Machine Learning Engineer Nanodegree
Capstone Project Proposal

Pavlos Sakoglou
November 21, 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.

We view this problem as a supervised classification where we train the model with a large dataset of nude and non-nude pictures of people by providing the correct labels, and then test the model with a newly received image and learn its class.

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, thus preventing harassment and unwanted information sharing.

Datasets and Inputs

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. The Tensorflow model to be used (see below) expects to get square 224x224 RGB images as input, thus the dataset must be pre-processed and adjusted before the training begins.

In order to achieve high accuracy with our model we will use two thousand data points (images) for each class, that is four thousand pictures overall. We will split the assembled datasets into an 80% training data and 20% testing data, so that we can evaluate the performance on various inputs.

The "**Nude**" dataset will consist of pictures that expose a lot of skin for both women and men of all races, and 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. Moreover, the size of the training data that are labeled "**Nude**" will be the same of the training data that are labeled "**Safe**" in order to achieve a balanced output for both classes.

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.

Solution Statement

Once we have assembled the data, we will write a function to load the datasets in the program and separate them in training and testing subsets. We will write another function to convert the data subsets into the appropriate dimensions and tensors, by using the Keras package *keras.preprocessing.image*.

When the data are pre-processed and converted into 224x224 tensors with RGB-scaled values, we will construct the network architecture using various Keras packages, such as keras.layers.Conv2D and keras.layers.MaxPooling2D. Finally we will compile and train the model, keeping track of the optimal set of coefficients using the Keras package keras.callbacks.ModelCheckpoint. We will experiment with several combinations of hyper parameter values, batch sizes, model optimizers, number of epochs, and network architectures.

Keep in mind that the penultimate layer will use an SVM classifier that will require separate tuning. In particular, we will use the output of the penultimate layer of the neural network as a feature vector to train the SVM classifier, from which we will derive the end results.

Finally, we will write a function that takes an image or an image path as input and incorporates the data pre-processing procedure described above. The function will also use the optimal model coefficients obtained via a prediction function call, and eventually will return a tuple of a processed image (blurred or not) and a warning message.

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.

Benchmark Model

The main assumption of this project is that a CNN with or without a SVM classifier will perform better than random. As we saw in the data exploration part above, the two class datasets have similar size, thus it is expected to assume that an arbitrary network architecture will perform with about 50% accuracy, recall, and precision. That is, for a random network choice, there is about 50% probability for a new image to be classified as **Nude** or **Safe**.

Evaluation Metrics

In order to evaluate the model's performance we will rely on three main metrics, namely 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:

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

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

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

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 Recall metric. Notice that since we value Recall more than precision here, for the F_b-Score we will choose a $\beta > 1$.

Project Design

The overall design layout of the training and testing part of the algorithm is summarized below, highlighting the key parts of the design process. Notice that details of prediction functions, code organization and refactoring, supplementary library usage and other components of the application mentioned above (e.g. OpenCV cascading, etc.) are ommited for simplicity.

1. Load and pre-process the training dataset using Keras
   (a) keras.preprocessing.image.load_img() loads RGB image as PIL.Image.Image type
   (b) keras.preprocessing.image.img_to_array() converts PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
2. Build the network architecture, compile, and determine optimal set of parameters.
   (a) model = keras.models.Sequential() will import the Sequential model to be used

(b) model.add() adds layers to the network such as Conv2D(), MaxPooling2D(), and GlobalAveragePooling2D() with parameters of our choice

(c) model.compile() compiles the model with an optional optimizer such as Adam or rmsprop and a loss function like categorical entropy

(d) keras.callbacks.ModelCheckpoint() and model.fit() will train the model and save the optimal set of parameters obtained in a file

3. Evaluate model using the testing set and metrics.

   (a) model.load_weights() imports the optimal set of parameters obtained by the training

   (b) Iterate in the testing set and classify all images. Then compare results with their true label

   (c) Calculate the confusion matrix values and use the results to compute the evaluation metrics.

4. Repeat steps 3-6 until we achieve over 90% accuracy by fine tuning the model

   (a) Redefine model architecture (number of layers and nodes)

   (b) Change optimizers

5. Build a function that uses the above trained model. The algorithm takes for input an image and returns a tuple of the form {image, warning message}

```
def NudityDetector(img_path) :
    # Preprocess data, and predict the class of the input
    # print results
    return (processedImage, warningMessage)
```

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.