Capstone Project

Machine Learning Nanodegree Program

Michael X. Ledesma

June 9, 2021

Identifying a Dog's Breed through Convolution Neural Network and Transfer Learning

Definition

Project Overview

Artificial intelligence (AI) assists machines to mimic human behavior. Machines needs human data to be given a task to restructure that data. The machine needs to learn on emulating the data by combining a model with that data. Therefore, we can say that Machine Learning is a subset of AI techniques.

Branching off from Machine Learning, we can discuss one technique, Deep Learning which uses Neural Networks. This project uses Convolution Neural Networks along with Transfer learning to predict a dog's breed classification.

Problem Statement

How can we assist machines to classify a dog's breed?

An algorithm based on Convolutional Neural Network (CNN) has weights and bias in its hidden layers. Training and testing its model can predict a dog's breed. If supplied an image of a human, the model will identify the resembling of a dog's breed. If the input image is neither a dog or a human, the output will simply indicate "It is not a dog, nor a human image, Neither.".

Metrics

Pytorch was written throughout this project. Evaluating the training and test set model, a train_loss and a valid_loss is updated from Figure 1:

```
train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data - train_loss))

valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valid_loss))
```

Figure 1

Training and test sets runs through iterations of Epochs. One Epoch is when the CNN learning algorithm goes through the entire dataset. Batch rates of samples are through the iterations of Epochs and are set by the CNN designer. As Epochs are running, training losses are displayed. If the validation losses are decreasing, the minimum validation loss are saved and displayed.

When the test set completes, the test_loss and accuracy score are defined by Figure 2:

```
test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data - test_loss))
Accuracy score  = correct predictions / total predictions.
```

Figure 2

The datasets can be balanced or Imbalanced. What does that mean? A large sample of the dataset can belong to one class. In other words, the distribution of samples across the classes of dogs is not equal.

Thus, three important metrics that can measure the imbalance of the dog classes are Precision, Recall and the F1 Score by Figure 3.

```
Recall = True Positive / (True Positive + False Negative)

Recall = True Positive / Total Actual Positive
```

```
Precision = True Positive / (True Positive + False Positive)

Precision = True Positive / Total Predictive Positive
```

Figure 3

Therefore, F1Score sees the balance between Precision and Recall by Figure 4.

```
F1 Score = 2 * Precision * Recall /(Precision + Recall)
```

Figure 4

From the CNN Architecture and running the training, validation and test model, the evaluation metrics are:

- **Training Loss: 3.630847**
- **Validation Loss: 3.873030**
- **Test Loss: 3.908329**
- **Test Accuracy Score: 11%**
- **Precision: 0.104356**
- **Recall: 0.114957**
- **F1 Score: 0.098336**

Analysis

Data Exploration

From Udacity, two large datasets (lfw.zip and dogImages.zip) compressed in zip forms are provided. Once the dataset are unzipped, the datasets consist of 8351 of dog image contents and 13233 of human image contents, respectively. The dog's image contents are used in providing the training, test and valid sets. The goal is to create an image classifier capable of determining a dog's breed from a dog's image or a resembling a dog breed from a human image.

Exploratory Visualization

The "dogImage.zip" file is unzipped to jpeg images, downloaded and saved to three objects, training_data, test_data, and valid_data shown in figure 5:

```
data_dir = '/content/dogImages/' # need to have the '/' for correct directory find
if not os.path.exists(data_dir):
    os.makedirs(data_dir)

training_data = os.path.join(data_dir, 'train')
test_data = os.path.join(data_dir, 'test')
valid_data = os.path.join(data_dir, 'valid')
```

Figure 5

Improving training and test processing, Jpeg images are reduced in size of 224 pixels. For preprocessing the data, Jpeg images are transformed by tensors by the transform compose and ImageFolder. Pytorch was used throughout the project and Figure 6 shows the transform:

```
# large images will slow down processing
size = 224 #  imag is a square 255 pixels on each side


normalize = transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225])
transform = transforms.Compose([transforms.Resize(size), #transforms.Resize(255)
                                transforms.RandomResizedCrop(224), # 224 pixels on each side
                                transforms.CenterCrop(224),
                                transforms.RandomHorizontalFlip(),
                                #transforms.RandomRotation(10),
                                transforms.ToTensor(),normalize])

train_data = datasets.ImageFolder(training_data,transform=transform)
test_data = datasets.ImageFolder(test_data,transform=transform)
valid_data = datasets.ImageFolder(valid_data,transform=transform)
```

Figure 6

Batch sizes are defined by trail and error. Data can be shuffled and implemented to the data loader and saved by three objects, train_dataloader, test_loader and valid_dataloader shown in Figure 7:

```
### TODO: Write data loaders for training, validation, and test sets
## Specify appropriate transforms, and batch_sizes
batch_size=20
# The batch size defines the number of samples that will be propagated through the network.
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=batch_size,shuffle=True)
valid_dataloader = DataLoader(valid_data, batch_size=batch_size, shuffle=True)
```

Figure 7

Train, valid and test dataloaders are saved to a loaders_scratch object by Figure 8:

```
loaders_scratch = {'train':train_dataloader, 'valid':valid_dataloader, 'test':test_dataloader}
```

Figure 8

A class_name object is defined for classification with a length of 133 predicted dog indexes. The following 2 pics is shown in Figure 9 and 10 :

```
# get classes of training datas
class_names = train_data.classes
number_classes = len(class_names)
parameter_size = number_classes
print(class_names,number_classes)
print(type(train_data),train_data,type(loaders_scratch), class_names)
```

Figure 9

```
<class 'str'> /content/dogImages/test
  Number of train images:  6680
  Number of test images:   836
  Number of valid images:  835
['001.Affenpinscher', '002.Afghan_hound', '003.Airedale_terrier', '004.Akita', '005.Alaskan_malamute', '006.American_es
<class 'torchvision.datasets.folder.ImageFolder'> Dataset ImageFolder
    Number of datapoints: 6680
    Root location: /content/dogImages/train
    StandardTransform
Transform: Compose(
               Resize(size=224, interpolation=bilinear)
               RandomResizedCrop(size=(224, 224), scale=(0.08, 1.0), ratio=(0.75, 1.3333), interpolation=bilinear)
               CenterCrop(size=(224, 224))
               RandomHorizontalFlip(p=0.5)
               ToTensor()
               Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
           ) <class 'dict'> ['001.Affenpinscher', '002.Afghan_hound', '003.Airedale_terrier', '004.Akita', '005.Alaskar
133
```

Figure 10

Algorithms and Techniques

To classify a dog's breed, the CNN architecture starts by a Class Net. The following steps are taken for the CNN algorithm:

- **I selected a stack of 3 convolution layers (RGB), a 3x3 filter convolutional Kernel with a paddling of 1 that gives the filter one more space to move in either direction, to find patterns in an image. The 2nd and 3rd convolutional layer are used to discover patterns within a pattern. I selected a stride of 2 for convolution layer 1 and 2 because it can find the image edges and patterns. The 3rd convolution layer is selected a stride of one to find the last of image patterns.**
- **The Max pooling layer with a kernel of (2x2) makes the image resolution dimensionality be reduced to a 4x4 pixel. The filter size and stride is (2,2). The original depth is 64. Three Max pooling are between the hidden layers which improves accuracy.**
- **To build the class for the Neural Network the inputs need to go to two input hidden layer. The 2nd nn.linear has the output of 30.**
- **To prevent overfitting, a dropout rate of 0.5 of is used.**
- **Three ReLu activations are used In the forward behavior to aid to standardized the output values. This is done on each of the three convolution layer.**
- **Two pool layers of 2x2 are used from the self.pool(x).**
- **The images tensor data are flatten by the view method for a 1 dimension.**
- **A dropout is used to prevent overfitting.**

- To instantiate the CNN Model, the model_scratch = Net() is called.
- Move tensor to GPU if CUDA is available is used and shown in Figure 11:

```
# move tensors to GPU if CUDA is available
if use_cuda:
    model_scratch.cuda()
```

Figure 11

- A cross Entropy loss function and an Adam optimizer was used and shown in Figure 12:

```
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)
```

Figure 12

- The learning rate parameter of .001 was used.

Transfer Learning

- Using Pytorch, the same transform, normalization and tensor code was used as well as the CNN architectural model. From the data loaders, the transfer loader name is used as Data Transfer.
- A VGG16 pretrained model is chosen by Figure 13:

```
# Load the pretrained model from pytorch
model_transfer = models.vgg16(pretrained=True)
```

Figure 13

- To accelerate my application, GPU cuda was used.

- **Freezing the weights in the hidden layer is by Figure 14:**

```
# To freeze the weights in the layer
for param in model_transfer.parameters():
    param.requires_grad = False
```

Figure 14

During the EPOCH iterations, the weights cannot be modified. The weights of the pre-trained part is frozen and only the last layers are trained. The learning rate can change the weights and bias in the layers.

- **Replacing the last fully connected layer with a Linear output layer of 133 by Figure 15:**

```
# replace the last fully connected layer with a Linnear layer with 133 out features (param_output_size)
model_transfer.classifier[6] = nn.Linear(4096, 133, bias=True)
```

Figure 15

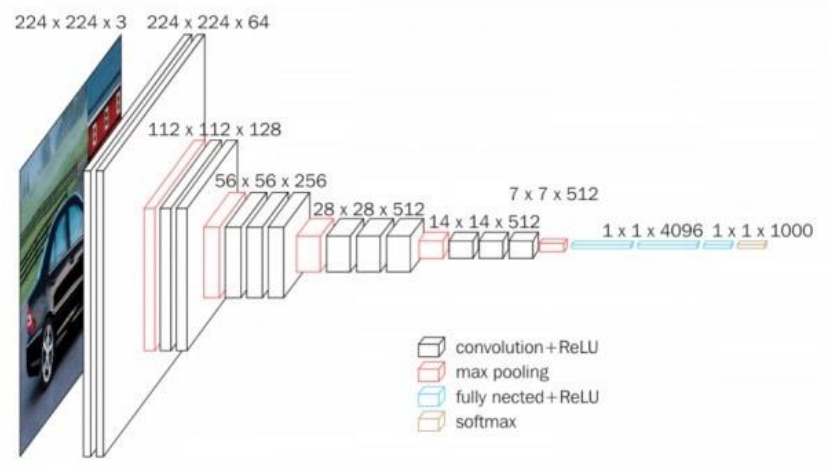
- **To accelerate the application, GPU cuda is used.**
- **A cross Entropy loss function and an Adam optimizer was used by Figure 16:**

```
import torch.optim as optim

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

optimizer_transfer = optim.Adam(filter(lambda p: p.requires_grad,model_transfer.parameters()), lr=0.0001)
```

Figure 16

- **A learning rate of 0.0001 was used.**

- **The VGG16 image model is shown in Figure 17:**



VGG-16 model architecture.

Figure 17

- **Running the CNN architecture with the transfer learning model of VGG-16 gave the following evaluation metrics:**
  - **Test Accuracy: 73%.**
  - **Training Loss: 0.716906**
  - **Validation Loss: 0.848791**
  - **Test Loss: 0.912295**
  - **Precision 0.753594**
  - **Recall: 0.726008**
  - **F1 Score: 0.724788**

Benchmark

Using Pytorch, the same transform, normalization and tensor code was used as well as the CNN architecture model. From the data loaders, the transfer loader name is used as loaders_scratch2.

- **A ResNet50 pretrained model is chosen and shown in Figure 18:**

```
## TODO: Specify model architecture
model_transfer2 = models.resnet50(pretrained=True)
```

Figure 18

- To accelerate my application, GPU cuda is used.
- Freezing the weights in the hidden layer is shown in Figure 19:

```
# Freezing the model
for param in model_transfer2.parameters():
    param.requires_grad = False
```

Figure 19

During the EPOCH iterations, the weights cannot be modified. The weights of the pre-trained part is frozen and only the last layers are trained. The learning rate can change the weights and bias in the layers.

- Replace the last fully connected layer with a Linear output layer 133 is shown in Figure 20:

```
model_transfer2.fc = nn.Linear(2048, 133, bias=True)
```

Figure 20

- To accelerate my application, GPU cuda is used.
- Update the weights and biases of the last fully connected layer by Figure 21:

```
for param in model_transfer.parameters():
    param.requires_grad = True # Update the weights and biases
```

Figure 21

- A cross Entropy loss function and an Adam optimizer was used by Figure 22:

```
## Specify Loss Function and Optimizer

criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.Adam(model_transfer2.fc.parameters(), lr=0.001,weight_decay=1e-5) # hyper parameters weight decay
```

Figure 22

- A learning rate of 0.001 was used.

- The ResNet-50 model is shown in Figure 23:

Figure 23

- For the Benchmark, running the CNN architecture with the transfer learning model of ResNet-50 gave the following metrics:
    - Training Loss: .607987
    - Validation Loss: .874565
    - Test Accuracy: 76%.
    - Test Loss: 0.902005
    - Precision 0.785567
    - Recall: 0.759500
    - F1 Score: 0.751775

Methodology

Data Preprocessing

To improve training and test processing, Jpeg images are reduced to a size of 224 pixels. If this step was not done, the computational time would have been slower. Jpeg images were transformed by tensors. Data augmentation is by the following transform compose shown in Figure 24:

```
transforms.RandomResizedCrop(224), # 224 pixels on each side
transforms.CenterCrop(224),
transforms.RandomHorizontalFlip()
```

Figure 24

and used to improve performance and accuracy.

Implementation

The popular VGG-16 Model was selected. The drawing of the VGG-16 model clearly shows the last fully connected layer of an input of 4096 and the output of 133 representing the dog prediction features. I kept getting errors because I did not specify the last 6th layer as shown in Figure 25:

```
# replace the last fully connected layer with a Linnear layer with 133 out features (param_output_size)
model_transfer.classifier[6] = nn.Linear(4096, 133, bias=True)
```

Figure 25

Additionally, using the VGG-16 model, I used the attribute .fc parameters instead of .classifier. Changing to .classifier resolved the issue and the VGG-16 model ran with no errors.

Refinement

My training set validation loss were in the (4.75 -5.09) range. The test accuracy were less than 11%. After numerous training set runs and researching the internet for best accuracy, I placed three a max pooling layer with a kernel size of (2x2) between the hidden linear layers.

Using the VGG-16 transfer learning model to increase test accuracy and a learning rate of .0001 is selected. The test accuracy is 11% with the CNN architecture. Including transfer learning increased the test accuracy to 73%. For imbalance dataset the evaluation metrics were run for Precision, Recall and F1Score.

Results

Model Evaluation and Validation

Performing many hours of training, validating and testing the model and having a reasonable within spec accuracy, I wrote an algorithm to Predict a Dog's Breed with the VGG-16 Model.

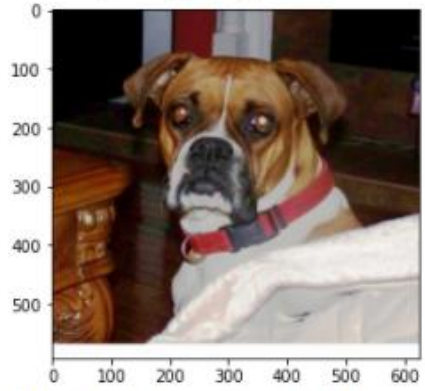
Class name was predicted and shown in Figure 26:

```
class_names = [item[4:].replace("_", " ") for item in data_transfer['train'].dataset.classes]
```

Figure 26

Figure 27 through 32, shows the results running the predictor algorithm using VGG-16 model:

```
Name of Dog Breed is Boxer.
<PIL.Image.Image image mode=RGB size=625x592 at 0x7FCBC7778D50> <class 'PIL.Image.Image'>
```




```
Name of Dog Breed is Keeshond.
```

Figure 27

```
Name of Dog Breed is Keeshond.
<PIL.Image.Image image mode=RGB size=587x612 at 0x7FCBC7716290> <class 'PIL.Image.Image'>
```

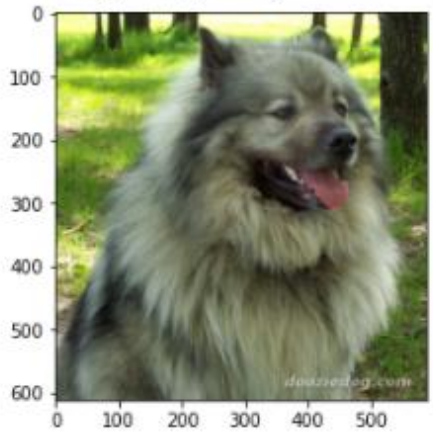


Figure 28

```
Name of Dog Breed is Japanese chin.
<PIL.Image.Image image mode=RGB size=275x200 at 0x7FCBC7E65B50> <class 'PIL.Image.Image'>
```

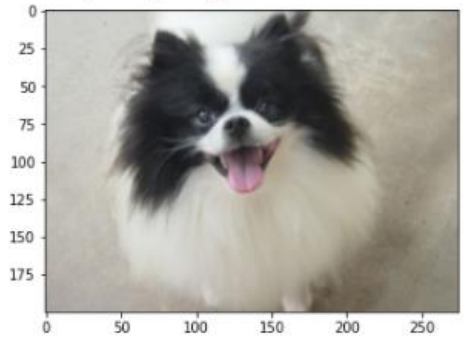


Figure 29

Figure 30

Figure 31

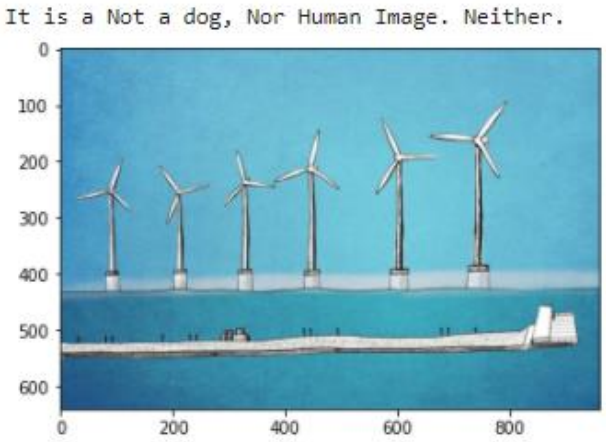


Figure 32

Justification

The CNN architecture with the transfer learning VGG-16 model gave a test accuracy of 73%.

The benchmark ResNet-50 model gave a test accuracy of 72%. Evaluating both models came out the same with an off percentage from one another of the models.

Conclusion

There are many different CNN architectures that software developers or Machine Learning engineers have made to accomplish the predictions of the dog's breed. By trial and error, the CNN convolution layers can be modified by the number of input layers, strides and kernels. I used three convolution layers for my CNN. To improve accuracy, many convolution layers could have been added along with the max pooling and ReLu activation functions.

There are many transfer learning models such as the Google inception, ResNet, and AlexNet models. I chose the most popular which is the VGG-16 Model and the ResNet-50 model.

Lastly, different python codes such as PySpark, tensorflow and Keras are widely used in maching learning. Udacity gives excellent Pytorch free videos which gave me the python coding choice of my CNN architectural model along with the transfer learning.

Reflections

Analyzing and increasing the convolution layers and hidden layers with adequate test accuracy was a challenge. Downloading the images using the glob function helped tremendously. I had to understand what datatype I was using such as a Jpeg file or numpy array. For plotting needed to convert a BGR image to RGB or keep the file as JPEG. For loops were used to ease the trouble in coding.

Lastly, writing a code with tensorflow and Keras or PySpark would have been a challenge.

Improvements

Bringing in the file from the list directory as shown in Figure 33:

```
data_dir = '/content/dogImages/' # need to have the '/' for correct directory find
if not os.path.exists(data_dir):
    os.makedirs(data_dir)

training_data = os.path.join(data_dir, 'train')
test_data = os.path.join(data_dir, 'test')
valid_data = os.path.join(data_dir, 'valid')
```

Figure 33

If the dog images were not from Udacity but another unknown source,

I could of included a try and exception code to remove gray scale images or no images at all such as Figure 34:

```
for folder in os.listdir(path) # Get all dog data train, valid and test data
    for img_file in os.listdir(os.path.join(path,folder)):
        image_file = os.path.join(path,folder, img_file) # creating full path for each image file

        try:
            img = Image.open(img_file)
            if img.mode != 'RGB':
                os.remove(img_file) # remove all gray scale images

        except:
                os.remove(img_file) # removing files type None images
```

Figure 34

Also, using another transfer learning model such as AlexNet could of increased the accuracy as well.

Thank you Udacity and the community for this Capstone Project of Predicting a Dog's Breed. I have learned tremendously.