LAB 1: WORKING WITH PRE-TRAINED MODELS

Machine Learning Hardware Course

OVERVIEW

This lab introduces you to the practical application of pre-trained convolutional neural networks (CNNs) using the MNIST or Fashion-MNIST dataset. You will experiment with established architectures such as MobileNet, ResNet, and VGG, analyzing their performance, efficiency, and hardware requirements. Through hands-on implementation, you will gain experience with transfer learning, model adaptation, and quantitative evaluation of model characteristics.

LEARNING OBJECTIVES

By the end of this lab, you will be able to:

1. Configure a Google Colab environment for deep learning development
2. Adapt pre-trained CNN architectures for small grayscale image datasets
3. Apply memory-efficient techniques when working with pre-trained models
4. Compare multiple model architectures based on performance metrics
5. Evaluate the impact of model complexity on hardware requirements
6. Implement transfer learning techniques for efficient model adaptation
7. Quantitatively analyze model performance versus computational cost

---

PREREQUISITES

- Basic Python programming knowledge
- Familiarity with deep learning concepts
- Google account for accessing Google Colab
- Understanding of CNN architectures (basic level)

TIME ALLOCATION

Total time: 2 hours (120 minutes)

Activity	Duration
Environment Setup	15 minutes
Dataset Preparation	15 minutes
Model Adaptation	30 minutes
Model Evaluation	30 minutes
Performance Analysis	20 minutes
Worksheet Completion	20 minutes

REQUIRED MATERIALS

- Computer with internet access
- Google account
- This lab guide
- Graded worksheet (provided separately)

LAB SETUP INSTRUCTIONS

Accessing Google Colab

1. Open your web browser and navigate to https://colab.research.google.com
2. Sign in with your Google account
3. Create a new notebook by clicking on "New Notebook"
4. Rename your notebook to "Lab1_PretrainedModels_YourName"

Mounting Google Drive (for saving your work)

python

Copy

```python
from google.colab import drive
drive.mount('/content/drive')

# Create a directory for this lab
!mkdir -p "/content/drive/My Drive/ML_Hardware_Course/Lab1"
```

Installing Required Libraries

Execute the following code to install and import the necessary libraries:

python

Copy

```python
# Import basic libraries
import numpy as np
import matplotlib.pyplot as plt
import time
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
import pandas as pd

# TensorFlow and Keras
import tensorflow as tf
from tensorflow.keras.datasets import mnist, fashion_mnist
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Flatten, Dropout, GlobalAveragePooling2D
from tensorflow.keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from tensorflow.keras.applications import MobileNetV2, ResNet50, VGG16
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping

# Check TensorFlow version
print("TensorFlow version:", tf.__version__)

# Check for GPU availability
print("GPU Available: ", tf.config.list_physical_devices('GPU'))
print("GPU Details:")
!nvidia-smi
```

PART 1: DATASET PREPARATION (15 minutes)

You can work with either the MNIST dataset (handwritten digits) or the Fashion-MNIST dataset (clothing items). Both datasets contain grayscale images with the same dimensions (28x28), but pre-trained models have specific input requirements we need to adapt to.

Step 1.1: Load the Dataset

python

Copy

```python
# Choose which dataset to use (uncomment one)
# Option 1: MNIST (Handwritten Digits)
(X_train, y_train), (X_test, y_test) = mnist.load_data()
class_names = [str(i) for i in range(10)]  # 0-9 digits

# Option 2: Fashion-MNIST (Clothing Items)
# (X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
# class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
#                'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Print dataset shapes
print("Training data shape:", X_train.shape)
print("Training labels shape:", y_train.shape)
print("Test data shape:", X_test.shape)
print("Test labels shape:", y_test.shape)
```

Step 1.2: Visualize Sample Images

python

Copy

```python
# Create a function to display multiple images
def display_sample_images(X, y, num_samples=10):
    plt.figure(figsize=(15, 3))
    for i in range(num_samples):
        plt.subplot(1, num_samples, i+1)
        plt.imshow(X[i], cmap='gray')
        plt.title(f"{class_names[y[i]]}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Display 10 sample images
display_sample_images(X_train, y_train)
```

Step 1.3: Preprocess the Data

We'll use a memory-efficient approach that doesn't require resizing the entire dataset:

python

Copy

```python
# Simple preprocessing function
def preprocess_mnist_simple(X_train, X_test):
    """
    Simple preprocessing for MNIST/Fashion-MNIST, normalizing and adding channel dimensio
    """
    # Normalize to [0,1]
    X_train = X_train.astype('float32') / 255.0
    X_test = X_test.astype('float32') / 255.0

    # Add channel dimension (28x28 -> 28x28x1)
    X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
    X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)

    return X_train, X_test

# Prepare the labels
y_train_encoded = to_categorical(y_train, 10)
y_test_encoded = to_categorical(y_test, 10)

# Create a validation set (20% of training data)
val_size = 12000  # 20% of 60,000
X_val = X_train[-val_size:]
y_val = y_train_encoded[-val_size:]
X_train_final = X_train[:-val_size]
y_train_final = y_train_encoded[:-val_size]

print("Training set size:", X_train_final.shape[0])
print("Validation set size:", X_val.shape[0])
print("Test set size:", X_test.shape[0])
```

PART 2: MODEL ADAPTATION (30 minutes)

In this section, you will adapt pre-trained CNN architectures for the dataset. Instead of resizing images to meet the models' requirements, we'll modify the model architectures to accept 28x28 images.

Step 2.1: Prepare MobileNetV2 Model

python

Copy

```
def create_mobilenet_model():
    """
    Create MobileNetV2 model with properly padded input for MNIST/Fashion-MNIST
    """
    # Preprocess data
    X_train_mobilenet, X_test_mobilenet = preprocess_mnist_simple(X_train_final, X_test)
    X_val_mobilenet, _ = preprocess_mnist_simple(X_val, np.zeros((1, 28, 28)))

    # Create model architecture
    inputs = Input(shape=(28, 28, 1))

    # Pad the input from 28x28 to 32x32 using zero padding
    x = ZeroPadding2D(padding=2)(inputs)  # Add 2 pixels on each side: 28x28 -> 32x32

    # Convert single-channel grayscale to 3-channel RGB format required by MobileNetV2
    x = Conv2D(16, kernel_size=3, padding='same', activation='relu')(x)
    x = Conv2D(3, kernel_size=1, padding='same', activation='relu')(x)  # Output 3 channe

    # Create sub-model with MobileNetV2
    base_model = MobileNetV2(
        include_top=False,
        weights='imagenet',
        input_shape=(32, 32, 3),
        pooling='avg'
    )

    # Freeze the base model layers
    base_model.trainable = False

    # Continue with the model architecture
    x = base_model(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.2)(x)
    outputs = Dense(10, activation='softmax')(x)

    mobilenet_model = Model(inputs, outputs)

    # Compile the model
    mobilenet_model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
```

```
    )

    # Display model summary
    mobilenet_model.summary()

    return mobilenet_model, X_train_mobilenet, X_val_mobilenet, X_test_mobilenet

# Create MobileNetV2 model
mobilenet_model, X_train_mobilenet, X_val_mobilenet, X_test_mobilenet = create_mobilenet_m
```

Step 2.2: Prepare ResNet50 Model

python

Copy

```python
def create_resnet_model():
    """
    Create ResNet50 model with properly padded input for MNIST/Fashion-MNIST
    """
    # Preprocess data
    X_train_resnet, X_test_resnet = preprocess_mnist_simple(X_train_final, X_test)
    X_val_resnet, _ = preprocess_mnist_simple(X_val, np.zeros((1, 28, 28)))

    # Create model architecture
    inputs = Input(shape=(28, 28, 1))

    # Pad the input from 28x28 to 32x32
    x = ZeroPadding2D(padding=2)(inputs)  # Add 2 pixels on each side

    # Convert single-channel to 3-channel input
    x = Conv2D(16, kernel_size=3, padding='same', activation='relu')(x)
    x = Conv2D(3, kernel_size=1, padding='same', activation='relu')(x)  # Output 3 channe

    # Load ResNet50 with proper input shape
    base_model = ResNet50(
        include_top=False,
        weights='imagenet',
        input_shape=(32, 32, 3),
        pooling='avg'
    )

    # Freeze the base model layers
    base_model.trainable = False

    # Continue with model architecture
    x = base_model(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.3)(x)
    outputs = Dense(10, activation='softmax')(x)

    resnet_model = Model(inputs, outputs)

    # Compile the model
    resnet_model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
```

```python
    )

    # Display model summary
    resnet_model.summary()

    return resnet_model, X_train_resnet, X_val_resnet, X_test_resnet

# Create ResNet50 model
resnet_model, X_train_resnet, X_val_resnet, X_test_resnet = create_resnet_model()
```

Step 2.3: Prepare VGG16 Model

python

Copy

```python
def create_vgg_model():
    """
    Create VGG16 model with properly padded input for MNIST/Fashion-MNIST
    """
    # Preprocess data
    X_train_vgg, X_test_vgg = preprocess_mnist_simple(X_train_final, X_test)
    X_val_vgg, _ = preprocess_mnist_simple(X_val, np.zeros((1, 28, 28)))

    # Create model architecture
    inputs = Input(shape=(28, 28, 1))

    # Pad the input from 28x28 to 32x32
    x = ZeroPadding2D(padding=2)(inputs)  # Add 2 pixels on each side

    # Convert single-channel to 3-channel
    x = Conv2D(16, kernel_size=3, padding='same', activation='relu')(x)
    x = Conv2D(3, kernel_size=1, padding='same', activation='relu')(x)  # Output 3 channe

    # Load VGG16 with proper input shape
    base_model = VGG16(
        include_top=False,
        weights='imagenet',
        input_shape=(32, 32, 3),
        pooling='avg'
    )

    # Freeze the base model layers
    base_model.trainable = False

    # Continue with model architecture
    x = base_model(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.3)(x)
    outputs = Dense(10, activation='softmax')(x)

    vgg_model = Model(inputs, outputs)

    # Compile the model
    vgg_model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
```

```
    )

    # Display model summary
    vgg_model.summary()

    return vgg_model, X_train_vgg, X_val_vgg, X_test_vgg

# Create VGG16 model
vgg_model, X_train_vgg, X_val_vgg, X_test_vgg = create_vgg_model()
```

PART 3: MODEL TRAINING AND EVALUATION (30 minutes)

Now you will train and evaluate each model, recording performance metrics.

Step 3.1: Train MobileNetV2 Model

python

Copy

```
# Early stopping callback
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=3,
    restore_best_weights=True
)

# Record start time
start_time = time.time()

# Train MobileNetV2 model
print("\n--- Training MobileNetV2 Model ---")
mobilenet_history = mobilenet_model.fit(
    X_train_mobilenet,
    y_train_final,
    epochs=10,
    batch_size=64,
    validation_data=(X_val_mobilenet, y_val),
    callbacks=[early_stopping],
    verbose=1
)

# Calculate training time
mobilenet_training_time = time.time() - start_time
print(f"MobileNetV2 - Training completed in {mobilenet_training_time:.2f} seconds")

# Evaluate on test set
mobilenet_loss, mobilenet_accuracy = mobilenet_model.evaluate(X_test_mobilenet, y_test_en
print(f"MobileNetV2 - Test accuracy: {mobilenet_accuracy*100:.2f}%")
```

Step 3.2: Train ResNet50 Model

python

Copy

```python
# Record start time
start_time = time.time()

# Train ResNet50 model
print("\n--- Training ResNet50 Model ---")
resnet_history = resnet_model.fit(
    X_train_resnet,
    y_train_final,
    epochs=10,
    batch_size=32,  # Smaller batch size due to larger model
    validation_data=(X_val_resnet, y_val),
    callbacks=[early_stopping],
    verbose=1
)

# Calculate training time
resnet_training_time = time.time() - start_time
print(f"ResNet50 - Training completed in {resnet_training_time:.2f} seconds")

# Evaluate on test set
resnet_loss, resnet_accuracy = resnet_model.evaluate(X_test_resnet, y_test_encoded)
print(f"ResNet50 - Test accuracy: {resnet_accuracy*100:.2f}%")
```

Step 3.3: Train VGG16 Model

python

Copy

```python
# Record start time
start_time = time.time()

# Train VGG16 model
print("\n--- Training VGG16 Model ---")
vgg_history = vgg_model.fit(
    X_train_vgg,
    y_train_final,
    epochs=10,
    batch_size=64,
    validation_data=(X_val_vgg, y_val),
    callbacks=[early_stopping],
    verbose=1
)

# Calculate training time
vgg_training_time = time.time() - start_time
print(f"VGG16 - Training completed in {vgg_training_time:.2f} seconds")

# Evaluate on test set
vgg_loss, vgg_accuracy = vgg_model.evaluate(X_test_vgg, y_test_encoded)
print(f"VGG16 - Test accuracy: {vgg_accuracy*100:.2f}%")
```

Step 3.4: Plot Training History

python

Copy

```python
# Function to plot training history
def plot_training_history(histories, titles):
    plt.figure(figsize=(15, 5))

    # Plot accuracy
    plt.subplot(1, 2, 1)
    for history, title in zip(histories, titles):
        plt.plot(history.history['accuracy'], label=f'{title} - Training')
        plt.plot(history.history['val_accuracy'], label=f'{title} - Validation')

    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Plot loss
    plt.subplot(1, 2, 2)
    for history, title in zip(histories, titles):
        plt.plot(history.history['loss'], label=f'{title} - Training')
        plt.plot(history.history['val_loss'], label=f'{title} - Validation')

    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

# Plot training history for all models
plot_training_history(
    [mobilenet_history, resnet_history, vgg_history],
    ['MobileNetV2', 'ResNet50', 'VGG16']
)
```

PART 4: PERFORMANCE ANALYSIS (20 minutes)

Analyze the performance of each model in terms of accuracy, training time, and model complexity.

Step 4.1: Generate Confusion Matrices

python

Copy

```python
# Function to generate predictions and confusion matrix
def analyze_model_performance(model, X_test, y_test, model_name):
    # Generate predictions
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_true_classes = np.argmax(y_test, axis=1)

    # Create confusion matrix
    cm = confusion_matrix(y_true_classes, y_pred_classes)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title(f'{model_name} - Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

    # Find the most confused pairs
    cm_normalized = cm.copy()
    np.fill_diagonal(cm_normalized, 0)  # Ignore correct predictions
    max_confusion = np.unravel_index(np.argmax(cm_normalized), cm_normalized.shape)
    print(f"Most confused pair: True {class_names[max_confusion[0]]} predicted as {class_

    # Generate classification report
    report = classification_report(y_true_classes, y_pred_classes, output_dict=True)
    report_df = pd.DataFrame(report).transpose()
    print(f"{model_name} Classification Report:")
    print(report_df.round(3))

    return y_pred_classes, report, max_confusion

# Analyze each model
print("\n--- MobileNetV2 Performance Analysis ---")
mobilenet_pred, mobilenet_report, mobilenet_confused_pair = analyze_model_performance(
    mobilenet_model, X_test_mobilenet, y_test_encoded, 'MobileNetV2'
)

print("\n--- ResNet50 Performance Analysis ---")
resnet_pred, resnet_report, resnet_confused_pair = analyze_model_performance(
    resnet_model, X_test_resnet, y_test_encoded, 'ResNet50'
)

print("\n--- VGG16 Performance Analysis ---")
```

```python
vgg_pred, vgg_report, vgg_confused_pair = analyze_model_performance(
    vgg_model, X_test_vgg, y_test_encoded, 'VGG16'
)
```

Step 4.2: Compare Model Metrics

python

Copy

```
# Function to count model parameters
def count_model_parameters(model):
    trainable_params = np.sum([np.prod(v.shape) for v in model.trainable_weights])
    non_trainable_params = np.sum([np.prod(v.shape) for v in model.non_trainable_weights]
    total_params = trainable_params + non_trainable_params
    return trainable_params, non_trainable_params, total_params

# Get parameter counts for each model
mobilenet_trainable, mobilenet_non_trainable, mobilenet_total = count_model_parameters(mol
resnet_trainable, resnet_non_trainable, resnet_total = count_model_parameters(resnet_mode
vgg_trainable, vgg_non_trainable, vgg_total = count_model_parameters(vgg_model)

# Create comparison table
model_metrics = {
    'Model': ['MobileNetV2', 'ResNet50', 'VGG16'],
    'Test Accuracy (%)': [
        mobilenet_accuracy * 100,
        resnet_accuracy * 100,
        vgg_accuracy * 100
    ],
    'Training Time (s)': [
        mobilenet_training_time,
        resnet_training_time,
        vgg_training_time
    ],
    'Trainable Parameters': [
        mobilenet_trainable,
        resnet_trainable,
        vgg_trainable
    ],
    'Total Parameters': [
        mobilenet_total,
        resnet_total,
        vgg_total
    ],
    'Parameters/Second': [
        mobilenet_total / mobilenet_training_time,
        resnet_total / resnet_training_time,
        vgg_total / vgg_training_time
    ],
    'Accuracy/Million Params': [
        (mobilenet_accuracy * 100) / (mobilenet_total / 1e6),
```

```
        (resnet_accuracy * 100) / (resnet_total / 1e6),
        (vgg_accuracy * 100) / (vgg_total / 1e6)
    ],
    'Most Confused Pair': [
        f"{class_names[mobilenet_confused_pair[0]]}-{class_names[mobilenet_confused_pair[
        f"{class_names[resnet_confused_pair[0]]}-{class_names[resnet_confused_pair[1]]}",
        f"{class_names[vgg_confused_pair[0]]}-{class_names[vgg_confused_pair[1]]}"
    ]
}

# Create and display DataFrame
metrics_df = pd.DataFrame(model_metrics).set_index('Model')
print("\n--- Model Comparison Metrics ---")
pd.set_option('display.float_format', '{:.2f}'.format)
print(metrics_df)

# Visualize the metrics
plt.figure(figsize=(15, 10))

# Accuracy comparison
plt.subplot(2, 2, 1)
plt.bar(model_metrics['Model'], model_metrics['Test Accuracy (%)'])
plt.title('Test Accuracy (%)')
plt.ylim(75, 100)  # Adjust as needed based on results
plt.grid(axis='y')

# Training time comparison
plt.subplot(2, 2, 2)
plt.bar(model_metrics['Model'], model_metrics['Training Time (s)'])
plt.title('Training Time (seconds)')
plt.grid(axis='y')

# Parameter count comparison (log scale)
plt.subplot(2, 2, 3)
plt.bar(model_metrics['Model'], [np.log10(p) for p in model_metrics['Total Parameters']])
plt.title('Log10(Total Parameters)')
plt.grid(axis='y')

# Efficiency comparison
plt.subplot(2, 2, 4)
plt.bar(model_metrics['Model'], model_metrics['Accuracy/Million Params'])
plt.title('Accuracy/Million Parameters')
plt.grid(axis='y')
```

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

Step 4.3: Measure Inference Time

python

Copy

```python
# Function to measure inference time
def measure_inference_time(model, X_test, batch_size=1, num_runs=50):
    # Warm-up
    for _ in range(10):
        _ = model.predict(X_test[:batch_size])

    # Measure time for inference
    start_time = time.time()
    for _ in range(num_runs):
        _ = model.predict(X_test[:batch_size])
    total_time = time.time() - start_time

    # Calculate average inference time per batch
    avg_time = total_time / num_runs
    return avg_time * 1000  # Convert to milliseconds

# Measure inference time for each model (single image)
mobilenet_inference_time = measure_inference_time(mobilenet_model, X_test_mobilenet)
resnet_inference_time = measure_inference_time(resnet_model, X_test_resnet)
vgg_inference_time = measure_inference_time(vgg_model, X_test_vgg)

print("\n--- Single Image Inference Time ---")
print(f"MobileNetV2 - Inference time (1 image): {mobilenet_inference_time:.2f} ms")
print(f"ResNet50 - Inference time (1 image): {resnet_inference_time:.2f} ms")
print(f"VGG16 - Inference time (1 image): {vgg_inference_time:.2f} ms")

# Measure inference time for batch of 32 images
mobilenet_batch_time = measure_inference_time(mobilenet_model, X_test_mobilenet, batch_si
resnet_batch_time = measure_inference_time(resnet_model, X_test_resnet, batch_size=32, nu
vgg_batch_time = measure_inference_time(vgg_model, X_test_vgg, batch_size=32, num_runs=20

print("\n--- Batch Inference Time (32 images) ---")
print(f"MobileNetV2 - Inference time (32 images): {mobilenet_batch_time:.2f} ms")
print(f"ResNet50 - Inference time (32 images): {resnet_batch_time:.2f} ms")
print(f"VGG16 - Inference time (32 images): {vgg_batch_time:.2f} ms")

# Create inference time comparison charts
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.bar(model_metrics['Model'], [mobilenet_inference_time, resnet_inference_time, vgg_inf
plt.title('Single Image Inference Time (ms)')
```

```
plt.grid(axis='y')

plt.subplot(1, 2, 2)
plt.bar(model_metrics['Model'], [mobilenet_batch_time, resnet_batch_time, vgg_batch_time]
plt.title('Batch Inference Time (32 images, ms)')
plt.grid(axis='y')

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

PART 5: WORKSHEET COMPLETION (20 minutes)

Now that you have trained and evaluated the models, complete the worksheet with the numerical results from your analysis. This worksheet will be submitted for grading.

Step 5.1: Create Results Summary

python

Copy

```
# Update the metrics dictionary with inference times
model_metrics.update({
    'Inference Time (ms)': [
        mobilenet_inference_time,
        resnet_inference_time,
        vgg_inference_time
    ],
    'Batch Inference Time (ms)': [
        mobilenet_batch_time,
        resnet_batch_time,
        vgg_batch_time
    ]
})

# Create a results summary file for the worksheet
results_summary = {
    "basic_metrics": {
        "mobilenet": {
            "trainable_params": mobilenet_trainable,
            "total_params": mobilenet_total,
            "test_accuracy": mobilenet_accuracy * 100,
            "training_time": mobilenet_training_time,
            "inference_time": mobilenet_inference_time,
        },
        "resnet": {
            "trainable_params": resnet_trainable,
            "total_params": resnet_total,
            "test_accuracy": resnet_accuracy * 100,
            "training_time": resnet_training_time,
            "inference_time": resnet_inference_time,
        },
        "vgg": {
            "trainable_params": vgg_trainable,
            "total_params": vgg_total,
            "test_accuracy": vgg_accuracy * 100,
            "training_time": vgg_training_time,
            "inference_time": vgg_inference_time,
        }
    },
    "efficiency_metrics": {
        "mobilenet": {
            "params_per_second": mobilenet_total / mobilenet_training_time,
```

```
            "accuracy_per_million_params": (mobilenet_accuracy * 100) / (mobilenet_total /
            "batch_inference_time": mobilenet_batch_time,
        },
        "resnet": {
            "params_per_second": resnet_total / resnet_training_time,
            "accuracy_per_million_params": (resnet_accuracy * 100) / (resnet_total / 1e6),
            "batch_inference_time": resnet_batch_time,
        },
        "vgg": {
            "params_per_second": vgg_total / vgg_training_time,
            "accuracy_per_million_params": (vgg_accuracy * 100) / (vgg_total / 1e6),
            "batch_inference_time": vgg_batch_time,
        }
    },
    "confusion_pairs": {
        "mobilenet": {
            "pair": f"{class_names[mobilenet_confused_pair[0]]}-{class_names[mobilenet_co
            "count": int(confusion_matrix(np.argmax(y_test_encoded, axis=1), mobilenet_pre
        },
        "resnet": {
            "pair": f"{class_names[resnet_confused_pair[0]]}-{class_names[resnet_confused_
            "count": int(confusion_matrix(np.argmax(y_test_encoded, axis=1), resnet_pred)
        },
        "vgg": {
            "pair": f"{class_names[vgg_confused_pair[0]]}-{class_names[vgg_confused_pair[1
            "count": int(confusion_matrix(np.argmax(y_test_encoded, axis=1), vgg_pred)[vg
        }
    },
    "best_model": {
        "name": best_model_name,
        "precision_by_class": {class_names[i]: best_model_report[str(i)]['precision'] for
    }
}

# Convert to DataFrame for easier viewing
results_df = pd.DataFrame({
    "Model": model_metrics['Model'],
    "Test Accuracy (%)": model_metrics['Test Accuracy (%)'],
    "Total Parameters": model_metrics['Total Parameters'],
    "Training Time (s)": model_metrics['Training Time (s)'],
    "Inference Time (ms)": model_metrics['Inference Time (ms)'],
    "Accuracy/Million Params": model_metrics['Accuracy/Million Params']
})

print("\n--- Results Summary for Worksheet ---")
```

```python
print(results_df)

# Uncomment to save results to a file
# results_df.to_csv("model_comparison_results.csv")
```

Step 5.2: Generate Worksheet Values

python

Copy

```python
print("\n===== WORKSHEET VALUES =====")

print("\n1.1 Basic Performance Metrics:")
for model_name, trainable, total, accuracy, train_time, infer_time in zip(
    model_metrics['Model'],
    model_metrics['Trainable Parameters'],
    model_metrics['Total Parameters'],
    model_metrics['Test Accuracy (%)'],
    model_metrics['Training Time (s)'],
    model_metrics['Inference Time (ms)']
):
    print(f"\n{model_name}:")
    print(f"  Trainable Parameters: {trainable}")
    print(f"  Total Parameters: {total}")
    print(f"  Test Accuracy: {accuracy:.2f}%")
    print(f"  Training Time: {train_time:.2f} seconds")
    print(f"  Inference Time: {infer_time:.2f} ms")

print("\n1.2 Efficiency Metrics:")
for model_name, params_per_sec, acc_per_mil, batch_time in zip(
    model_metrics['Model'],
    model_metrics['Parameters/Second'],
    model_metrics['Accuracy/Million Params'],
    [mobilenet_batch_time, resnet_batch_time, vgg_batch_time]
):
    print(f"\n{model_name}:")
    print(f"  Parameters/Second: {params_per_sec:.2f}")
    print(f"  Accuracy/Million Params: {acc_per_mil:.2f}")
    print(f"  Batch Inference Time: {batch_time:.2f} ms")

print("\n2.1 Most Confused Pairs:")
for model_name, confused_pair in zip(
    model_metrics['Model'],
    model_metrics['Most Confused Pair']
):
    print(f"  {model_name}: {confused_pair}")

# Get the best performing model
best_model_idx = np.argmax(model_metrics['Test Accuracy (%)'])
best_model_name = model_metrics['Model'][best_model_idx]
best_model_report = [mobilenet_report, resnet_report, vgg_report][best_model_idx]
```

```python
print(f"\n2.2 Per-Class Precision for Best Model ({best_model_name}):")
for i in range(10):
    print(f"  Class {class_names[i]}: {best_model_report[str(i)]['precision']:.4f}")
```

Model Parameters and Performance

Complete the following table with the values from your experiment:

Model	Trainable Parameters	Total Parameters	Test Accuracy (%)	Training Time (s)	Inference Time (ms)
MobileNetV2					
ResNet50					
VGG16					

Efficiency Metrics

Calculate and record the following efficiency metrics:

Model	Parameters/Second	Accuracy/Million Params	FLOPs/Inference
MobileNetV2			
ResNet50			
VGG16			

Analysis Questions

Answer the following questions based on your results:

1. Which model provides the best balance between accuracy and computational efficiency? Why?
2. How does model size affect training time versus inference time? Explain the differences you observed.
3. Why might you choose MobileNetV2 over ResNet50 or VGG16 for a mobile application?
4. What hardware factors significantly impact the performance of these pre-trained models?

SAVING YOUR WORK

Save your notebook to Google Drive using the following code:

python

Copy

```python
# Save the notebook to Google Drive
notebook_path = "/content/drive/My Drive/ML_Hardware_Course/Lab1/Lab1_PretrainedModels_Yo
print(f"Saving notebook to: {notebook_path}")
```

Also, export your models for future use:

python

Copy

```python
# Save models
mobilenet_model.save("/content/drive/My Drive/ML_Hardware_Course/Lab1/mobilenet_mnist.h5"
resnet_model.save("/content/drive/My Drive/ML_Hardware_Course/Lab1/resnet_mnist.h5")
vgg_model.save("/content/drive/My Drive/ML_Hardware_Course/Lab1/vgg_mnist.h5")
print("Models saved successfully!")
```

SUBMISSION REQUIREMENTS

Submit the following items:

1. Completed Colab notebook (exported as .ipynb file)
2. Filled worksheet with numerical results
3. Short reflection (100 words) on model selection criteria for different applications

---

ADDITIONAL CHALLENGES (Optional)

If you complete the lab early, try these extensions:

1. Implement fine-tuning by unfreezing some layers of the base models
2. Try model quantization to improve inference speed
3. Experiment with other pre-trained architectures (e.g., EfficientNet, DenseNet)
4. Implement pruning techniques to reduce model size
5. Test the models on other simple datasets (e.g., Fashion MNIST)

TROUBLESHOOTING TIPS

- If you encounter "Out of Memory" errors, try reducing the batch size
- If training is too slow, check if GPU is enabled in Colab (Runtime > Change runtime type)
- If preprocessing is taking too long, consider using a smaller subset of the data for testing
- If you get shape mismatch errors, check the input shapes required by each model

---

REFERENCES AND ADDITIONAL RESOURCES

1. MobileNetV2 Paper: arXiv:1801.04381
2. ResNet Paper: arXiv:1512.03385
3. VGG Paper: arXiv:1409.1556

4. TensorFlow Keras Applications: https://www.tensorflow.org/api_docs/python/tf/keras/applications
5. Transfer Learning Guide: https://www.tensorflow.org/tutorials/images/transfer_learning