Part A: Implementing CNN from Scratch

1. Data Understanding, Analysis, Visualization and Cleaning **bold text**

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras import optimizers
from sklearn.metrics import confusion_matrix, classification_report
import time
from tensorflow.keras.applications import VGG16, ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D
```

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
# Set paths
train_path = '/content/gdrive/MyDrive/assignment_for_ML/GroceryStoreDataset/dataset/train'
validation_path = '/content/gdrive/MyDrive/assignment_for_ML/GroceryStoreDataset/dataset/val'
test_path = '/content/gdrive/MyDrive/assignment_for_ML/GroceryStoreDataset/dataset/test'
```

```
# Data Analysis
def analyze_dataset(path):
    import os
    classes = os.listdir(path)
    print(f"Classes: {classes}")

    # Count images in each class (including subdirectories)
    for cls in classes:
        cls_path = os.path.join(path, cls)
        image_count = 0
        # Walk through all subdirectories
        for root, dirs, files in os.walk(cls_path):
            image_count += len([f for f in files if f.lower().endswith(('.png', '.jpg', '.jpeg'))])
        print(f"{cls}: {image_count} images")

    # Plot sample images
    plt.figure(figsize=(15,5))
    for i, cls in enumerate(classes):
        cls_path = os.path.join(path, cls)
        # Find first image in class directory or subdirectories
        img_path = None
        for root, dirs, files in os.walk(cls_path):
            for file in files:
                if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                    img_path = os.path.join(root, file)
                    break
            if img_path is not None:
                break

        if img_path is not None:
            img = plt.imread(img_path)
            plt.subplot(1,3,i+1)
            plt.imshow(img)
            plt.title(cls)
    plt.show()

print("Training set analysis:")
analyze_dataset(train_path)
print("\nValidation set analysis:")
analyze_dataset(validation_path)
```

```
    print("\nTest set analysis:")
    analyze_dataset(test_path)
```

```
Training set analysis:
Classes: ['Vegetables', 'Packages', 'Fruit']
Vegetables: 634 images
Packages: 864 images
Fruit: 1142 images
```


```
Validation set analysis:
Classes: ['Vegetables', 'Packages', 'Fruit']
Vegetables: 85 images
Packages: 100 images
Fruit: 111 images
```


```
Test set analysis:
Classes: ['Vegetables', 'Packages', 'Fruit']
Vegetables: 587 images
Packages: 781 images
Fruit: 1117 images
```

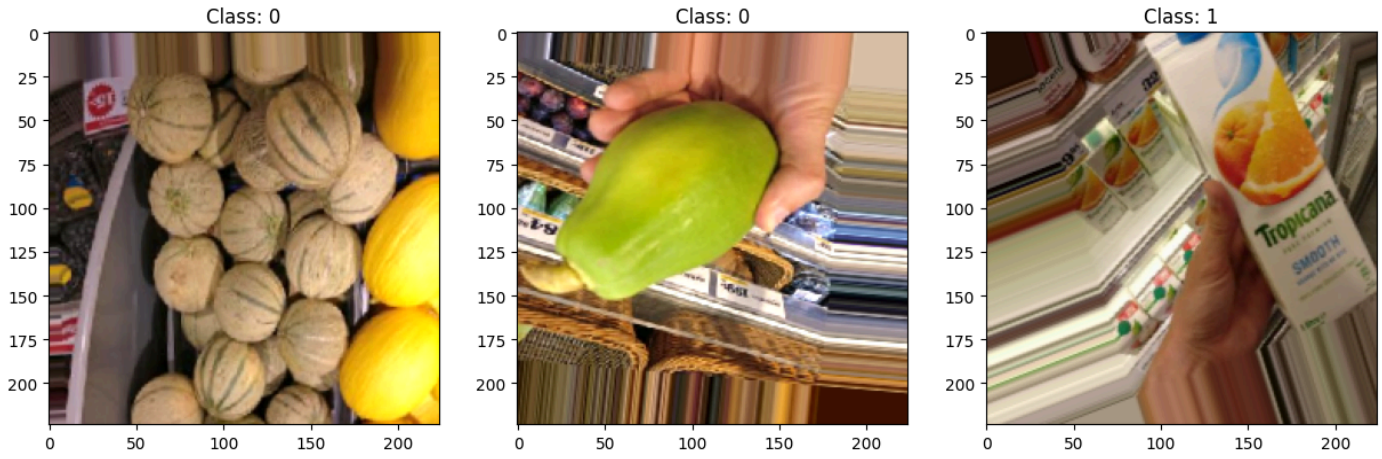
2. Data Augmentation to Increase Dataset Size

```
# Enhanced Data Augmentation Configuration
train_datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
    vertical_flip=True,
    brightness_range=[0.7, 1.3],
    fill_mode='nearest',
    rescale=1./255)

test_datagen = ImageDataGenerator(rescale=1./255)


# Create generators
batch_size = 32
target_size = (224, 224)

train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=target_size,
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_path,
    target_size=target_size,
    batch_size=batch_size,
    class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=target_size,
    batch_size=1,
    class_mode='categorical',
    shuffle=False)
```

```
Found 2640 images belonging to 3 classes.
Found 296 images belonging to 3 classes.
Found 2485 images belonging to 3 classes.
```

```
# Calculate steps_per_epoch to effectively increase dataset size
num_classes = len(train_generator.class_indices)
desired_images_per_class = 1200  # Target 1200 images per class
steps_per_epoch = (desired_images_per_class * num_classes) // batch_size


# Verify augmented images
def plot_augmented_images(generator):
    x, y = next(generator)
    plt.figure(figsize=(15,5))
    for i in range(3):
        plt.subplot(1,3,i+1)
        plt.imshow(x[i])
        plt.title(f"Class: {np.argmax(y[i])}")
    plt.show()

print("Sample augmented images:")
plot_augmented_images(train_generator)
```

```
Sample augmented images:
```



3. Baseline CNN Model

```
# Build baseline model
def build_baseline_model(input_shape=(224, 224, 3), num_classes=3):
    model = Sequential([
        Conv2D(32, (3,3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2,2)),

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

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

        Flatten(),
        Dense(512, activation='relu'),
        Dense(256, activation='relu'),
        Dense(128, activation='relu'),
        Dense(num_classes, activation='softmax')
    ])
    return model

baseline_model_cnn = build_baseline_model()
baseline_model_cnn.summary()
```

```
/usr/local/lib/python3.11/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)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
flatten (Flatten)	(None, 86528)	0
dense (Dense)	(None, 512)	44,302,848
dense_1 (Dense)	(None, 256)	131,328
dense_2 (Dense)	(None, 128)	32,896
dense_3 (Dense)	(None, 3)	387

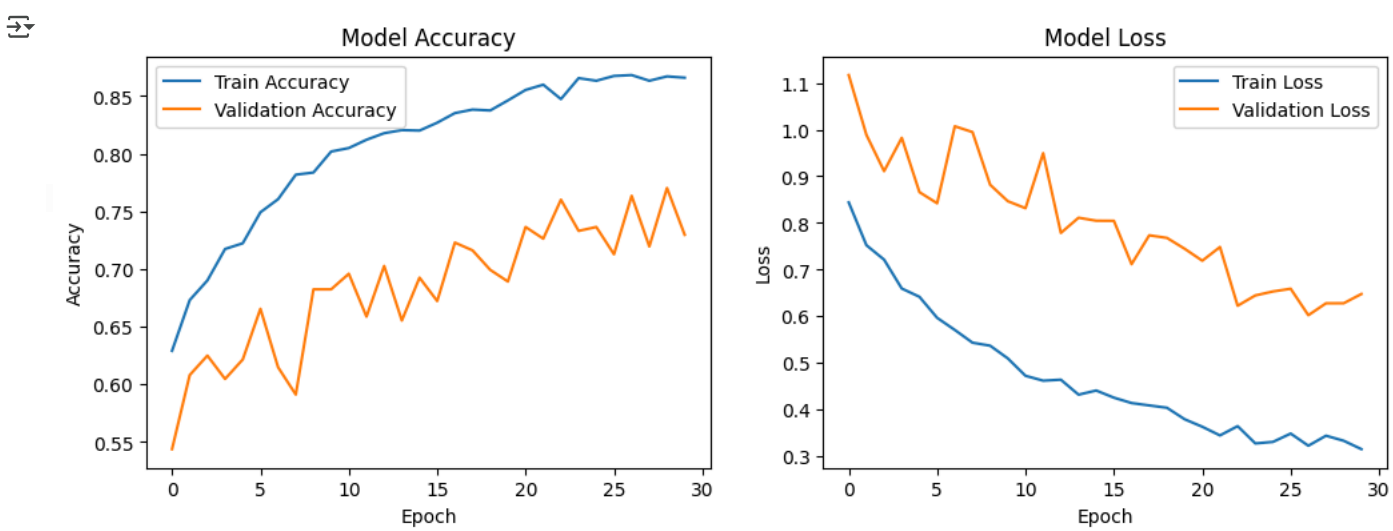
Total params: 44,560,707 (169.99 MB)
Trainable params: 44,560,707 (169.99 MB)
Non-trainable params: 0 (0.00 B)

```
# Compile and train baseline model with adjusted learning rate
baseline_model_cnn.compile(optimizer=optimizers.Adam(learning_rate=0.0001),
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

start_time = time.time()
history_baseline = baseline_model_cnn.fit(
    train_generator,
    steps_per_epoch=steps_per_epoch,
    epochs=30,
    validation_data=validation_generator,
    validation_steps=len(validation_generator))
training_time_baseline = time.time() - start_time
```

```
Epoch 1/30
 83/112 ━━━━━━━━━━━━━━━━━━━━ 16s 563ms/step - accuracy: 0.5991 - loss: 0.9158/usr/local/lib/python3.11/dist-packages/keras/src/traine
  self._interrupted_warning()
112/112 ━━━━━━━━━━━━━━━━━━━━ 53s 436ms/step - accuracy: 0.6069 - loss: 0.8972 - val_accuracy: 0.5439 - val_loss: 1.1167
Epoch 2/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 48s 424ms/step - accuracy: 0.6466 - loss: 0.8018 - val_accuracy: 0.6081 - val_loss: 0.9887
Epoch 3/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 83s 435ms/step - accuracy: 0.6930 - loss: 0.7203 - val_accuracy: 0.6250 - val_loss: 0.9111
Epoch 4/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 81s 429ms/step - accuracy: 0.7166 - loss: 0.6592 - val_accuracy: 0.6047 - val_loss: 0.9824
Epoch 5/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 81s 419ms/step - accuracy: 0.7301 - loss: 0.6283 - val_accuracy: 0.6216 - val_loss: 0.8661
Epoch 6/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 48s 428ms/step - accuracy: 0.7545 - loss: 0.5876 - val_accuracy: 0.6655 - val_loss: 0.8423
Epoch 7/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 47s 417ms/step - accuracy: 0.7627 - loss: 0.5718 - val_accuracy: 0.6149 - val_loss: 1.0072
Epoch 8/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 84s 433ms/step - accuracy: 0.7811 - loss: 0.5461 - val_accuracy: 0.5912 - val_loss: 0.9947
Epoch 9/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 48s 423ms/step - accuracy: 0.7681 - loss: 0.5614 - val_accuracy: 0.6824 - val_loss: 0.8819
Epoch 10/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 84s 438ms/step - accuracy: 0.8005 - loss: 0.5055 - val_accuracy: 0.6824 - val_loss: 0.8467
Epoch 11/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 81s 432ms/step - accuracy: 0.8041 - loss: 0.4741 - val_accuracy: 0.6959 - val_loss: 0.8317
Epoch 12/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 83s 439ms/step - accuracy: 0.8112 - loss: 0.4580 - val_accuracy: 0.6588 - val_loss: 0.9495
Epoch 13/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 80s 422ms/step - accuracy: 0.8260 - loss: 0.4650 - val_accuracy: 0.7027 - val_loss: 0.7788
Epoch 14/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 49s 432ms/step - accuracy: 0.8178 - loss: 0.4410 - val_accuracy: 0.6554 - val_loss: 0.8113
Epoch 15/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 48s 424ms/step - accuracy: 0.8178 - loss: 0.4547 - val_accuracy: 0.6926 - val_loss: 0.8049
Epoch 16/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 83s 433ms/step - accuracy: 0.8206 - loss: 0.4258 - val_accuracy: 0.6723 - val_loss: 0.8047
Epoch 17/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 82s 432ms/step - accuracy: 0.8401 - loss: 0.4013 - val_accuracy: 0.7230 - val_loss: 0.7119
```

```
Epoch 18/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 82s 437ms/step - accuracy: 0.8445 - loss: 0.3905 - val_accuracy: 0.7162 - val_loss: 0.7737
Epoch 19/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 82s 434ms/step - accuracy: 0.8270 - loss: 0.4216 - val_accuracy: 0.6993 - val_loss: 0.7681
Epoch 20/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 83s 447ms/step - accuracy: 0.8388 - loss: 0.3862 - val_accuracy: 0.6892 - val_loss: 0.7449
Epoch 21/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 49s 433ms/step - accuracy: 0.8662 - loss: 0.3504 - val_accuracy: 0.7365 - val_loss: 0.7192
Epoch 22/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 82s 435ms/step - accuracy: 0.8623 - loss: 0.3494 - val_accuracy: 0.7264 - val_loss: 0.7486
Epoch 23/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 48s 423ms/step - accuracy: 0.8572 - loss: 0.3564 - val_accuracy: 0.7601 - val_loss: 0.6228
Epoch 24/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 83s 435ms/step - accuracy: 0.8707 - loss: 0.3241 - val_accuracy: 0.7331 - val_loss: 0.6449
Epoch 25/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 81s 426ms/step - accuracy: 0.8667 - loss: 0.3313 - val_accuracy: 0.7365 - val_loss: 0.6534
Epoch 26/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 49s 437ms/step - accuracy: 0.8622 - loss: 0.3590 - val_accuracy: 0.7128 - val_loss: 0.6593
Epoch 27/30
112/112 ━━━━━━━━━━━━━━━━━━━━ 49s 441ms/step - accuracy: 0.8669 - loss: 0.3233 - val_accuracy: 0.7635 - val_loss: 0.6026
```

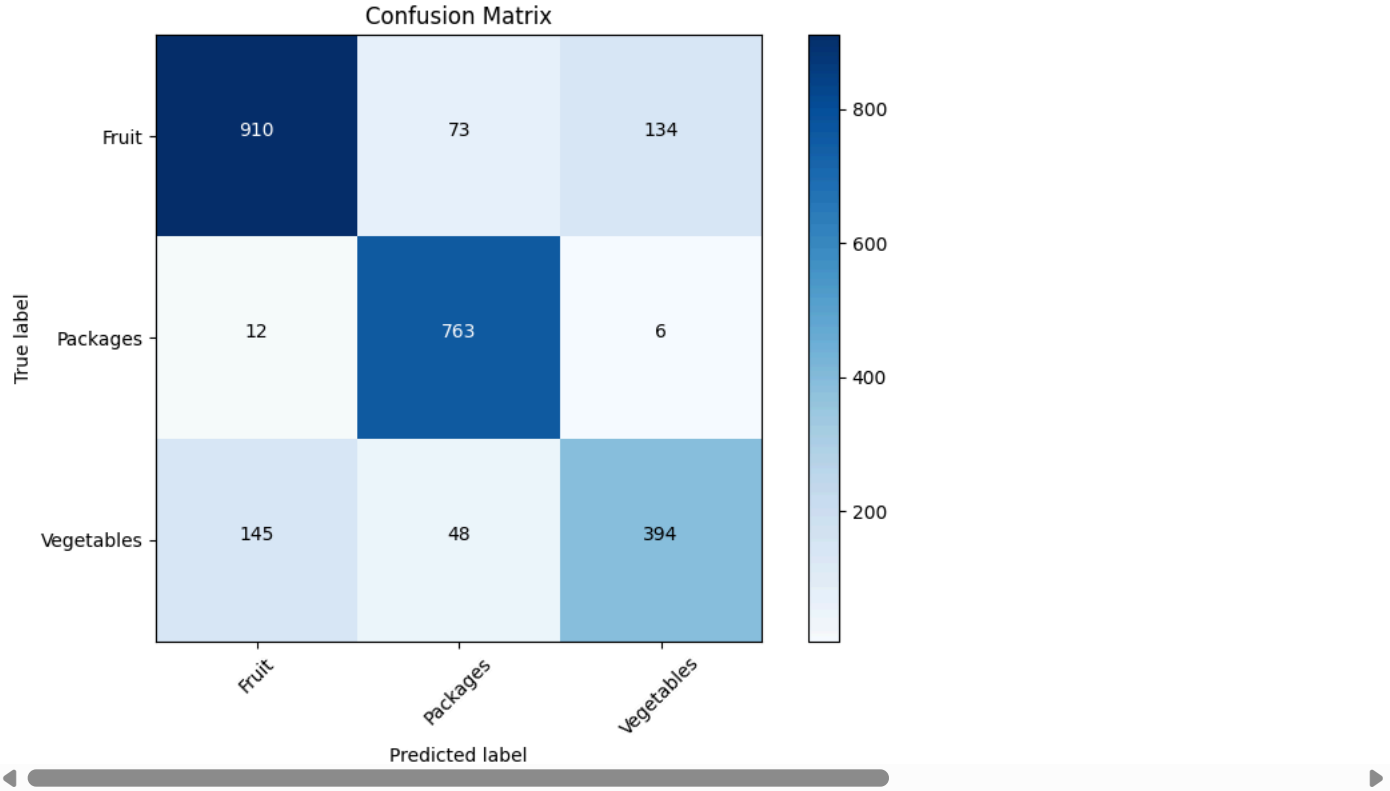
```
# Plot training history
def plot_history(history):
    plt.figure(figsize=(12,4))

    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')
    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')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()

    plt.show()

plot_history(history_baseline)
```




```
# Evaluate baseline model
def evaluate_model(model, generator):
    y_true = generator.classes
    y_pred = model.predict(generator, steps=len(generator)).argmax(axis=1)

    print("Classification Report:")
    print(classification_report(y_true, y_pred, target_names=generator.class_indices.keys()))

    cm = confusion_matrix(y_true, y_pred)
```

```
    plt.figure(figsize=(8,6))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()
    tick_marks = np.arange(len(generator.class_indices))
    plt.xticks(tick_marks, generator.class_indices.keys(), rotation=45)
    plt.yticks(tick_marks, generator.class_indices.keys())

    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()


print("Baseline Model Evaluation:")
evaluate_model(baseline_model_cnn, test_generator)
```

```
Baseline Model Evaluation:
2485/2485 ━━━━━━━━━━━━━━━━━━━━ 13s 5ms/step
Classification Report:
              precision    recall  f1-score   support

       Fruit       0.85      0.81      0.83      1117
    Packages       0.86      0.98      0.92       781
  Vegetables       0.74      0.67      0.70       587

    accuracy                           0.83      2485
   macro avg       0.82      0.82      0.82      2485
weighted avg       0.83      0.83      0.83      2485
```



4. Deeper CNN Model with Regularization

```
# Improved Deeper CNN Model with Regularization
def build_deeper_model(input_shape=(224, 224, 3), num_classes=3):
    model = Sequential([
        # Block 1
```

```
            Conv2D(32, (3,3), activation='relu', padding='same', input_shape=input_shape),
            BatchNormalization(),
            Conv2D(32, (3,3), activation='relu', padding='same'),
            BatchNormalization(),
            MaxPooling2D((2,2)),
            Dropout(0.2),

            # Block 2
            Conv2D(64, (3,3), activation='relu', padding='same'),
            BatchNormalization(),
            Conv2D(64, (3,3), activation='relu', padding='same'),
            BatchNormalization(),
            MaxPooling2D((2,2)),
            Dropout(0.3),

            # Block 3
            Conv2D(128, (3,3), activation='relu', padding='same'),
            BatchNormalization(),
            Conv2D(128, (3,3), activation='relu', padding='same'),
            BatchNormalization(),
            Conv2D(128, (3,3), activation='relu', padding='same'),
            BatchNormalization(),
            MaxPooling2D((2,2)),
            Dropout(0.4),

            # Block 4
            Conv2D(256, (3,3), activation='relu', padding='same'),
            BatchNormalization(),
            Conv2D(256, (3,3), activation='relu', padding='same'),
            BatchNormalization(),
            Conv2D(256, (3,3), activation='relu', padding='same'),
            BatchNormalization(),
            MaxPooling2D((2,2)),
            Dropout(0.5),

            # Classifier
            Flatten(),
            Dense(1024, activation='relu', kernel_regularizer='l2'),
            BatchNormalization(),
            Dropout(0.6),
            Dense(512, activation='relu', kernel_regularizer='l2'),
            BatchNormalization(),
            Dropout(0.5),
            Dense(num_classes, activation='softmax')
        ])
        return model

    deeper_model = build_deeper_model()
    deeper_model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 224, 224, 32)	896
batch_normalization (BatchNormalization)	(None, 224, 224, 32)	128
conv2d_4 (Conv2D)	(None, 224, 224, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 224, 224, 32)	128
max_pooling2d_3 (MaxPooling2D)	(None, 112, 112, 32)	0
dropout (Dropout)	(None, 112, 112, 32)	0
conv2d_5 (Conv2D)	(None, 112, 112, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 112, 112, 64)	256
conv2d_6 (Conv2D)	(None, 112, 112, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 112, 112, 64)	256
max_pooling2d_4 (MaxPooling2D)	(None, 56, 56, 64)	0
dropout_1 (Dropout)	(None, 56, 56, 64)	0
conv2d_7 (Conv2D)	(None, 56, 56, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 56, 56, 128)	512
conv2d_8 (Conv2D)	(None, 56, 56, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 56, 56, 128)	512
conv2d_9 (Conv2D)	(None, 56, 56, 128)	147,584
batch_normalization_6 (BatchNormalization)	(None, 56, 56, 128)	512
max_pooling2d_5 (MaxPooling2D)	(None, 28, 28, 128)	0
dropout_2 (Dropout)	(None, 28, 28, 128)	0
conv2d_10 (Conv2D)	(None, 28, 28, 256)	295,168
batch_normalization_7 (BatchNormalization)	(None, 28, 28, 256)	1,024
conv2d_11 (Conv2D)	(None, 28, 28, 256)	590,080
batch_normalization_8	(None, 28, 28, 256)	1,024

```
# Compile and train deeper model with learning rate scheduling
initial_learning_rate = 0.0005
lr_schedule = optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=steps_per_epoch*5,
    decay_rate=0.9,
    staircase=True)

deeper_model.compile(optimizer=optimizers.Adam(learning_rate=lr_schedule),
                     loss='categorical_crossentropy',
                     metrics=['accuracy'])
```

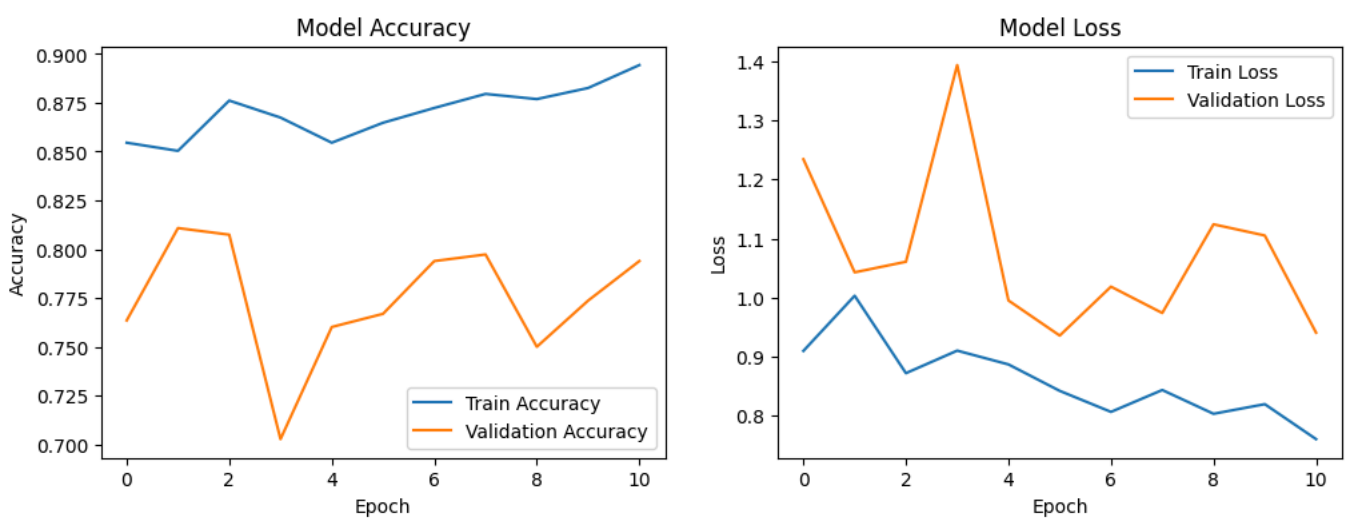
dense_4 (Dense)	(None, 1024)	51,381,248

```
# Add early stopping
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

start_time = time.time()
history_deeper = deeper_model.fit(
    train_generator,
    steps_per_epoch=steps_per_epoch,
    epochs=12,  # Increased epochs
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
```

```
    callbacks=[early_stopping])
training_time_deeper = time.time() - start_time
```

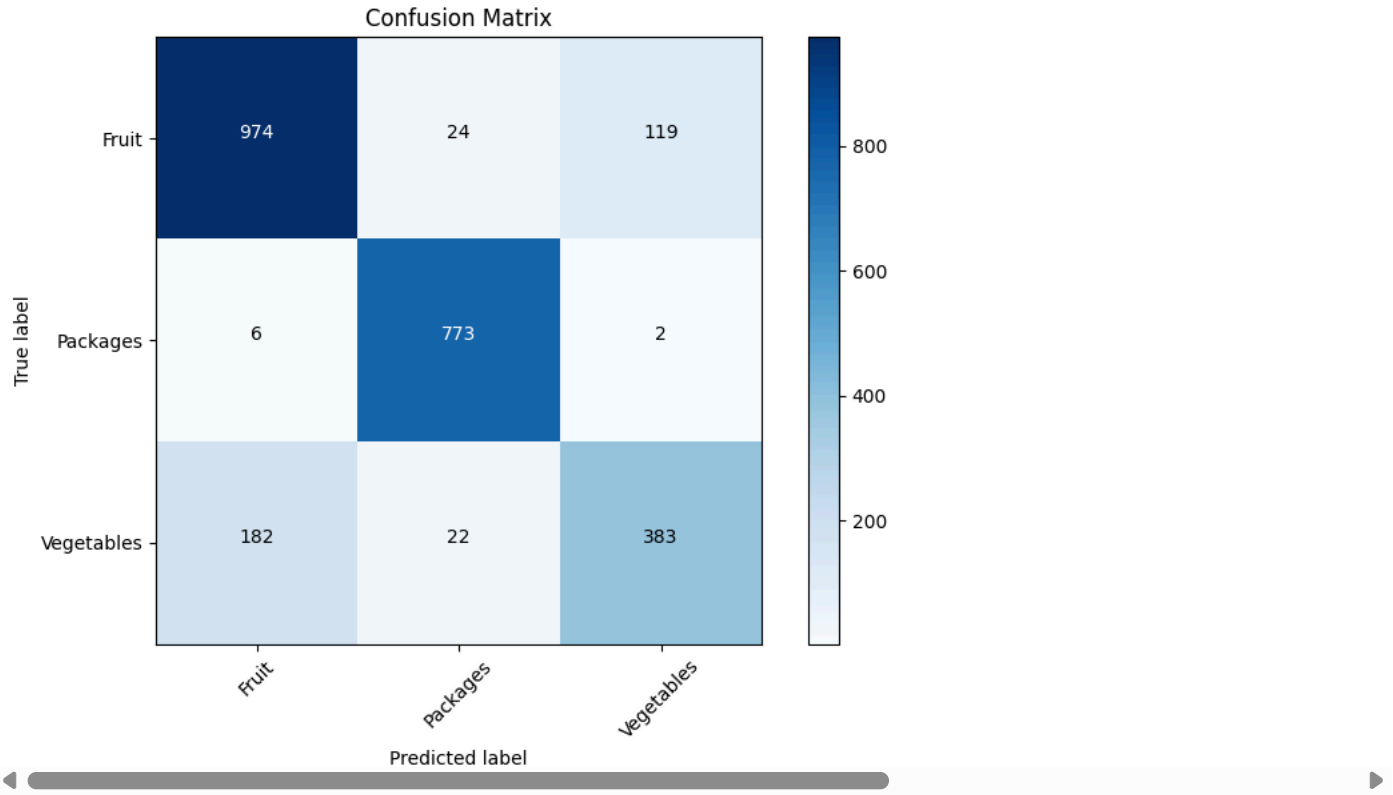
```
 Non-trainable params: 5,760 (22.50 KB)
Epoch 1/12
112/112 ━━━━━━━━━━━━━━━━━━━━ 54s 479ms/step - accuracy: 0.8610 - loss: 0.8911 - val_accuracy: 0.7635 - val_loss: 1.2340
Epoch 2/12
112/112 ━━━━━━━━━━━━━━━━━━━━ 53s 474ms/step - accuracy: 0.8598 - loss: 0.9938 - val_accuracy: 0.8108 - val_loss: 1.0427
Epoch 3/12
112/112 ━━━━━━━━━━━━━━━━━━━━ 52s 464ms/step - accuracy: 0.8754 - loss: 0.8543 - val_accuracy: 0.8074 - val_loss: 1.0606
Epoch 4/12
112/112 ━━━━━━━━━━━━━━━━━━━━ 53s 472ms/step - accuracy: 0.8776 - loss: 0.8779 - val_accuracy: 0.7027 - val_loss: 1.3934
Epoch 5/12
112/112 ━━━━━━━━━━━━━━━━━━━━ 54s 474ms/step - accuracy: 0.8513 - loss: 0.9426 - val_accuracy: 0.7601 - val_loss: 0.9951
Epoch 6/12
112/112 ━━━━━━━━━━━━━━━━━━━━ 54s 482ms/step - accuracy: 0.8677 - loss: 0.8493 - val_accuracy: 0.7669 - val_loss: 0.9357
Epoch 7/12
112/112 ━━━━━━━━━━━━━━━━━━━━ 80s 470ms/step - accuracy: 0.8734 - loss: 0.7696 - val_accuracy: 0.7939 - val_loss: 1.0185
Epoch 8/12
112/112 ━━━━━━━━━━━━━━━━━━━━ 82s 469ms/step - accuracy: 0.8849 - loss: 0.8350 - val_accuracy: 0.7973 - val_loss: 0.9739
Epoch 9/12
112/112 ━━━━━━━━━━━━━━━━━━━━ 84s 487ms/step - accuracy: 0.8797 - loss: 0.7963 - val_accuracy: 0.7500 - val_loss: 1.1240
Epoch 10/12
112/112 ━━━━━━━━━━━━━━━━━━━━ 81s 483ms/step - accuracy: 0.8882 - loss: 0.8105 - val_accuracy: 0.7736 - val_loss: 1.1050
Epoch 11/12
112/112 ━━━━━━━━━━━━━━━━━━━━ 52s 466ms/step - accuracy: 0.8976 - loss: 0.7373 - val_accuracy: 0.7939 - val_loss: 0.9407
```

```
# Plot training history
plot_history(history_deeper)
```




```
# Evaluate deeper model
print("Deeper Model Evaluation:")
evaluate_model(deeper_model, test_generator)
```

```
Deeper Model Evaluation:
2485/2485 ━━━━━━━━━━━━━━━━━━━━ 16s 6ms/step
Classification Report:
              precision    recall  f1-score   support

       Fruit       0.84      0.87      0.85      1117
    Packages       0.94      0.99      0.97       781
  Vegetables       0.76      0.65      0.70       587

    accuracy                           0.86      2485
   macro avg       0.85      0.84      0.84      2485
weighted avg       0.85      0.86      0.85      2485
```



1. Save Models After Training Add these lines after training each model:

```
# For baseline model
baseline_model_cnn.save('/content/gdrive/MyDrive/assignment_for_ML/models/baseline_model.h5')

# For deeper model
deeper_model.save('/content/gdrive/MyDrive/assignment_for_ML/models/deeper_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

2. Save Training History (Optional) To save the training history for plotting later:

```
import pickle

# For baseline model
with open('/content/gdrive/MyDrive/assignment_for_ML/models/baseline_history.pkl', 'wb') as f:
    pickle.dump(history_baseline.history, f)

# For deeper model
with open('/content/gdrive/MyDrive/assignment_for_ML/models/deeper_history.pkl', 'wb') as f:
    pickle.dump(history_deeper.history, f)


from tensorflow.keras.models import load_model
import pickle
from tensorflow.keras import Model

# Initialize history variables (important!)
```

```
history_baseline = Model()
history_deeper = Model()

try:
    # Load baseline model
    baseline_model_cnn = load_model('/content/gdrive/MyDrive/assignment_for_ML/models/baseline_model.h5')

    # Load deeper model
    deeper_model = load_model('/content/gdrive/MyDrive/assignment_for_ML/models/deeper_model.h5')

    # Load histories
    with open('/content/gdrive/MyDrive/assignment_for_ML/models/baseline_history.pkl', 'rb') as f:
        history_baseline.history = pickle.load(f)

    with open('/content/gdrive/MyDrive/assignment_for_ML/models/deeper_history.pkl', 'rb') as f:
        history_deeper.history = pickle.load(f)

    print("Models loaded successfully!")

    # Recompile models to remove the warning (optional)
    baseline_model_cnn.compile(optimizer='adam',
                               loss='categorical_crossentropy',
                               metrics=['accuracy'])
    deeper_model.compile(optimizer='adam',
                         loss='categorical_crossentropy',
                         metrics=['accuracy'])

except Exception as e:  # Catch specific exceptions
    print(f"Error loading models: {str(e)}")
    print("Training from scratch...")
    # Your existing model building code here
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you t
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you t
Models loaded successfully!
```

```
!ls '/content/gdrive/MyDrive/assignment_for_ML/models'
```

```
baseline_history.pkl  baseline_model.h5  deeper_history.pkl  deeper_model.h5
```

5. Experimentation and Comparative Analysis

```
# Compare accuracy
baseline_val_acc = max(history_baseline.history['val_accuracy'])
deeper_val_acc = max(history_deeper.history['val_accuracy'])
print(f"\nBaseline Model Best Validation Accuracy: {baseline_val_acc:.4f}")
print(f"Deeper Model Best Validation Accuracy: {deeper_val_acc:.4f}")
print(f"Improvement: {(deeper_val_acc - baseline_val_acc)*100:.2f}%")
```

```
Baseline Model Best Validation Accuracy: 0.7703
Deeper Model Best Validation Accuracy: 0.8108
Improvement: 4.05%
```

```
# Compare loss
baseline_val_loss = min(history_baseline.history['val_loss'])
deeper_val_loss = min(history_deeper.history['val_loss'])
print(f"\nBaseline Model Best Validation Loss: {baseline_val_loss:.4f}")
print(f"Deeper Model Best Validation Loss: {deeper_val_loss:.4f}")
print(f"Improvement: {(baseline_val_loss - deeper_val_loss)*100:.2f}% reduction")
```

```
Baseline Model Best Validation Loss: 0.6026
Deeper Model Best Validation Loss: 0.9357
Improvement: -33.31% reduction
```

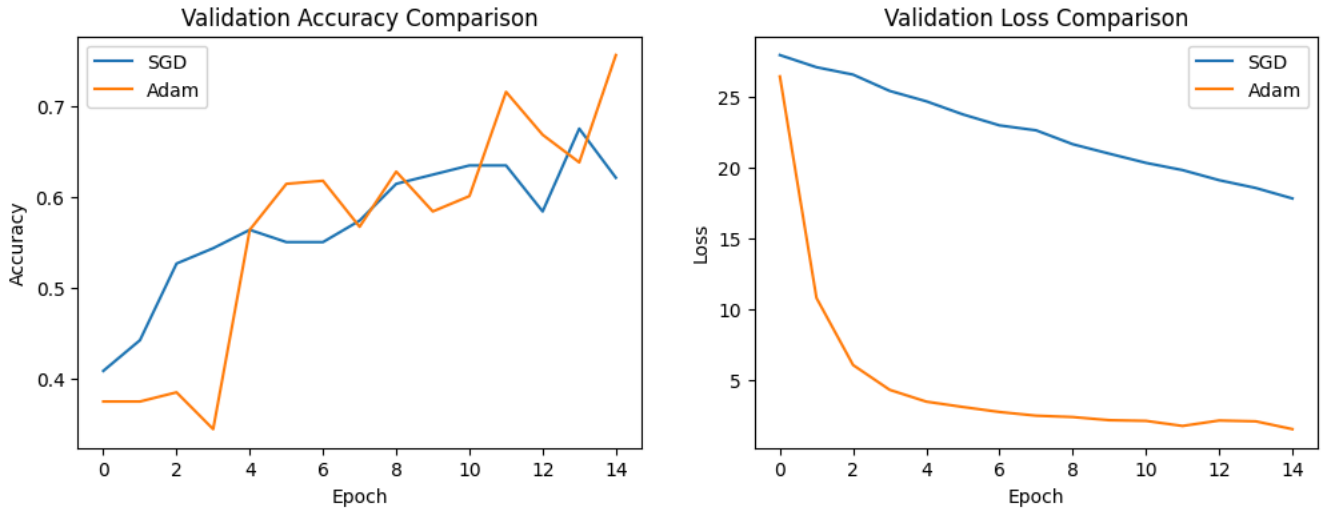
```
# Optimizer comparison (SGD vs Adam)
def train_with_optimizer(optimizer):
    model = build_deeper_model()
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

```
    history = model.fit(
        train_generator,
        steps_per_epoch=len(train_generator),
        epochs=15,  # Reduced for comparison
        validation_data=validation_generator,
        validation_steps=len(validation_generator))

    return history

print("\nTraining with SGD optimizer...")
sgd = optimizers.SGD(learning_rate=0.01)
history_sgd = train_with_optimizer(sgd)

print("\nTraining with Adam optimizer...")
adam = optimizers.Adam(learning_rate=0.001)
history_adam = train_with_optimizer(adam)
```

```
Training with SGD optimizer...
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cla
  self._warn_if_super_not_called()
Epoch 1/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 1978s 24s/step - accuracy: 0.4201 - loss: 28.6168 - val_accuracy: 0.4088 - val_loss: 27.9632
Epoch 2/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 76s 590ms/step - accuracy: 0.5836 - loss: 27.5269 - val_accuracy: 0.4426 - val_loss: 27.1066
Epoch 3/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 48s 577ms/step - accuracy: 0.6193 - loss: 26.5936 - val_accuracy: 0.5270 - val_loss: 26.5830
Epoch 4/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 48s 574ms/step - accuracy: 0.6480 - loss: 25.7218 - val_accuracy: 0.5439 - val_loss: 25.4303
Epoch 5/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 49s 588ms/step - accuracy: 0.6661 - loss: 24.8240 - val_accuracy: 0.5642 - val_loss: 24.6995
Epoch 6/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 81s 573ms/step - accuracy: 0.6544 - loss: 24.0304 - val_accuracy: 0.5507 - val_loss: 23.7738
Epoch 7/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 49s 588ms/step - accuracy: 0.6757 - loss: 23.2494 - val_accuracy: 0.5507 - val_loss: 22.9987
Epoch 8/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 81s 578ms/step - accuracy: 0.7061 - loss: 22.4893 - val_accuracy: 0.5743 - val_loss: 22.6490
Epoch 9/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 49s 591ms/step - accuracy: 0.7107 - loss: 21.7744 - val_accuracy: 0.6149 - val_loss: 21.6697
Epoch 10/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 81s 579ms/step - accuracy: 0.7320 - loss: 21.0604 - val_accuracy: 0.6250 - val_loss: 21.0006
Epoch 11/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 82s 587ms/step - accuracy: 0.7570 - loss: 20.3500 - val_accuracy: 0.6351 - val_loss: 20.3511
Epoch 12/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 82s 592ms/step - accuracy: 0.7487 - loss: 19.7085 - val_accuracy: 0.6351 - val_loss: 19.8427
Epoch 13/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 48s 581ms/step - accuracy: 0.7525 - loss: 19.0722 - val_accuracy: 0.5845 - val_loss: 19.1282
Epoch 14/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 83s 588ms/step - accuracy: 0.7807 - loss: 18.4499 - val_accuracy: 0.6757 - val_loss: 18.5781
Epoch 15/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 49s 587ms/step - accuracy: 0.7796 - loss: 17.8472 - val_accuracy: 0.6216 - val_loss: 17.8392

Training with Adam optimizer...
Epoch 1/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 76s 704ms/step - accuracy: 0.5335 - loss: 31.2727 - val_accuracy: 0.3750 - val_loss: 26.4531
Epoch 2/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 63s 585ms/step - accuracy: 0.6232 - loss: 17.5406 - val_accuracy: 0.3750 - val_loss: 10.8122
Epoch 3/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 48s 581ms/step - accuracy: 0.6873 - loss: 8.1693 - val_accuracy: 0.3851 - val_loss: 6.0771
Epoch 4/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 82s 577ms/step - accuracy: 0.7029 - loss: 5.2851 - val_accuracy: 0.3446 - val_loss: 4.3261
Epoch 5/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 49s 592ms/step - accuracy: 0.7195 - loss: 3.8486 - val_accuracy: 0.5642 - val_loss: 3.4927
Epoch 6/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 82s 589ms/step - accuracy: 0.7645 - loss: 2.9795 - val_accuracy: 0.6149 - val_loss: 3.1154
Epoch 7/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 49s 595ms/step - accuracy: 0.7347 - loss: 2.8987 - val_accuracy: 0.6182 - val_loss: 2.7620
Epoch 8/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 81s 580ms/step - accuracy: 0.7512 - loss: 2.3795 - val_accuracy: 0.5676 - val_loss: 2.5015
Epoch 9/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 49s 586ms/step - accuracy: 0.7690 - loss: 2.0692 - val_accuracy: 0.6284 - val_loss: 2.4051
Epoch 10/15
83/83 ━━━━━━━━━━━━━━━━━━━━ 82s 580ms/step - accuracy: 0.7588 - loss: 2.0454 - val_accuracy: 0.5845 - val_loss: 2.1828
Epoch 11/15
```

```
# Plot comparison
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history_sgd.history['val_accuracy'], label='SGD')
plt.plot(history_adam.history['val_accuracy'], label='Adam')
```

```
plt.title('Validation Accuracy Comparison')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history_sgd.history['val_loss'], label='SGD')
plt.plot(history_adam.history['val_loss'], label='Adam')
plt.title('Validation Loss Comparison')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

plt.show()
```



Part B: Fine-Tuning a Pre-Trained Model

```
from tensorflow.keras.applications import VGG16, ResNet50
from tensorflow.keras.models import Model

# Load pre-trained VGG16 model
def build_vgg_model():
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,224,3))

    # Freeze convolutional layers
    for layer in base_model.layers:
        layer.trainable = False

    # Add custom layers
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    predictions = Dense(3, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=predictions)
    return model

vgg_model = build_vgg_model()
vgg_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
Model: "functional_192"
```

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense_13 (Dense)	(None, 512)	262,656
dropout_18 (Dropout)	(None, 512)	0
dense_14 (Dense)	(None, 3)	1,539

```
Total params: 14,978,883 (57.14 MB)
Trainable params: 264,195 (1.01 MB)
```

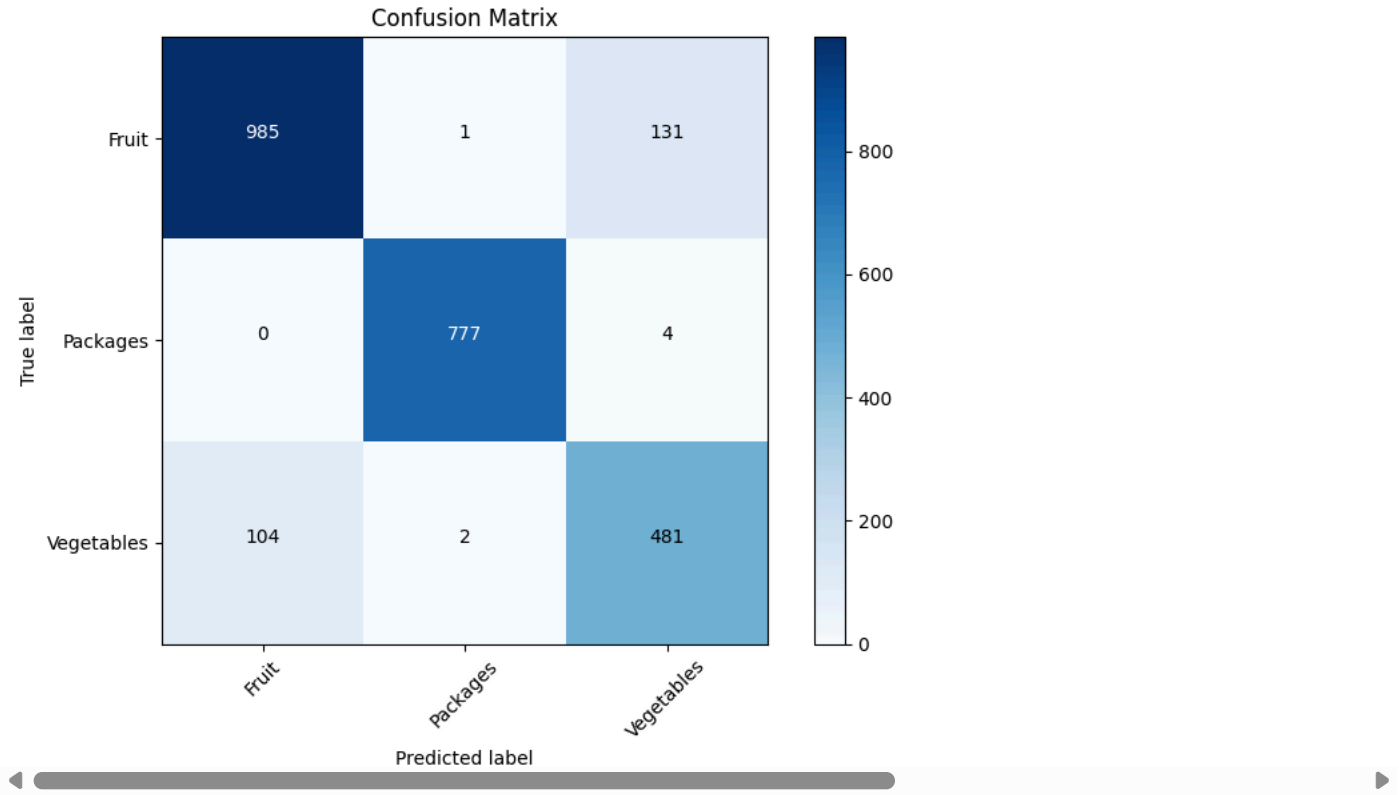
```
# Compile and train VGG model
vgg_model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

history_vgg = vgg_model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=20,
    validation_data=validation_generator,
    validation_steps=len(validation_generator))
```

```
Epoch 1/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 74s 743ms/step - accuracy: 0.6717 - loss: 0.7357 - val_accuracy: 0.8378 - val_loss: 0.3497
Epoch 2/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 49s 594ms/step - accuracy: 0.8317 - loss: 0.3709 - val_accuracy: 0.8682 - val_loss: 0.2925
Epoch 3/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 81s 587ms/step - accuracy: 0.8543 - loss: 0.3401 - val_accuracy: 0.8750 - val_loss: 0.3007
Epoch 4/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 83s 596ms/step - accuracy: 0.8488 - loss: 0.3459 - val_accuracy: 0.8716 - val_loss: 0.2808
Epoch 5/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 49s 585ms/step - accuracy: 0.8433 - loss: 0.3354 - val_accuracy: 0.8682 - val_loss: 0.2793
Epoch 6/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 49s 586ms/step - accuracy: 0.8596 - loss: 0.3127 - val_accuracy: 0.8716 - val_loss: 0.2761
Epoch 7/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 82s 585ms/step - accuracy: 0.8804 - loss: 0.2810 - val_accuracy: 0.8649 - val_loss: 0.2774
Epoch 8/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 83s 598ms/step - accuracy: 0.8815 - loss: 0.2710 - val_accuracy: 0.8919 - val_loss: 0.2671
```

```
Epoch 9/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 80s 577ms/step - accuracy: 0.8809 - loss: 0.2901 - val_accuracy: 0.8649 - val_loss: 0.2479
Epoch 10/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 84s 602ms/step - accuracy: 0.8714 - loss: 0.2680 - val_accuracy: 0.8818 - val_loss: 0.2558
Epoch 11/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 49s 588ms/step - accuracy: 0.8916 - loss: 0.2600 - val_accuracy: 0.8885 - val_loss: 0.2680
Epoch 12/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 83s 601ms/step - accuracy: 0.8901 - loss: 0.2500 - val_accuracy: 0.8716 - val_loss: 0.2456
Epoch 13/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 81s 588ms/step - accuracy: 0.8834 - loss: 0.2624 - val_accuracy: 0.8885 - val_loss: 0.2786
Epoch 14/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 83s 603ms/step - accuracy: 0.8829 - loss: 0.2792 - val_accuracy: 0.8953 - val_loss: 0.2401
Epoch 15/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 81s 582ms/step - accuracy: 0.9018 - loss: 0.2354 - val_accuracy: 0.8919 - val_loss: 0.2522
Epoch 16/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 82s 590ms/step - accuracy: 0.8934 - loss: 0.2314 - val_accuracy: 0.8851 - val_loss: 0.2417
Epoch 17/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 48s 575ms/step - accuracy: 0.9053 - loss: 0.2260 - val_accuracy: 0.8953 - val_loss: 0.2426
Epoch 18/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 83s 591ms/step - accuracy: 0.9116 - loss: 0.2104 - val_accuracy: 0.8986 - val_loss: 0.2320
Epoch 19/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 82s 589ms/step - accuracy: 0.9071 - loss: 0.2116 - val_accuracy: 0.8953 - val_loss: 0.2199
Epoch 20/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 50s 596ms/step - accuracy: 0.9039 - loss: 0.2331 - val_accuracy: 0.9088 - val_loss: 0.2344
```

```
def evaluate_model(model, generator):
    y_true = generator.classes
    y_pred = model.predict(generator, steps=len(generator)).argmax(axis=1)

    print("Classification Report:")
    print(classification_report(y_true, y_pred,
                                target_names=generator.class_indices.keys()))

    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8,6))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()
    tick_marks = np.arange(len(generator.class_indices))
    plt.xticks(tick_marks, generator.class_indices.keys(), rotation=45)
    plt.yticks(tick_marks, generator.class_indices.keys())

    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()



# Evaluate VGG model
print("VGG16 Model Evaluation:")
evaluate_model(vgg_model, test_generator)
```

```
VGG16 Model Evaluation:
2485/2485 ━━━━━━━━━━━━━━━━━━━━ 1664s 669ms/step
Classification Report:
              precision    recall  f1-score   support

       Fruit       0.90      0.88      0.89      1117
    Packages       1.00      0.99      1.00       781
  Vegetables       0.78      0.82      0.80       587

    accuracy                           0.90      2485
   macro avg       0.89      0.90      0.90      2485
weighted avg       0.90      0.90      0.90      2485
```




```
# Load pre-trained ResNet50 model
def build_resnet_model():
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,224,3))

    # Freeze convolutional layers
    for layer in base_model.layers:
        layer.trainable = False

    # Add custom layers
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    predictions = Dense(3, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=predictions)
    return model

resnet_model = build_resnet_model()
resnet_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
Model: "functional_193"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_5 (InputLayer)	(None, 224, 224, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_layer_5[0]…
conv1_conv (Conv2D)	(None, 112, 112, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalizatio…	(None, 112, 112, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 112, 112, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block1_1_c…
conv2_block1_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_block1_1_b…
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_block1_1_r…
conv2_block1_2_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block1_2_c…
conv2_block1_2_relu (Activation)	(None, 56, 56, 64)	0	conv2_block1_2_b…
conv2_block1_0_conv	(None, 56, 56,	16,640	pool1_pool[0][0]

```
# Compile and train ResNet model
resnet_model.compile(optimizer='adam',
                     loss='categorical_crossentropy',
                     metrics=['accuracy'])

history_resnet = resnet_model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=20,
    validation_data=validation_generator,
    validation_steps=len(validation_generator))
```

(Activation)	256)		
conv2_block2_1_conv (Conv2D)	(None, 56, 56, 64)	16,448	conv2_block1_out…
conv2_block2_1_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block2_1_c…
conv2_block2_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_block2_1_b…
conv2_block2_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_block2_1_r…
conv2_block2_2_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block2_2_c…
conv2_block2_2_relu	(None, 56, 56,	0	conv2_block2_2_b…

```
Epoch 1/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 71s 695ms/step - accuracy: 0.3766 - loss: 1.2976 - val_accuracy: 0.3953 - val_loss: 1.0704
Epoch 2/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 65s 556ms/step - accuracy: 0.4427 - loss: 1.0641 - val_accuracy: 0.4189 - val_loss: 1.0634
Epoch 3/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 82s 553ms/step - accuracy: 0.4645 - loss: 1.0551 - val_accuracy: 0.4628 - val_loss: 1.0432
Epoch 4/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 45s 545ms/step - accuracy: 0.4537 - loss: 1.0519 - val_accuracy: 0.5135 - val_loss: 1.0369
Epoch 5/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 46s 554ms/step - accuracy: 0.5040 - loss: 1.0257 - val_accuracy: 0.4696 - val_loss: 1.0379
Epoch 6/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 81s 543ms/step - accuracy: 0.4990 - loss: 1.0368 - val_accuracy: 0.4561 - val_loss: 1.0464
Epoch 7/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 83s 559ms/step - accuracy: 0.4537 - loss: 1.0527 - val_accuracy: 0.5236 - val_loss: 1.0167
Epoch 8/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 46s 549ms/step - accuracy: 0.5065 - loss: 1.0223 - val_accuracy: 0.5135 - val_loss: 1.0166
Epoch 9/20
```