Machine Learning Engineer Nanodegree

Capstone Proposal

Paresh Pradhan
May 13th, 2018

NotPineapple Image Classifier

Domain Background

Image Classification - It is the task of assigning an input image one label from a fixed set of categories. This is one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications. Many other Computer Vision tasks such as object detection and/or segmentation can be reduced to image classification.

Data-driven approach - Unlike writing an algorithm for, for example, sorting a list of numbers, it is not obvious how one might write an algorithm for identifying cats in images. Therefore, instead of trying to specify what every one of the categories of interest look like directly in code, we're going to provide the computer with many examples of each class and then develop learning algorithms that look at these examples and learn about the visual appearance of each class. This approach is referred to as a data-driven approach, since it relies on first accumulating a training dataset of labeled images.

Image classification pipeline - The task in Image Classification is to take an array of pixels that represents a single image and assign a label to it. Our complete pipeline can be formalized as follows:

- Input: Our input consists of a set of N images, each labeled with one of K different classes. We refer to this data as the training set.
- Learning: Our task is to use the training set to learn what every one of the classes looks like. We refer to this step as training a classifier, or learning a model.
- Evaluation: In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare the true labels of these images to the ones predicted by the classifier. Intuitively, we're hoping that a lot of the predictions match up with the true answers (which we call the ground truth).

Convolutional Neural Networks (CNN) - They are very similar to ordinary Neural Networks. They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks for learning regular Neural Networks still apply.

So what does change? ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

Source: http://cs231n.github.io/

Personal Motivation - This project was inspired from the following:

- NotHotdog app created in the TV series - Silicon Valley
- NotSanta project by Adrian Rosebrock (https://www.pyimagesearch.com/2017/12/11/image-classification-with-keras-and-deep-learning/)

Problem Statement

Our goal is to take images of fruits as input and determine whether that fruit is Pineapple or not. The project is a Binary Classification between Pineapple and NotPineapple labels. The output is the testing accuracy of the model.

Datasets and Inputs

The dataset used in this project was sourced from the Fruits-360 dataset on Kaggle (https://www.kaggle.com/moltean/fruits).

The dataset has been zipped into a single file - Training_Testing_Images.zip

The contents of the file are as follows:

- Training_Images
  - Pineapple - 490 images (100x100 pixels) taken directly from the Fruits-360/Training directory
  - NotPineapple - 490 images (100x100 pixels) randomly selected from all other fruits in the Fruits-360/Training directory
- Testing_Images
  - Pineapple - 166 images (100x100 pixels) taken directly from the Fruits-360/Validation directory
  - NotPineapple - 166 images (100x100 pixels) randomly selected from all other fruits in the Fruits-360/Validation directory

The images were extracted from the Fruits-360 dataset using shell commands. All images are 100x100 jpg images.

Solution Statement

We are going to use Convolutional Neural Networks to train a model on the training images and then classify the test images into one of two categories - Pineapple or NotPineapple. Finally we will calculate the accuracy of the predictions.

Benchmark Model

We are calculating testing accuracy for the model. Since this is a Binary Classification, simple random selection will give us ~50% accuracy. Thus, we can take 50% testing accuracy as our benchmark.

Evaluation Metrics

$$accuracy = \frac{Number\ of\ matches\ between\ predicted\ labels\ and\ true\ labels}{Number\ of\ input\ images}$$

Project Design

1. Create Dataset

- The input images were extracted from Kaggle's Fruits-360 dataset.
- The Pineapple and NotPineapple images were copied to their respective directories to create a dataset with the structure shown above.

2. Create Model File

- Create one or more classes for different CNN architectures - LeNet (AlexNet, Custom architecture).
- Use keras library with tensorflow as backend.

3. Train Model

- Read images from Training_Images directory
- Preprocess images
  - Normalize pixel intensities to range [0,1]
  - Resize images based on the architecture
  - Convert images to numpy array
- Extract image labels from the directory paths
- Divide the training data into Train & Test splits - 75% and 25% respectively
- One-Hot-Encode the labels
- Construct image generator for data augmentation
- Initialize the model using the architecture classes from Model File
- Initialize the optimizer
- Compile the model with:
  - loss = binary crossentropy

    - metrics = accuracy
- Train the model with augmentation and generate a history object
- Use the history object to plot a graph of training/validation loss/accuracy
- Save trained model to disk

4. Test Model

- Load in the trained model file
- Read images from Testing_Images directory
- Preprocess images
    - Normalize pixel intensities to range [0,1]
    - Resize images based on the architecture
    - Convert images to numpy array
- Extract image labels from the directory paths
- Make predictions for each image
- Compare the predictions with the true labels extracted from the image paths
- Calculate Model's Testing accuracy.