Dog Breed Classification

Comparing transfer learning models

The Stanford Dog Dataset

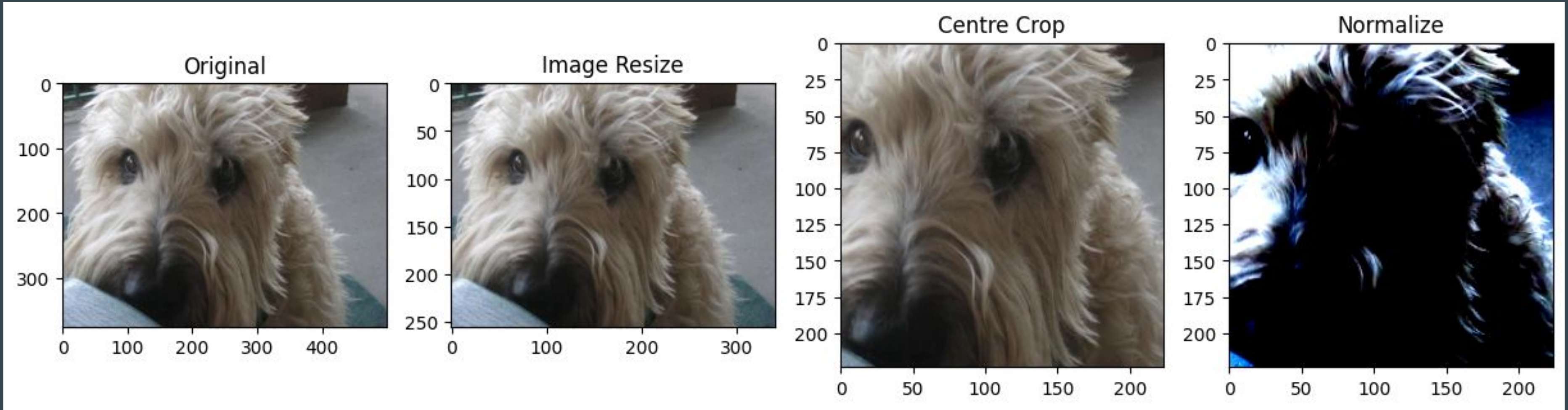
Contains 20k images from 120 dog breeds, ruff-ly 150 images per breed

Our Tasks

Transfer learning in PyTorch with ResNet & VGG

- Pre-process images in dataset
- Create Trainer classes for pre-trained models
- Fine tune models
- Plot training and test accuracies between the models
- Analyze results and compare the models

Image Pre-processing



Trainer Class for Pre-trained Models

- Training and validation dataset is converted to dataloader
  - Contains images of size 224x224
- Cross entropy loss
- VGG and ResNet model is passed to respective class
  - ResNet-18, ResNet-50, VGG-16, VGG-11
  - Different fine-tuning methods per network
- Fully connected linear layer is changed for both networks
  - 120 output channels
- Train and save models

Comparison of ResNet and VGG

- For our project we chose to do a comparative study of two pretrained machine learning models.
- The aim was to create a model which would take an image of a dog and classify it as the correct breed.
- The dataset we used was the Stanford Dogs dataset.

The ResNet Model

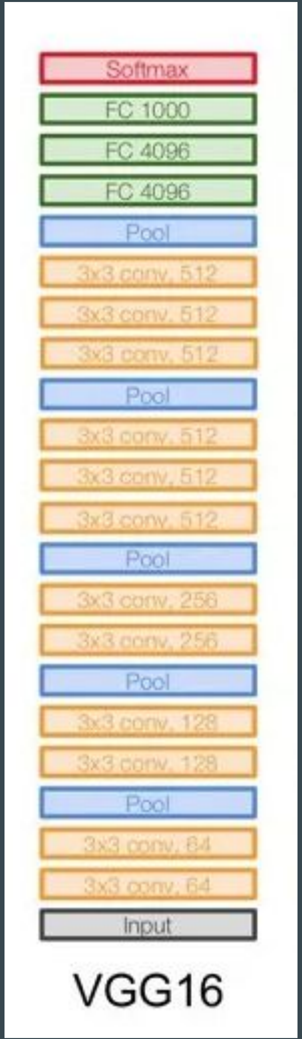
- All ResNet models have a similar architecture, just with different layers.
- We used ResNet-50 and ResNet-18.
- The architecture consists of a number of residual blocks.
- Within each block we have multiple convolution layers, each of which are followed by batch normalization.
- All these blocks are stacked, so it becomes a very deep architecture.

Fine-tuning ResNet

```
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)
```

The VGG model

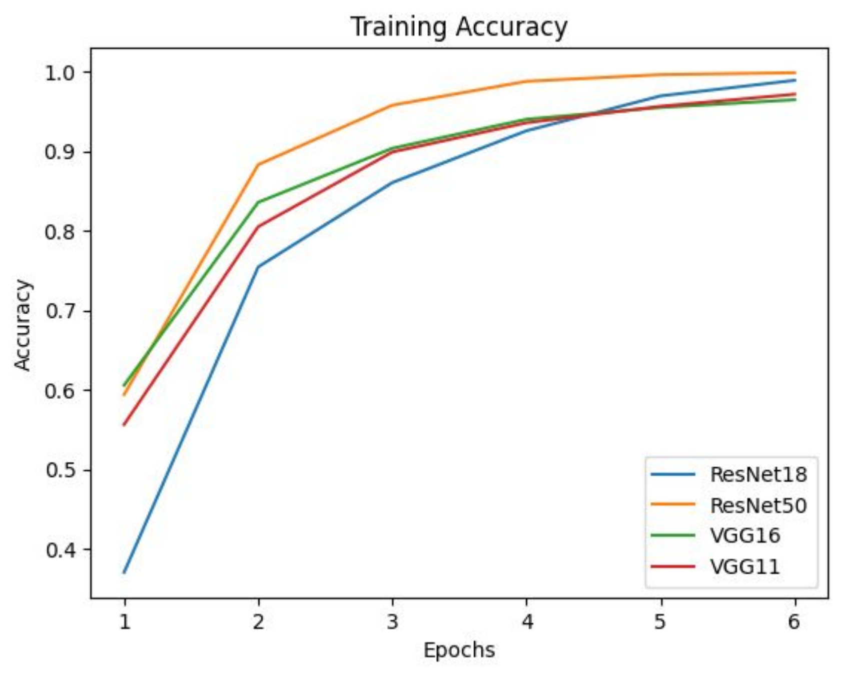
- All VGG models have a similar architecture, just with a different set of layers.
- We used VGG-11 and VGG-16.
- Ther VGG architecture uses small sets of convolution layers (2-4) followed by a pooling layer.
- This small grouping is stacked and followed by a fully connected layer.

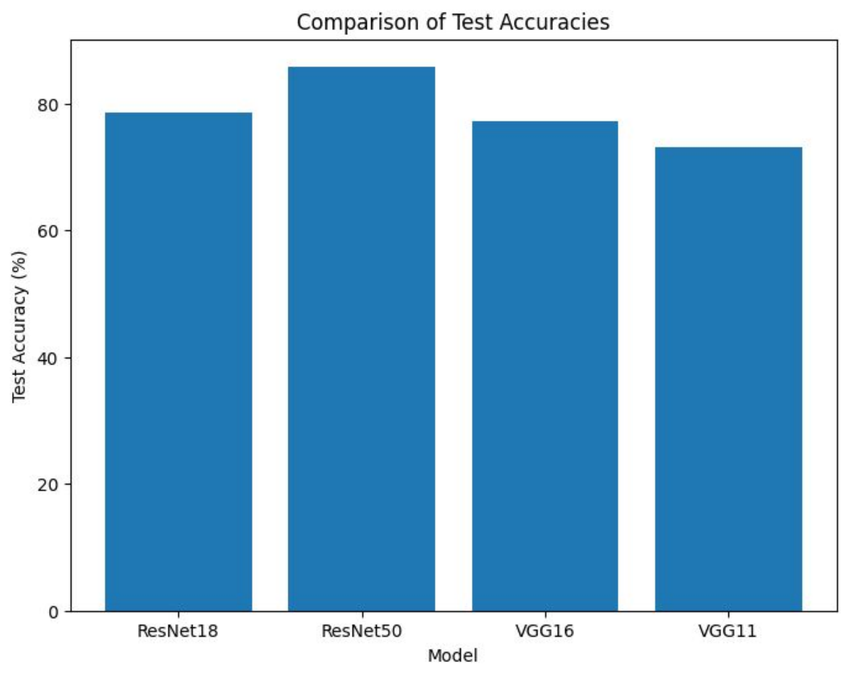


Fine-tuning VGG

```
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)
```

Accuracy between models





Training of ResNet models

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

```
[1 (116.7618s)]: train_loss=3.1977 train_acc=0.3707, val_loss=1.7912 val_acc=0.6651
[2 (112.5373s)]: train_loss=1.2983 train_acc=0.7546, val_loss=1.1403 val_acc=0.7395
[3 (110.0501s)]: train_loss=0.7498 train_acc=0.8606, val_loss=0.8885 val_acc=0.7715
[4 (109.4918s)]: train_loss=0.4584 train_acc=0.9257, val_loss=0.8229 val_acc=0.7712
[5 (110.6676s)]: train_loss=0.2780 train_acc=0.9696, val_loss=0.7666 val_acc=0.7858
[6 (109.7251s)]: train_loss=0.1687 train_acc=0.9892, val_loss=0.7392 val_acc=0.7895
== Total training time 669.2391 seconds ==
```

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

```
[1 (178.1292s)]: train_loss=2.1301 train_acc=0.5942, val_loss=0.7935 val_acc=0.8158
[2 (179.3901s)]: train_loss=0.4763 train_acc=0.8831, val_loss=0.5481 val_acc=0.8496
[3 (179.2760s)]: train_loss=0.1975 train_acc=0.9579, val_loss=0.4886 val_acc=0.8487
[4 (178.7121s)]: train_loss=0.0819 train_acc=0.9879, val_loss=0.4601 val_acc=0.8583
[5 (179.2338s)]: train_loss=0.0382 train_acc=0.9964, val_loss=0.4593 val_acc=0.8637
[6 (178.5510s)]: train_loss=0.0204 train_acc=0.9988, val_loss=0.4662 val_acc=0.8618
== Total training time 1073.2975 seconds ==
```

Training of VGG models

```
# 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 (228.1726s)]: train_loss=1.4506 train_acc=0.6059, val_loss=0.6936 val_acc=0.7863
[2 (227.6527s)]: train_loss=0.5095 train_acc=0.8358, val_loss=0.6828 val_acc=0.7867
[3 (228.2055s)]: train_loss=0.2872 train_acc=0.9038, val_loss=0.7121 val_acc=0.7858
[4 (228.2102s)]: train_loss=0.1825 train_acc=0.9402, val_loss=0.8456 val_acc=0.7585
[5 (226.7153s)]: train_loss=0.1378 train_acc=0.9551, val_loss=0.8479 val_acc=0.7777
[6 (228.2807s)]: train_loss=0.1013 train_acc=0.9648, val_loss=0.8041 val_acc=0.7799
== Total training time 1367.2402 seconds ==
```

```
# 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 (156.2669s)]: train_loss=1.6267 train_acc=0.5566, val_loss=0.8424 val_acc=0.7508
[2 (156.7959s)]: train_loss=0.6028 train_acc=0.8052, val_loss=0.7919 val_acc=0.7607
[3 (156.1768s)]: train_loss=0.2997 train_acc=0.8991, val_loss=0.7921 val_acc=0.7595
[4 (155.6686s)]: train_loss=0.1827 train_acc=0.9358, val_loss=0.9015 val_acc=0.7375
[5 (156.2055s)]: train_loss=0.1320 train_acc=0.9566, val_loss=0.9846 val_acc=0.7386
[6 (156.2990s)]: train_loss=0.0866 train_acc=0.9717, val_loss=1.0465 val_acc=0.7342
== Total training time 937.4153 seconds ==
```

Deployment

- Python file that identifies the breed of dog
  - Dog images would be taken from a directory
  - Each image will be processed before evaluation
- Loads saved model from file
- Images are classified and the breed is named
- Possible future integration to front-end apps or to other models

References

- https://cv-tricks.com/keras/understand-implement-resnets/
- https://towardsdatascience.com/architecture-comparison-of-alexnet-vggnet-resnet-inception-densenet-beb8b116866d
- http://vision.stanford.edu/aditya86/ImageNetDogs/