Machine Learning Engineer Nanodegree
Capstone Project Proposal

Pavlos Sakoglou
November 13, 2017

Image Classification using Deep Learning with Support Vector Machines

Domain Background

Convolutional Neural Networks (CNNs) are very similar to regular Neural Networks and aim to find an optimal predictive model that assigns the input variable to the correct label. CNNs are a subset of Supervised Learning class of algorithms which mimics how the human brain works, by creating connected layers of nodes that model neurons and their connectivities.

In contrast to the Multilayer Perceptron Architecture (MLP) that uses fully connected network layers, CNNs receive as input a matrix of features (pixels), encoded as greyscale values, and create connections that are informed by the 2-D structure of the matrix. That is to say, a CNN does not need to provide information of the entire feature space to all hidden layer nodes, but instead it breaks the input matrix into regions and then connects each region to a single hidden node. With this regional breakdown and assignment of small local groups of features to different hidden nodes, CNNs are performing very well for image recognition tasks.

In his paper "Deep Learning using Linear Support Vector Machines", Yichuan Tang from University of Toronto argued that there is universal performance improvement of the network by replacing the softmax function of the penultimate network layer with a linear Support Vector Machine classifier. In this project we will investigate performance differences in image classification using this technique, and confirm performance improvement versus performance of the standard CNN model.

Problem Statement

With the immense usage of smart phones in developed countries, people are sharing information via various types of messenger applications in unimaginable volumes. A natural and unfortunate consequence of this is message abuse in written and visual form with written texts and images. In this project we aim to partially tackle this societal issue using deep learning with SVC.

The idea is to develop and train a smart algorithm that takes an image from a message as input and detects if the image contains nudity or not. That is, the algorithm will classify the picture as "**Nude**" or "**Safe**". The receiver of the picture will have the chance to either block or open the message; having seen the class of the newly arrived image. The goal of this application is to reduce message harassment and prevent unwanted information sharing.

Datasets and Inputs

In order to teach the CNN what it means for an image to be labeled as "**Nude**" or "**Safe**", we need to assemble datasets of nude and non-nude images in JPG format, and train the network with them. Moreover, we will split the assembled datasets into an 80% training data and 20% testing data, so that we can evaluate the performance on various input.

In order to achieve high accuracy with our model we will use about two thousand data points (images) for each class. There are several websites that provide raw image samples for our purposes, and thus we will use a technology called *Content Grabber* (https://contentgrabber.com/), which allows us to download content from websites in the format of our choice. These datasets will then be used by our algorithm which will process and adjust the data in a uniform manner that maximizes the speed of our algorithm. The data then will be encoded and ready to be used as input space.

In particular, the "**Nude**" dataset will consist of pictures the expose a lot of skin for both women and men of all races, and similarly, the "**Safe**" dataset will consist of pictures of both women and men of all races that do not reveal a lot of skin, but instead wearing clothes.

Solution Statement

As soon as the training of the CNN with SVC is finished, we can test the model with new input. That is, we are going to use the testing part of the data and confirm accuracy.

A successfully trained algorithm should be able to receive a new image as input and display a warning message as per the class it should belong. In case the predicted class label of the new input is "**Safe**", the warning message will read "Safe Content" and the user will normally view the content. Alternatively, the warning message will read "Image Contains Nudity" and the content will be temporarily blurred. The user will have the option to either un-blur it or disregard it; having seen the warning message.

As a result, the user will not get harassed or disturbed by surprise unwanted content, unless he/she choices to.

Benchmark Model

Since this project requires high performance and accuracy, we need to design a CNN architecture for image recognition. We will need to tune the architecture's hyper-parameters to optimize the process and avoid overfitting or other training issues. Finally, we will rely on *transfer learning* and bottlenecking features to advantage of past high-end work of data professionals and engineers.

The CNN architecture for image processing will be enhanced and tuned by hyper parameter settings suggested in the Tensorflow and Keras framework documentation. For example, popular bottleneck feature models that will be considered include: InceptionV3, ResNet-50, and Xception.

To fully construct and properly tune the model, we will consider the documentation of these features and find an optimal set of parameters that maximize the likelihood of correctness of each prediction.

Evaluation Metrics

In order to evaluate the model's performance we will rely on four main metrics, namely Accuracy, Precision, F_b-Score, and Recall. Notice that this application serves as an inappropriateness filter and thus we prefer any potential errors to falsely predict an image as "**Nude**", than to falsely predict it as "**Safe**".

In other words, we want to penalize more for False Negatives, which will be described by a high Recall value. To clearly demonstrate how these evaluation metrics are eligible to determine our model's efficiency, let's take a closer look to them:

$$\text{Accuracy} = \frac{\text{True Negatives} + \text{True Positives}}{\text{All predictions}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positive} + \text{False Negatives}}$$

$$F_b\text{-Score} = (1+\beta^2)\frac{\text{Precision} * \text{Recall}}{\beta^2\text{Precision} + \text{Recall}}$$

All metrics concern both the benchmark model and its solution, and will be used in the testing process. We will compare results between the different parameter choices and architectures, and we will pick those which maximize the Accuracy and the Recall metrics. Notice that since we value Recall more than precision here, for the F_b-Score we will choose a $\beta > 1$.

Project Design

Since the input of the algorithm is an image, we need to design the application as follows:

1. Load the training dataset.
2. Pre-process and encode the input separately
3. Build the network architecture using an SVC at the penultimate layer and determine optimal set of parameters. Facilitate transfer learning.
4. Compile and train the model.
5. Evaluate model using the testing set and metrics.
6. Repeat steps 3-6 until we achieve over 90% accuracy.
7. Build algorithm that uses the above trained model. The algorithm takes for input an image and returns a tuple of the form {image, warning message}

Once we are satisfied with the results we are getting from the testing set, we will further test the application with more random input, and finally we will deploy the code with the optimal parameters as a mobile application. The application's input and output will then be as follows:

1. **Input:** {image or image path }
2. **Output:** {output-formatted image, warning message}.

The main technologies to be used in this project will be Tensorflow, Keras, and standard Python libraries. Supplementary technologies might include open source code and references taught throughout the Udacity Machine Learning Engineer Nanodegree.