Capstone Project

Randheer Ramesh K

Machine Learning Engineer Nanodegree

May 14th, 2020

Definition

Project OverView

Welcome to the Convolutional Neural Networks (CNN) project! In this project, I will build a pipeline to process real-world, user-supplied images. Given an image of a dog, my algorithm will identify an estimate of the canine's breed. If supplied with an image of a human, the code will identify the resembling dog breed.

Along with exploring state-of-the-art CNN models for classification, we have to make important design decisions about the performance of the model. Our goal is that by completing this lab, you understand the challenges involved in piecing together a series of models designed to perform various tasks in a data processing pipeline.

Problem Statement

The goal is to create a dog breed classification model using CNN. The task involved are the following:

1. Import Datasets
2. Detect Humans
3. Detect Dogs
4. Create a CNN to Classify Dog Breeds (from Scratch)
5. Create a CNN to Classify Dog Breeds (using Transfer Learning)
6. Write your Algorithm
7. Test Your Algorithm

The final model is expected to detect the dog breed from the input image. If the image contains a human face, then output the dog breed that is most similar to the face.

Metrics

Since this is a multi-class classification problem, we use accuracy as the main metrics to evaluate the model performance. Mainly because the classes are nearly balanced in the data.

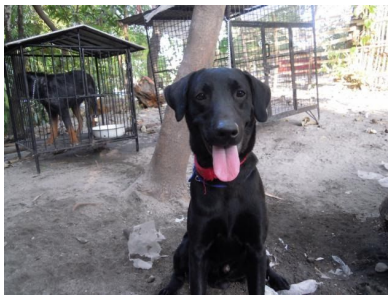
Accuracy = $\frac{TP + TN}{data\ size}$

Where TP is true positive, TN is true negative and data size is the total number of the input images.

Analysis

Data Exploration

We are using the images provided by Udacity to train the model. The dataset contains 13233 total human images and 8351 total dog images. The number of images later will be split into training and testing data.



This is an image from the dataset

Algorithms and Techniques

The classifier is a Convolutional Neural Network, which is the state-of-the-art algorithm for most image processing tasks, including classification. It needs a large amount of training data compared to other approaches; fortunately, the COCO and COCO-Text datasets are big

enough. The algorithm outputs an assigned probability for each class; this can be used to reduce the number of false positives using a threshold. (The tradeoff is that this increases the number of false negatives.)
The following parameters can be tuned to optimize the classifier:

- ❖ Classification threshold (see above)
- ❖ Training parameters
  - Training length (number of epochs)
  - Batch size (how many images to look at once during a single training step)
  - Solver type (what algorithm to use for learning)
  - Learning rate (how fast to learn; this can be dynamic)
  - Weight decay (prevents the model being dominated by a few “neurons”)
  - Momentum (takes the previous learning step into account when calculating the next one)
- ❖ Neural network architecture
  - Number of layers
  - Layer types (convolutional, fully-connected, or pooling)
  - Layer parameters (see links above)
- ❖ Preprocessing parameters (see the Data Preprocessing section)

During training, both the training and the validation sets are loaded into the RAM. After that, random batches are selected to be loaded into the GPU memory for processing. The training is done using the Mini-batch gradient descent algorithm (with momentum).
In contrast to this, inference (that is, inference in the Android app) is done using only the CPU, because TensorFlow doesn’t support smartphone GPUs.

Benchmark

I tried three different architecture and selected the model with the best accuracy.

Methodology

Data Preprocessing

The image we given as input to the model process the image before passing it to the network. In this step, the image is normalized, resize, crop the image to speed up the training. Then the image is converted into tensors.
In this step, there are some parameters we can choose, like the parameter for resizing, center crop, and normalize methods.

Implementation

In this step, the model is designed. We will be using 5 convolutional layers for this project and 3 fully connected layers. A Dropout layer is added at the end so that it will reduce the complexity of the network.
5 convolutional layers are created with max-pooling layers in between them to learn a hierarchy of high-level features. Max pooling layer is added to reduce the dimensionality. Dropout was used along with the flattening layer before using the fully connected layer to reduce overfitting and ensure that the network generalizes well. The number of nodes in the last fully connected layer was set up as 133. Relu activation function was used for most of the layers.

The below given are the steps followed in the implementation part.

- Load both the training and validation images into memory, preprocessing them as described in the previous section.
- Define network architecture and training parameters.
- Define the loss function, accuracy.
- Train the network, logging the validation/training loss and validation accuracy.
- If the accuracy is not high enough, retry these steps, else save and freeze the trained network.

```
trainLoader = torch.utils.data.DataLoader(train_dataset,
                                          batch_size=128,
                                          shuffle=True,
                                          num_workers=0)

validLoader = torch.utils.data.DataLoader(valid_dataset,
                                          batch_size=128,
                                          shuffle=True,
                                          num_workers=0)

testLoader = torch.utils.data.DataLoader(test_dataset,
                                          batch_size=64,
                                          shuffle=False,
                                          num_workers=0)
```

This is how the images are loaded.

```
class Net(nn.Module):
      ### TODO: choose an architecture, and complete the class
      def __init__(self):
```

```
        super(Net, self).__init__()
        ## Define layers of a CNN
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(256, 512, kernel_size=3, padding=1)
        self.conv5 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(25088, 512)
        self.fc2 = nn.Linear(512, 512)
        self.fc3 = nn.Linear(512, 133)
        self.dropout = nn.Dropout(0.5)

        def forward(self, x):
        ## Define forward behavior
        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 = self.pool(F.relu(self.conv4(x)))
        x = self.pool(F.relu(self.conv5(x)))
        x = x.view(-1, 25088)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x
```

This is the model architecture.

Model Evaluation and Validation

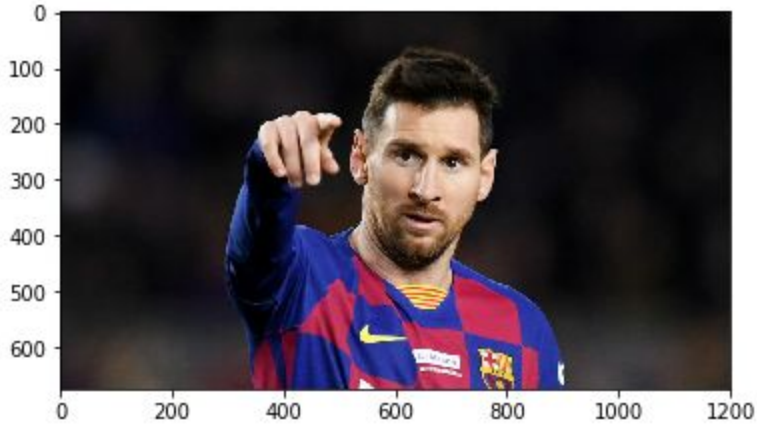
During development, a validation set was used to evaluate the model. The final architecture and hyperparameters were chosen because they performed the best among the tried combinations. The following list:

- The details about the network can be obtained from the above figure.

The validation set is used for hyperparameter tuning and the test set is used to test the performance of the model. After completing the training, and testing of the model we got an accuracy of 77%

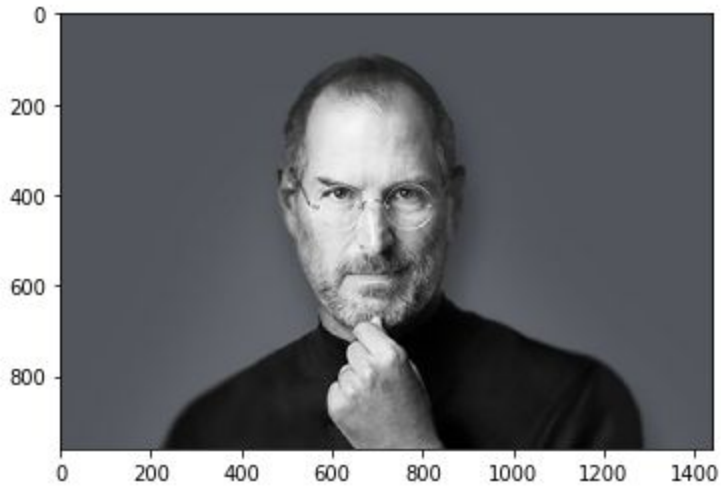
At last, I used 6 images to test the model and found that the model correctly classified the images

```
Hello Human!
```




```
You look like a ... Dachshund
```

```
Hello Human!
```




```
You look like a ... Italian greyhound
```

Reflection

The process used for this project can be summarized using the following steps:

- An initial problem and relevant, public datasets were found
- The data was downloaded and preprocessed
- The classifier was trained using the data (multiple times, until a good set of parameters, were found)

Out of these steps, the 3rd step bit hard compared to other steps since it contains network

construction and training. But after many tries, I found a good pair of parameters to train the model and got a good model at the last.

Improvement

Even though I trained the model to predict good results, It doesn't mean that we can't further improve the model. Below are some steps that could result in a better model.

- Increase the number convolutional layer
- Adding more dropout layer
- Convolutional Network Parameters
- Altering the Multi-Layer Feed-Forward NN at the end of the network

Conclusion

The model for detecting Dog breed from the input image if constructed with an accuracy of 77%