In [ ]:
```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

In [ ]:
```
model = tf.keras.Sequential([

# Fill this block with all layers
    Conv2D(6, (3,3), activation='relu', input_shape=(128, 128, 3)),
    MaxPooling2D((2,2)),
    Dropout(0.3),

    Conv2D(10, (5,5), activation='relu'),
    MaxPooling2D((2,2)),
    Dropout(0.3),

    Conv2D(16, (5,5), activation='relu'),
    MaxPooling2D((2,2)),
    Dropout(0.3),
    Flatten(),

    Dense(1024, activation='relu'),
    Dense(512, activation='relu'),
    Dense(128, activation='relu'),

    Dense(10, activation='softmax')
])
```

```
C:\Users\Hunter\AppData\Roaming\Python\Python311\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Optimizer and Loss Function

Q4 Lets set an optimizer for the model training. Use Tensorflow Adam as the optimizer with the learning rate of 0.001 and the loss function of sparse categorical crossentropy. We will keep accuracy as the evaluation measure. Print model summary and see the number of trainable weights

A4 Replace ??? with code in the code cell below

In [ ]:
```
#optimiser = tf.keras.optimizers.?(learning_rate=?)

#model.compile(
#    optimizer=?,
#    loss=?,
#    metrics=['?'])

#model.summary()
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 6)	168
max_pooling2d (MaxPooling2D)	(None, 63, 63, 6)	0
dropout (Dropout)	(None, 63, 63, 6)	0
conv2d_1 (Conv2D)	(None, 59, 59, 10)	1,510
max_pooling2d_1 (MaxPooling2D)	(None, 29, 29, 10)	0
dropout_1 (Dropout)	(None, 29, 29, 10)	0
conv2d_2 (Conv2D)	(None, 25, 25, 16)	4,016
max_pooling2d_2 (MaxPooling2D)	(None, 12, 12, 16)	0
dropout_2 (Dropout)	(None, 12, 12, 16)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 1024)	2,360,320
dense_1 (Dense)	(None, 512)	524,800
dense_2 (Dense)	(None, 128)	65,664
dense_3 (Dense)	(None, 10)	1,290

Total params: 2,957,768 (11.28 MB)

Trainable params: 2,957,768 (11.28 MB)

Non-trainable params: 0 (0.00 B)

We have loaded the training data, and compiled the model, now it is time to train it. The validation dataset has been included here so that the model validates its accuracy after each step by making predictions against the validation dataset. Each epoch takes under a minute to complete so this may take a few minutes to run depending on your machine.

Train the CNN and Plot the Accuracy

Q5 Start the training using fit function

1. Pass train data inside fit function
2. Provide validation data inside fit function
3. Define number of epochs(For faster training set it equal to 15)

A5 Replace ??? with code in the code cell below

In [ ]:
```
#train_log = model.fit(
#    ?,
```

```
#    validation_data=?,
#    epochs=?
#)

train_log = model.fit(
    tf_train_data,
    validation_data=tf_val_data,
    epochs=15
)
```

```
Epoch 1/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 4s 80ms/step - accuracy: 0.3391 - loss: 59.5702 - val_accu
racy: 0.3372 - val_loss: 1.6443
Epoch 2/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 74ms/step - accuracy: 0.3474 - loss: 1.1773 - val_accur
acy: 0.3837 - val_loss: 1.6428
Epoch 3/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 77ms/step - accuracy: 0.3631 - loss: 1.1181 - val_accur
acy: 0.4264 - val_loss: 1.4182
Epoch 4/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.4259 - loss: 1.0907 - val_accur
acy: 0.3798 - val_loss: 1.3090
Epoch 5/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 74ms/step - accuracy: 0.4481 - loss: 1.0479 - val_accur
acy: 0.4884 - val_loss: 1.3074
Epoch 6/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - accuracy: 0.4196 - loss: 1.0438 - val_accur
acy: 0.4419 - val_loss: 1.2527
Epoch 7/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 74ms/step - accuracy: 0.4478 - loss: 1.0698 - val_accur
acy: 0.4767 - val_loss: 1.2269
Epoch 8/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - accuracy: 0.4671 - loss: 1.0490 - val_accur
acy: 0.5116 - val_loss: 1.2050
Epoch 9/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 76ms/step - accuracy: 0.4420 - loss: 1.0368 - val_accur
acy: 0.5116 - val_loss: 1.1461
Epoch 10/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - accuracy: 0.4972 - loss: 1.0217 - val_accur
acy: 0.5543 - val_loss: 1.2242
Epoch 11/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.4691 - loss: 1.0105 - val_accur
acy: 0.5775 - val_loss: 1.1307
Epoch 12/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.4977 - loss: 0.9965 - val_accur
acy: 0.5349 - val_loss: 1.1787
Epoch 13/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - accuracy: 0.5566 - loss: 0.9376 - val_accur
acy: 0.5465 - val_loss: 1.1601
Epoch 14/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 73ms/step - accuracy: 0.5796 - loss: 0.8938 - val_accur
acy: 0.5930 - val_loss: 1.1465
Epoch 15/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 74ms/step - accuracy: 0.6050 - loss: 0.9046 - val_accur
acy: 0.5775 - val_loss: 1.1606
```

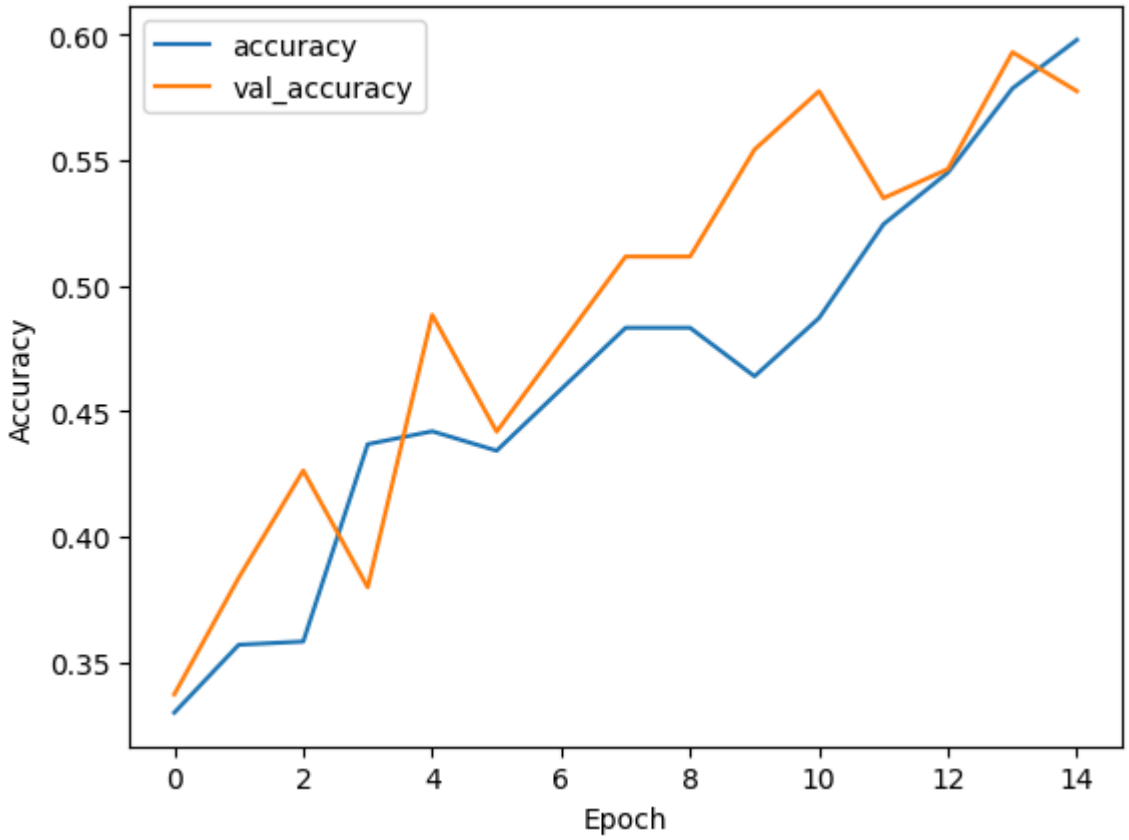
Q6 Create plot on the training set, and validation set. Use `history` attribute from the above train_log for creating accuracy curve.

A6 Replace ??? with code in the code cell below

In [ ]:
```
#plt.plot(train_log.??, label='accuracy')
#plt.plot(train_log.??, label = 'val_accuracy')
plt.plot(train_log.history['accuracy'], label='accuracy')
plt.plot(train_log.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

#print('Training accuracy: %f' % ?)
#print('Validation accuracy: %f' % ?)
print(f'Training accuracy: {train_log.history["accuracy"][-1]:.2f}')
print(f'Validation accuracy: {train_log.history["val_accuracy"][-1]:.2f}')
```

```
Training accuracy: 0.60
Validation accuracy: 0.58
```



Plot few validation set prediction

Now that we have the trained model, lets use it to classify some images. The model will output a probability for each possible class. Numpys argmax is thus used to find the class with the highest probability and this class is used for the image.

Q7 Use the trained model to predict the first batch of the validation set

1. Pass first batch validation data inside predict function of trained model
2. Use numpy `argmax` to get the index with best classificaion score.
3. Find the corresponding class name from the prediction class id.

4. Plot the predicted class name with the images same as you did for the `Q. No 2` (Use 4X4 graph for plotting)

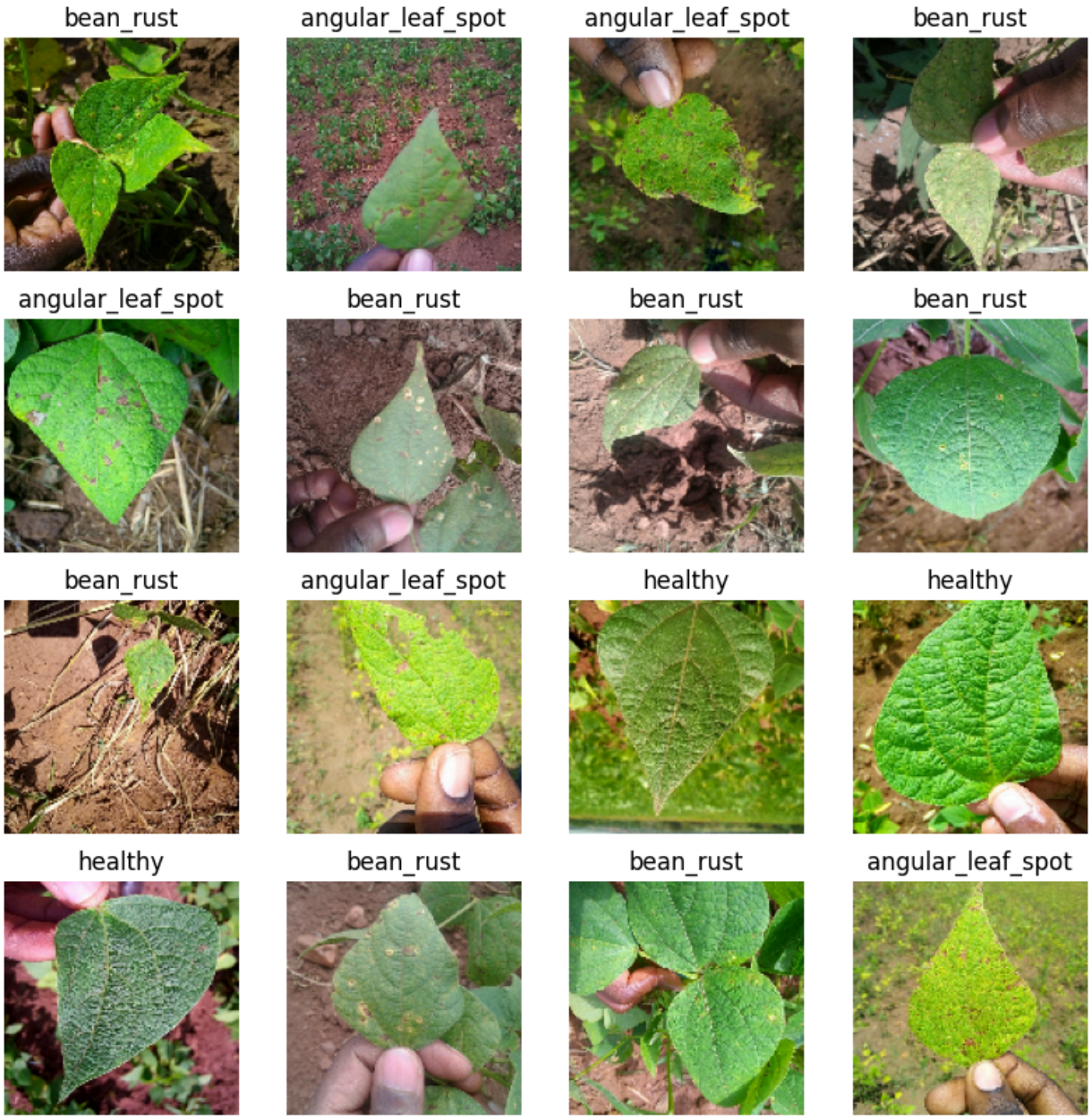
A7 Add any number of cells you need

In [ ]:
```
#predictions = model.predict(??)
#predictions = np.argmax(??)
predictions = model.predict(tf_val_data.take(1))
predictions = np.argmax(predictions, axis=1)
print(predictions)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
[2 2 1 0 2 2 1 2 0 0 2 0 2 0 1 2]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
[2 2 1 0 2 2 1 2 0 0 2 0 2 0 1 2]
```

In [ ]:
```
predicted_class_names = [class_names[i] for i in predictions]

plt.figure(figsize=(10, 10))
for images, labels in tf_val_data.take(1):
    for i in range(16):
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
    plt.show()
```



Additional Reading

- Neural Network
- Convolutional Neural Network
- Tensorflow Framework