About Us

- Our team consists of the following members, with their contributions listed:
  - Wenbo Zhang (Falanan) - 100778036
    - Initial code & analysis
    - Code for data download
    - DataLoader code
    - ResNet Trainer & VGG Trainer code
    - ResNet18 & ResNet50 & VGG11 & VGG16 fine tuning code
  - Martin Truong (martru118) - 100708410
    - VGG models and Trainer
    - Code refactor
    - Plotting
    - Presentation
  - Wyatt Ritchie (wyattRitchie) - 100483764
    - Plotting
    - Presentation
    - Coding

About Dataset

This dataset is a dataset on dog breed classification.

The dataset comes from Stanford University. (Stanford Dogs Dataset)

```
# import os
# os.environ['CUDA_LAUNCH_BLOCKING'] = "1"

import warnings
warnings.filterwarnings("ignore")

import torch
from torch import (nn, optim)
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
import pandas as pd
```

```
from torch.utils.data import Dataset, DataLoader, random_split

import numpy as np
import itertools
import torch.nn.functional as F
import torchvision
from torchvision import transforms

import matplotlib.pyplot as plt
import cv2 as cv
import scipy.io
import re

from PIL import Image
import time
import random

print("PyTorch version:", torch.__version__)
# device = "cuda" if torch.cuda.is_available() else "cpu"
device = "mps" if getattr(torch,'has_mps',False) else "cuda" if torch.cuda.is_available() else "cpu"
print("Using {} device".format(device))
# device = "cpu"
devices = [torch.device(f'cuda:{i}')
            for i in range(torch.cuda.device_count())]
```

```
PyTorch version: 2.0.0+cu118
Using cuda device
```

Pre-processing of images

We download the dataset and split it into training images and test images. The dataset is split based on the file list.

```
! wget http://vision.stanford.edu/aditya86/ImageNetDogs/images.tar
```

```
! wget http://vision.stanford.edu/aditya86/ImageNetDogs/lists.tar
```

```
! tar -xvf images.tar | wc
```

```
! tar -xvf lists.tar | wc
```

```
! ls
```

```
# load training data
train_data_list = scipy.io.loadmat('train_list.mat')
```

```
# get file names
train_data_file_name_list = [inner_list[0] for inner_list in train_data_list["file_list"]]
train_data_file_name_list = [inner_list[0] for inner_list in train_data_file_name_list]

# get breeds
pattern = r'-(.*?)/'
train_data_breeds_list = [[re.search(pattern, item[0]).group(1) for item in sublist] for sublist in train_data_list["file_list"].tolist()]
train_data_breeds_list = [inner_list[0] for inner_list in train_data_breeds_list]

# get labels
train_data_labels = train_data_list["labels"].tolist()
train_data_labels = [label[0]-1 for label in train_data_labels]

# get paths of images
train_data_file_path_list = ["Images/"+inner_list for inner_list in train_data_file_name_list]


train_data_df = pd.DataFrame({"File_Name": train_data_file_name_list, "File_Path": train_data_file_path_list, "Breed":train_data_breeds_list, "Breed_Label": train_
train_data_df.head()
```

```
Training dataframe has a shape of: (12000, 4)
```

```
print("Training dataframe has a shape of:", train_data_df.shape)


# load test data
test_data_list = scipy.io.loadmat('test_list.mat')

test_data_file_name_list = [inner_list[0] for inner_list in test_data_list["file_list"]]
test_data_file_name_list = [inner_list[0] for inner_list in test_data_file_name_list]

pattern = r'-(.*?)/'
test_data_breeds_list = [[re.search(pattern, item[0]).group(1) for item in sublist] for sublist in test_data_list["file_list"].tolist()]
test_data_breeds_list = [inner_list[0] for inner_list in test_data_breeds_list]

test_data_labels = test_data_list["labels"].tolist()
test_data_labels = [label[0]-1 for label in test_data_labels]

test_data_file_path_list = ["Images/"+inner_list for inner_list in test_data_file_name_list]


test_data_df = pd.DataFrame({"File_Name": test_data_file_name_list, "File_Path": test_data_file_path_list, "Breed":test_data_breeds_list, "Breed_Label": test_data_
test_data_df.head()
```

```
Test dataframe has a shape of: (8580, 4)
```

```
print("Test dataframe has a shape of:",test_data_df.shape)
```

Below is a short sample of images in the dataset. From this sample we can see that there is very little consistency in the images, both in terms of subject and of quality.

```
img = cv.imread(train_data_file_path_list[0])
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)


# get random images from the dataset
random_sample = [random.randint(1, 8000) for _ in range(10)]
fig, axs = plt.subplots(1, 10, figsize=(17,17))

for i, ax in enumerate(axs):
    img = cv.imread(train_data_file_path_list[random_sample[i]])
    img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
    ax.imshow(img)
    ax.tick_params(axis='both', which='both', labelbottom=False, labelleft=False, labelright=False, labeltop=False)

plt.show()
```

Below is an example of image pre-processing done on the dataset.

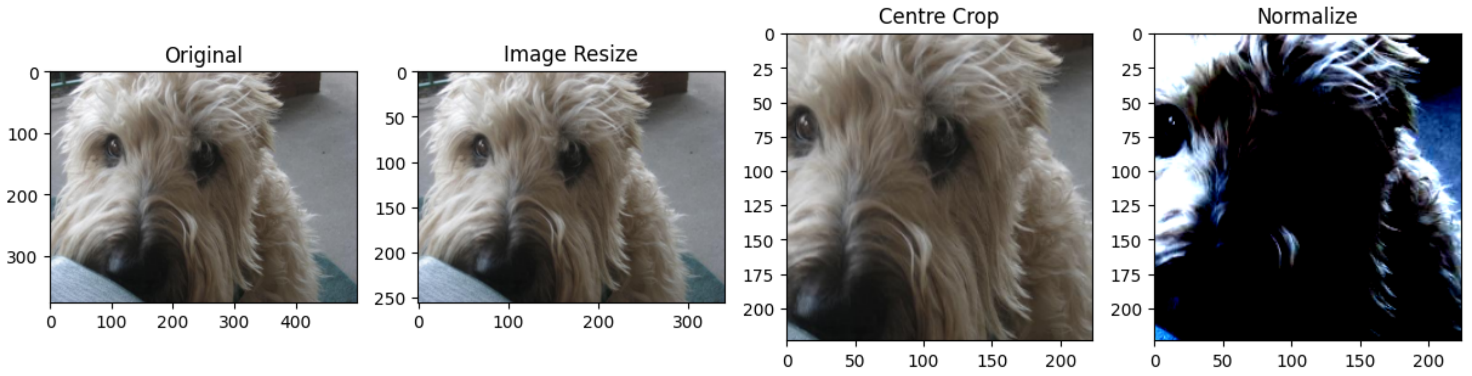
```
trans = transforms.Compose([transforms.ToTensor(), transforms.Resize(256)])
trans2 = transforms.Compose([transforms.CenterCrop(224)])
trans3 = transforms.Compose([transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])

img1_trans1 = trans(img)
img1_trans2 = trans2(img1_trans1)
img1_trans3 = trans3(img1_trans2)
img1_trans1 = np.transpose(img1_trans1, (1, 2, 0))
img1_trans2 = np.transpose(img1_trans2, (1, 2, 0))
img1_trans3 = np.transpose(img1_trans3, (1, 2, 0))
fig, axs = plt.subplots(1, 4, figsize=(15,15))
axs[0].imshow(img)
axs[1].imshow(img1_trans1)
axs[2].imshow(img1_trans2)
axs[3].imshow(img1_trans3)

axs[0].set_title('Original')
```

```
axs[1].set_title('Image Resize')
axs[2].set_title('Centre Crop')
axs[3].set_title('Normalize')
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..25
```



With the data pre-processed, we create our dataset.

```
class DogsDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        img_path = self.dataframe.iloc[idx]['File_Path']
        breed_label = self.dataframe.iloc[idx]['Breed_Label']
        img = Image.open(img_path)
        img = img.convert('RGB')
        if self.transform:
            img = self.transform(img)
        return img, breed_label

# transform the data
data_transforms = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
```

```
])

# create training and test datasets
dogs_dataset = DogsDataset(train_data_df, transform=data_transforms)
train_dataset, val_dataset = random_split(dogs_dataset, (0.8, 0.2))

# get shape of datasets
train_dataloader_for_display = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_dataloader_for_display = DataLoader(val_dataset, batch_size=128, shuffle=False)
```

```
for xs, ys in train_dataloader_for_display:
    break
xs.shape, ys.shape
```

```
(torch.Size([64, 3, 224, 224]), torch.Size([64]))
```

Evaluating our models

We create a function that evaluates a model using `test_dataset.npz`.

```
! mkdir records
```

```
! mkdir models
```

```
def test_model(model = None, saved_model = None):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    test_dog_dataset = DogsDataset(test_data_df, transform=data_transforms)

    if saved_model != None:
        #print("Loading from saved model.")
        model = torch.load(saved_model).to(device)
    elif model != None:
        model.to(device)

    loss = nn.CrossEntropyLoss()
    dataloader = DataLoader(test_dog_dataset, batch_size=128, shuffle=True)

    accuracy = 0

    with torch.no_grad():
        for xs, targets in dataloader:
            xs, targets = xs.to(device), targets.to(device)
            ys = model(xs)
            accuracy += (ys.argmax(axis=1) == targets).sum().item()
```

```
    #print("Saved model has test accuracy = %.4f" % acc)
    accuracy = accuracy / len(test_dog_dataset) * 100
    return accuracy
```

```
epochs = 6
```

Apply ResNet models

Here, we will apply ResNet model (`resnet18` and `resnet50`) for transfer learning and create a trainer class for the model.

```
# trainer for resnet model
class ResNet_Trainer:
    def __init__(self, model, train_dataset, val_dataset, learning_rate, batch_size, param_group=True):
        self.model = model
        self.train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
        self.val_dataloader = DataLoader(val_dataset, batch_size=batch_size*2, shuffle=True)
        self.loss = nn.CrossEntropyLoss()

        if param_group:
            # fine-tune model
            params_1x = [param for name, param in model.named_parameters()
                if name not in ["fc.weight", "fc.bias"]]
            self.optimizer = torch.optim.Adam([{'params': params_1x},
                                    {'params': model.fc.parameters(),
                                        'lr': learning_rate * 10}],
                                    lr=learning_rate, weight_decay=0.00001)
        else:
            self.optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate,
                                    weight_decay=0.00001)


    def categorical_accuracy(self, y_out, y_true):
      # calculate accuracy
      pred = y_out.argmax(axis=1)
      success = (pred == y_true).sum().item()
      total = len(y_true)
      return success / total

    def train_one_epoch(self):
        self.model.train()
        l_list = []
        acc_list = []
        for (i, (xs, targets)) in enumerate(self.train_dataloader):
            xs, targets = xs.to(device), targets.to(device)
```

```
            # calculate loss
            self.optimizer.zero_grad()
            opt = self.model(xs)
            loss = self.loss(opt, targets)
            loss.backward()
            self.optimizer.step()

            # log loss and accuracy
            l_list.append(loss.item())
            acc_list.append(self.categorical_accuracy(opt, targets))

        return np.mean(l_list), np.mean(acc_list)

    def val_one_epoch(self):
        self.model.eval()
        l_list = []
        acc_list = []
        with torch.no_grad():
            for (xs, targets) in self.val_dataloader:
                xs, targets = xs.to(device), targets.to(device)
                opt = self.model(xs)
                loss = self.loss(opt, targets)

                # log loss and accuracy
                l_list.append(loss.item())
                acc_list.append(self.categorical_accuracy(opt, targets))

        return np.mean(l_list), np.mean(acc_list)

    def train(self, epochs):
        history = {
            'train_loss': [],
            'train_accuracy': [],
            'val_loss': [],
            'val_accuracy': [],
            'epoch_duration': [],
        }

        start0 = time.time()

        for epoch in range(epochs):
            start = time.time()
            train_loss, train_acc = self.train_one_epoch()
            val_loss, val_acc = self.val_one_epoch()

            duration = time.time() - start
            history['train_loss'].append(train_loss)
            history['train_accuracy'].append(train_acc)
            history['val_loss'].append(val_loss)
            history['val_accuracy'].append(val_acc)
```

```
        history['epoch_duration'].append(duration)

        print("[%d (%.4fs)]: train_loss=%.4f train_acc=%.4f, val_loss=%.4f val_acc=%.4f" % (epoch+1, duration, train_loss, train_acc, val_loss, val_acc))

    duration0 = time.time() - start0
    print("== Total training time %.4f seconds ==" % duration0)
    return pd.DataFrame(history)
```

```
resnet18_model = torchvision.models.resnet18(pretrained=True)
resnet18_model.fc = nn.Linear(resnet18_model.fc.in_features,120)    # change to 120 labels
resnet18_model = resnet18_model.to(device)
resnet18trainer = ResNet_Trainer(resnet18_model, train_dataset, val_dataset, 5e-5,128)
```

```
# train model
resnet18_log = resnet18trainer.train(epochs)
```

```
[1 (115.8350s)]: train_loss=3.2303 train_acc=0.3543, val_loss=1.8950 val_acc=0.6217
[2 (114.6463s)]: train_loss=1.3256 train_acc=0.7493, val_loss=1.1727 val_acc=0.7283
[3 (128.0369s)]: train_loss=0.7632 train_acc=0.8575, val_loss=0.9530 val_acc=0.7617
[4 (111.0800s)]: train_loss=0.4659 train_acc=0.9282, val_loss=0.8526 val_acc=0.7707
[5 (109.1434s)]: train_loss=0.2820 train_acc=0.9704, val_loss=0.7916 val_acc=0.7706
[6 (108.5363s)]: train_loss=0.1727 train_acc=0.9881, val_loss=0.7759 val_acc=0.7712
== Total training time 687.2795 seconds ==
```

```
resnet18_csv_data = resnet18_log.to_csv(index=False)
with open('records/resnet18_csv_data.csv', 'w') as f:
    f.write(resnet18_csv_data)
```

```
resnet18_model = resnet18_model.to("cpu")
torch.save(resnet18_model, 'models/resnet18_model.pt')
torch.cuda.empty_cache()
```

Fine-tune and train a different model. This time, we are using ResNet50.

```
resnet50_model = torchvision.models.resnet50(pretrained=True)
resnet50_model.fc = nn.Linear(resnet50_model.fc.in_features,120)    # change to 120 labels
resnet50_model = resnet50_model.to(device)
resnet50trainer = ResNet_Trainer(resnet50_model, train_dataset, val_dataset, 5e-5, 128)
```

```
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|████████████████| 97.8M/97.8M [00:00<00:00, 103MB/s]
```

```
# train model
resnet50_log = resnet50trainer.train(epochs)
```

```
[1 (173.7187s)]: train_loss=2.1465 train_acc=0.5844, val_loss=0.8377 val_acc=0.7975
[2 (175.0209s)]: train_loss=0.4892 train_acc=0.8817, val_loss=0.5778 val_acc=0.8285
[3 (173.3371s)]: train_loss=0.1935 train_acc=0.9592, val_loss=0.5018 val_acc=0.8477
[4 (173.3833s)]: train_loss=0.0849 train_acc=0.9867, val_loss=0.4745 val_acc=0.8583
[5 (175.0527s)]: train_loss=0.0395 train_acc=0.9949, val_loss=0.4809 val_acc=0.8516
[6 (173.1704s)]: train_loss=0.0236 train_acc=0.9973, val_loss=0.4818 val_acc=0.8566
== Total training time 1043.6845 seconds ==
```

```
resnet50_csv_data = resnet50_log.to_csv(index=False)
with open('records/resnet50_csv_data.csv', 'w') as f:
    f.write(resnet50_csv_data)
```

```
resnet50_model = resnet50_model.to("cpu")
torch.save(resnet50_model, 'models/resnet50_model.pt')
torch.cuda.empty_cache()
```

Apply VGG models

Here, we will apply VGG model (`VGG16` and `VGG11`) for transfer learning and create a trainer class for the model.

```
# trainer for vgg model
class VGG_Trainer:
    def __init__(self, model, train_dataset, val_dataset, learning_rate, batch_size, param_group=True):
        self.model = model
        self.train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
        self.val_dataloader = DataLoader(val_dataset, batch_size=batch_size*2, shuffle=True)
        self.loss = nn.CrossEntropyLoss()

        if param_group:
            # fine-tune model
            params_1x = [param for name, param in model.named_parameters()
                if name not in ["classifier.6.weight", "classifier.6.bias"]]
            self.optimizer = torch.optim.Adam([{'params': params_1x},
                                    {'params': model.classifier[6].parameters(),
                                        'lr': learning_rate * 10}],
                                    lr=learning_rate, weight_decay=0.00001)
        else:
            self.optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, weight_decay=0.00001)

    def categorical_accuracy(self, y_out, y_true):
      # calculate accuracy
      pred = y_out.argmax(axis=1)
      success = (pred == y_true).sum().item()
      total = len(y_true)
      return success / total
```

```
def train_one_epoch(self):
    self.model.train()
    l_list = []
    acc_list = []
    for (i, (xs, targets)) in enumerate(self.train_dataloader):
        xs, targets = xs.to(device), targets.to(device)

        # calculate loss
        self.optimizer.zero_grad()
        opt = self.model(xs)
        loss = self.loss(opt, targets)
        loss.backward()
        self.optimizer.step()

        # log loss and accuracy
        l_list.append(loss.item())
        acc_list.append(self.categorical_accuracy(opt, targets))

    return np.mean(l_list), np.mean(acc_list)

def val_one_epoch(self):
    self.model.eval()
    l_list = []
    acc_list = []

    with torch.no_grad():
        for (xs, targets) in self.val_dataloader:
            xs, targets = xs.to(device), targets.to(device)
            opt = self.model(xs)
            loss = self.loss(opt, targets)

            # log loss and accuracy
            l_list.append(loss.item())
            acc_list.append(self.categorical_accuracy(opt, targets))

    return np.mean(l_list), np.mean(acc_list)

def train(self, epochs):
    history = {
        'train_loss': [],
        'train_accuracy': [],
        'val_loss': [],
        'val_accuracy': [],
        'epoch_duration': [],
    }

    start0 = time.time()

    for epoch in range(epochs):
        start = time.time()
```

```
            train_loss, train_acc = self.train_one_epoch()
            val_loss, val_acc = self.val_one_epoch()

            duration = time.time() - start
            history['train_loss'].append(train_loss)
            history['train_accuracy'].append(train_acc)
            history['val_loss'].append(val_loss)
            history['val_accuracy'].append(val_acc)
            history['epoch_duration'].append(duration)

            print("[%d (%.4fs)]: train_loss=%.4f train_acc=%.4f, val_loss=%.4f val_acc=%.4f" % (epoch+1, duration, train_loss, train_acc, val_loss, val_acc))

        duration0 = time.time() - start0
        print("== Total training time %.4f seconds ==" % duration0)

        return pd.DataFrame(history)


vgg16_model = torchvision.models.vgg16(pretrained=True)

# change classifier to 120 categories of dogs
vgg16_model.classifier[6] = nn.Linear(vgg16_model.classifier[6].in_features, 120)


# train model
vgg16_model = vgg16_model.to(device)
vgg16_trainer = VGG_Trainer(vgg16_model, train_dataset, val_dataset, 5e-5, 64)
vgg16_log = vgg16_trainer.train(epochs)
```

```
[1 (226.6084s)]: train_loss=1.4484 train_acc=0.6079, val_loss=0.7257 val_acc=0.7736
[2 (220.9710s)]: train_loss=0.5084 train_acc=0.8302, val_loss=0.6864 val_acc=0.7919
[3 (222.6453s)]: train_loss=0.2753 train_acc=0.9047, val_loss=0.7297 val_acc=0.7736
[4 (222.7849s)]: train_loss=0.1693 train_acc=0.9443, val_loss=0.7493 val_acc=0.7917
[5 (221.7520s)]: train_loss=0.1372 train_acc=0.9549, val_loss=0.8195 val_acc=0.7936
[6 (221.6562s)]: train_loss=0.1245 train_acc=0.9578, val_loss=0.8482 val_acc=0.7889
== Total training time 1336.4207 seconds ==
```

```
vgg16_csv_data = vgg16_log.to_csv(index=False)
with open('records/vgg16_csv_data.csv', 'w') as f:
    f.write(vgg16_csv_data)


vgg16_model = vgg16_model.to("cpu")
torch.save(vgg16_model, 'models/vgg16_model.pt')
torch.cuda.empty_cache()
```

Fine-tune and train a different model. This time, we are using VGG11.

```
vgg11_model = torchvision.models.vgg11(pretrained=True)

# change classifier to 120 categories of dogs
vgg11_model.classifier[6] = nn.Linear(vgg11_model.classifier[6].in_features, 120)


# train model
vgg11_model = vgg11_model.to(device)
vgg11_trainer = VGG_Trainer(vgg11_model, train_dataset, val_dataset, 5e-5, 64)
vgg11_log = vgg11_trainer.train(epochs)
```

```
    [1 (154.2641s)]: train_loss=1.6445 train_acc=0.5583, val_loss=0.8092 val_acc=0.7519
    [2 (156.9024s)]: train_loss=0.5931 train_acc=0.8090, val_loss=0.7906 val_acc=0.7526
    [3 (157.6562s)]: train_loss=0.3039 train_acc=0.8990, val_loss=0.8524 val_acc=0.7485
    [4 (155.5143s)]: train_loss=0.1814 train_acc=0.9402, val_loss=0.8587 val_acc=0.7685
    [5 (156.6619s)]: train_loss=0.1319 train_acc=0.9546, val_loss=0.9286 val_acc=0.7475
    [6 (156.1656s)]: train_loss=0.0821 train_acc=0.9749, val_loss=1.0220 val_acc=0.7486
    == Total training time 937.1675 seconds ==
```

```
vgg11_csv_data = vgg11_log.to_csv(index=False)
with open('records/vgg11_csv_data.csv', 'w') as f:
    f.write(vgg11_csv_data)
```

```
vgg11_model = vgg11_model.to("cpu")
torch.save(vgg11_model, 'models/vgg11_model.pt')
torch.cuda.empty_cache()
```

Now that we have the records and models for both networks, we save the outputs to a zip file.

```
! zip -r records.zip records
```

```
! zip -r models.zip models
```

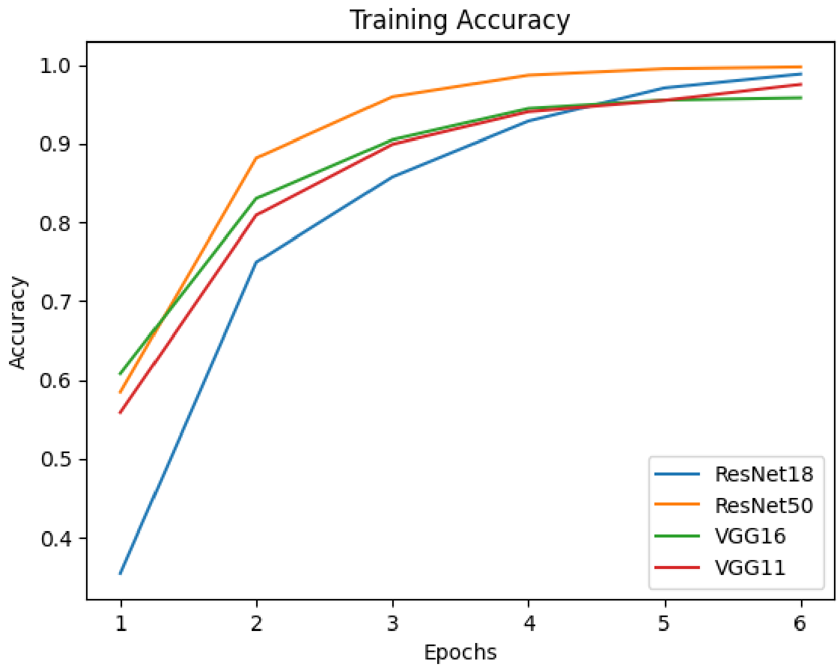
Plot training accuracy

We plot the training accuracies between the fine-tuned ResNet and VGG models.

```
plt.figure()
plt.plot(resnet18_log.index+1, resnet18_log.train_accuracy)
plt.plot(resnet50_log.index+1, resnet50_log.train_accuracy)
plt.plot(vgg16_log.index+1, vgg16_log.train_accuracy)
plt.plot(vgg11_log.index+1, vgg11_log.train_accuracy)

plt.title('Training Accuracy')
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['ResNet18', 'ResNet50', 'VGG16', 'VGG11']);
```



Plot test accuracy

Test the fine-tuned ResNet and VGG models and plot the test accuracy (in a bar plot).

```
resnet50_test_acc = test_model(model = resnet50_model)
resnet18_test_acc = test_model(model = resnet18_model)
vgg16_test_acc = test_model(model = vgg16_model)
vgg11_test_acc = test_model(model = vgg11_model)

print("ResNet18 model test accuracy: %.4f" % resnet18_test_acc)
print("ResNet50 model test accuracy: %.4f" % resnet50_test_acc)
print("VGG11 model test accuracy: %.4f" % vgg16_test_acc)
print("VGG16 model test accuracy: %.4f" % vgg16_test_acc)
```

```
ResNet18 model test accuracy: 78.2634
ResNet50 model test accuracy: 85.8508
VGG11 model test accuracy: 77.8089
VGG16 model test accuracy: 77.8089
```

```
test_accs = [resnet18_test_acc, resnet50_test_acc, vgg16_test_acc, vgg11_test_acc]
model_names = ['ResNet18', 'ResNet50', 'VGG16', 'VGG11']

fig, ax = plt.subplots(figsize=(8, 6))
ax.bar(model_names, test_accs)

ax.set_xlabel('Model')
ax.set_ylabel('Test Accuracy (%)')
ax.set_title('Comparison of Test Accuracies')
```

```
Text(0.5, 1.0, 'Comparison of Test Accuracies')
```