Udacity

Machine Learning Nanodegree

Capstone Project: Dog Breed Classifier

Ayran Olckers

Ayran Thane Olckers
03/02/2020

Project Definition Overview

This project originates from one of the capstone proposals provided by Udacity in the Machine Learning Engineer Nanodegree. Artificial Neural Networks have rapidly growing applications in the area of image-based classification problems. This project is a *dog breed classifier* which integrates several components to create a simple application to identify dogs and humans and to classify the type of dog breed, or the type of dog breed a human most closely resembles. The project falls under the research domain of image recognition, a field in machine learning where major breakthroughs have been achieved, such as a convolutional neural network (CNN).

- *The training, validation and test dataset in this project was provided by Udacity.*

Problem Statement

The project primarily addresses classification problems using supervised learning. Input images are mostly of Animals, in this case dogs and also human faces but the application can actually accept any image.

The first classification problem is to determine if an image contains a dog (the model will classify an image true of false). If there is a dog in the image, the CNN will take the dog image as an input and output a prediction of the breed of the dog. If there is not a dog (Fale) in the image, the application will detect if there is a human face in the image and if one has been found, the CNN will guess which kind of dog breed the human face most closely resembles.

In the case where an image does not contain a dog or human, the application will display a message noting this state.

Metrics

In this project accuracy (number of correct predictions / total number of predictions) is used as the primary metric for evaluation and is suitable because the data is fairly balanced, meaning the training, validation and test data have roughly the same number of samples of dogs of each breed. A balanced dataset is important to avoid misleading accuracy results, such as the case in which 90% of the test data images are of a specific breed - then a default prediction of that breed would yield 90% accuracy.

Although accuracy is not necessarily the best metric to evaluate the performance of a model, it relatively sufficient here as the distribution of training/validation/testing data across the different classes is not extreme.

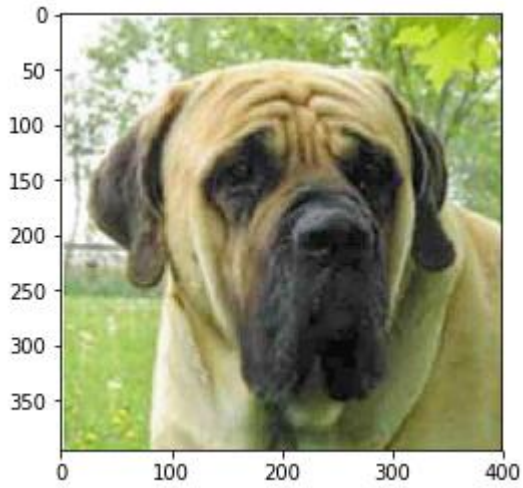
Data Analysis

Data Exploration and Visualization

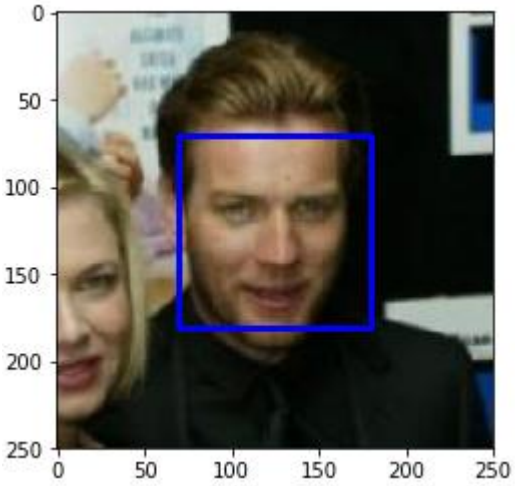
The dataset contains 8351 dog images and 13233 human images. Each dog and human image in its raw form has **R G B(Red Green Blue)** values and may vary in its pixel dimensions. For example, an image tensor could have the following dimensions: [1, 3, 224, 224], 1 image, a list RGB values of length 3 and width and length of 224.

Sample Dog Image

Ayran Thane Olckers
03/02/2020



Sample Human Image



Methodology

Data Pre-processing

The data comes in the form of images contained in numbered folders where each numbered folder is for a specific dog-breed. It's important to note that the photos are not of the same size; hence, randomized cropping needs to be implemented as part of the data augmentation.

Therefore images were retrieved using the **Python Image Library (PIL)**, resized, converted to tensors and normalized. After the image transformations, a sample dog or human image would have, for example, dimensions 1 x 3 x 224 x 224, meaning the first tensor would contain 3 tensors representing RGB values of a 224 x 224 image.

Each pixel is an input into the initial layer of the neural network. For the model built using transfer learning, images were transformed into uniform dimensions by first resizing them to 256 pixels and then centre cropping to 224 pixels. Since **PyTorch's ResNet** model was used as the starting point for transfer learning.

Ayran Thane Olckers
03/02/2020

The images were normalized with mean values [0.487,0.467,0.397]] and standard deviation values [0.235,0.23,0.23] which are also from **ResNet**'s recommended mean and standard deviation values. For the model built from scratch, the images were resized and cropped to 32 x 32 pixels to reduce training time and normalized with mean values.

Implementation and Refinement

First I defined, trained, validated and tested a convolutional neural network from scratch. The CNN contains 6 layers of convolutions which produce features used by a final, fully connected layer that takes in the features and outputs predictions.

As a prerequisite to training the model, the data need to be loaded, normalized, and augmented. For example, common kernel sizes in convolutional layers are 3x3, 5x5 and 7x7. Smaller strides tend to encode more information and maintain translational invariance, meaning the effect of the position of the object in the image is reduced. The second convolutional layer has 6 input channels which is the same number of output channels as from layer 1, 16 output channels and a kernel size of 5.

With a small kernel size and stride in pooling prevents discarding too much information from the previous layer. Finally, fully connected layers take in the features from the convolutional layers and produces predictions

There are a few empirically driven rules of thumb when choosing the number of hidden neurons. In general, the number of hidden neurons should be between the number of input and output parameters. Too few hidden neurons may result in underfitting and too many hidden neurons may lead to overfitting.

A common rule is to have the number of hidden neurons be roughly the average of the number of neurons in the input and output layers. The output layer contains 133 neurons which corresponds to the 133 dog breeds in our dataset. Each neuron of the output layer is a probability assigned to each dog breed, with the highest probability representing the model's prediction of the type of dog breed in the image.

To improve the performance, I employed fixed feature extraction, a form of transfer learning to extract the features from a pre-trained neural network, replace the final layer of the network and only optimize the weights of that layer. I started with a pre-trained neural network, **ResNet152**, a residual neural network, which is a convolutional neural network containing "skip connections" to simplify the network by using fewer layers for training and to avoid "vanishing gradients" problem in which the gradients in the early layers of a neural network become extremely small, inhibiting the network's ability to accurately reflect how a small change in a parameter's value will affect the output. The final layer was re-trained with Cross Entropy Loss criterion and stochastic gradient descent optimizer to update the parameters with a learning rate (how much to adjust each parameter based on the gradient) of .001.

```
: import torch.optim as optim

  ### TODO: select loss function
  criterion_scratch = nn.CrossEntropyLoss()

  ### TODO: select optimizer
  optimizer_scratch = optim.Adam(model_scratch.parameters(), lr=0.001)
```

Learning rates (typically ranging from .1 - .001) that are larger enable models to train faster but may

lead to suboptimal final weights while smaller learning rates train more slowly but may lead to better results. The result, discussed in the next section, substantially improved.

```
import torch
from torchvision import datasets,transforms
from torch.utils.data import DataLoader
### TODO: Write data loaders for training, validation, and test sets
## Specify appropriate transforms, and batch_sizes
train_dir ='/data/dog_images/train'
valid_dir = '/data/dog_images/valid'
test_dir = '/data/dog_images/test'
train_transforms = transforms.Compose([transforms.RandomResizedCrop(256
),
                                       transforms.ColorJitter(saturation
=0.2),
                                       transforms.RandomHorizontalFlip()
,
                                       transforms.ToTensor(),
                                       transforms.Normalize(mean_train_s
et,std_train_set)])
valid_test_transforms= transforms.Compose([transforms.Resize(300),
                                           transforms.CenterCrop(256),
                                           transforms.ToTensor(),
                                           transforms.Normalize(mean_tra
in_set,std_train_set)])

train_dataset = datasets.ImageFolder(train_dir,transform=train_transfor
ms)
valid_dataset = datasets.ImageFolder(valid_dir,transform=valid_test_tra
nsforms)
test_dataset = datasets.ImageFolder(test_dir,transform=valid_test_trans
forms)

trainloader = DataLoader(train_dataset,batch_size=32,shuffle=True)
validloader = DataLoader(valid_dataset,batch_size=32,shuffle=False)
testloader = DataLoader(test_dataset,batch_size=32,shuffle=False)

loaders_scratch={}
loaders_scratch['train'] = trainloader
loaders_scratch['valid'] = validloader
loaders_scratch['test'] = testloader
use_cuda = torch.cuda.is_available()
```

Test and Validate Model

```
Epoch: 1       Training Loss: 3.163183      Validation Loss: 3.351496
Saving Model...
Epoch: 2       Training Loss: 3.110660      Validation Loss: 3.362549
Epoch: 3       Training Loss: 3.094015      Validation Loss: 3.302557
Saving Model...
Epoch: 4       Training Loss: 3.045681      Validation Loss: 3.276846
Saving Model...
Epoch: 5       Training Loss: 2.993825      Validation Loss: 3.233835
Saving Model...
Epoch: 6       Training Loss: 2.929757      Validation Loss: 3.265956
Epoch: 7       Training Loss: 2.924087      Validation Loss: 3.234437
Epoch: 8       Training Loss: 2.886171      Validation Loss: 3.248855
```

Ayran Thane Olckers
03/02/2020

```
Epoch: 9      Training Loss: 2.864938      Validation Loss: 3.202823
Saving Model...
Epoch: 10     Training Loss: 2.837034      Validation Loss: 3.141700
Saving Model...
Epoch: 11     Training Loss: 2.767841      Validation Loss: 3.127835
Saving Model...
Epoch: 12     Training Loss: 2.772565      Validation Loss: 3.096114
Saving Model...
Epoch: 13     Training Loss: 2.729465      Validation Loss: 3.208153
Epoch: 14     Training Loss: 2.699877      Validation Loss: 3.122743
Epoch: 15     Training Loss: 2.680818      Validation Loss: 3.102226
Epoch: 16     Training Loss: 2.657998      Validation Loss: 3.059349
Saving Model...
Epoch: 17     Training Loss: 2.650375      Validation Loss: 3.138282
Epoch: 18     Training Loss: 2.607886      Validation Loss: 3.048496
Saving Model...
Epoch: 19     Training Loss: 2.542328      Validation Loss: 3.013177
Saving Model...
Epoch: 20     Training Loss: 2.581949      Validation Loss: 3.041686
```

Test Model:

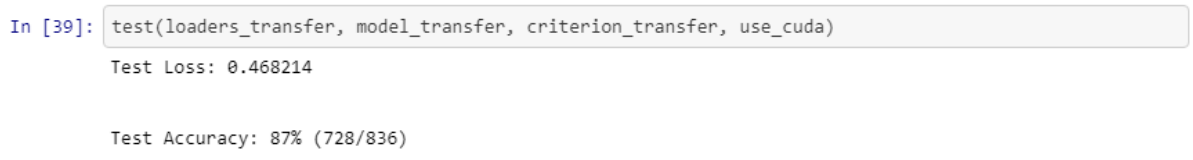
```
Test Loss: 2.814672


Test Accuracy: 30% (258/836)
```

Results

Model Evaluation, Validation and Justification

To evaluate both the model created from scratch and the transfer learning model, I used the following validation process.



I accumulated the loss during each forward pass and computed the average validation loss by dividing the total loss in each epoch by the number of samples in the validation set.

To track the progress of the training, I accumulated the training loss in each epoch and calculated the average training loss.

```
model.eval()
    for batch_idx, (data, target) in enumerate(loaders['test']):
        # move to GPU
        if use_cuda:
            data, target = data.cuda(), target.cuda()
        # forward pass: compute predicted outputs by passing inputs to
the model
        output = model(data)
        # calculate the loss
        loss = criterion(output, target)
```

```
        # update average test loss
        test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data - t
est_loss))
        # convert output probabilities to predicted class
        pred = output.data.max(1, keepdim=True)[1]
        # compare predictions to true label
        correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred))
).cpu().numpy())
        total += data.size(0)
```

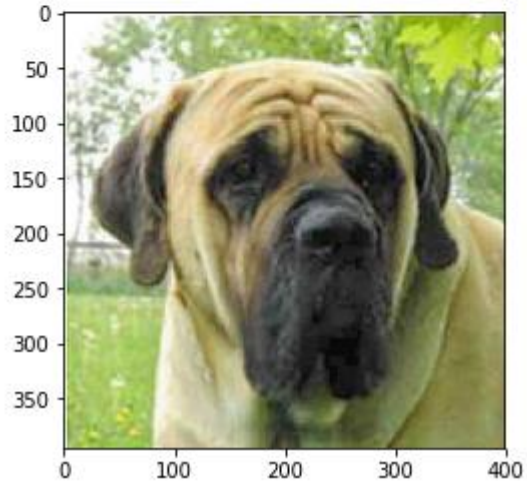
For the convolutional network built and trained from scratch, the results were poor. After training for 20 epochs, Epoch 1 started with training loss 4.525344 and Validation loss at 3.808864, all the way to Epoch 20 with Training loss of 0.894771 and Validation loss at 0.446516.

```
Epoch: 1      Training Loss: 4.515244      Validation Loss: 3.808864
Saving Model...
Epoch: 2      Training Loss: 3.419878      Validation Loss: 2.502836
Saving Model...
Epoch: 3      Training Loss: 2.511337      Validation Loss: 1.710158
Saving Model...
Epoch: 4      Training Loss: 1.982668      Validation Loss: 1.280219
Saving Model...
Epoch: 5      Training Loss: 1.704548      Validation Loss: 1.041078
Saving Model...
Epoch: 6      Training Loss: 1.493711      Validation Loss: 0.884728
Saving Model...
Epoch: 7      Training Loss: 1.352886      Validation Loss: 0.782032
Saving Model...
Epoch: 8      Training Loss: 1.288674      Validation Loss: 0.710056
Saving Model...
Epoch: 9      Training Loss: 1.203638      Validation Loss: 0.680317
Saving Model...
Epoch: 10     Training Loss: 1.154084      Validation Loss: 0.615035
Saving Model...
Epoch: 11     Training Loss: 1.125447      Validation Loss: 0.582500
Saving Model...
Epoch: 12     Training Loss: 1.073280      Validation Loss: 0.560404
Saving Model...
Epoch: 13     Training Loss: 1.061358      Validation Loss: 0.532912
Saving Model...
Epoch: 14     Training Loss: 1.016455      Validation Loss: 0.530950
Saving Model...
Epoch: 15     Training Loss: 1.006813      Validation Loss: 0.495559
Saving Model...
Epoch: 16     Training Loss: 0.964747      Validation Loss: 0.472909
Saving Model...
Epoch: 17     Training Loss: 0.948202      Validation Loss: 0.470832
Saving Model...
Epoch: 18     Training Loss: 0.921800      Validation Loss: 0.446577
Saving Model...
Epoch: 19     Training Loss: 0.879347      Validation Loss: 0.450817
Epoch: 20     Training Loss: 0.894771      Validation Loss: 0.446516
Saving Model...
```

Ayran Thane Olckers
03/02/2020

Test Cases:

```
Hello Human,
You look like a English toy spaniel
Hello Human,
You look like a American foxhound
Hello Human,
You look like a American water spaniel
Hello Human,
You look like a Chinese crested
Hello Human,
You look like a Dogue de bordeaux
It is a dog!
It looks like a Mastiff
It is a dog!
It looks like a Mastiff
It is a dog!
It looks like a Mastiff
It is a dog!
It looks like a Mastiff
It is a dog!
It looks like a Mastiff
```



Conclusion

I have successfully built and trained a Deep Neural Net model based on the VGG16 pretrained network to predict the breed of a dog by image input also images of dogs were pre-processed, a convolutional neural network was trained, validated and tested, and a function was written to simulate an application that takes in an input image path, predict the breed of the dog or the type of breed a human most closely resembles.

```
Test Loss: 0.468214
```

```
Test Accuracy: 87% (728/836)
```

Ayran Thane Olckers
03/02/2020

To improve the results of the model trained from scratch, I can try adding more convolutional layers and to boost the accuracy of the transfer model, perhaps I can experiment with adding more hidden layers to the final fully connected layers of the network with more training time.

Sources:

https://arxiv.org/abs/1512.03385

https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363

https://www.quora.com/How-can-I-decide-the-kernel-size-output-maps-and-layers-of-CNN

https://arxiv.org/pdf/1606.02228v2.pdf

https://www.quora.com/How-does-one-determine-stride-size-in-CNN-filters

https://github.com/KaimingHe/deep-residual-networks

https://stats.stackexchange.com/questions/207195/translational-variance-in-convolutional-neural-networks

https://en.wikipedia.org/wiki/Residual_neural_network

https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-param eter-42efca01e5d7

https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

https://www.quora.com/What-is-the-vanishing-gradient-problem

https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

Ayran Thane Olckers

03/02/2020