```
import os
import tensorflow as tf
from tensorflow.keras.applications import VGG16, ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout, GlobalAveragePooling2D
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import time
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
from fpdf import FPDF
import pandas as pd

# Output directories
output_dir = "model_comparison_results"
os.makedirs(output_dir, exist_ok=True)
model_dir = os.path.join(output_dir, "models")
os.makedirs(model_dir, exist_ok=True)

# Configure GPU
print("TensorFlow version:", tf.__version__)
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        # Set memory growth for all GPUs
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)

        # Set visible devices to first GPU
        tf.config.set_visible_devices(gpus[0], 'GPU')
        logical_gpus = tf.config.list_logical_devices('GPU')
        print(f"✅ {len(gpus)} Physical GPUs, {len(logical_gpus)} Logical GPUs")
    except RuntimeError as e:
        print(e)
else:
    print("⚠ No GPU found. Running on CPU.")

# Enable Mixed Precision
tf.keras.mixed_precision.set_global_policy('mixed_float16')

# Define constants
IMG_SIZE = (128, 128)  # Smaller size for faster training
BATCH_SIZE = 64  # Adjust based on your GPU memory
EPOCHS = 10  # Set to 10 or 30 as needed
AUTOTUNE = tf.data.AUTOTUNE
CLASSES = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# Load CIFAR-10 Dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
print(f"Training data shape: {x_train.shape}, Test data shape: {x_test.shape}")

# Convert labels to categorical
y_train_cat = tf.keras.utils.to_categorical(y_train, 10)
y_test_cat = tf.keras.utils.to_categorical(y_test, 10)

# Data preprocessing function
def preprocess(image, label):
    image = tf.cast(image, tf.float32)
    image = tf.image.resize(image, IMG_SIZE)
    image = image / 255.0  # Normalize
    return image, label

# Create tf.data Pipeline
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train_cat))
train_dataset = train_dataset.map(preprocess, num_parallel_calls=AUTOTUNE)
train_dataset = train_dataset.batch(BATCH_SIZE).prefetch(AUTOTUNE)

test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test_cat))
test_dataset = test_dataset.map(preprocess, num_parallel_calls=AUTOTUNE)
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(AUTOTUNE)

# Data augmentation for training
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.1
)
```

```python
# Function to build model
def build_model(base_model, model_name):
    # Freeze base layers
    base_model.trainable = False

    inputs = base_model.input
    x = base_model(inputs, training=False)
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    outputs = Dense(10, activation='softmax', dtype='float32')(x)

    model = Model(inputs, outputs, name=model_name)

    # Compile with Adam optimizer
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
        loss='categorical_crossentropy',
        metrics=['accuracy',
                tf.keras.metrics.Precision(name='precision'),
                tf.keras.metrics.Recall(name='recall')]
    )

    return model

# Function to count model parameters
def count_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])
    return {
        'trainable': trainable_params,
        'non_trainable': non_trainable_params,
        'total': trainable_params + non_trainable_params
    }

# Function to measure inference time
def measure_inference_time(model, test_dataset, num_runs=50):
    # Warm up
    for images, _ in test_dataset.take(1):
        model.predict(images, verbose=0)

    # Measure inference time
    times = []
    for images, _ in test_dataset.take(num_runs):
        start_time = time.time()
        model.predict(images, verbose=0)
        end_time = time.time()
        times.append(end_time - start_time)

    return {
        'mean': np.mean(times),
        'std': np.std(times),
        'min': np.min(times),
        'max': np.max(times)
    }

# Function to train and evaluate model
def train_and_evaluate(model, model_name):
    print(f"\n{'='*50}\nTraining {model_name}...\n{'='*50}")

    # Callbacks
    early_stopping = EarlyStopping(
        monitor='val_accuracy',
        patience=5,
        restore_best_weights=True
    )

    checkpoint = ModelCheckpoint(
        filepath=os.path.join(model_dir, f"{model_name}_best.keras"),
        monitor='val_accuracy',
        save_best_only=True
    )

    reduce_lr = ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,
        patience=3,
        min_lr=1e-6
    )

    # Train model
    start_time = time.time()
    history = model.fit(
        train_dataset,
        validation_data=test_dataset,
        epochs=EPOCHS,
        callbacks=[early_stopping, checkpoint, reduce_lr],
        verbose=1
    )
    training_time = time.time() - start_time

    # Evaluate model
    print(f"\nEvaluating {model_name}...")
    test_results = model.evaluate(test_dataset, verbose=1)

    # Create metrics dictionary with proper handling of metric names
    metrics_names = model.metrics_names
    test_metrics = {}

    # Print the actual metric names for debugging
    print(f"Available metrics: {metrics_names}")

    # Map metrics to standardized names
    for i, name in enumerate(metrics_names):
        # Handle different naming conventions in TensorFlow versions
        if 'accuracy' in name or name == 'acc':
            test_metrics['accuracy'] = test_results[i]
        elif 'precision' in name:
            test_metrics['precision'] = test_results[i]
        elif 'recall' in name:
            test_metrics['recall'] = test_results[i]
        elif 'loss' in name:
            test_metrics['loss'] = test_results[i]
        else:
            # Store with original name as fallback
            test_metrics[name] = test_results[i]

    # Ensure all required metrics exist
    if 'accuracy' not in test_metrics:
        print("Warning: 'accuracy' metric not found. Using first non-loss metric.")
        for i, name in enumerate(metrics_names):
            if 'loss' not in name:
                test_metrics['accuracy'] = test_results[i]
                break

    # Get predictions for confusion matrix
    y_pred = []
    y_true = []

    for images, labels in test_dataset:
        predictions = model.predict(images, verbose=0)
        y_pred.extend(np.argmax(predictions, axis=1))
        y_true.extend(np.argmax(labels.numpy(), axis=1))

    # Calculate confusion matrix
    cm = confusion_matrix(y_true, y_pred)

    # Get classification report
    report = classification_report(y_true, y_pred, target_names=CLASSES, output_dict=True)

    # Count parameters
    params = count_parameters(model)

    # Measure inference time
    inference_times = measure_inference_time(model, test_dataset)

    return {
        'history': history,
        'training_time': training_time,
        'test_metrics': test_metrics,
        'confusion_matrix': cm,
        'classification_report': report,
        'parameters': params,
        'inference_times': inference_times
    }

# Load base models
print("Loading VGG16 model...")
vgg16_base = VGG16(weights='imagenet', include_top=False, input_shape=(*IMG_SIZE, 3))
print("Loading ResNet50 model...")
resnet50_base = ResNet50(weights='imagenet', include_top=False, input_shape=(*IMG_SIZE, 3))

# Build models
vgg16_model = build_model(vgg16_base, "VGG16")
resnet50_model = build_model(resnet50_base, "ResNet50")
```

```python
resnet50_model = build_model(resnet50_base, "ResNet50")

# Print model summaries
vgg16_model.summary()
resnet50_model.summary()

# Train and evaluate models
vgg16_results = train_and_evaluate(vgg16_model, "VGG16")
resnet50_results = train_and_evaluate(resnet50_model, "ResNet50")

# Save models
vgg16_model.save(os.path.join(model_dir, "vgg16_final.keras"))
resnet50_model.save(os.path.join(model_dir, "resnet50_final.keras"))

# Helper function to safely get metric value
def get_metric_value(results, metric_name, default=0.0):
    """Safely get a metric value from results dictionary"""
    if metric_name in results['test_metrics']:
        return results['test_metrics'][metric_name]
    print(f"Warning: Metric '{metric_name}' not found. Using default value.")
    return default

# Create visualizations
def plot_training_history(vgg16_history, resnet50_history):
    fig, axes = plt.subplots(2, 2, figsize=(15, 10))

    # Accuracy
    axes[0, 0].plot(vgg16_history.history['accuracy'], label='VGG16 Train', marker='o')
    axes[0, 0].plot(vgg16_history.history['val_accuracy'], label='VGG16 Val', marker='o')
    axes[0, 0].plot(resnet50_history.history['accuracy'], label='ResNet50 Train', marker='s')
    axes[0, 0].plot(resnet50_history.history['val_accuracy'], label='ResNet50 Val', marker='s')
    axes[0, 0].set_title('Accuracy')
    axes[0, 0].set_xlabel('Epoch')
    axes[0, 0].set_ylabel('Accuracy')
    axes[0, 0].legend()
    axes[0, 0].grid(True)

    # Loss
    axes[0, 1].plot(vgg16_history.history['loss'], label='VGG16 Train', marker='o')
    axes[0, 1].plot(vgg16_history.history['val_loss'], label='VGG16 Val', marker='o')
    axes[0, 1].plot(resnet50_history.history['loss'], label='ResNet50 Train', marker='s')
    axes[0, 1].plot(resnet50_history.history['val_loss'], label='ResNet50 Val', marker='s')
    axes[0, 1].set_title('Loss')
    axes[0, 1].set_xlabel('Epoch')
    axes[0, 1].set_ylabel('Loss')
    axes[0, 1].legend()
    axes[0, 1].grid(True)

    # Check if precision and recall are in the history
    if 'precision' in vgg16_history.history and 'precision' in resnet50_history.history:
        axes[1, 0].plot(vgg16_history.history['precision'], label='VGG16 Train', marker='o')
        axes[1, 0].plot(vgg16_history.history['val_precision'], label='VGG16 Val', marker='o')
        axes[1, 0].plot(resnet50_history.history['precision'], label='ResNet50 Train', marker='s')
        axes[1, 0].plot(resnet50_history.history['val_precision'], label='ResNet50 Val', marker='s')
        axes[1, 0].set_title('Precision')
        axes[1, 0].set_xlabel('Epoch')
        axes[1, 0].set_ylabel('Precision')
        axes[1, 0].legend()
        axes[1, 0].grid(True)

    if 'recall' in vgg16_history.history and 'recall' in resnet50_history.history:
        axes[1, 1].plot(vgg16_history.history['recall'], label='VGG16 Train', marker='o')
        axes[1, 1].plot(vgg16_history.history['val_recall'], label='VGG16 Val', marker='o')
        axes[1, 1].plot(resnet50_history.history['recall'], label='ResNet50 Train', marker='s')
        axes[1, 1].plot(resnet50_history.history['val_recall'], label='ResNet50 Val', marker='s')
        axes[1, 1].set_title('Recall')
        axes[1, 1].set_xlabel('Epoch')
        axes[1, 1].set_ylabel('Recall')
        axes[1, 1].legend()
        axes[1, 1].grid(True)

    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'training_history.png'), dpi=300)
    plt.close()

def plot_confusion_matrices(vgg16_cm, resnet50_cm):
    fig, axes = plt.subplots(1, 2, figsize=(20, 8))

    # VGG16 confusion matrix
    sns.heatmap(vgg16_cm, annot=True, fmt='d', cmap='Blues', xticklabels=CLASSES, yticklabels=CLASSES, ax=axes[0])
    axes[0].set_title('VGG16 Confusion Matrix')
    axes[0].set_xlabel('Predicted')
    axes[0].set_ylabel('True')
    # ResNet50 confusion matrix
    sns.heatmap(resnet50_cm, annot=True, fmt='d', cmap='Blues', xticklabels=CLASSES, yticklabels=CLASSES, ax=axes[1])
    axes[1].set_title('ResNet50 Confusion Matrix')
    axes[1].set_xlabel('Predicted')
    axes[1].set_ylabel('True')

    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'confusion_matrices.png'), dpi=300)
    plt.close()

def plot_performance_comparison():
    # Metrics for comparison - Using the safe getter function
    metrics = {
        'Accuracy': [
            get_metric_value(vgg16_results, 'accuracy'),
            get_metric_value(resnet50_results, 'accuracy')
        ],
        'Precision': [
            get_metric_value(vgg16_results, 'precision'),
            get_metric_value(resnet50_results, 'precision')
        ],
        'Recall': [
            get_metric_value(vgg16_results, 'recall'),
            get_metric_value(resnet50_results, 'recall')
        ],
        'Training Time (s)': [
            vgg16_results['training_time'],
            resnet50_results['training_time']
        ],
        'Inference Time (ms)': [
            vgg16_results['inference_times']['mean'] * 1000,
            resnet50_results['inference_times']['mean'] * 1000
        ]
    }

    # Create figure
    fig, axes = plt.subplots(1, 2, figsize=(15, 6))

    # Bar chart
    models = ['VGG16', 'ResNet50']
    x = np.arange(len(models))
    width = 0.15
    multiplier = 0

    for metric, values in metrics.items():
        offset = width * multiplier
        axes[0].bar(x + offset, values, width, label=metric)
        multiplier += 1

    axes[0].set_xticks(x + width * 2)
    axes[0].set_xticklabels(models)
    axes[0].set_title('Performance Metrics Comparison')
    axes[0].legend(loc='upper left', bbox_to_anchor=(1, 1))

    # Parameters comparison
    param_data = {
        'Trainable': [vgg16_results['parameters']['trainable'] / 1e6, resnet50_results['parameters']['trainable'] / 1e6],
        'Non-Trainable': [vgg16_results['parameters']['non_trainable'] / 1e6, resnet50_results['parameters']['non_trainable'] / 1e6]
    }

    bottom = np.zeros(2)
    for param, values in param_data.items():
        axes[1].bar(models, values, label=param, bottom=bottom)
        bottom += values

    axes[1].set_title('Model Parameters (Millions)')
    axes[1].legend()

    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'performance_comparison.png'), dpi=300)
    plt.close()

def plot_class_performance():
    # Get per-class metrics
    vgg16_report = vgg16_results['classification_report']
    resnet50_report = resnet50_results['classification_report']

    # Create dataframes
    vgg16_df = pd.DataFrame({cls: [vgg16_report[cls]['precision'], vgg16_report[cls]['recall'], vgg16_report[cls]['f1-score']]
                            for cls in CLASSES}, index=['Precision', 'Recall', 'F1-Score'])

    resnet50_df = pd.DataFrame({cls: [resnet50_report[cls]['precision'], resnet50_report[cls]['recall'], resnet50_report[cls]['f1-score']
                                for cls in CLASSES}, index=['Precision', 'Recall', 'F1-Score'])

    # Plot
    fig, axes = plt.subplots(2, 1, figsize=(15, 12))

    vgg16_df.T.plot(kind='bar', ax=axes[0])
    axes[0].set_title('VGG16 Per-Class Performance')
    axes[0].set_xlabel('Class')
    axes[0].set_ylabel('Score')
    axes[0].legend()
    axes[0].grid(axis='y')

    resnet50_df.T.plot(kind='bar', ax=axes[1])
    axes[1].set_title('ResNet50 Per-Class Performance')
    axes[1].set_xlabel('Class')
    axes[1].set_ylabel('Score')
    axes[1].legend()
    axes[1].grid(axis='y')

    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'class_performance.png'), dpi=300)
    plt.close()

# Generate all plots
print("Generating visualizations...")
plot_training_history(vgg16_results['history'], resnet50_results['history'])
plot_confusion_matrices(vgg16_results['confusion_matrix'], resnet50_results['confusion_matrix'])
plot_performance_comparison()
plot_class_performance()

# Create PDF report
def create_pdf_report():
    pdf = FPDF()
    pdf.add_page()

    # Title
    pdf.set_font("Arial", 'B', 16)
    pdf.cell(200, 10, "VGG16 vs ResNet50 Comparison on CIFAR-10", ln=True, align="C")
    pdf.ln(5)

    # Summary
    pdf.set_font("Arial", 'B', 12)
    pdf.cell(200, 10, "Performance Summary", ln=True)
    pdf.set_font("Arial", '', 10)

    # Create summary table with safe metric access
    summary_data = [
        ["Metric", "VGG16", "ResNet50"],
        ["Accuracy", f"{get_metric_value(vgg16_results, 'accuracy'):.4f}", f"{get_metric_value(resnet50_results, 'accuracy'):.4f}"],
        ["Precision", f"{get_metric_value(vgg16_results, 'precision'):.4f}", f"{get_metric_value(resnet50_results, 'precision'):.4f}"],
        ["Recall", f"{get_metric_value(vgg16_results, 'recall'):.4f}", f"{get_metric_value(resnet50_results, 'recall'):.4f}"],
        ["Training Time", f"{vgg16_results['training_time']:.2f} sec", f"{resnet50_results['training_time']:.2f} sec"],
        ["Inference Time", f"{vgg16_results['inference_times']['mean']*1000:.2f} ms", f"{resnet50_results['inference_times']['mean']*1000
        ["Total Parameters", f"{vgg16_results['parameters']['total']/1e6:.2f}M", f"{resnet50_results['parameters']['total']/1e6:.2f}M"],
        ["Trainable Parameters", f"{vgg16_results['parameters']['trainable']/1e6:.2f}M", f"{resnet50_results['parameters']['trainable']/1
    ]

    # Add summary table
    col_width = 60
    row_height = 7
    for row in summary_data:
        for item in row:
            pdf.cell(col_width, row_height, str(item), border=1)
        pdf.ln(row_height)

    pdf.ln(10)

    # Add training history plot
    pdf.set_font("Arial", 'B', 12)
    pdf.cell(200, 10, "Training History", ln=True)
    pdf.image(os.path.join(output_dir, 'training_history.png'), x=10, y=pdf.get_y(), w=180)
    pdf.ln(120)  # Adjust based on image height

    # Add new page
    pdf.add_page()

    # Add confusion matrices
    pdf.set_font("Arial", 'B', 12)
    pdf.cell(200, 10, "Confusion Matrices", ln=True)
    pdf.image(os.path.join(output_dir, 'confusion_matrices.png'), x=10, y=pdf.get_y(), w=180)
    pdf.ln(100)  # Adjust based on image height

    # Add performance comparison
    # Add performance comparison
    pdf.set_font("Arial", 'B', 12)
    pdf.cell(200, 10, "Performance Comparison", ln=True)
    pdf.image(os.path.join(output_dir, 'performance_comparison.png'), x=10, y=pdf.get_y(), w=180)
    pdf.ln(80)  # Adjust based on image height

    # Add new page
    pdf.add_page()

    # Add class performance
    pdf.set_font("Arial", 'B', 12)
    pdf.cell(200, 10, "Per-Class Performance", ln=True)
    pdf.image(os.path.join(output_dir, 'class_performance.png'), x=10, y=pdf.get_y(), w=180)

    # Add conclusion
    pdf.add_page()
    pdf.set_font("Arial", 'B', 12)
    pdf.cell(200, 10, "Conclusion", ln=True)
    pdf.set_font("Arial", '', 10)

    # Determine which model performed better
    vgg16_acc = get_metric_value(vgg16_results, 'accuracy')
    resnet50_acc = get_metric_value(resnet50_results, 'accuracy')

    if vgg16_acc > resnet50_acc:
        better_model = "VGG16"
        accuracy_diff = vgg16_acc - resnet50_acc
    else:
        better_model = "ResNet50"
        accuracy_diff = resnet50_acc - vgg16_acc

    conclusion_text = f"""
    Based on the experiments conducted, {better_model} achieved better accuracy with a difference of {accuracy_diff:.4f}.

    VGG16 Summary:
    - Accuracy: {get_metric_value(vgg16_results, 'accuracy'):.4f}
    - Training Time: {vgg16_results['training_time']:.2f} seconds
    - Inference Time: {vgg16_results['inference_times']['mean']*1000:.2f} ms
    - Total Parameters: {vgg16_results['parameters']['total']/1e6:.2f} million

    ResNet50 Summary:
    - Accuracy: {get_metric_value(resnet50_results, 'accuracy'):.4f}
    - Training Time: {resnet50_results['training_time']:.2f} seconds
    - Inference Time: {resnet50_results['inference_times']['mean']*1000:.2f} ms
    - Total Parameters: {resnet50_results['parameters']['total']/1e6:.2f} million

    Key Observations:
    1. {better_model} showed better overall performance on the CIFAR-10 dataset.
    2. ResNet50 has more parameters but may converge faster due to its skip connections.
    3. VGG16 has a simpler architecture but may require more training time to achieve comparable results.

    The models were trained for {EPOCHS} epochs with a batch size of {BATCH_SIZE} on images resized to {IMG_SIZE}.
    """

    # Add multiline text
    pdf.multi_cell(0, 5, conclusion_text)

    # Save PDF
    pdf_path = os.path.join(output_dir, 'vgg16_resnet50_comparison.pdf')
    pdf.output(pdf_path)
    return pdf_path

# Create PDF report
pdf_path = create_pdf_report()

print(f"\nResults saved in {output_dir}:")
print(f" - PDF Report: {pdf_path}")
print(f" - Models: {model_dir}")
print(f" - Visualizations: {output_dir}/*.png")
```

```
TensorFlow version: 2.18.0
✅ 1 Physical GPUs, 1 Logical GPUs
Training data shape: (50000, 32, 32, 3), Test data shape: (10000, 32, 32, 3)
Loading VGG16 model...
Loading ResNet50 model...
Model: "VGG16"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ input_layer_4 (InputLayer)           │ (None, 128, 128, 3)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ vgg16 (Functional)                   │ (None, 4, 4, 512)           │      14,714,688 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ global_average_pooling2d_4           │ (None, 512)                 │               0 │
│ (GlobalAveragePooling2D)             │                             │                 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_8 (Dense)                      │ (None, 512)                 │         262,656 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_4 (Dropout)                  │ (None, 512)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ cast_10 (Cast)                       │ (None, 512)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_9 (Dense)                      │ (None, 10)                  │           5,130 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 14,982,474 (57.15 MB)
 Trainable params: 267,786 (1.02 MB)
 Non-trainable params: 14,714,688 (56.13 MB)
Model: "ResNet50"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ input_layer_5 (InputLayer)           │ (None, 128, 128, 3)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ resnet50 (Functional)                │ (None, 4, 4, 2048)          │      23,587,712 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ global_average_pooling2d_5           │ (None, 2048)                │               0 │
│ (GlobalAveragePooling2D)             │                             │                 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_10 (Dense)                     │ (None, 512)                 │       1,049,088 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_5 (Dropout)                  │ (None, 512)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ cast_11 (Cast)                       │ (None, 512)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_11 (Dense)                     │ (None, 10)                  │           5,130 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 24,641,930 (94.00 MB)
 Trainable params: 1,054,218 (4.02 MB)
 Non-trainable params: 23,587,712 (89.98 MB)

==================================================
Training VGG16...
==================================================
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 46ms/step - accuracy: 0.4432 - loss: 1.5894 - precision: 0.7633 - recall: 0.1805 - val_accuracy:
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 39ms/step - accuracy: 0.6226 - loss: 1.0890 - precision: 0.7714 - recall: 0.4559 - val_accuracy:
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 42s 40ms/step - accuracy: 0.6489 - loss: 1.0084 - precision: 0.7764 - recall: 0.5074 - val_accuracy:
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 40ms/step - accuracy: 0.6654 - loss: 0.9610 - precision: 0.7880 - recall: 0.5374 - val_accuracy:
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 31s 40ms/step - accuracy: 0.6756 - loss: 0.9338 - precision: 0.7910 - recall: 0.5528 - val_accuracy:
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 31s 39ms/step - accuracy: 0.6812 - loss: 0.9114 - precision: 0.7956 - recall: 0.5633 - val_accuracy:
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 31s 39ms/step - accuracy: 0.6873 - loss: 0.8918 - precision: 0.7961 - recall: 0.5781 - val_accuracy:
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 40ms/step - accuracy: 0.6925 - loss: 0.8820 - precision: 0.7972 - recall: 0.5867 - val_accuracy:
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 31s 39ms/step - accuracy: 0.7013 - loss: 0.8646 - precision: 0.8031 - recall: 0.5954 - val_accuracy:
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 39ms/step - accuracy: 0.7019 - loss: 0.8519 - precision: 0.8047 - recall: 0.6015 - val_accuracy:

Evaluating VGG16...
157/157 ━━━━━━━━━━━━━━━━━━━━ 5s 32ms/step - accuracy: 0.7052 - loss: 0.8650 - precision: 0.8085 - recall: 0.5907
Available metrics: ['loss', 'compile_metrics']
Warning: 'accuracy' metric not found. Using first non-loss metric.

==================================================
Training ResNet50...
==================================================
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 47s 44ms/step - accuracy: 0.1407 - loss: 2.2860 - precision: 0.1806 - recall: 4.2086e-05 - val_accur
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 25ms/step - accuracy: 0.2303 - loss: 2.0578 - precision: 0.4889 - recall: 0.0013 - val_accuracy:
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.2548 - loss: 1.9903 - precision: 0.5518 - recall: 0.0059 - val_accuracy:
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.2725 - loss: 1.9494 - precision: 0.5298 - recall: 0.0091 - val_accuracy:
Epoch 5/10
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 29ms/step - accuracy: 0.2753 - loss: 1.9254 - precision: 0.5672 - recall: 0.0137 - val_accuracy:
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 29ms/step - accuracy: 0.2870 - loss: 1.9054 - precision: 0.5807 - recall: 0.0175 - val_accuracy:
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.2973 - loss: 1.8848 - precision: 0.5771 - recall: 0.0211 - val_accuracy:
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 40s 28ms/step - accuracy: 0.3045 - loss: 1.8756 - precision: 0.5786 - recall: 0.0228 - val_accuracy:
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.3064 - loss: 1.8645 - precision: 0.5987 - recall: 0.0258 - val_accuracy:
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 26ms/step - accuracy: 0.3060 - loss: 1.8594 - precision: 0.6085 - recall: 0.0278 - val_accuracy:

Evaluating ResNet50...
157/157 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.3444 - loss: 1.7836 - precision: 0.7049 - recall: 0.0174
Available metrics: ['loss', 'compile_metrics']
Warning: 'accuracy' metric not found. Using first non-loss metric.
Generating visualizations...
Warning: Metric 'precision' not found. Using default value.
Warning: Metric 'precision' not found. Using default value.
Warning: Metric 'recall' not found. Using default value.
Warning: Metric 'recall' not found. Using default value.
Warning: Metric 'precision' not found. Using default value.
Warning: Metric 'precision' not found. Using default value.
Warning: Metric 'recall' not found. Using default value.
Warning: Metric 'recall' not found. Using default value.

Results saved in model_comparison_results:
 - PDF Report: model_comparison_results/vgg16_resnet50_comparison.pdf
```

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

```
Mounted at /content/drive
```