Executive Summary

Introduction:

In the agricultural industry, pests can be the matter of a good or bad yield for the farmers. Timely and accurate identification of pests is essential for pest management. The learner tackled the problem of agricultural pest detection using machine learning. The learner was provided 2,479 images of 5 different types of agricultural pests including Ants, Bees, Grasshoppers, Moths and Wasps.

Business Problem

Pest infestation in the agricultural industry can lead to decreased crop yield and economic losses. Accurate pest detection can help businesses to make timely and accurate decisions to deal with the pests. The learners goal is to develop a machine learning model that can automatically identify these agricultural pests from images, aiding farmers in pest management.

Data Set:

We were provided with a dataset containing 2,479 images of five agricultural pests. The dataset was divided into five categories: Ants, Bees, Grasshoppers, Moths, and Wasps. The learner used a 70/30 split ratio for training and testing sets, ensuring a sufficient amount of data for model training and evaluation.

Methodology:

We employed deep learning techniques to address the pest recognition problem. Specifically, we created two deep learning models with different architectures:

Model 1: A Convolutional Neural Network (CNN) model with multiple convolutional and pooling layers followed by fully connected layers. This model aims to capture intricate features from the pest images.

Model 2: Another CNN model with a different architecture involving more convolutional and pooling layers and different node sizes. This model explores a different feature extraction strategy.

Experimental Results:

Model 1: Achieved a test accuracy of approximately 67.47%.

Model 2: Achieved a test accuracy of approximately 73.13%.

Interpretation

Both models demonstrated the ability to identify agricultural pests to a reasonable degree. Model 2 outperformed Model 1, showing that model 2 made more accurate predictions for this task. This system can be integrated into farm management softwares/applications, allowing farmers to capture images of pests in their fields and receive immediate pest identification and recommended actions. This technology can enhance pest management practices, leading to increased crop yields and reduced economic losses for farmers.

Improvements

- Data Augmentation - Rotating/flipping data while training model would help to make more accurate predictions in real-life scenarios.
- Parameter Tuning - Changing batch sizes, learning rate etc can lead to better model performance.
- Ensemble Learning - Ensemble learning for models like stacking or voting can lead to higer accuracy.

```
import numpy as np
import matplotlib.pyplot as plt
import random
from tensorflow import keras
import tensorflow as tf

from sklearn.metrics import classification_report
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils, to_categorical
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D, GlobalAveragePooling2D, Flatten
from keras.layers.normalization.batch_normalization import BatchNormalization
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard, Callback
from keras.optimizers import SGD, RMSprop, Adam, Nadam
from keras.losses import categorical_crossentropy

tf.config.list_physical_devices('GPU')
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
import warnings
warnings.filterwarnings('ignore')


from google.colab import drive
#it will open a webpage for verifying your google account. if it is successful, the Google colab can link the Google drive
drive.mount('/content/drive')

# to show the folders under the dataset
!ls "/content/drive/MyDrive/Colab Notebooks/A2_Datasets/Part2_agricultural_pests"
```

```
Mounted at /content/drive
ants  bees  grasshopper  moth  wasp
```

```
import os

#setting paths
ants_path = '/content/drive/MyDrive/Colab Notebooks/A2_Datasets/Part2_agricultural_pests/ants'
bees_path = '/content/drive/MyDrive/Colab Notebooks/A2_Datasets/Part2_agricultural_pests/bees'
grasshopper_path = '/content/drive/MyDrive/Colab Notebooks/A2_Datasets/Part2_agricultural_pests/grasshopper'
moth_path = '/content/drive/MyDrive/Colab Notebooks/A2_Datasets/Part2_agricultural_pests/moth'
wasp_path = '/content/drive/MyDrive/Colab Notebooks/A2_Datasets/Part2_agricultural_pests/wasp'

# get a list of all files in the folder
ants_file_list = os.listdir(ants_path)
bees_file_list = os.listdir(bees_path)
grasshopper_file_list = os.listdir(grasshopper_path)
moth_file_list = os.listdir(moth_path)
wasp_file_list = os.listdir(wasp_path)

# print the total number of files
print(f'Total number of files under ants folder are: {len(ants_file_list)}')
print(f'Total number of files under bees folder are: {len(bees_file_list)}')
print(f'Total number of files under grasshopper folder are: {len(grasshopper_file_list)}')
print(f'Total number of files under moth folder are: {len(moth_file_list)}')
print(f'Total number of files under wasp folder are: {len(wasp_file_list)}')
```

```
Total number of files under ants folder are: 499
Total number of files under bees folder are: 500
Total number of files under grasshopper folder are: 485
Total number of files under moth folder are: 497
Total number of files under wasp folder are: 498
```

```
import os
import matplotlib.pyplot as plt
import random
from PIL import Image

# Function to display images from a folder
def display_images_from_folder(folder_path, num_images=5):
    file_list = os.listdir(folder_path)
    selected_images = random.sample(file_list, num_images)

    for i, image_filename in enumerate(selected_images):
        image_path = os.path.join(folder_path, image_filename)
        image = Image.open(image_path)

        plt.subplot(1, num_images, i + 1)
        plt.imshow(image)
        plt.axis('off')

# Display random images from each category
plt.figure(figsize=(15, 3))

plt.subplot(151)
plt.title('Ants')
display_images_from_folder(ants_path)

plt.subplot(152)
plt.title('Bees')
display_images_from_folder(bees_path)

plt.subplot(153)
plt.title('Grasshopper')
display_images_from_folder(grasshopper_path)

plt.subplot(154)
plt.title('Moth')
display_images_from_folder(moth_path)

plt.subplot(155)
plt.title('Wasp')
display_images_from_folder(wasp_path)
```

```
display_images_from_folder(wasp_path)

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


```
#CNN Model 1
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

def create_cnn_model(input_shape, num_classes):
    model = Sequential()

    # Convolutional layers
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))

    # Flatten layer to transition from convolutional to fully connected layers
    model.add(Flatten())

    # Fully connected layers
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))

    # Output layer with softmax activation for classification
    model.add(Dense(num_classes, activation='softmax'))

    return model
```

Double-click (or enter) to edit

```
# Continue from Method 1 code

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

# Define a function to load and preprocess images
def load_and_preprocess_images(folder_path, label):
    data = []

    for file in os.listdir(folder_path):
        if file.endswith('.jpeg') or file.endswith('.jpg'):
            img = tf.io.read_file(os.path.join(folder_path, file))
            img = tf.image.decode_jpeg(img)
            img = tf.image.resize(img, (100, 100))
            data.append((img, label))

    return data


# Load and preprocess images from each category
ants_data = load_and_preprocess_images(ants_path, 'Ants')
bees_data = load_and_preprocess_images(bees_path, 'Bees')
grasshopper_data = load_and_preprocess_images(grasshopper_path, 'Grasshopper')
moth_data = load_and_preprocess_images(moth_path, 'Moth')
```

```
wasp_data = load_and_preprocess_images(wasp_path, 'Wasp')

# Combine data from all categories
data = ants_data + bees_data + grasshopper_data + moth_data + wasp_data

# Shuffle the data
random.shuffle(data)

# Split the data into training and testing sets
train_size = int(0.8 * len(data))
train_data = data[:train_size]
test_data = data[train_size:]


# Extract image data and labels from the training and testing sets
x_train, y_train = zip(*train_data)
x_test, y_test = zip(*test_data)

# Convert image data and labels into NumPy arrays
x_train = np.array(x_train)
x_test = np.array(x_test)

# Normalize image data
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# Encode labels
label_map = {'Ants': 0, 'Bees': 1, 'Grasshopper': 2, 'Moth': 3, 'Wasp': 4}
y_train = np.array([label_map[label] for label in y_train])
y_test = np.array([label_map[label] for label in y_test])

# One-hot encode labels
y_train = to_categorical(y_train, num_classes=5)
y_test = to_categorical(y_test, num_classes=5)


# Continue with the CNN model and training from Method 1

# Define the CNN model
input_shape = (100, 100, 3)
num_classes = 5
model = create_cnn_model(input_shape, num_classes)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Print model summary
model.summary()

# Train the model
history = model.fit(x_train, y_train,
                    batch_size=32,
                    epochs=30,
                    validation_data=(x_test, y_test))

# Evaluate the model on the test data
evaluation = model.evaluate(x_test, y_test)

# Print the test loss and accuracy
test_loss, test_accuracy = evaluation
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
62/62 [==============================] - 1s 15ms/step - loss: 0.8912 - accuracy: 0.6836 - val_loss: 0.9656 - val_accuracy
Epoch 14/30
62/62 [==============================] - 1s 14ms/step - loss: 0.8554 - accuracy: 0.6788 - val_loss: 0.9377 - val_accuracy
Epoch 15/30
62/62 [==============================] - 1s 14ms/step - loss: 0.8390 - accuracy: 0.6798 - val_loss: 0.9402 - val_accuracy
Epoch 16/30
62/62 [==============================] - 1s 17ms/step - loss: 0.8181 - accuracy: 0.7035 - val_loss: 1.0041 - val_accuracy
Epoch 17/30
62/62 [==============================] - 1s 17ms/step - loss: 0.8010 - accuracy: 0.6949 - val_loss: 0.9645 - val_accuracy
Epoch 18/30
62/62 [==============================] - 1s 18ms/step - loss: 0.7914 - accuracy: 0.7126 - val_loss: 0.9198 - val_accuracy
Epoch 19/30
62/62 [==============================] - 1s 17ms/step - loss: 0.7443 - accuracy: 0.7237 - val_loss: 0.9221 - val_accuracy
Epoch 20/30
62/62 [==============================] - 1s 14ms/step - loss: 0.7367 - accuracy: 0.7348 - val_loss: 0.9093 - val_accuracy
Epoch 21/30
62/62 [==============================] - 1s 15ms/step - loss: 0.7095 - accuracy: 0.7333 - val_loss: 0.9480 - val_accuracy
Epoch 22/30
62/62 [==============================] - 1s 14ms/step - loss: 0.7297 - accuracy: 0.7379 - val_loss: 0.9941 - val_accuracy
Epoch 23/30
62/62 [==============================] - 1s 14ms/step - loss: 0.6806 - accuracy: 0.7485 - val_loss: 0.8786 - val_accuracy
Epoch 24/30
62/62 [==============================] - 1s 14ms/step - loss: 0.6296 - accuracy: 0.7702 - val_loss: 0.8684 - val_accuracy
Epoch 25/30
62/62 [==============================] - 1s 15ms/step - loss: 0.6065 - accuracy: 0.7869 - val_loss: 0.8751 - val_accuracy
Epoch 26/30
62/62 [==============================] - 1s 15ms/step - loss: 0.6196 - accuracy: 0.7859 - val_loss: 0.8625 - val_accuracy
Epoch 27/30
62/62 [==============================] - 1s 15ms/step - loss: 0.5985 - accuracy: 0.7939 - val_loss: 0.8718 - val_accuracy
Epoch 28/30
62/62 [==============================] - 1s 15ms/step - loss: 0.5456 - accuracy: 0.8081 - val_loss: 0.9878 - val_accuracy
Epoch 29/30
62/62 [==============================] - 1s 15ms/step - loss: 0.5715 - accuracy: 0.7970 - val_loss: 0.8527 - val_accuracy
Epoch 30/30
62/62 [==============================] - 1s 15ms/step - loss: 0.5527 - accuracy: 0.8061 - val_loss: 0.9319 - val_accuracy
16/16 [==============================] - 0s 7ms/step - loss: 0.9319 - accuracy: 0.6747
Test Loss: 0.9319
Test Accuracy: 0.6747
```

```
# Import necessary libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

def create_second_cnn_model(input_shape, num_classes):
    model = Sequential()

    # Convolutional layers
    model.add(Conv2D(64, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(256, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))

    # Flatten layer to transition from convolutional to fully connected layers
    model.add(Flatten())

    # Fully connected layers
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))

    # Output layer with softmax activation for classification
    model.add(Dense(num_classes, activation='softmax'))

    return model



# Define the revised second CNN model (with different architecture)
input_shape_second = (100, 100, 3)
num_classes_second = 5
second_model = create_second_cnn_model(input_shape_second, num_classes_second)



# Compile the second model
second_model.compile(optimizer=Adam(learning_rate=0.0001),
                     loss='categorical_crossentropy',
                     metrics=['accuracy'])



# Train the second model
history_second = second_model.fit(x_train, y_train,
                                  batch_size=32,
```

```
                    epochs=30,
                    validation_data=(x_test, y_test))
```

```
Epoch 1/30
62/62 [==============================] - 4s 30ms/step - loss: 1.5695 - accuracy: 0.2758 - val_loss: 1.4722 - val_accuracy
Epoch 2/30
62/62 [==============================] - 2s 29ms/step - loss: 1.3727 - accuracy: 0.4384 - val_loss: 1.2323 - val_accuracy
Epoch 3/30
62/62 [==============================] - 2s 26ms/step - loss: 1.2402 - accuracy: 0.5061 - val_loss: 1.2025 - val_accuracy
Epoch 4/30
62/62 [==============================] - 2s 26ms/step - loss: 1.1440 - accuracy: 0.5591 - val_loss: 1.1501 - val_accuracy
Epoch 5/30
62/62 [==============================] - 2s 26ms/step - loss: 1.0644 - accuracy: 0.5995 - val_loss: 1.0586 - val_accuracy
Epoch 6/30
62/62 [==============================] - 2s 25ms/step - loss: 0.9908 - accuracy: 0.6217 - val_loss: 0.9963 - val_accuracy
Epoch 7/30
62/62 [==============================] - 2s 27ms/step - loss: 0.9232 - accuracy: 0.6601 - val_loss: 0.9563 - val_accuracy
Epoch 8/30
62/62 [==============================] - 1s 24ms/step - loss: 0.8513 - accuracy: 0.6884 - val_loss: 0.9282 - val_accuracy
Epoch 9/30
62/62 [==============================] - 2s 25ms/step - loss: 0.7951 - accuracy: 0.7010 - val_loss: 0.9280 - val_accuracy
Epoch 10/30
62/62 [==============================] - 2s 25ms/step - loss: 0.7463 - accuracy: 0.7187 - val_loss: 0.8588 - val_accuracy
Epoch 11/30
62/62 [==============================] - 2s 25ms/step - loss: 0.7085 - accuracy: 0.7465 - val_loss: 0.9291 - val_accuracy
Epoch 12/30
62/62 [==============================] - 2s 26ms/step - loss: 0.6630 - accuracy: 0.7667 - val_loss: 0.8333 - val_accuracy
Epoch 13/30
62/62 [==============================] - 2s 27ms/step - loss: 0.6190 - accuracy: 0.7682 - val_loss: 0.8311 - val_accuracy
Epoch 14/30
62/62 [==============================] - 2s 26ms/step - loss: 0.5764 - accuracy: 0.7899 - val_loss: 0.8374 - val_accuracy
Epoch 15/30
62/62 [==============================] - 2s 25ms/step - loss: 0.5521 - accuracy: 0.8121 - val_loss: 0.8216 - val_accuracy
Epoch 16/30
62/62 [==============================] - 2s 24ms/step - loss: 0.5002 - accuracy: 0.8232 - val_loss: 0.7526 - val_accuracy
Epoch 17/30
62/62 [==============================] - 2s 24ms/step - loss: 0.4993 - accuracy: 0.8227 - val_loss: 0.7747 - val_accuracy
Epoch 18/30
62/62 [==============================] - 2s 24ms/step - loss: 0.4546 - accuracy: 0.8414 - val_loss: 0.7851 - val_accuracy
Epoch 19/30
62/62 [==============================] - 2s 24ms/step - loss: 0.4086 - accuracy: 0.8621 - val_loss: 0.8310 - val_accuracy
Epoch 20/30
62/62 [==============================] - 2s 24ms/step - loss: 0.4088 - accuracy: 0.8571 - val_loss: 0.7944 - val_accuracy
Epoch 21/30
62/62 [==============================] - 2s 26ms/step - loss: 0.3633 - accuracy: 0.8768 - val_loss: 0.7702 - val_accuracy
Epoch 22/30
62/62 [==============================] - 2s 26ms/step - loss: 0.3337 - accuracy: 0.8813 - val_loss: 0.7737 - val_accuracy
Epoch 23/30
62/62 [==============================] - 2s 26ms/step - loss: 0.2958 - accuracy: 0.9000 - val_loss: 0.7988 - val_accuracy
Epoch 24/30
62/62 [==============================] - 2s 25ms/step - loss: 0.2866 - accuracy: 0.9056 - val_loss: 0.8094 - val_accuracy
Epoch 25/30
62/62 [==============================] - 2s 25ms/step - loss: 0.2808 - accuracy: 0.9086 - val_loss: 0.8463 - val_accuracy
Epoch 26/30
62/62 [==============================] - 2s 24ms/step - loss: 0.2682 - accuracy: 0.9091 - val_loss: 0.8065 - val_accuracy
Epoch 27/30
62/62 [==============================] - 2s 25ms/step - loss: 0.2251 - accuracy: 0.9278 - val_loss: 0.8334 - val_accuracy
Epoch 28/30
62/62 [==============================] - 2s 25ms/step - loss: 0.2152 - accuracy: 0.9318 - val_loss: 0.8165 - val_accuracy
Epoch 29/30
62/62 [==============================] - 2s 25ms/step - loss: 0.2034 - accuracy: 0.9374 - val_loss: 0.8662 - val_accuracy
```

```
# Evaluate the second model on the test data
evaluation_second = second_model.evaluate(x_test, y_test)

# Print the test loss and accuracy for the second model
test_loss_second, test_accuracy_second = evaluation_second
print(f'Test Loss (Second Model): {test_loss_second:.4f}')
print(f'Test Accuracy (Second Model): {test_accuracy_second:.4f}')
```

```
16/16 [==============================] - 0s 10ms/step - loss: 0.9039 - accuracy: 0.7313
Test Loss (Second Model): 0.9039
Test Accuracy (Second Model): 0.7313
```

Experiment Report

In this business scenario the learner aimed to develop machine learning models for the recognition of agricultural pests using a dataset containing 2,479 real images of five different types of pests: Ants, Bees, Grasshoppers, Moths, and Wasps. The learner conducted two primary experiments, each involving a different deep learning architecture, and evaluated their performance.

Experiment 1: Model 1

Architecture: Convolutional Neural Network (CNN) with multiple convolutional and pooling layers followed by fully connected layers.

Test Accuracy: 67.47%.

Experiment 2: Model 2

Architecture: CNN with a different structure involving more convolutional and pooling layers and different node sizes.

Test Accuracy: 73.13%.

Interpretation

- Both models showed reasonable capability to identify pests
- Model 2 outperformed model one, showed higher accuracy while identifying pests.

Business implication

The model created by the learner can serve as the foundation for a pest detection that assists farmers in identifying and taking appropriate action against pests, leading to higher yield of crops and lower economical losses. The model can be improved over time, becoming more and more accurate.