Connected to Data370_25au (Python 3.12.11)

In [ ]:
```
# %pip install openimages
#
```

In [ ]:
```
# Download Open Images subset
# import os
# from dataset_utils import download_openimages_subset, prev

# output = download_openimages_subset(
#     classes=["Dog", "Cat", "Car", "Tree", "Bicycle"],
#     max_samples=1000,
#     export_dir=r"C:\Users\glenm\Downloads\_git\DATA37000\d
# )

# preview_imagefolder(output)
# print("Download Done.")
```

In [ ]:
```
# Exploratory Data Analysis
import os
import matplotlib.pyplot as plt
from collections import Counter
from PIL import Image
import seaborn as sns

# Path to downloaded dataset
DATA_DIR = r"C:\Users\glenm\Downloads\_git\DATA37000\data\bi

# Helper: get class folders
classes = [d for d in os.listdir(DATA_DIR) if os.path.isdir(
print("Classes found:", classes)

# 1. Show examples of images
def show_examples_per_class(classes, n=3):
    fig, axes = plt.subplots(len(classes), n, figsize=(n*3,
    for i, cls in enumerate(classes):
        cls_dir = os.path.join(DATA_DIR, cls)
        images = [f for f in os.listdir(cls_dir) if f.lower(
        for j in range(n):
            img_path = os.path.join(cls_dir, images[j])
```

```
            img = Image.open(img_path)
            axes[i, j].imshow(img)
            axes[i, j].axis("off")
            if j == 0:
                axes[i, j].set_title(cls)
    plt.tight_layout()
    plt.show()

show_examples_per_class(classes, n=3)

# 2. Class distribution
counts = {cls: len(os.listdir(os.path.join(DATA_DIR, cls)))
print("Class counts:", counts)

plt.figure(figsize=(8,5))
sns.barplot(x=list(counts.keys()), y=list(counts.values()))
plt.title("Class Distribution")
plt.ylabel("Number of Images")
plt.show()

# 3. Image sizes & quality
sizes = []
for cls in classes:
    cls_dir = os.path.join(DATA_DIR, cls)
    images = [f for f in os.listdir(cls_dir) if f.lower().er
    for f in images[:50]:  # sample first 50 per class for s
        img_path = os.path.join(cls_dir, f)
        try:
            with Image.open(img_path) as img:
                sizes.append(img.size)  # (width, height)
        except Exception as e:
            print(f"Error opening {img_path}: {e}")

# Convert to width/height lists
widths = [w for w,h in sizes]
heights = [h for w,h in sizes]

plt.figure(figsize=(6,4))
sns.scatterplot(x=widths, y=heights)
plt.xlabel("Width")
plt.ylabel("Height")
plt.title("Image Size Distribution (sampled)")
plt.show()
```

```
print(f"Average width: {sum(widths)/len(widths):.1f}, Averag
print(f"Min size: {min(widths)}x{min(heights)}, Max size: {m
```

```
Classes found: ['Bicycle', 'Car', 'Cat', 'Dog', 'Tree']
```

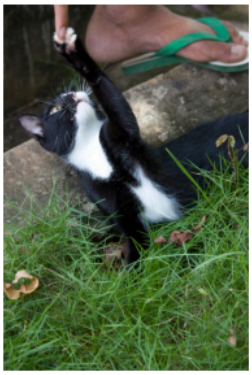
Bicycle



Car

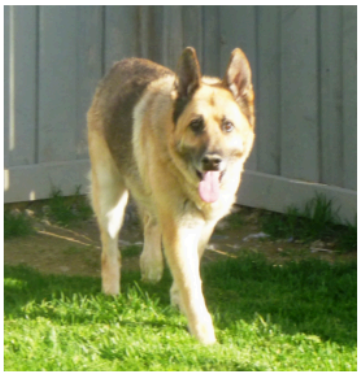

Cat





Dog

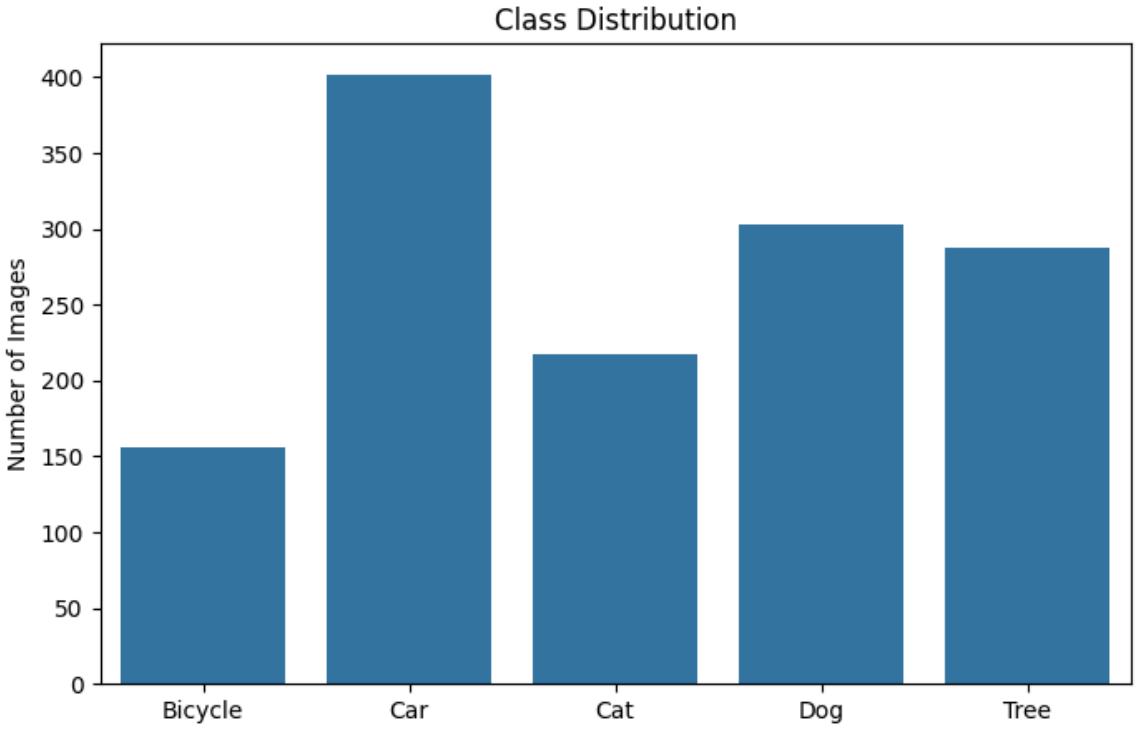


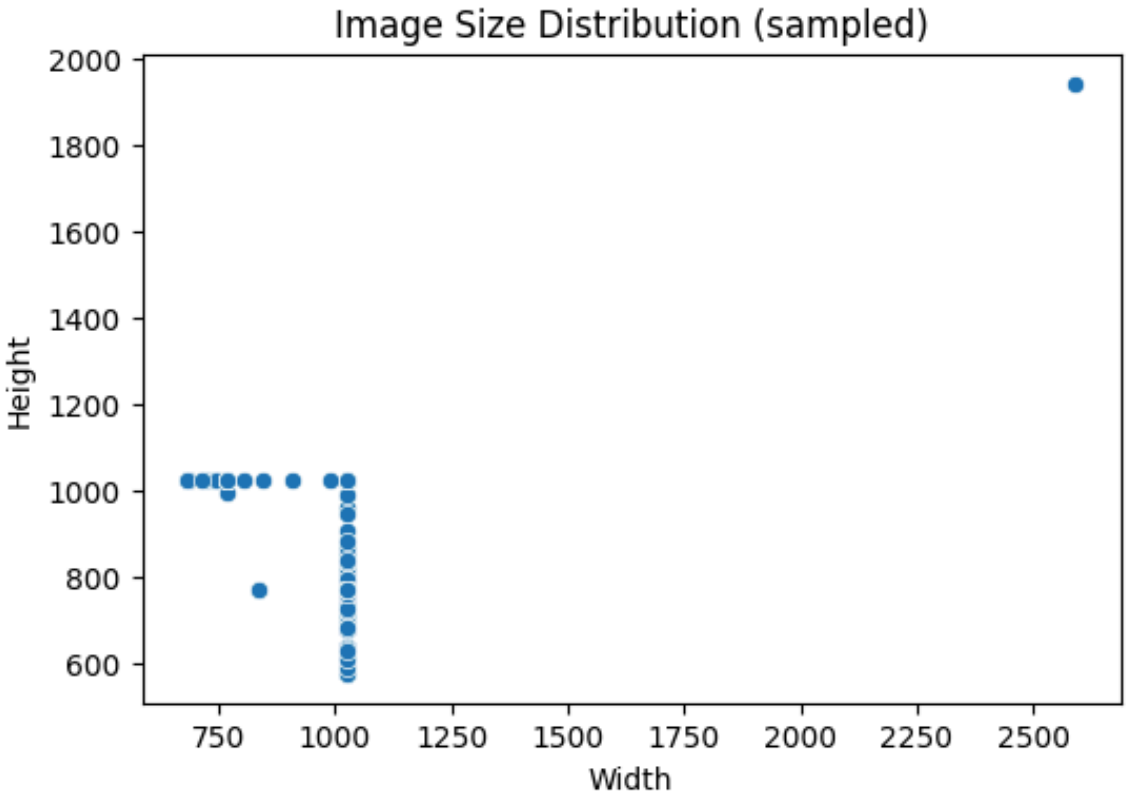


Tree



```
Class counts: {'Bicycle': 156, 'Car': 402, 'Cat': 217, 'Dog':
303, 'Tree': 288}
```






```
Average width: 991.8, Average height: 786.6
Min size: 680x576, Max size: 2592x1944
```

In [ ]:
```
import torch
import torch.nn as nn
```

```
import torch.nn.functional as F
from torchvision import transforms, datasets, models
from torch.utils.data import DataLoader, random_split
from sklearn.metrics import confusion_matrix, ConfusionMatri
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np

# ----------------------------
# Transforms
# ----------------------------
imagenet_mean = [0.485, 0.456, 0.406]
imagenet_std = [0.229, 0.224, 0.225]

train_transforms = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=imagenet_mean, std=imagenet_st
])

val_transforms = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=imagenet_mean, std=imagenet_st
])

# ----------------------------
# Dataset + Loaders
# ----------------------------
DATA_DIR = r"C:\Users\glenm\Downloads\_git\DATA37000\data\bi

full_dataset = datasets.ImageFolder(DATA_DIR, transform=trai
train_size = int(0.8 * len(full_dataset))
val_size = len(full_dataset) - train_size
train_dataset, val_dataset = random_split(full_dataset, [tra
val_dataset.dataset.transform = val_transforms

train_loader = DataLoader(train_dataset, batch_size=32, shuf
val_loader = DataLoader(val_dataset, batch_size=32)

class_names = val_dataset.dataset.classes
num_classes = len(class_names)
```

```
# ----------------------------
# Baseline CNN
# ----------------------------
class BaselineCNN(nn.Module):
    def __init__(self, num_classes=5):
        super(BaselineCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, 1, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1, 1)
        self.conv3 = nn.Conv2d(64, 128, 3, 1, 1)
        self.pool = nn.MaxPool2d(2,2)
        self.fc1 = nn.Linear(128*28*28, 256)
        self.fc2 = nn.Linear(256, num_classes)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 128*28*28)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

# ----------------------------
# ResNet18 Transfer Learning
# ----------------------------
def build_resnet18(num_classes=5):
    model = models.resnet18(weights=models.ResNet18_Weights.
    for param in model.parameters():
        param.requires_grad = False
    model.fc = nn.Sequential(
        nn.Linear(model.fc.in_features, 256),
        nn.ReLU(),
        nn.Dropout(0.3),
        nn.Linear(256, num_classes)
    )
    return model

# ----------------------------
# Training function
# ----------------------------
```

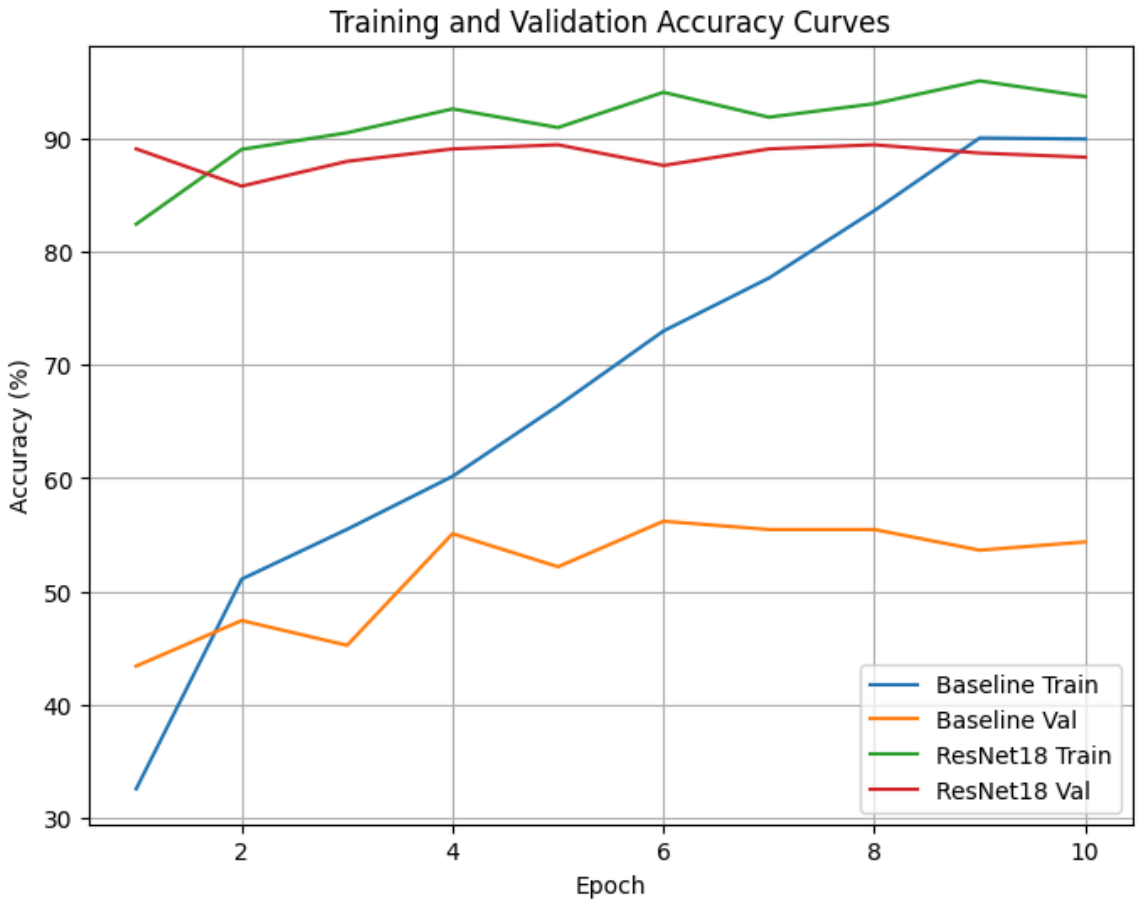
```
def train_model(model, train_loader, val_loader, num_epochs=
    device = torch.device("cuda" if torch.cuda.is_available(
    model = model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.AdamW(model.parameters(), lr=lr,

    best_val_acc, counter = 0, 0
    train_accs, val_accs = [], []

    for epoch in range(num_epochs):
        model.train()
        correct, total, running_loss = 0, 0, 0.0
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(de
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        train_acc = 100 * correct / total
        train_accs.append(train_acc)

        # Validation
        model.eval()
        correct, total = 0, 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.t
                outputs = model(images)
                _, predicted = torch.max(outputs, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item(
        val_acc = 100 * correct / total
        val_accs.append(val_acc)

        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {runni

        # Early stopping
        if val_acc > best_val_acc:
```

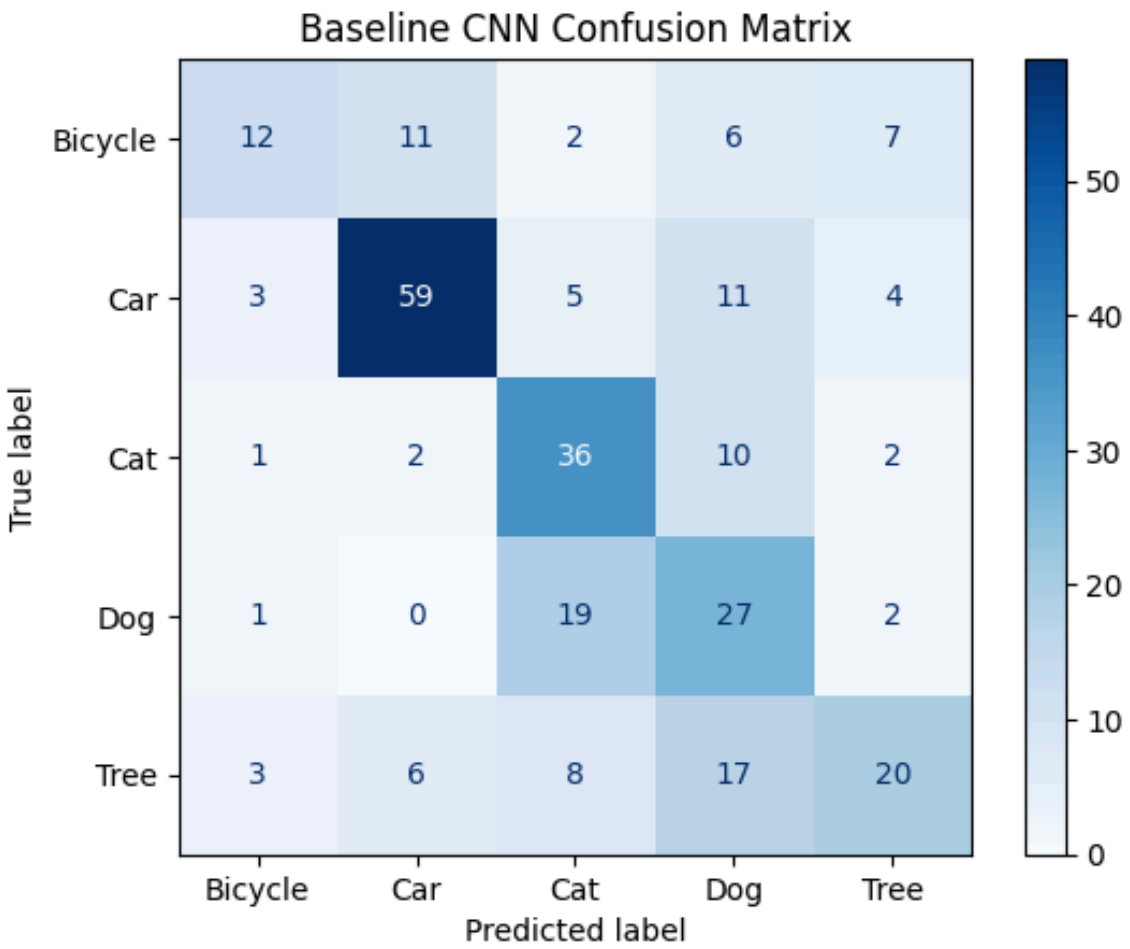
```
            best_val_acc = val_acc
            counter = 0
            torch.save(model.state_dict(), save_path)
        else:
            counter += 1
            if counter >= patience:
                print("Early stopping triggered")
                break

    return train_accs, val_accs

# ----------------------------
# Evaluation: Confusion Matrix + ROC/AUC
# ----------------------------
def evaluate_model(model, val_loader, model_name="Model"):
    device = torch.device("cuda" if torch.cuda.is_available(
    model = model.to(device)
    model.eval()

    y_true, y_pred, y_scores = [], [], []

    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(de
            outputs = model(images)
            probs = F.softmax(outputs, dim=1)
            _, predicted = torch.max(probs, 1)

            y_true.extend(labels.cpu().numpy())
            y_pred.extend(predicted.cpu().numpy())
            y_scores.extend(probs.cpu().numpy())

    # Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, displ
    disp.plot(cmap="Blues")
    plt.title(f"{model_name} Confusion Matrix")
    plt.show()

    # Classification Report
    print(f"\n{model_name} Classification Report:")
    print(classification_report(y_true, y_pred, target_names
```

```
    # ROC/AUC (One-vs-Rest)
    y_true_bin = label_binarize(y_true, classes=list(range(r
    fpr, tpr, roc_auc = {}, {}, {}

    for i in range(num_classes):
        fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], np.a
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure()
    for i in range(num_classes):
        plt.plot(fpr[i], tpr[i], label=f"{class_names[i]} (A
    plt.plot([0, 1], [0, 1], "k--")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"{model_name} ROC Curves")
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.show()

# ----------------------------
# Train and Evaluate Both Models
# ----------------------------
baseline_model = BaselineCNN(num_classes=num_classes)
baseline_train_accs, baseline_val_accs = train_model(baselir
                                                     num_epc

resnet_model = build_resnet18(num_classes=num_classes)
resnet_train_accs, resnet_val_accs = train_model(resnet_mode
                                                 num_epochs=

# Plot training/validation curves
plt.figure(figsize=(8,6))
plt.plot(range(1, len(baseline_train_accs)+1), baseline_trai
plt.plot(range(1, len(baseline_val_accs)+1), baseline_val_ac
plt.plot(range(1, len(resnet_train_accs)+1), resnet_train_ac
plt.plot(range(1, len(resnet_val_accs)+1), resnet_val_accs,
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.title("Training and Validation Accuracy Curves")
plt.legend()
plt.grid(True)
plt.show()
```

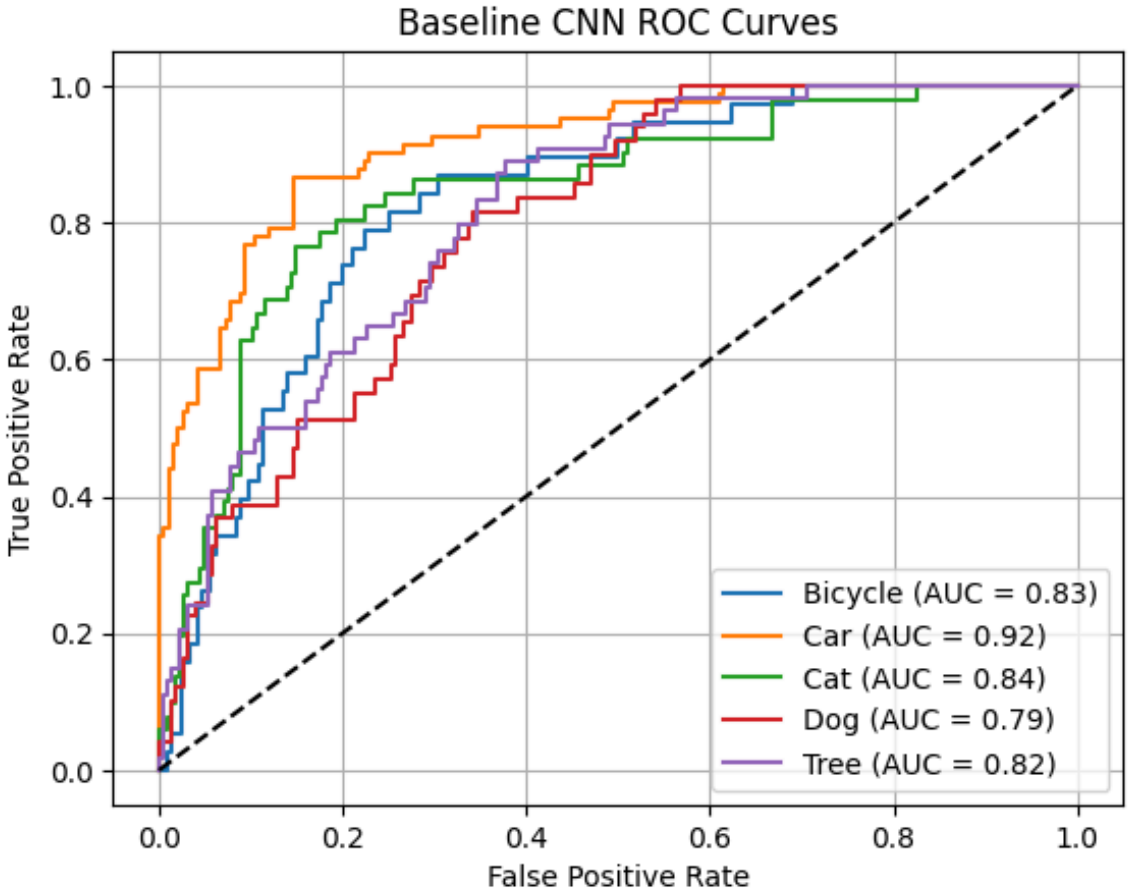
```
# Reload best weights before evaluation
baseline_model.load_state_dict(torch.load("baseline_best.pth
evaluate_model(baseline_model, val_loader, model_name="Basel

resnet_model.load_state_dict(torch.load("resnet_best.pth", w
evaluate_model(resnet_model, val_loader, model_name="ResNet1
```

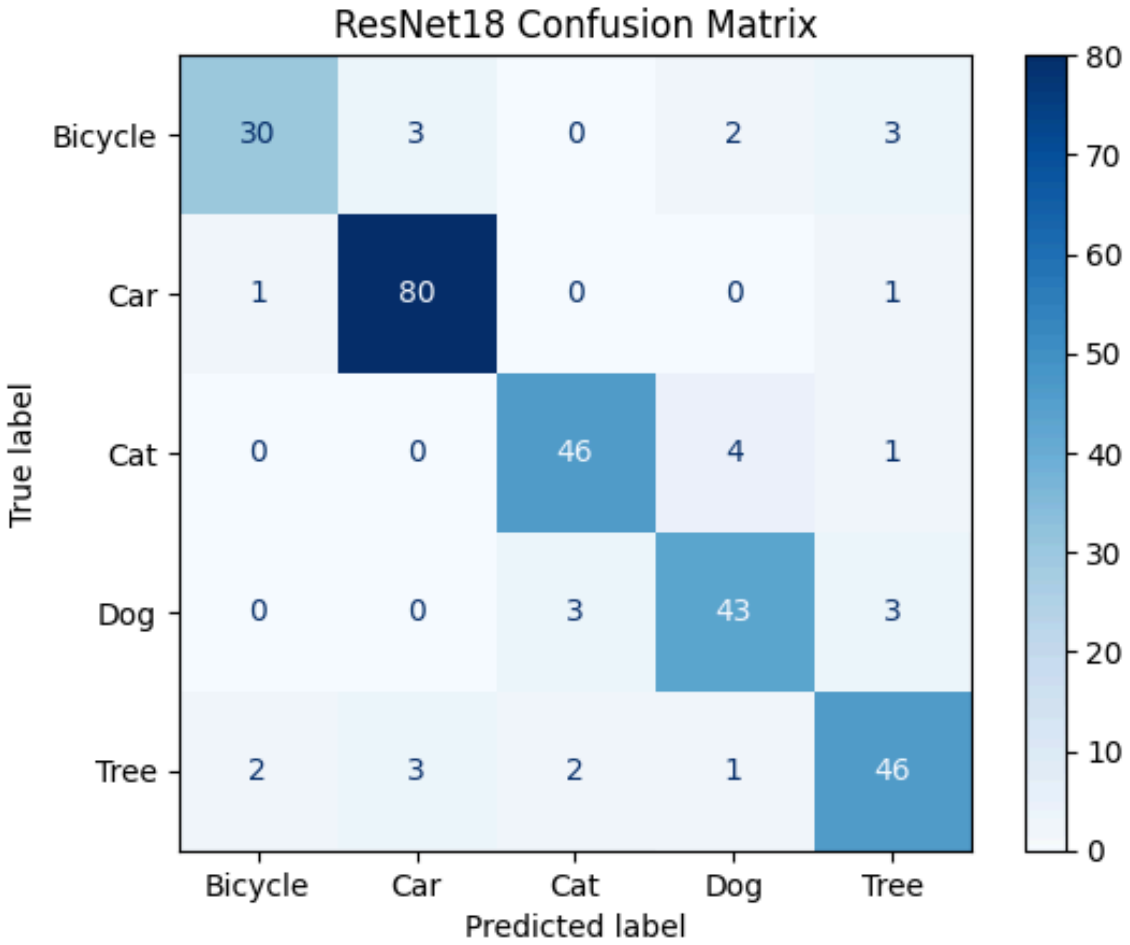
```
Epoch [1/10], Loss: 60.9634, Train Acc: 32.60%, Val Acc: 43.4
3%
Epoch [2/10], Loss: 42.9659, Train Acc: 51.10%, Val Acc: 47.4
5%
Epoch [3/10], Loss: 38.0697, Train Acc: 55.49%, Val Acc: 45.2
6%
Epoch [4/10], Loss: 33.5410, Train Acc: 60.16%, Val Acc: 55.1
1%
Epoch [5/10], Loss: 28.7574, Train Acc: 66.39%, Val Acc: 52.1
9%
Epoch [6/10], Loss: 24.5109, Train Acc: 72.99%, Val Acc: 56.2
0%
Epoch [7/10], Loss: 20.4764, Train Acc: 77.66%, Val Acc: 55.4
7%
Epoch [8/10], Loss: 14.7558, Train Acc: 83.61%, Val Acc: 55.4
7%
Epoch [9/10], Loss: 10.5049, Train Acc: 90.02%, Val Acc: 53.6
5%
Epoch [10/10], Loss: 9.3753, Train Acc: 89.93%, Val Acc: 54.3
8%
Epoch [1/15], Loss: 20.8699, Train Acc: 82.42%, Val Acc: 89.0
5%
Epoch [2/15], Loss: 12.5680, Train Acc: 89.01%, Val Acc: 85.7
7%
Epoch [3/15], Loss: 10.5588, Train Acc: 90.48%, Val Acc: 87.9
6%
Epoch [4/15], Loss: 8.7057, Train Acc: 92.58%, Val Acc: 89.0
5%
Epoch [5/15], Loss: 9.0611, Train Acc: 90.93%, Val Acc: 89.4
2%
Epoch [6/15], Loss: 6.8172, Train Acc: 94.05%, Val Acc: 87.5
9%
Epoch [7/15], Loss: 8.2593, Train Acc: 91.85%, Val Acc: 89.0
5%
Epoch [8/15], Loss: 7.0338, Train Acc: 93.04%, Val Acc: 89.4
2%
Epoch [9/15], Loss: 6.2759, Train Acc: 95.05%, Val Acc: 88.6
9%
Epoch [10/15], Loss: 6.1549, Train Acc: 93.68%, Val Acc: 88.3
2%
Early stopping triggered
```


Training and Validation Accuracy Curves
Accuracy (%)
Epoch
Baseline Train
Baseline Val
ResNet18 Train
ResNet18 Val


Baseline CNN Confusion Matrix
True label
Predicted label
Bicycle
Car
Cat
Dog
Tree

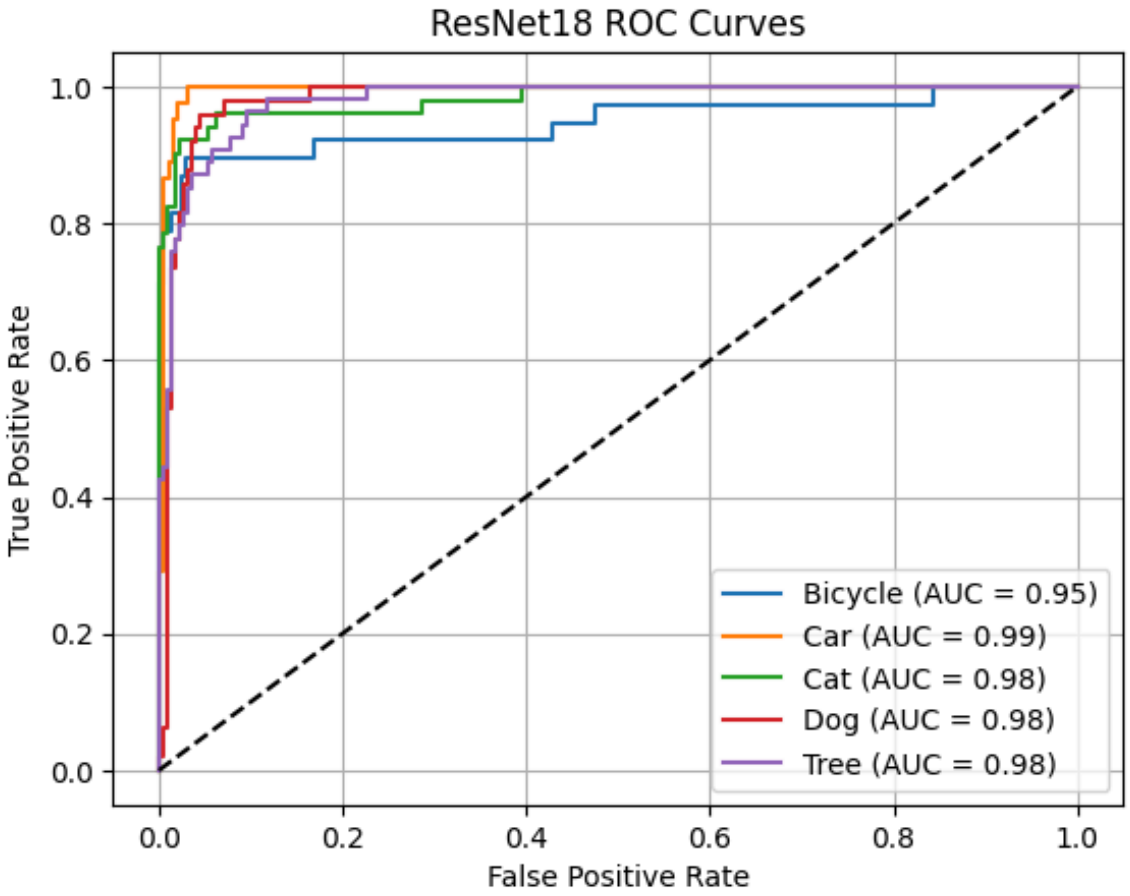
```
Baseline CNN Classification Report:
              precision    recall  f1-score   support

     Bicycle       0.60      0.32      0.41        38
         Car       0.76      0.72      0.74        82
         Cat       0.51      0.71      0.60        51
         Dog       0.38      0.55      0.45        49
        Tree       0.57      0.37      0.45        54

    accuracy                           0.56       274
   macro avg       0.56      0.53      0.53       274
weighted avg       0.59      0.56      0.56       274
```





ResNet18 Classification Report:

	precision	recall	f1-score	support
Bicycle	0.91	0.79	0.85	38
Car	0.93	0.98	0.95	82
Cat	0.90	0.90	0.90	51
Dog	0.86	0.88	0.87	49
Tree	0.85	0.85	0.85	54
accuracy			0.89	274
macro avg	0.89	0.88	0.88	274
weighted avg	0.89	0.89	0.89	274


ResNet18 ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Bicycle (AUC = 0.95)
Car (AUC = 0.99)
Cat (AUC = 0.98)
Dog (AUC = 0.98)
Tree (AUC = 0.98)