Name: Gabriel Bentley

Class: CS 4375.003

Explaining the imported data

- Data set name: Agriculture crop images
- List of crop types: There are 5 different crop types; maize, wheat, jute, rice and sugarcane.
- Image amount: In the original data set there are 40 images in each category but, when downloaded there was a file containing an additional 10 pictures for each category. There is a total of 251 images in the data set
- What the model should predict: The purpose of the set is to build a model that can decide if a given image is of maize, wheat, jute, rice, or sugercane.
- Website: https://www.kaggle.com/datasets/aman2000jaiswal/agriculture-crop-images

Importing the data and putting it into training and testing groups

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import zipfile
import os


cwd = os.getcwd()
print(cwd)

    /content
```

Unzip the crop file

```
os.environ['KAGGLE_CONFIG_DIR'] = "/content"

zip_ref = zipfile.ZipFile('Crops.zip', 'r') #Opens the zip file in read mode
zip_ref.extractall('/content') #Extracts the files into the /content folder
zip_ref.close()
```

read in the images into training_data

```
IMG_SIZE = 60

DATADIR = "Crops"
CATEGORIES = ["jute", "maize", "rice", "sugarcane", "wheat"]
training_data = []

for category in CATEGORIES:
  path = os.path.join(DATADIR, category)
  class_num = CATEGORIES.index(category)
  for img in os.listdir(path):
    try:
      img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
      new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
      training_data.append([new_array, class_num])
    except Exception as e:
        pass
```

Randomizing the order of the images

```
import random

random.shuffle(training_data)
```

Seperate the data from the labels (x is data, y is label)

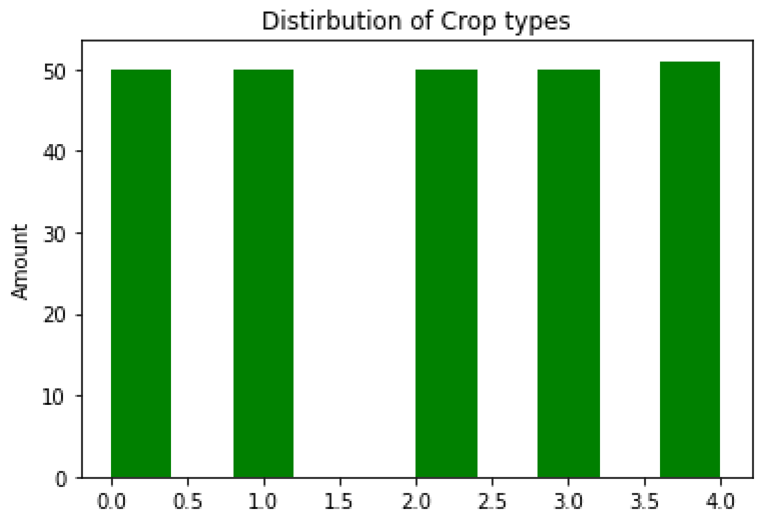
```
X = []
Y = []

for i, j in training_data:
  X.append(i)
  Y.append(j)

X = np.array(X).reshape(-1,IMG_SIZE, IMG_SIZE)
Y = np.array(Y)
```

```
print(X.shape, "X shape")
print(Y.shape, "Y shape")
```

```
    (251, 60, 60) X shape
    (251,) Y shape
```

Plot the spread of classes in the data set. There is an even distribution of crop types with each crop type having 50 pictures (except for wheat which has 51) coming to a total of 251 pictures to run through the model.

```
plt.hist(Y, color='g')
plt.gca().set(title='Distirbution of Crop types', ylabel='Amount')
plt.show()
```



Split into train and test

```
X_train = X[:200]
X_test =  X[-51:]

Y_train = Y[:200]
Y_test = Y[-51:]

print(len(X_train))
print(len(X_test))
```

```
    200
    51
```

```
import tensorflow as tf
```

```
batch_size = 8
num_classes = 5
epochs = 20
```

Normalize the data by dividing them by 255.0

```
X_train, X_test = X_train / 255.0, X_test / 255.0

print(X_train.shape, 'train samples')
print(X_test.shape, 'test samples')
```

```
(200, 60, 60) train samples
(51, 60, 60) test samples
```

```
Y_train = tf.keras.utils.to_categorical(Y_train, num_classes)
Y_test = tf.keras.utils.to_categorical(Y_test, num_classes)
```

Create the sequential model

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(IMG_SIZE, IMG_SIZE)),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(num_classes, activation='softmax'),
])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 3600)              0

 dense (Dense)               (None, 512)               1843712

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 512)               262656

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 5)                 2565

=================================================================
Total params: 2,108,933
Trainable params: 2,108,933
Non-trainable params: 0
_________________________________________________________________
```

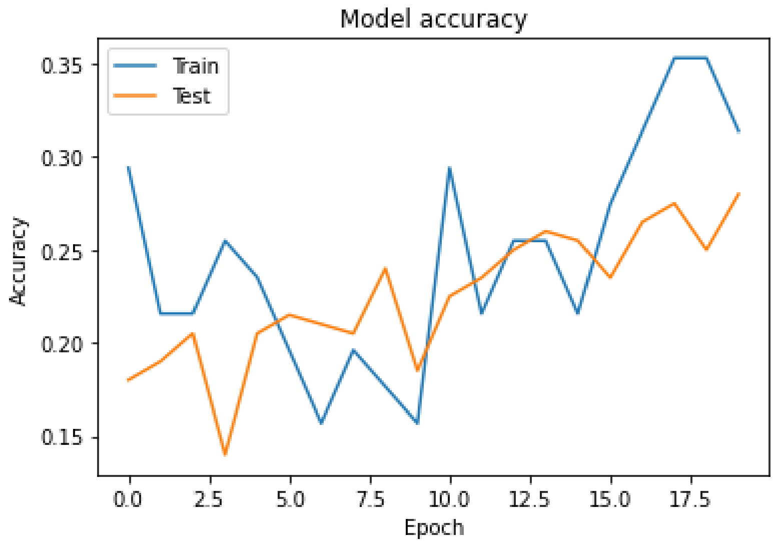
```
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

history = model.fit(X_train, Y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_test, Y_test))
```

```
Epoch 1/20
25/25 [==============================] - 2s 33ms/step - loss: 4.5796 - accuracy:
Epoch 2/20
25/25 [==============================] - 1s 25ms/step - loss: 1.7636 - accuracy:
Epoch 3/20
25/25 [==============================] - 1s 25ms/step - loss: 1.6615 - accuracy:
Epoch 4/20
25/25 [==============================] - 1s 26ms/step - loss: 1.6234 - accuracy:
Epoch 5/20
25/25 [==============================] - 1s 25ms/step - loss: 1.6196 - accuracy:
Epoch 6/20
25/25 [==============================] - 1s 26ms/step - loss: 1.6187 - accuracy:
Epoch 7/20
25/25 [==============================] - 1s 25ms/step - loss: 1.6122 - accuracy:
Epoch 8/20
25/25 [==============================] - 1s 25ms/step - loss: 1.6095 - accuracy:
Epoch 9/20
25/25 [==============================] - 1s 25ms/step - loss: 1.6169 - accuracy:
Epoch 10/20
25/25 [==============================] - 1s 23ms/step - loss: 1.6102 - accuracy:
Epoch 11/20
25/25 [==============================] - 1s 25ms/step - loss: 1.5888 - accuracy:
Epoch 12/20
25/25 [==============================] - 1s 24ms/step - loss: 1.6057 - accuracy:
Epoch 13/20
25/25 [==============================] - 1s 25ms/step - loss: 1.5992 - accuracy:
Epoch 14/20
25/25 [==============================] - 1s 26ms/step - loss: 1.5836 - accuracy:
Epoch 15/20
25/25 [==============================] - 1s 24ms/step - loss: 1.5841 - accuracy:
Epoch 16/20
25/25 [==============================] - 1s 26ms/step - loss: 1.5811 - accuracy:
Epoch 17/20
25/25 [==============================] - 1s 24ms/step - loss: 1.5727 - accuracy:
Epoch 18/20
25/25 [==============================] - 1s 26ms/step - loss: 1.5696 - accuracy:
Epoch 19/20
25/25 [==============================] - 1s 26ms/step - loss: 1.5681 - accuracy:
Epoch 20/20
25/25 [==============================] - 1s 26ms/step - loss: 1.5777 - accuracy:
```

```
history.history.keys()

    dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Plot the results of the sequential model

```
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

    Test loss: 1.5588717460632324
    Test accuracy: 0.3137255012989044
```

CNN method

```
batch_size = 8
num_classes = 5
epochs = 20
```

```
num_filters = 8
filter_size = 3
pool_size = 2
```

```
IMG_SIZE = 60
```

Create the CNN model

```
model = tf.keras.models.Sequential(
    [
        tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 1)),
        tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 58, 58, 32)        320

 max_pooling2d_2 (MaxPooling  (None, 29, 29, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 27, 27, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 13, 13, 64)       0
 2D)

 flatten_2 (Flatten)         (None, 10816)             0

 dropout_3 (Dropout)         (None, 10816)             0

 dense_4 (Dense)             (None, 5)                 54085

=================================================================
Total params: 72,901
Trainable params: 72,901
Non-trainable params: 0
_________________________________________________________________
```

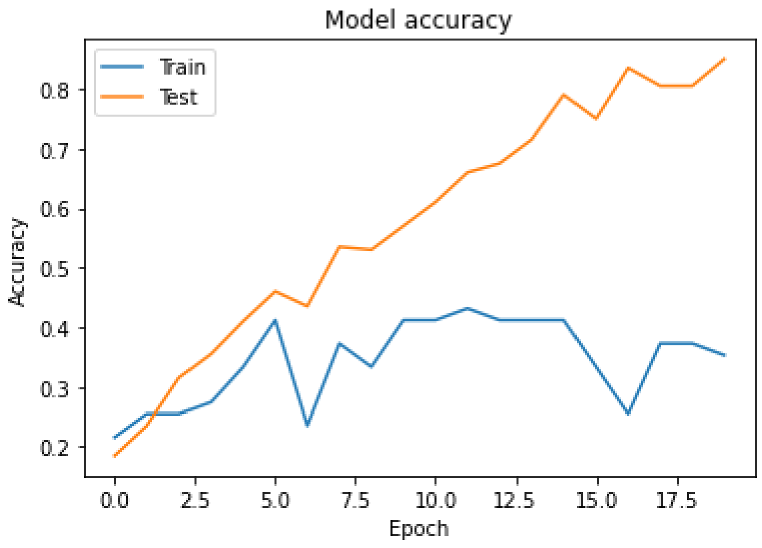
```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
history = model.fit(X_train, Y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_test, Y_test))
```

```
Epoch 1/20
25/25 [==============================] - 2s 56ms/step - loss: 1.6230 - accuracy:
Epoch 2/20
25/25 [==============================] - 1s 45ms/step - loss: 1.5814 - accuracy:
Epoch 3/20
25/25 [==============================] - 1s 49ms/step - loss: 1.5227 - accuracy:
Epoch 4/20
25/25 [==============================] - 1s 46ms/step - loss: 1.4894 - accuracy:
Epoch 5/20
25/25 [==============================] - 1s 47ms/step - loss: 1.4341 - accuracy:
Epoch 6/20
25/25 [==============================] - 1s 46ms/step - loss: 1.3844 - accuracy:
Epoch 7/20
25/25 [==============================] - 1s 47ms/step - loss: 1.3690 - accuracy:
Epoch 8/20
25/25 [==============================] - 1s 48ms/step - loss: 1.2745 - accuracy:
Epoch 9/20
25/25 [==============================] - 1s 46ms/step - loss: 1.2303 - accuracy:
Epoch 10/20
25/25 [==============================] - 1s 47ms/step - loss: 1.1721 - accuracy:
Epoch 11/20
25/25 [==============================] - 1s 48ms/step - loss: 1.0440 - accuracy:
Epoch 12/20
25/25 [==============================] - 1s 48ms/step - loss: 0.9552 - accuracy:
Epoch 13/20
25/25 [==============================] - 1s 47ms/step - loss: 0.8864 - accuracy:
Epoch 14/20
25/25 [==============================] - 1s 47ms/step - loss: 0.7979 - accuracy:
Epoch 15/20
25/25 [==============================] - 1s 47ms/step - loss: 0.7030 - accuracy:
Epoch 16/20
25/25 [==============================] - 1s 48ms/step - loss: 0.6950 - accuracy:
Epoch 17/20
25/25 [==============================] - 1s 47ms/step - loss: 0.5899 - accuracy:
Epoch 18/20
25/25 [==============================] - 1s 48ms/step - loss: 0.6075 - accuracy:
Epoch 19/20
25/25 [==============================] - 1s 46ms/step - loss: 0.6027 - accuracy:
Epoch 20/20
25/25 [==============================] - 1s 50ms/step - loss: 0.5008 - accuracy:
```

Plot the results of the CNN model

```
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
    Test loss: 1.5327306985855103
    Test accuracy: 0.3529411852359772
```

RNN method

```
num_classes = 5
epochs = 20
```

Create the RNN model using LSTM

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.LSTM(100, input_shape=(IMG_SIZE,IMG_SIZE)))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 100)               64400

 dense_14 (Dense)            (None, 10)                1010

=================================================================
Total params: 65,410
Trainable params: 65,410
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss= tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              optimizer='adam',
              metrics=['accuracy'])
```

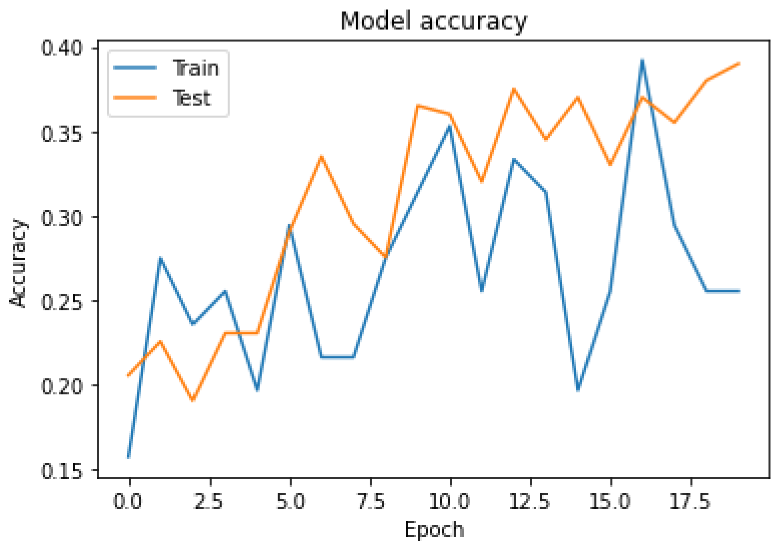
```
history = model.fit(X_train, Y_train,
                    epochs= epochs,
                    validation_data=(X_test, Y_test))
```

```
Epoch 1/20
7/7 [==============================] - 3s 211ms/step - loss: 2.0437 - accuracy: 
Epoch 2/20
7/7 [==============================] - 0s 58ms/step - loss: 1.7540 - accuracy: 0
Epoch 3/20
7/7 [==============================] - 0s 56ms/step - loss: 1.6712 - accuracy: 0
Epoch 4/20
7/7 [==============================] - 0s 57ms/step - loss: 1.6344 - accuracy: 0
Epoch 5/20
7/7 [==============================] - 0s 54ms/step - loss: 1.6200 - accuracy: 0
Epoch 6/20
7/7 [==============================] - 0s 54ms/step - loss: 1.5940 - accuracy: 0
Epoch 7/20
7/7 [==============================] - 0s 59ms/step - loss: 1.5544 - accuracy: 0
Epoch 8/20
7/7 [==============================] - 0s 59ms/step - loss: 1.5378 - accuracy: 0
Epoch 9/20
7/7 [==============================] - 0s 51ms/step - loss: 1.5373 - accuracy: 0
Epoch 10/20
7/7 [==============================] - 0s 55ms/step - loss: 1.5169 - accuracy: 0
Epoch 11/20
7/7 [==============================] - 0s 56ms/step - loss: 1.4920 - accuracy: 0
Epoch 12/20
7/7 [==============================] - 0s 57ms/step - loss: 1.4759 - accuracy: 0
Epoch 13/20
7/7 [==============================] - 0s 55ms/step - loss: 1.4822 - accuracy: 0
Epoch 14/20
7/7 [==============================] - 0s 56ms/step - loss: 1.4593 - accuracy: 0
Epoch 15/20
7/7 [==============================] - 0s 57ms/step - loss: 1.4478 - accuracy: 0
Epoch 16/20
```

```
7/7 [==============================] - 0s 53ms/step - loss: 1.4953 - accuracy: 0
Epoch 17/20
7/7 [==============================] - 0s 55ms/step - loss: 1.4936 - accuracy: 0
Epoch 18/20
7/7 [==============================] - 0s 57ms/step - loss: 1.4613 - accuracy: 0
Epoch 19/20
7/7 [==============================] - 0s 54ms/step - loss: 1.4559 - accuracy: 0
Epoch 20/20
7/7 [==============================] - 0s 55ms/step - loss: 1.4205 - accuracy: 0
```

Plot the RNN model's results

```
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



```
score = model.evaluate(X_test, Y_test, verbose=2)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
2/2 - 0s - loss: 1.5921 - accuracy: 0.2549 - 46ms/epoch - 23ms/step
Test loss: 1.592103362083435
Test accuracy: 0.2549019753932953
```

Pretrained Model

```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
```

Extract the pretrain-crops.zip file and make variables to hold directory paths

```
local_zip = '/tmp/pretrain-crops.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()

base_dir = '/tmp/pretrain-crops'
train_dir = os.path.join(base_dir, 'Train')
validation_dir = os.path.join(base_dir, 'Validation')

# Directory with our training jute pictures
train_jute_dir = os.path.join(train_dir, 'jute')

# Directory with our training maize pictures
train_maize_dir = os.path.join(train_dir, 'maize')

# Directory with our training rice pictures
train_rice_dir = os.path.join(train_dir, 'rice')

# Directory with our training sugarcane pictures
train_sugarcane_dir = os.path.join(train_dir, 'sugarcane')

# Directory with our training wheat pictures
train_wheat_dir = os.path.join(train_dir, 'wheat')

# Directory with our validation jute pictures
validation_jute_dir = os.path.join(validation_dir, 'jute')

# Directory with our validation maize pictures
validation_maize_dir = os.path.join(validation_dir, 'maize')

# Directory with our validation rice pictures
validation_rice_dir = os.path.join(validation_dir, 'rice')

# Directory with our validation sugarcane pictures
validation_sugarcane_dir = os.path.join(validation_dir, 'sugarcane')

# Directory with our validation wheat pictures
validation_wheat_dir = os.path.join(validation_dir, 'wheat')


# Set up matplotlib fig, and size it to fit 4x4 pics
import matplotlib.image as mpimg
```

```
nrows = 4
ncols = 4

fig = plt.gcf()
fig.set_size_inches(ncols*4, nrows*4)
pic_index = 100
train_jute_fnames = os.listdir( train_jute_dir )
train_maize_fnames = os.listdir( train_maize_dir )
train_rice_fnames = os.listdir( train_rice_dir )
train_sugarcane_fnames = os.listdir( train_sugarcane_dir )
train_wheat_fnames = os.listdir( train_wheat_dir )



next_jute_pix = [os.path.join(train_jute_dir, fname)
                for fname in train_jute_fnames[ pic_index-8:pic_index]
               ]

next_maize_pix = [os.path.join(train_maize_dir, fname)
                for fname in train_maize_fnames[ pic_index-8:pic_index]
               ]

next_rice_pix = [os.path.join(train_rice_dir, fname)
                for fname in train_rice_fnames[ pic_index-8:pic_index]
               ]

next_sugarcane_pix = [os.path.join(train_sugarcane_dir, fname)
                for fname in train_sugarcane_fnames[ pic_index-8:pic_index]
               ]

next_wheat_pix = [os.path.join(train_wheat_dir, fname)
                for fname in train_wheat_fnames[ pic_index-8:pic_index]
               ]


for i, img_path in enumerate(next_jute_pix+next_maize_pix+next_rice_pix+next_sugarcane
  # Set up subplot; subplot indices start at 1
  sp = plt.subplot(nrows, ncols, i + 1)
  sp.axis('Off') # Don't show axes (or gridlines)

  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```

```
    <Figure size 1152x1152 with 0 Axes>
```

Create more images using ImageDataGenerator

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator


train_datagen = ImageDataGenerator(rescale = 1./255.,rotation_range = 40, width_shift_

test_datagen = ImageDataGenerator( rescale = 1.0/255. )



train_generator = train_datagen.flow_from_directory(train_dir, batch_size = 20, class_

# Flow validation images in batches of 20 using test_datagen generator
validation_generator = test_datagen.flow_from_directory( validation_dir,  batch_size =
```

```
    Found 200 images belonging to 5 classes.
    Found 50 images belonging to 5 classes.
```

Import the VGG16 pretrained model to use for predicting on the crops images.

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras import layers
base_model = VGG16(input_shape = (224, 224, 3), # Shape of our images
include_top = False, # Leave out the last fully connected layer
weights = 'imagenet')


for layer in base_model.layers:
    layer.trainable = False
```

Us VGG16 to build the model

```
# Flatten the output layer to 1 dimension
x = layers.Flatten()(base_model.output)

# Add a fully connected layer with 512 hidden units and ReLU activation
x = layers.Dense(512, activation='relu')(x)

# Add a dropout rate of 0.5
x = layers.Dropout(0.5)(x)

# Add a final sigmoid layer with 1 node for classification output
x = layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.models.Model(base_model.input, x)

model.compile(optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001), loss = 'bi
```

Fit the model

```
vgghist = model.fit(train_generator,
                    validation_data = validation_generator,
                    steps_per_epoch = 3, epochs = 5)
```

```
Epoch 1/5
3/3 [==============================] - 149s 27s/step - loss: -58.9973 - acc: 0.2
Epoch 2/5
3/3 [==============================] - 59s 24s/step - loss: -205.5149 - acc: 0.2
Epoch 3/5
3/3 [==============================] - 59s 24s/step - loss: -368.1542 - acc: 0.1
Epoch 4/5
3/3 [==============================] - 59s 24s/step - loss: -658.1708 - acc: 0.1
Epoch 5/5
3/3 [==============================] - 60s 24s/step - loss: -709.8444 - acc: 0.1
```

Analysis

Simple model

- Accuracy: The accuracy of the sequential model was around 31% slightly higher than random.

CNN model

- Accuracy: The accuracy of the CNN model was around 35% the best accuracy I could manage messing with all models.

RNN model

- Accuracy: The accuracy for the model was around 26% a little better than random.

Transfer Learning

- Accuracy: The accuracy for the model was random being 20%.

Conclusion

- There was no where near enough images to create an accurate model to predict different crop types. I should have researched more on how to generate usable images by copying existing images in the data set and transforming them slightly.

- I imported the images in greyscale by following an example of how to import images, this probably made the model less accurate because colors would likely be a big facter in differentiating crop types. I should have tried to find a way to import and store them in colored format.
- All of the models I used were really inaccurate even after continuing the tune them by changing batch size, epoach number, optimizer, number of layers, and activation functions for those layers. None of the tuning I did could significantly increase the accuracy due to the small sample size and the similarity of the greyscaled crop images.