Bean Disease Assignment Solution

I hope you enjoyed using Generators to help develop a model to solve a real-world problem!

There are many possible solutions to this assignment.

For the generators, at a minimum you should have had both the training and validation Generators rescale the image data. This can be done using the training Generator as an example:

```
train_datagen = ImageDataGenerator(
            rescale=1./255
)
```

If you wanted you could have gone further and applied additional image augmentation techniques to the training Generator to have a better dataset for training. A possible solution could be:

```
train_datagen = ImageDataGenerator(
            rescale=1./255,
            rotation_range=40,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            fill_mode='nearest'
)
```

You then needed to supply the following values to get the Generators to work:

```
TRAIN_DIRECTORY_LOCATION = '/tmp/train'
VAL_DIRECTORY_LOCATION = '/tmp/validation'
TARGET_SIZE = (224,224)
CLASS_MODE = 'categorical'
```

For the model, again there isn't one answer. One good approach (as suggested by the hints), is again to use convolutional and pooling layers to find high level features and then dense layers to classify those features. A possible solution to the model design is shown below:

```
model = tf.keras.models.Sequential([
                # Find the features with Convolutions and Pooling
            tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(224, 224, 3)),
            tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
            tf.keras.layers.MaxPooling2D(2,2),
            tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
            tf.keras.layers.MaxPooling2D(2,2),
            tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
            tf.keras.layers.MaxPooling2D(2,2),
            # Flatten the results to feed into a DNN
            tf.keras.layers.Flatten(),
            # 512 neuron hidden layer
            tf.keras.layers.Dense(512, activation='relu'),
            tf.keras.layers.Dense(3, activation='softmax')
])
```

For this assignment an optimizer with an adaptive learning rate often performs quite well and for the loss we want something that can help optimize for differences across categories. Again there are many possible options that will work. A possible solution is:

```
OPTIMIZER = 'adam'
LOSS = 'categorical_crossentropy'
```

Another optimizer that works well is:

```
from tensorflow.keras.optimizers import RMSprop
OPTIMIZER = RMSprop(lr=0.0001)
```

As for Epochs as mentioned in the hint, something in the low tens is a good place to start. Try with 20 Epochs.