Dataset Preparation & Visualization

```
# --------------------------
# Step 1: Environment Setup
# --------------------------
import tensorflow as tf
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import os
import gdown
import zipfile
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, TensorBoard
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNetV2, ResNet50, VGG16
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.models import load_model
from tensorflow.keras import layers, models

print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.18.0
```

Explanation:

- Import essential libraries
- Verify TensorFlow version (should be 2.x)
- We'll use Keras for model building and ImageDataGenerator for data loading

```
# ------------------------------
# Step 2: Download & Prepare Data
# ------------------------------

# Download dataset
output = 'ninjacart_data.zip'

# Extract files
with zipfile.ZipFile(output, 'r') as zip_ref:
    zip_ref.extractall('dataset')

# Verify structure
train_dir = 'dataset/ninjacart_data/train'
test_dir = 'dataset/ninjacart_data/test'

print("\nTrain directory structure:")
print(os.listdir(train_dir))
print("\nTest directory structure:")
print(os.listdir(test_dir))
```

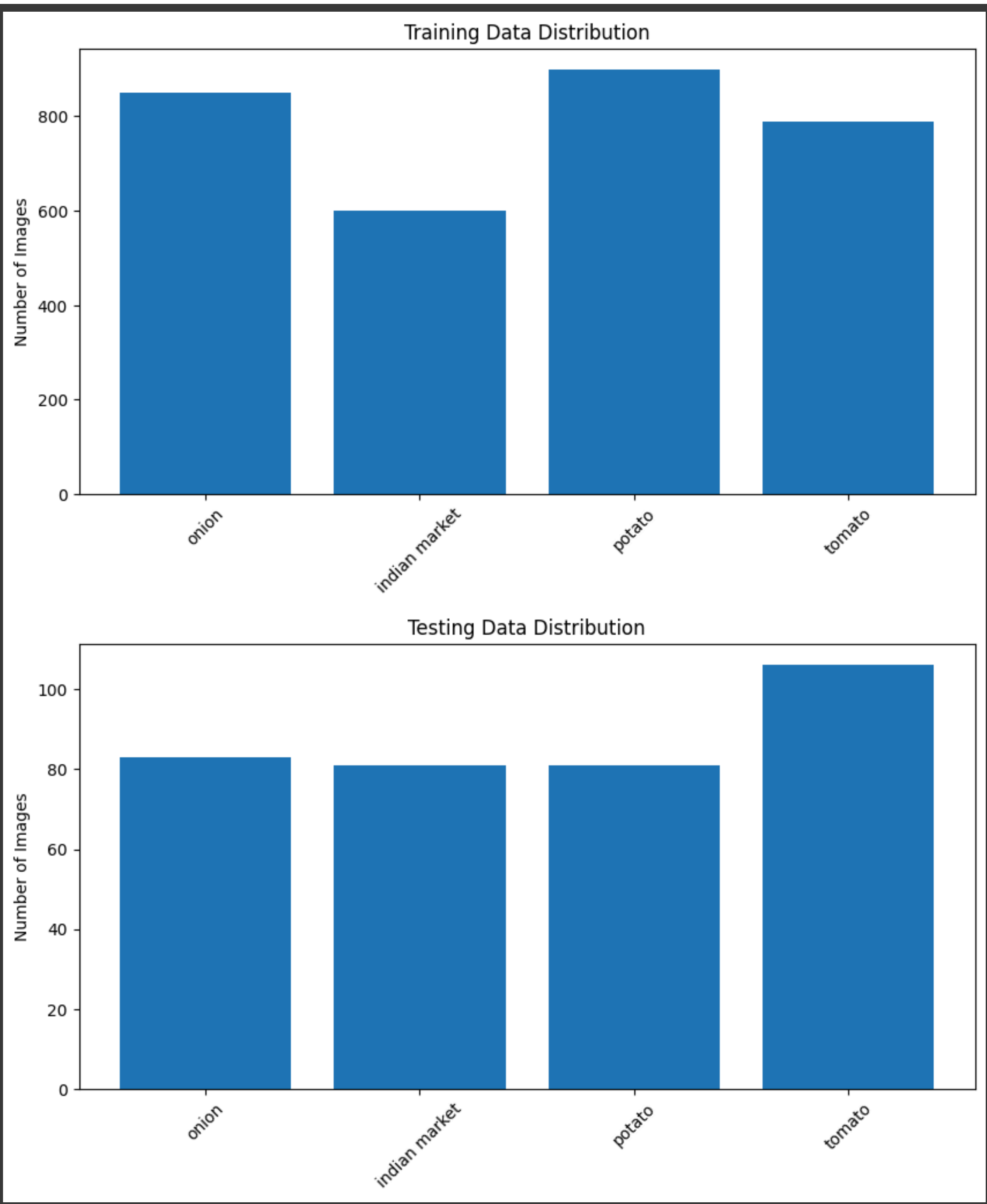
```

Train directory structure:
['onion', 'indian market', 'potato', 'tomato']

Test directory structure:
['onion', 'indian market', 'potato', 'tomato']
```

Explanation:

- Extract ZIP file to 'dataset' directory
- Verify folder structure matches expected classes

```
# ----------------------------------
# Step 3: Exploratory Data Analysis
# ----------------------------------

def plot_class_distribution(directory, title):
    class_counts = [len(os.listdir(os.path.join(directory, cls)))
                    for cls in os.listdir(directory)]
    classes = os.listdir(directory)
```

```
    plt.figure(figsize=(10,5))
    plt.bar(classes, class_counts)
    plt.title(title)
    plt.ylabel('Number of Images')
    plt.xticks(rotation=45)
    plt.show()

plot_class_distribution(train_dir, 'Training Data Distribution')
plot_class_distribution(test_dir, 'Testing Data Distribution')

# Check image dimensions
def analyze_image_sizes(directory):
    sizes = []
    for cls in os.listdir(directory):
        cls_dir = os.path.join(directory, cls)
        for img_file in os.listdir(cls_dir)[:100]:  # Check first 100 images
            img_path = os.path.join(cls_dir, img_file)
            img = plt.imread(img_path)
            sizes.append(img.shape)
    return sizes

train_sizes = analyze_image_sizes(train_dir)
unique_sizes = set(train_sizes)
print(f"\nUnique image sizes in training data: {unique_sizes}")
```




```
Unique image sizes in training data: {(225, 300, 3), (177, 285, 3), (540, 500, 3), (480, 640, 3), (169, 299, 3), (182, 277, 3), (52
```

Explanation:

- Visualize class distribution using bar plots
- Analyze image dimensions variation
- Helps identify if resizing is needed (we'll need to standardize sizes)

```
# ------------------------------
# Step 4: Data Preprocessing
# ------------------------------

# Constants
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
NUM_CLASSES = 4

# Create data generators with augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
```

```
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Training data flow
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training'
)

# Validation data flow
val_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'
)

# Test data flow
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)

print("\nClass indices:", train_generator.class_indices)
```

```
Found 2511 images belonging to 4 classes.
Found 624 images belonging to 4 classes.
Found 351 images belonging to 4 classes.

Class indices: {'indian market': 0, 'onion': 1, 'potato': 2, 'tomato': 3}
```

Explanation:

- Standardize image size to 224x224 (standard for transfer learning models)
- Use data augmentation for training set
- Split training data into 80% train / 20% validation
- Create separate generators for each dataset
- Note the class indices for reference

Model Development & Training

```
# ------------------------------
# Step 5: Build Base CNN Model
# ------------------------------

def create_cnn_model():
    model = models.Sequential([
        layers.Conv2D(32, (3,3), activation='relu', input_shape=(224,224,3)),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(128, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(NUM_CLASSES, activation='softmax')
    ])

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

```
    return model

cnn_model = create_cnn_model()
print(cnn_model.summary())
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  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
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 4)	2,052

```
 Total params: 44,398,148 (169.37 MB)
 Trainable params: 44,398,148 (169.37 MB)
 Non-trainable params: 0 (0.00 B)
None
```

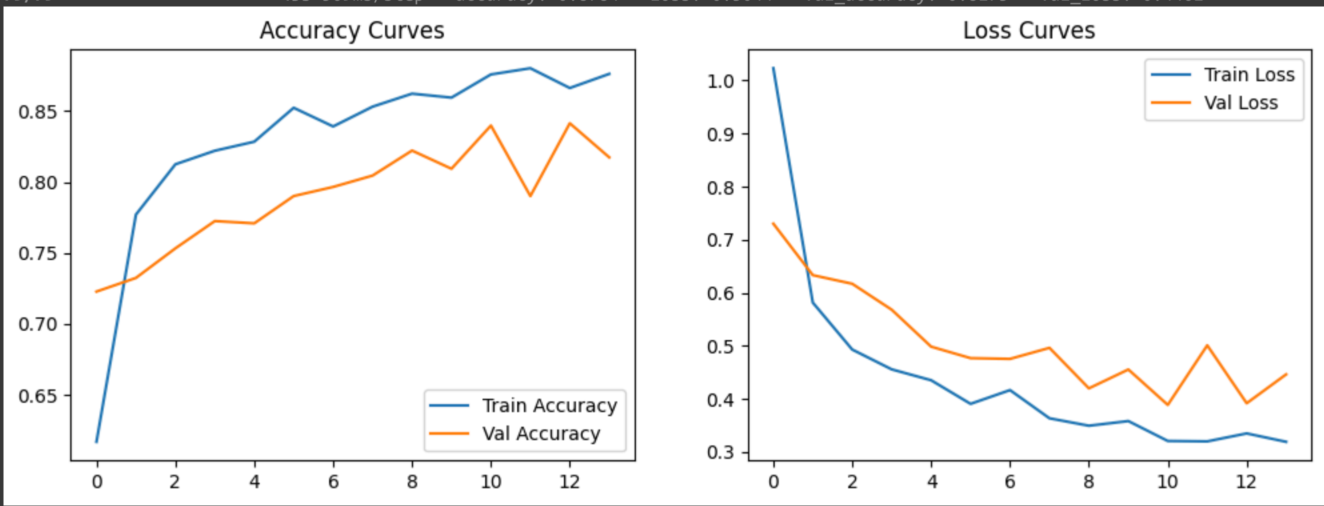
Explanation:

- Basic CNN architecture with increasing filters
- MaxPooling for dimensionality reduction
- Dropout layer to prevent overfitting
- Softmax output for multi-class classification

```
# ------------------------------
# Step 6: Train Base Model
# ------------------------------

# Define callbacks
callbacks = [
    EarlyStopping(patience=3, monitor='val_loss'),
    ModelCheckpoint('best_cnn_model.keras', save_best_only=True),
    TensorBoard(log_dir='logs/cnn')
]

# Train model
history = cnn_model.fit(
    train_generator,
    epochs=20,
    validation_data=val_generator,
    callbacks=callbacks
)

# 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='Val Accuracy')
    plt.title('Accuracy Curves')
    plt.legend()

    plt.subplot(1,2,2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title('Loss Curves')
    plt.legend()

    plt.show()
```

```
plot_history(history)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` c
  self._warn_if_super_not_called()
Epoch 1/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 72s 810ms/step - accuracy: 0.4975 - loss: 1.5613 - val_accuracy: 0.7228 - val_loss: 0.7301
Epoch 2/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 51s 648ms/step - accuracy: 0.7593 - loss: 0.6303 - val_accuracy: 0.7324 - val_loss: 0.6331
Epoch 3/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 51s 641ms/step - accuracy: 0.8141 - loss: 0.5117 - val_accuracy: 0.7532 - val_loss: 0.6168
Epoch 4/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 57s 724ms/step - accuracy: 0.8103 - loss: 0.4814 - val_accuracy: 0.7724 - val_loss: 0.5680
Epoch 5/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 55s 690ms/step - accuracy: 0.8221 - loss: 0.4561 - val_accuracy: 0.7708 - val_loss: 0.4983
Epoch 6/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 50s 631ms/step - accuracy: 0.8513 - loss: 0.4025 - val_accuracy: 0.7901 - val_loss: 0.4767
Epoch 7/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 90s 740ms/step - accuracy: 0.8477 - loss: 0.4041 - val_accuracy: 0.7965 - val_loss: 0.4753
Epoch 8/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 75s 648ms/step - accuracy: 0.8635 - loss: 0.3485 - val_accuracy: 0.8045 - val_loss: 0.4961
Epoch 9/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 94s 1s/step - accuracy: 0.8614 - loss: 0.3500 - val_accuracy: 0.8221 - val_loss: 0.4198
Epoch 10/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 94s 586ms/step - accuracy: 0.8641 - loss: 0.3506 - val_accuracy: 0.8093 - val_loss: 0.4553
Epoch 11/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 51s 644ms/step - accuracy: 0.8667 - loss: 0.3497 - val_accuracy: 0.8397 - val_loss: 0.3886
Epoch 12/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 603ms/step - accuracy: 0.8750 - loss: 0.3244 - val_accuracy: 0.7901 - val_loss: 0.5008
Epoch 13/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 44s 559ms/step - accuracy: 0.8642 - loss: 0.3185 - val_accuracy: 0.8413 - val_loss: 0.3917
Epoch 14/20
79/79 ━━━━━━━━━━━━━━━━━━━━ 45s 569ms/step - accuracy: 0.8784 - loss: 0.3044 - val_accuracy: 0.8173 - val_loss: 0.4461
```



Explanation:

- Use EarlyStopping to prevent overfitting
- Save best model using ModelCheckpoint
- Track metrics with TensorBoard
- Visualize training progress with accuracy/loss curves

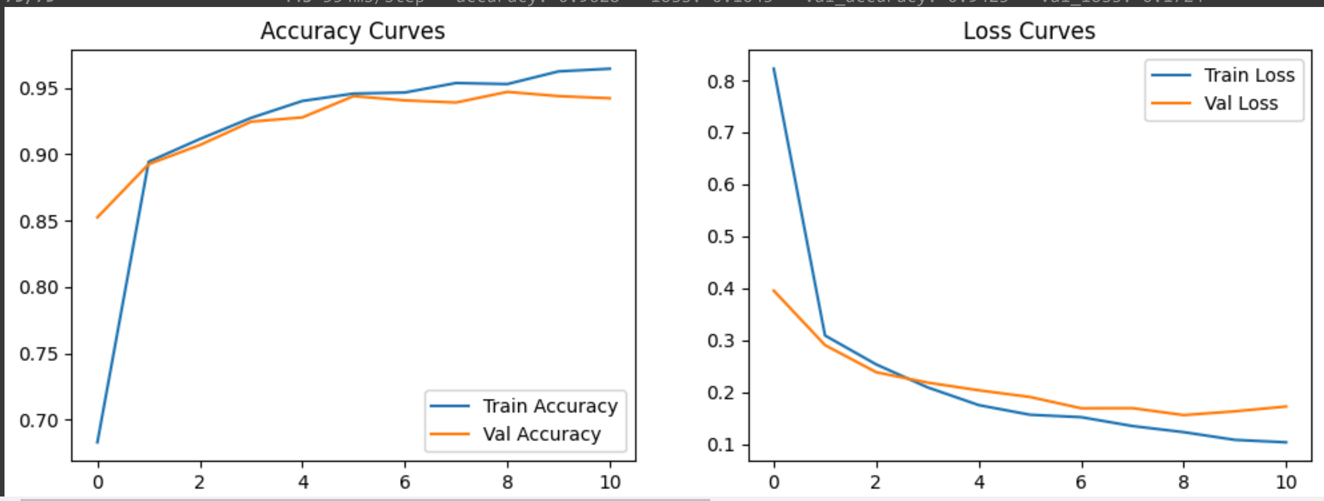
```
# --------------------------------------
# Step 7: Transfer Learning with MobileNet
# --------------------------------------

def create_transfer_model():
    base_model = tf.keras.applications.MobileNetV2(
        input_shape=(224,224,3),
        include_top=False,
        weights='imagenet'
    )

    # Freeze base model layers
    base_model.trainable = False

    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
```

```
        layers.Dropout(0.3),
        layers.Dense(NUM_CLASSES, activation='softmax')
    ])

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])
    return model

mobile_model = create_transfer_model()
print(mobile_model.summary())

# Train transfer model
history_transfer = mobile_model.fit(
    train_generator,
    epochs=15,
    validation_data=val_generator,
    callbacks=[
        ModelCheckpoint('best_mobilenet.keras', save_best_only=True),
        EarlyStopping(patience=2)
    ]
)

plot_history(history_transfer)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 128)	163,968
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 4)	516

```
 Total params: 2,422,468 (9.24 MB)
 Trainable params: 164,484 (642.52 KB)
 Non-trainable params: 2,257,984 (8.61 MB)
None
Epoch 1/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 66s 717ms/step - accuracy: 0.5023 - loss: 1.2157 - val_accuracy: 0.8526 - val_loss: 0.3954
Epoch 2/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 45s 570ms/step - accuracy: 0.8833 - loss: 0.3360 - val_accuracy: 0.8926 - val_loss: 0.2905
Epoch 3/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 47s 592ms/step - accuracy: 0.9003 - loss: 0.2772 - val_accuracy: 0.9071 - val_loss: 0.2383
Epoch 4/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 45s 566ms/step - accuracy: 0.9341 - loss: 0.1986 - val_accuracy: 0.9247 - val_loss: 0.2184
Epoch 5/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 43s 547ms/step - accuracy: 0.9480 - loss: 0.1709 - val_accuracy: 0.9279 - val_loss: 0.2036
Epoch 6/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 45s 569ms/step - accuracy: 0.9374 - loss: 0.1748 - val_accuracy: 0.9439 - val_loss: 0.1907
Epoch 7/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 43s 548ms/step - accuracy: 0.9493 - loss: 0.1479 - val_accuracy: 0.9407 - val_loss: 0.1690
Epoch 8/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 45s 569ms/step - accuracy: 0.9467 - loss: 0.1515 - val_accuracy: 0.9391 - val_loss: 0.1692
Epoch 9/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 45s 567ms/step - accuracy: 0.9570 - loss: 0.1113 - val_accuracy: 0.9471 - val_loss: 0.1559
Epoch 10/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 43s 546ms/step - accuracy: 0.9618 - loss: 0.1065 - val_accuracy: 0.9439 - val_loss: 0.1631
Epoch 11/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 44s 554ms/step - accuracy: 0.9628 - loss: 0.1043 - val_accuracy: 0.9423 - val_loss: 0.1724
```



Explanation:

- Use pre-trained MobileNetV2 as feature extractor
- Add custom classification head
- Lower learning rate for fine-tuning
- GlobalAveragePooling instead of Flatten for better performance
- Fewer epochs needed due to pre-trained weights

```
# For CNN model
callbacks_cnn = [
    ModelCheckpoint('best_cnn_model.keras',
                   monitor='val_accuracy',
                   save_best_only=True,
                   mode='max',
                   ),
    # Other callbacks...
```

```
]

# For MobileNet
callbacks_mobilenet = [
    ModelCheckpoint('best_mobilenet.keras',
                   monitor='val_accuracy',
                   save_best_only=True,
                   mode='max',
                   ),
    # Other callbacks...
]
```

VGG16:

```
# VGG16 Model
def create_vgg16_model():
    # Load pre-trained VGG16 without top layers
    base_model = tf.keras.applications.VGG16(
        weights='imagenet',
        include_top=False,
        input_shape=(224, 224, 3)
    )

    # Freeze base model layers
    base_model.trainable = False

    # Add custom head
    model = tf.keras.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.BatchNormalization(),
        layers.Dense(256, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(4, activation='softmax')
    ])

    # Compile
    model.compile(
        optimizer=tf.keras.optimizers.Adam(0.0001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

# Training VGG16
vgg_model = create_vgg16_model()
vgg_model.summary()

# Callbacks
vgg_callbacks = [
    EarlyStopping(patience=3, restore_best_weights=True),
    ModelCheckpoint('best_vgg.keras', save_best_only=True),
    TensorBoard(log_dir='logs/vgg')
]

# Calculate class weights (add this section)
from sklearn.utils import class_weight
class_weights = class_weight.compute_class_weight(
    'balanced',
    classes=np.unique(train_generator.classes),
    y=train_generator.classes
)
class_weights = dict(enumerate(class_weights))

# Train
history_vgg = vgg_model.fit(
    train_generator,
    epochs=15,
    validation_data=val_generator,
    class_weight=class_weights,
    callbacks=vgg_callbacks
)

# Fine-tuning
vgg_model.layers[0].trainable = True
for layer in vgg_model.layers[0].layers[:15]:  # Unfreeze last few layers
    layer.trainable = False
```

```
vgg_model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history_vgg_finetune = vgg_model.fit(
    train_generator,
    epochs=5,
    validation_data=val_generator,
    callbacks=[ModelCheckpoint('best_vgg_tuned.keras', save_best_only=True)]
)
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 512)	0
batch_normalization_1 (BatchNormalization)	(None, 512)	2,048
dense_6 (Dense)	(None, 256)	131,328
dropout_3 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 4)	1,028

```
 Total params: 14,849,092 (56.64 MB)
 Trainable params: 133,380 (521.02 KB)
 Non-trainable params: 14,715,712 (56.14 MB)
Epoch 1/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 81s 839ms/step - accuracy: 0.4446 - loss: 1.2443 - val_accuracy: 0.5705 - val_loss: 1.1016
Epoch 2/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 54s 681ms/step - accuracy: 0.7340 - loss: 0.6505 - val_accuracy: 0.7244 - val_loss: 0.8937
Epoch 3/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 52s 663ms/step - accuracy: 0.7978 - loss: 0.4902 - val_accuracy: 0.8237 - val_loss: 0.6799
Epoch 4/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 623ms/step - accuracy: 0.8361 - loss: 0.4074 - val_accuracy: 0.8462 - val_loss: 0.5265
Epoch 5/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 625ms/step - accuracy: 0.8396 - loss: 0.3882 - val_accuracy: 0.8734 - val_loss: 0.4266
Epoch 6/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 53s 671ms/step - accuracy: 0.8507 - loss: 0.3562 - val_accuracy: 0.8622 - val_loss: 0.3770
Epoch 7/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 627ms/step - accuracy: 0.8690 - loss: 0.3239 - val_accuracy: 0.8862 - val_loss: 0.3273
Epoch 8/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 617ms/step - accuracy: 0.8870 - loss: 0.3013 - val_accuracy: 0.8878 - val_loss: 0.2921
Epoch 9/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 82s 621ms/step - accuracy: 0.8775 - loss: 0.2968 - val_accuracy: 0.8974 - val_loss: 0.2837
Epoch 10/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 627ms/step - accuracy: 0.8943 - loss: 0.2805 - val_accuracy: 0.8990 - val_loss: 0.2737
Epoch 11/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 619ms/step - accuracy: 0.8928 - loss: 0.2723 - val_accuracy: 0.8958 - val_loss: 0.2894
Epoch 12/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 618ms/step - accuracy: 0.8984 - loss: 0.2531 - val_accuracy: 0.9183 - val_loss: 0.2498
Epoch 13/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 609ms/step - accuracy: 0.9071 - loss: 0.2338 - val_accuracy: 0.9038 - val_loss: 0.2593
Epoch 14/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 616ms/step - accuracy: 0.8944 - loss: 0.2615 - val_accuracy: 0.9151 - val_loss: 0.2469
Epoch 15/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 622ms/step - accuracy: 0.8969 - loss: 0.2471 - val_accuracy: 0.9135 - val_loss: 0.2458
Epoch 1/5
79/79 ━━━━━━━━━━━━━━━━━━━━ 61s 682ms/step - accuracy: 0.9169 - loss: 0.2326 - val_accuracy: 0.9263 - val_loss: 0.2035
Epoch 2/5
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 626ms/step - accuracy: 0.9151 - loss: 0.2204 - val_accuracy: 0.9183 - val_loss: 0.2062
Epoch 3/5
79/79 ━━━━━━━━━━━━━━━━━━━━ 52s 657ms/step - accuracy: 0.9392 - loss: 0.1733 - val_accuracy: 0.9327 - val_loss: 0.1775
Epoch 4/5
79/79 ━━━━━━━━━━━━━━━━━━━━ 50s 631ms/step - accuracy: 0.9506 - loss: 0.1540 - val_accuracy: 0.9215 - val_loss: 0.1805
Epoch 5/5
79/79 ━━━━━━━━━━━━━━━━━━━━ 50s 636ms/step - accuracy: 0.9638 - loss: 0.1159 - val_accuracy: 0.9295 - val_loss: 0.1632
```

ResNet50:

```
# ResNet50 Model
def create_resnet50_model():
    # Load pre-trained ResNet50 without top layers
    base_model = tf.keras.applications.ResNet50(
        weights='imagenet',
        include_top=False,
        input_shape=(224, 224, 3)
    )
```

```
    # Freeze base model layers
    base_model.trainable = False

    # Add custom head
    model = tf.keras.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.BatchNormalization(),
        layers.Dense(256, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(4, activation='softmax')
    ])

    # Compile
    model.compile(
        optimizer=tf.keras.optimizers.Adam(0.0001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

# Training ResNet50
resnet_model = create_resnet50_model()
resnet_model.summary()

# Callbacks
resnet_callbacks = [
    EarlyStopping(patience=3, restore_best_weights=True),
    ModelCheckpoint('best_resnet.keras', save_best_only=True),
    TensorBoard(log_dir='logs/resnet')
]

# Train
history_resnet = resnet_model.fit(
    train_generator,
    epochs=15,
    validation_data=val_generator,
    class_weight=class_weights,
    callbacks=resnet_callbacks
)

# Fine-tuning
resnet_model.layers[0].trainable = True
for layer in resnet_model.layers[0].layers[:100]:  # Unfreeze last few layers
    layer.trainable = False

resnet_model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history_resnet_finetune = resnet_model.fit(
    train_generator,
    epochs=5,
    validation_data=val_generator,
    callbacks=[ModelCheckpoint('best_resnet_tuned.keras', save_best_only=True)]
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kerne
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
global_average_pooling2d_3 (GlobalAveragePooling2D)	(None, 2048)	0
batch_normalization_2 (BatchNormalization)	(None, 2048)	8,192
dense_8 (Dense)	(None, 256)	524,544
dropout_4 (Dropout)	(None, 256)	0
dense_9 (Dense)	(None, 4)	1,028

```
 Total params: 24,121,476 (92.02 MB)
 Trainable params: 529,668 (2.02 MB)
 Non-trainable params: 23,591,808 (90.00 MB)
Epoch 1/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 75s 775ms/step - accuracy: 0.4091 - loss: 1.3212 - val_accuracy: 0.2869 - val_loss: 1.3876
Epoch 2/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 63s 610ms/step - accuracy: 0.5342 - loss: 1.0380 - val_accuracy: 0.3301 - val_loss: 1.2638
Epoch 3/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 51s 642ms/step - accuracy: 0.6069 - loss: 0.9392 - val_accuracy: 0.5160 - val_loss: 1.1673
Epoch 4/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 50s 626ms/step - accuracy: 0.6035 - loss: 0.9002 - val_accuracy: 0.5593 - val_loss: 1.0723
Epoch 5/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 50s 633ms/step - accuracy: 0.6229 - loss: 0.8724 - val_accuracy: 0.5978 - val_loss: 1.0002
Epoch 6/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 79s 606ms/step - accuracy: 0.6335 - loss: 0.8262 - val_accuracy: 0.6250 - val_loss: 0.9244
Epoch 7/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 603ms/step - accuracy: 0.6600 - loss: 0.8044 - val_accuracy: 0.6346 - val_loss: 0.8789
Epoch 8/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 605ms/step - accuracy: 0.6422 - loss: 0.7856 - val_accuracy: 0.6538 - val_loss: 0.8311
Epoch 9/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 605ms/step - accuracy: 0.6726 - loss: 0.8020 - val_accuracy: 0.6875 - val_loss: 0.7716
Epoch 10/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 617ms/step - accuracy: 0.6633 - loss: 0.7528 - val_accuracy: 0.6538 - val_loss: 0.7901
Epoch 11/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 618ms/step - accuracy: 0.6842 - loss: 0.7201 - val_accuracy: 0.6747 - val_loss: 0.7561
Epoch 12/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 617ms/step - accuracy: 0.6880 - loss: 0.7287 - val_accuracy: 0.6875 - val_loss: 0.7264
Epoch 13/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 46s 584ms/step - accuracy: 0.6997 - loss: 0.7239 - val_accuracy: 0.6811 - val_loss: 0.7454
Epoch 14/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 82s 583ms/step - accuracy: 0.6890 - loss: 0.7136 - val_accuracy: 0.6779 - val_loss: 0.7559
Epoch 15/15
79/79 ━━━━━━━━━━━━━━━━━━━━ 48s 600ms/step - accuracy: 0.6869 - loss: 0.7013 - val_accuracy: 0.6923 - val_loss: 0.7163
Epoch 1/5
79/79 ━━━━━━━━━━━━━━━━━━━━ 105s 906ms/step - accuracy: 0.4337 - loss: 1.7639 - val_accuracy: 0.3333 - val_loss: 1.6686
Epoch 2/5
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 618ms/step - accuracy: 0.5406 - loss: 1.2210 - val_accuracy: 0.3077 - val_loss: 2.3817
Epoch 3/5
79/79 ━━━━━━━━━━━━━━━━━━━━ 49s 616ms/step - accuracy: 0.5682 - loss: 1.0858 - val_accuracy: 0.3686 - val_loss: 2.1056
Epoch 4/5
79/79 ━━━━━━━━━━━━━━━━━━━━ 54s 681ms/step - accuracy: 0.6113 - loss: 1.0415 - val_accuracy: 0.4567 - val_loss: 1.5942
Epoch 5/5
79/79 ━━━━━━━━━━━━━━━━━━━━ 79s 646ms/step - accuracy: 0.6160 - loss: 0.9760 - val_accuracy: 0.5994 - val_loss: 0.9856
```

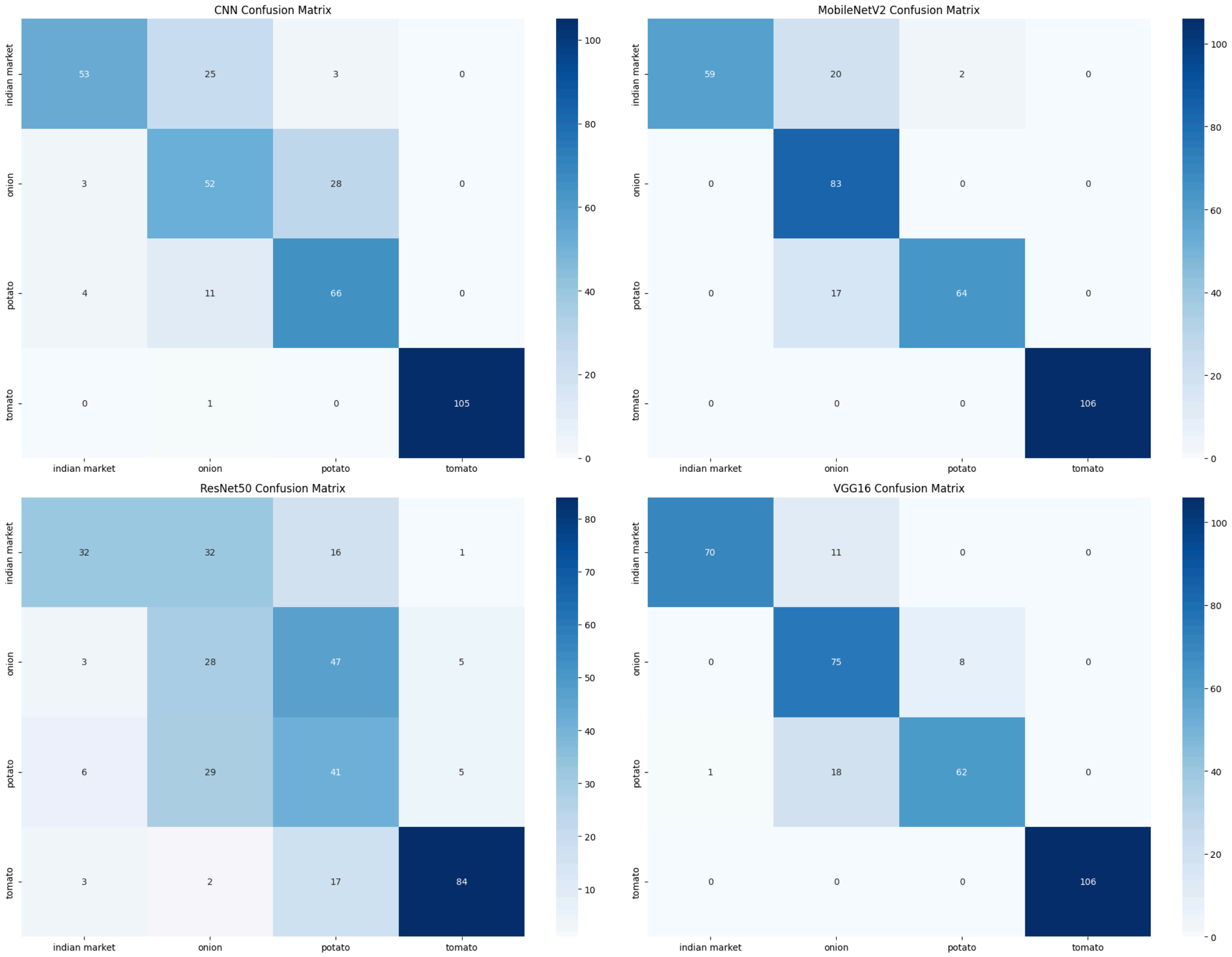
Evaluation & Insights

```
# ------------------------------
# Step 8: Model Evaluation
# ------------------------------
def evaluate_all_models(test_gen):
    models = {
        'CNN': 'best_cnn_model.keras',
        'MobileNetV2': 'best_mobilenet.keras',
        'ResNet50': 'best_resnet_tuned.keras',
        'VGG16': 'best_vgg_tuned.keras'
    }

    metrics = []
    plt.figure(figsize=(20, 15))

    for idx, (name, path) in enumerate(models.items()):
        try:
            # Load model
            model = tf.keras.models.load_model(path)
```

```
            # Evaluate
            loss, acc = model.evaluate(test_gen, verbose=0)
            preds = model.predict(test_gen)
            y_pred = np.argmax(preds, axis=1)
            y_true = test_gen.classes

            # Classification report
            report = classification_report(y_true, y_pred,
                                          target_names=test_gen.class_indices.keys(),
                                          output_dict=True)

            # Store metrics
            metrics.append({
                'Model': name,
                'Test Accuracy': f"{acc:.4f}",
                'Test Loss': f"{loss:.4f}",
                'Precision (Macro)': f"{report['macro avg']['precision']:.4f}",
                'Recall (Macro)': f"{report['macro avg']['recall']:.4f}",
                'F1 (Macro)': f"{report['macro avg']['f1-score']:.4f}",
                'Precision (Weighted)': f"{report['weighted avg']['precision']:.4f}",
                'Recall (Weighted)': f"{report['weighted avg']['recall']:.4f}",
                'F1 (Weighted)': f"{report['weighted avg']['f1-score']:.4f}"
            })

            # Confusion Matrix
            plt.subplot(2, 2, idx+1)
            cm = confusion_matrix(y_true, y_pred)
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                        xticklabels=test_gen.class_indices.keys(),
                        yticklabels=test_gen.class_indices.keys())
            plt.title(f'{name} Confusion Matrix')

        except Exception as e:
            print(f"Error evaluating {name}: {str(e)}")
            continue

    # Display metrics
    metrics_df = pd.DataFrame(metrics)
    # Convert 'Test Accuracy' and 'Test Loss' columns to numeric
    metrics_df['Test Accuracy'] = pd.to_numeric(metrics_df['Test Accuracy'])
    metrics_df['Test Loss'] = pd.to_numeric(metrics_df['Test Loss'])

    print("\n\n=== Model Comparison Table ===")
    display(metrics_df.style
            .background_gradient(cmap='Blues', subset=['Test Accuracy'])
            .format({'Test Accuracy': '{:.2f}', 'Test Loss': '{:.4f}'}))

    plt.tight_layout()
    plt.show()

# Run evaluation
evaluate_all_models(test_generator)
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 158ms/step
11/11 ━━━━━━━━━━━━━━━━━━━━ 8s 479ms/step
11/11 ━━━━━━━━━━━━━━━━━━━━ 10s 692ms/step
11/11 ━━━━━━━━━━━━━━━━━━━━ 3s 267ms/step


=== Model Comparison Table ===
```

	Model	Test Accuracy	Test Loss	Precision (Macro)	Recall (Macro)	F1 (Macro)	Precision (Weighted)	Recall (Weighted)	F1 (Weighted)
0	CNN	0.79	0.5648	0.7870	0.7716	0.7733	0.8010	0.7863	0.7882
1	MobileNetV2	0.89	0.3183	0.9153	0.8796	0.8828	0.9201	0.8889	0.8908
2	ResNet50	0.53	1.1483	0.5645	0.5078	0.5189	0.5858	0.5271	0.5403
3	VGG16	0.89	0.4136	0.8982	0.8833	0.8861	0.9044	0.8917	0.8937



Explanation:

- Compare performance of all models

- Generate confusion matrices to understand class-wise performance
- Load best saved weights before evaluation