Machine Learning Engineer Nanodegree – Capstone Proposal

Proposal

Build an application that can interpret number strings in real world images.
Reference: This project is taken from the deep learning course @ Udacity.

Domain Background

Identification of numbers in images has multiple real world applications like

- Automatic detection of postal codes that will enable sorting posts for different regions into different bins without human intervention. Automatic number recognition can also be used in banks for cheque processing.
- We can apply this technique to automatic number plate recognition (ANPR) which can be used by police forces for law enforcement purposes (for example, to check whether a vehicle is registered or not)
- Recognizing numbers in photographs captured at street level is used in map making.

References:

[1] https://en.wikipedia.org/wiki/Automatic_number_plate_recognition
[2] http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf

Problem Statement

We are given a real world image which contains a sequence of numbers. These numbers can be things like street view numbers, vehicle license plate numbers or postal codes. The problem is to train a model that can recognize the number present in the image. The project will be implemented as a python script.
The script should be able to accurately (i.e with low misclassification error) detect the number present in the image (assuming that we test the model on different images that contain many different strings of numbers in them).

Datasets and Inputs

For training and testing, we will use the publicly available Street View House Numbers (SVHN) dataset, which is a large scale dataset of house numbers in Google Street View images.

The SVHN dataset requires minimum preprocessing and it is similar to the MNIST dataset, but contains significantly more number of images (around 600,000) captured from real world images.

References:

[1] http://ufldl.stanford.edu/housenumbers/
[2] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*

Solution Statement

The solution to this problem will be to build a supervised model using the dataset. Supervised because, the dataset contains images with the actual labels (i.e number present in the label) and we can use this information for learning a model and then evaluate the model on the testing set. Specifically, we will split the dataset into training, validation and testing subsets and train the model using the training dataset and evaluate the performance of the model on the validation datasets so that we can tune the hyper-parameters of the model. Finally we will report the performance of the model on the testing set.
Alternatively, we can also evaluate the performance of the model using cross validation.
For the dataset, instead of using the full-fledged dataset initially, we can consider starting off with synthetic dataset (i.e 4 digits taken from MNIST stitched together) and then after this model works reasonably well, consider switching to the more complex SVHN dataset.

Benchmark Model

As a benchmark model, we consider the model implemented in the paper
Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks by Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet.

This uses the DistBelief implementation of deep neural networks to train large, distributed neural networks. This model uses a convolutional neural network with the best performance at 11 hidden units. This achieves an accuracy of over 96% on the SVHN dataset.

DistBelief network is described in *Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q., Mao, M., Ranzato, M., Senior, A., Tucker, P.,Yang, K., and Ng, A. Y. (2012). Large scale distributed deep networks. In NIPS'2012*

Evaluation Metrics

How well the model performs can be quantified by the score of the model on the testing set. We will use a simple evaluation metric like misclassification error. i.e for a particular image to be classified correctly, all the digits in the image should be labelled correctly in order. Partial correctness of the digits in an image would be considered as misclassification. We can also consider the digit level accuracy (i.e how many digits in the image the model got right) as another evaluation metric.

Sometimes when there is class imbalance in the data, classification accuracy alone cannot give a good performance metric. So in this case I would also consider a metric like F1 score which uses both precision and recall to overcome class imbalance issues.

Project Design

Since this is a classification problem we could consider approaches like Logistic Regression, SVM or Neural networks.

Logistic regression is a simple model in the sense that it is equivalent to an ANN without any hidden units and with a sigmoidal activation unit. Also LR cannot capture non-linear relationships between the dependent and independent variables. So LR will only work well if the classes are linearly separable. For our dataset, the features are pixels in the images and could result in highly non-linear classes. Also there could be co-relation between the independent variables and this cannot be

captured by LR models. Note that ANNs can capture both non-linear relationships and correlation via hidden units.

With regards to comparison to SVM's, note that we don't have just 2 classes of outputs to classify, but rather a wide range of outputs; Possibly SVMs would work ok if the image consists of only 1 digit, so in that case we would have 10 SVMs one for each digit. But in the case where the image can have n digits, it is not clear how an SVM can be trained for the whole set of digits in an image. Maybe we should localize the digits in the image and then use the multi class classification using the SVM. And also in the paper http://yann.lecun.com/exdb/publis/pdf/bengio-lecun-07.pdf it is argued that SVM architectures are shallower compared to the deep neural network models. Shallower meaning "*They consist of one layer of fixed kernel functions, whose role is to match the incoming pattern with templates extracted from a training set, followed by a linear combination of the matching scores. Since the templates are extracted from the training set, the first layer of a kernel machine can be seen as being trained in a somewhat trivial unsupervised way. The only components subject to supervised training are the coefficients of the linear combination*"

In http://www.kdnuggets.com/2016/04/deep-learning-vs-svm-random-forest.html, it is argued that we have to worry less about feature engineering when we use deep neural nets compared to classifiers like SVMs (but for this to be accomplished in a small amount of training time and limited computing power, we need to ensure that the number of features is reasonably small to begin with. We can reduce the dimension of the original image by resizing or gray scaling the image)

So even though deep neural networks take a lot of time to train and require a relatively large number of training examples, considering that this is a complex problem and we really need not worry about designing great features upfront and let the hidden layers do the feature abstraction part, I would go with deep neural net (also we can experiment with convolutional neural nets apart from the deep neural nets).

Following is a theoretical sequence of steps I would follow (of course some of these steps would be enhanced in more detail as the solution evolves; but following is a rough outline):

1. Dataset Selection: First experiment with a smaller synthetic dataset before going to the larger SVHN dataset. The synthetic dataset can be constructed by either using single digits from MNIST or stitching together a few digits from MNIST dataset
2. Data Preprocessing: since the data will be very high dimensional and it will take very long to train our NNs in this case, we can reduce the number of features by gray scaling or by resizing the image to a smaller dimension. We could also use PCA for dimensionality reduction to convert the original data points to a reduced feature space. This will help in requiring fewer data points for training as well as faster training.
3. Experiment with one/both of the following architectures by training and evaluating the models and repeating the process until we get good results
    a. Deep neural network: Experiment with different number hidden layers and neurons in each layer; Also experiment with the activation function: sigmoid, tanh, ReLU etc.
    b. Convolutional neural network: Experiment with sharing of weights between different softmax classifiers at the final layer.

4. During training (step 3), we need to ensure that the model does not over fit. We will do this by experimenting with different mechanisms like L1 regularization, L2 regularization and Dropout.
5. During training (step 3), we need to be able to tune the hyper parameters of the model like number of hidden layers for example. We will do this by using either a holdout cross validation or k-fold cross validation. We will use randomized grid search for tuning the parameters.
6. At regular iterations of training, we will observe the performance of the model using learning and validation curves.
7. We will also use gradient checking mechanism to debug the neural network.
8. Finally, the best model as per the validation will be used and the test set will be run on this model and the final performance of the model will be reported.

References:
[1] https://gdrenice2015.sciencesconf.org/53551/document