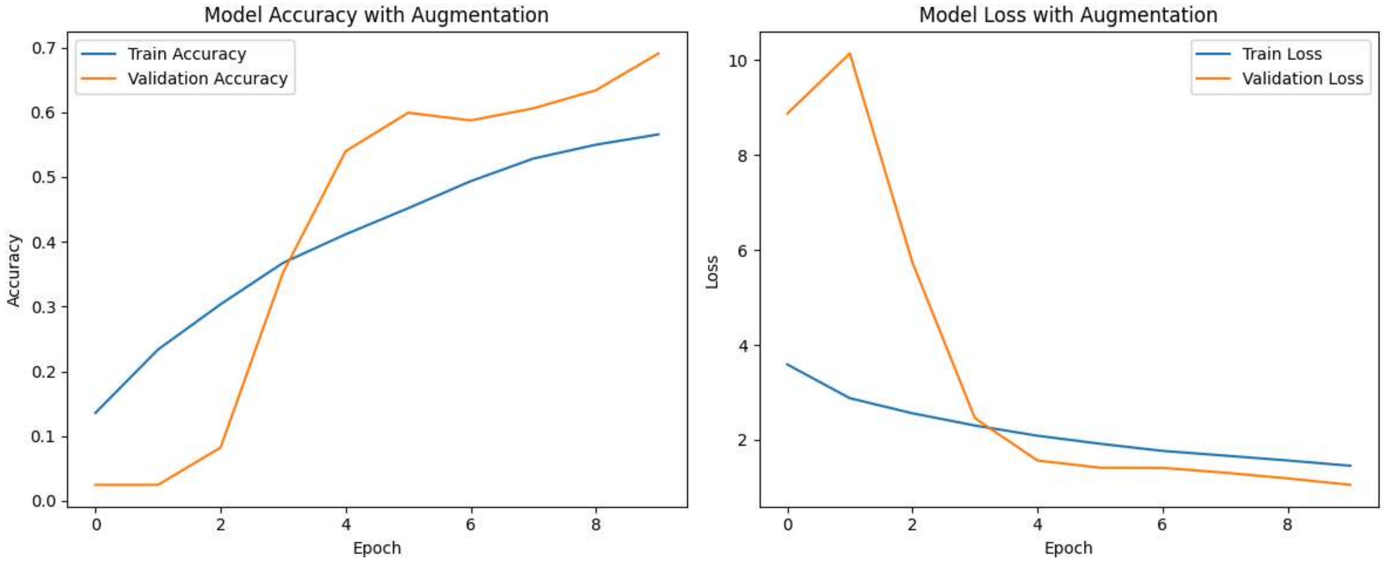
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Define dataset path
dataset_path = "/content/drive/MyDrive/Medicinal_plant_dataset"

# Image data generator for augmenting and normalizing the dataset
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2  # Reserve 20% of the data for validation
)

# Create training and validation generators
train_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=(128, 128),  # Resize images
    batch_size=32,
    class_mode='sparse',  # Use sparse if labels are integers
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=(128, 128),
    batch_size=32,
    class_mode='sparse',  # Use sparse if labels are integers
    subset='validation'
)

# Define the CNN model with Dropout and Batch Normalization
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),  # Dropout layer to reduce overfitting
    layers.Dense(len(train_generator.class_indices), activation='softmax')  # Number of classes
])

# Compile the model with a lower learning rate
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    epochs=10,  # Increased epochs for better training
    validation_data=validation_generator
)

# Evaluate the model
loss, accuracy = model.evaluate(validation_generator)
print(f"Validation Loss: {loss:.4f}, Validation Accuracy: {accuracy:.4f}")

# Plot training & validation accuracy and loss
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy with Augmentation')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss with Augmentation')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

plt.tight_layout()
plt.show()
```

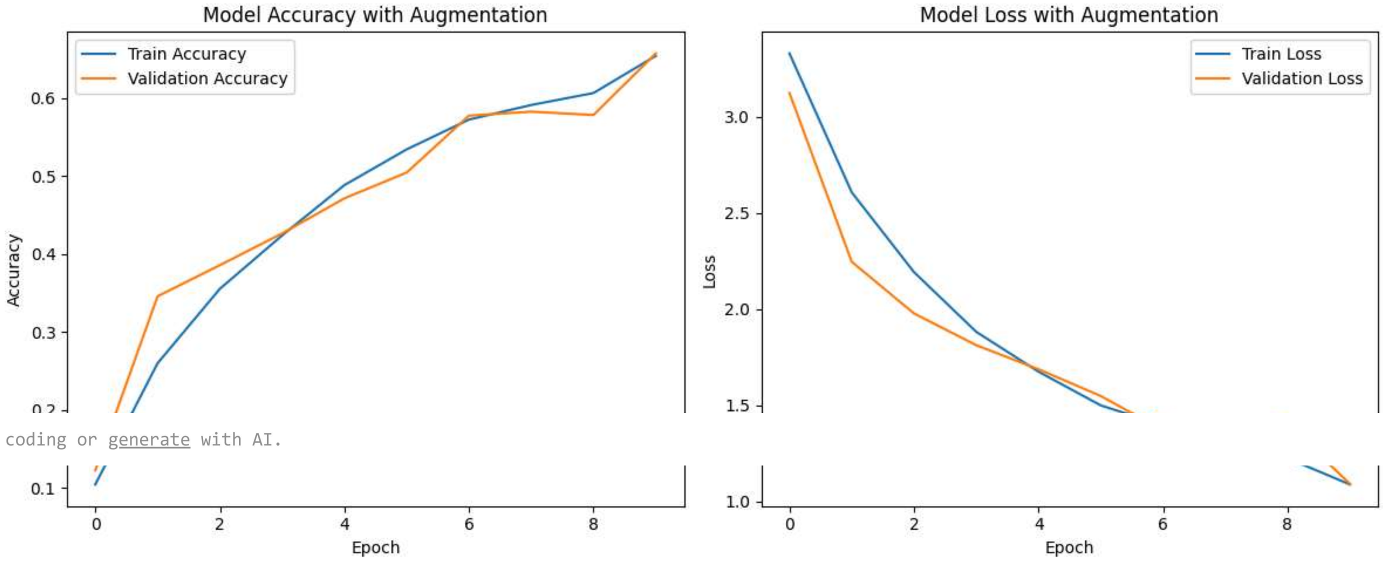
```
Found 4765 images belonging to 40 classes.
Found 1180 images belonging to 40 classes.
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 376s 2s/step - accuracy: 0.0949 - loss: 4.2364 - val_accuracy: 0.0246 - val_loss: 8.8724
Epoch 2/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 379s 2s/step - accuracy: 0.2244 - loss: 2.9382 - val_accuracy: 0.0246 - val_loss: 10.1392
Epoch 3/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 388s 2s/step - accuracy: 0.2806 - loss: 2.6407 - val_accuracy: 0.0822 - val_loss: 5.7389
Epoch 4/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 366s 2s/step - accuracy: 0.3565 - loss: 2.3287 - val_accuracy: 0.3525 - val_loss: 2.4551
Epoch 5/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 345s 2s/step - accuracy: 0.3991 - loss: 2.1157 - val_accuracy: 0.5398 - val_loss: 1.5644
Epoch 6/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 350s 2s/step - accuracy: 0.4566 - loss: 1.9383 - val_accuracy: 0.5992 - val_loss: 1.4110
Epoch 7/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 361s 2s/step - accuracy: 0.4757 - loss: 1.7947 - val_accuracy: 0.5873 - val_loss: 1.4080
Epoch 8/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 342s 2s/step - accuracy: 0.5163 - loss: 1.7108 - val_accuracy: 0.6059 - val_loss: 1.3068
Epoch 9/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 336s 2s/step - accuracy: 0.5492 - loss: 1.5779 - val_accuracy: 0.6339 - val_loss: 1.1871
Epoch 10/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 395s 2s/step - accuracy: 0.5438 - loss: 1.4976 - val_accuracy: 0.6907 - val_loss: 1.0503
37/37 ━━━━━━━━━━━━━━━━━━━━ 26s 683ms/step - accuracy: 0.6650 - loss: 1.0683
Validation Loss: 1.0643, Validation Accuracy: 0.6653
```



```
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import Sequence
import os
import tensorflow as tf
```

```
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt


class ImageMaskGenerator(Sequence):
    def __init__(self, image_dir, mask_dir, batch_size, img_size=(128, 128), augment=False):
        # Only keep files, not directories
        self.image_filenames = [f for f in os.listdir(image_dir) if os.path.isfile(os.path.join(image_dir, f))]
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.batch_size = batch_size
        self.img_size = img_size
        self.augment = augment
        self.datagen = ImageDataGenerator(
            rescale=1.0/255.0,
            rotation_range=20,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            fill_mode='nearest') if augment else ImageDataGenerator(rescale=1.0/255.0)
# Define a CNN model for classification + segmentation
inputs = layers.Input(shape=(128, 128, 3))

# Encoder
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)

x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)

x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
encoded = layers.MaxPooling2D(pool_size=(2, 2))(x)

# Decoder for segmentation
segmentation_output = layers.Conv2DTranspose(64, (3, 3), strides=(2, 2), padding='same', activation='relu')(encoded)
segmentation_output = layers.Conv2DTranspose(32, (3, 3), strides=(2, 2), padding='same', activation='relu')(segmentation_output)
segmentation_output = layers.Conv2DTranspose(1, (3, 3), strides=(2, 2), padding='same', activation='sigmoid', name='segmentation')(segmentat

# Flatten and Classification output
x_flattened = layers.Flatten()(encoded)
x_dense = layers.Dense(512, activation='relu')(x_flattened)
x_dropout = layers.Dropout(0.5)(x_dense)
classification_output = layers.Dense(30, activation='softmax', name='classification')(x_dropout)  # Assume 30 classes

# Combine both outputs
model = models.Model(inputs=inputs, outputs=[classification_output, segmentation_output])

# Compile the model for multi-task learning
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss={'classification': 'sparse_categorical_crossentropy', 'segmentation': 'binary_crossentropy'},
              metrics={'classification': 'accuracy', 'segmentation': 'accuracy'})


# Define dataset path
dataset_path = "/content/drive/MyDrive/Medicinal_plant_dataset"

# Image data generator for augmenting and normalizing the dataset
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2  # Reserve 20% of the data for validation
)

# Create training and validation generators
train_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=(128, 128),  # Resize images
    batch_size=32,
    class_mode='sparse',  # Use sparse if labels are integers
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=(128, 128),
    batch_size=32,
    class_mode='sparse',  # Use sparse if labels are integers
    subset='validation'
)

# Define the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dense(len(train_generator.class_indices), activation='softmax')  # Number of classes
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)

# Evaluate the model
loss, accuracy = model.evaluate(validation_generator)
print(f"Validation Loss: {loss:.4f}, Validation Accuracy: {accuracy:.4f}")

# Plot training & validation accuracy and loss
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy with Augmentation')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss with Augmentation')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Found 4765 images belonging to 40 classes.
Found 1180 images belonging to 40 classes.
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class
  self._warn_if_super_not_called()
149/149 ━━━━━━━━━━━━━━━━━━━━ 1651s 11s/step - accuracy: 0.0577 - loss: 3.5372 - val_accuracy: 0.1229 - val_loss: 3.1231
Epoch 2/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 293s 2s/step - accuracy: 0.2209 - loss: 2.7584 - val_accuracy: 0.3458 - val_loss: 2.2448
Epoch 3/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 243s 1s/step - accuracy: 0.3401 - loss: 2.2213 - val_accuracy: 0.3856 - val_loss: 1.9762
Epoch 4/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 279s 2s/step - accuracy: 0.4107 - loss: 1.9120 - val_accuracy: 0.4263 - val_loss: 1.8109
Epoch 5/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 221s 1s/step - accuracy: 0.4770 - loss: 1.7008 - val_accuracy: 0.4712 - val_loss: 1.6860
Epoch 6/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 262s 1s/step - accuracy: 0.5440 - loss: 1.4766 - val_accuracy: 0.5042 - val_loss: 1.5465
Epoch 7/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 224s 1s/step - accuracy: 0.5617 - loss: 1.4100 - val_accuracy: 0.5771 - val_loss: 1.3689
Epoch 8/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 224s 1s/step - accuracy: 0.5811 - loss: 1.3469 - val_accuracy: 0.5822 - val_loss: 1.3523
Epoch 9/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 220s 1s/step - accuracy: 0.6038 - loss: 1.2396 - val_accuracy: 0.5780 - val_loss: 1.3988
Epoch 10/10
149/149 ━━━━━━━━━━━━━━━━━━━━ 237s 2s/step - accuracy: 0.6650 - loss: 1.0683 - val_accuracy: 0.6568 - val_loss: 1.0900
37/37 ━━━━━━━━━━━━━━━━━━━━ 24s 657ms/step - accuracy: 0.6605 - loss: 1.1201
Validation Loss: 1.0862, Validation Accuracy: 0.6669
```



Start coding or generate with AI.