Machine Learning Engineer Nanodegree

Capstone Project: Street View House Numbers Recognition using Deep Convolutional Neural Networks

1. DEFINITION

Domain Background

Optical character recognition (OCR) is highly challenging, but can help solve many problems. One of OCR problems is to identify house number digit classification based on photos from Google Maps. The difficulty arises due to the size of internal features, such as fonts, colors, styles, orientations, and character arrangements, as well as external features, such as lighting, shadows, resolution, motion, and focus blurs.

Traditional approaches to solve this problem typically separate out the localization, segmentation, and recognition steps. In this project, we propose using Convolutional Neural Networks[1]. Like most neural networks, they contain several filtering layers with each layer applying an affine transformation to the vector input followed by an elementwise non-linearity. This makes convolutional networks computationally efficient, allowing them to scale to large images.

Problem Statement

Street View House Numbers (SVHN) is a similar problem as MNIST digits recognition. Given an image, the task is to identify the number in the image. It is extremely important to have at least human level accuracy. Users of maps find it very time consuming and frustrating to be led to the wrong location, so it is essential to minimize the amount of incorrect transcriptions entered into the map.

Evaluation Metrics

We are going to use “Accuracy” as our evaluation metrics. Accuracy is defined in percentage point and calculated based on how well our model can predict the actual result.

---

[1] Fukushima, 1980; LeCun et al., 1998

We define accuracy method in our model as follows:

```
In [14]: def accuracy(predictions, labels):
           return (100.0 * np.sum(np.argmax(predictions, 1) == np.argmax(labels, 1))
                   / predictions.shape[0])
```

II. ANALYSIS

Data Exploration

The Street View House Numbers (SVHN)[2] dataset is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

SVHN has 10 classes (1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10). The dataset consists of 73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training data. And comes in two formats: (i) original images with character level bounding boxes, and (ii) MNIST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides).

Exploratory Visualization

Example of SVHN dataset:

[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*

The format and size of Matlab files for the cropped SVHN images is as follows:

```
# Load matlab files using scipy.io library
import scipy.io as sio

train_data = sio.loadmat('train_32x32.mat', variable_names='X').get('X')
train_labels = sio.loadmat('train_32x32.mat', variable_names='y').get('y')
test_data = sio.loadmat('test_32x32.mat', variable_names='X').get('X')
test_labels = sio.loadmat('test_32x32.mat', variable_names='y').get('y')
# extra_data = sio.loadmat('extra_32x32.mat', variable_names='X').get('X')
# extra_labels = sio.loadmat('extra_32x32.mat', variable_names='y').get('y')

print("train data: ", train_data.shape, train_labels.shape)
print("test data: ", test_data.shape, test_labels.shape)
# print("extra data: ", extra_data.shape, extra_labels.shape)
```

```
train data:  (32, 32, 3, 73257) (73257, 1)
test data:  (32, 32, 3, 26032) (26032, 1)
```

- Each image is cropped into 32x32 pixels format with 3 channels (RGB).
- Train data consists of 73,257 images and labels.
- Test data consists of 26,032 images and labels.

10 samples of train data in RGB image including the label on top:

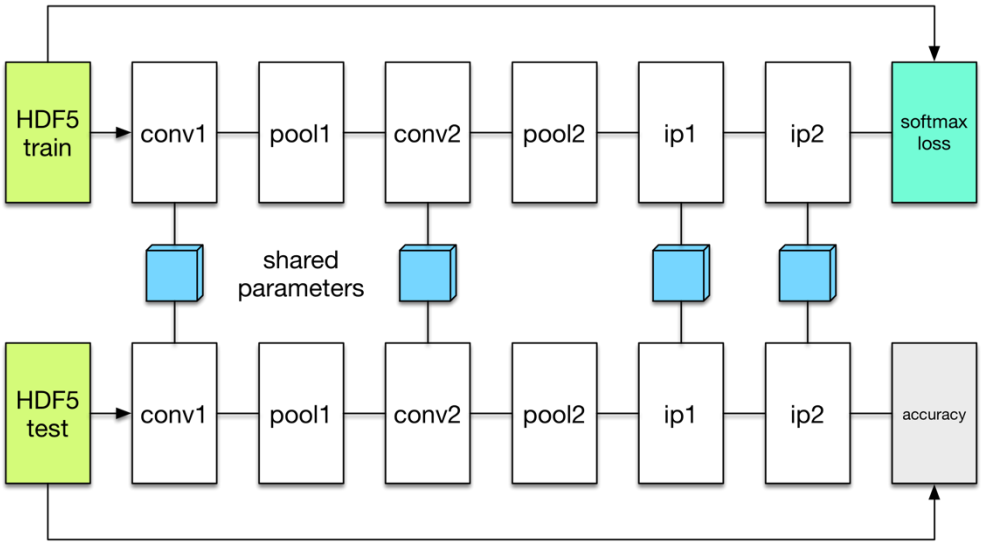
10 samples of train data in grayscale image:

Algorithms and Techniques

The Street View House Number Dataset (SVHN) is similar as MINST Dataset. Machine learning researchers have found that a deep convolutional neural network (also known as Covnet) can be achieve reasonable performance on hard visual recognition tasks.

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech[3].

Example of a CNN architecture:



[3] Yann LeCun, Yoshua Bengio & Geoffrey Hinton, Deep Learning, Nature May 28, 2015

Techniques:

- Convolutional layer: consists of a set of kernels that compute the entries of the filters and inputs and use RELU to activate some specific features.
- Max_pooling: helps to reduce the memory usage after the activations are triggered by the convolutional layer.
- Fully connected: this layer connects all the neurons after several convolutional layers and pooling.
- Dropout: helps to reduce overfitting by eliminating some neurons during training iterations.
- Softmax: is the probability of categorical distribution to predict the output.
- AdamOptimizer optimization: uses moving-averages of the parameters. This optimizer controls the learning rate and it will converge without fine tuning.
- Exponential decay learning rate: to lower the learning rate as the training progresses.

Benchmark

According to this website[4], the top 5 SVHN dataset benchmark are:

Method	Summary	Result
Generalizing Pooling Functions in CNN: Mixed, Gated, and Tree[5]	Seek to improve deep neural networks by generalizing the pooling operations.	1.69%
Competitive Multi-scale Convolution[6]	Introduce a new deep convolutional neural network (ConvNet) module that promotes competition among a set of multi-scale convolutional filters.	1.76%
Recurrent CNN for Object Recognition[7]	Demonstrate the advantage of the recurrent structure over purely feed-forward structure for object	1.77%

[4] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#5356484e

[5] Chen-Yu Lee, Patrick Gallagher, Zhuowen Tu, AISTATS 10 Oct 2015

[6] Zhibin Liao, Gustavo Carneiro, asXiv 18 Nov 2015

[7] Ming Liang, Xiaolin Hu, CVPR 2015

	recognition.	
Batch-normalized Maxout Network in Network[8]	Employ maxout MLP to learn a variety of piecewise linear activation functions and to mediate the problem of vanishing gradients that can occur when using rectifier units.	1.81%
Deeply-Supervised Nets[9]	Simultaneously minimizes classification error while improving the directness and transparency of the hidden layer learning process.	1.92%

III. METHODOLOGY

Data Preprocessing

1. Normalize train and test data
2. Apply one-hot encoding to train label

```
train data:  (73257, 32, 32, 3) (73257, 10)
test data:  (26032, 32, 32, 3) (26032, 10)
sample one-hot encoding train label:  [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
```

3. Split train dataset to create a subset of valid dataset
4. Shuffle train, test, and valid data to remove the possibility SVHN images may be in sequential order

```
train data:  (65931, 32, 32, 3) (65931, 10)
test data:  (26032, 32, 32, 3) (26032, 10)
valid data:  (7326, 32, 32, 3) (7326, 10)
```

5. Convert RGB (3 channels) to grayscale (1 channel)

[8] Jia-Ren Chang, Yong-Shen Chen, arXiv 2014

[9] Chen-Yu Lee*, Saining Xie*, Patrick Gallagher, Zhengyou Zhang, Zhuowen Tu, arXiv, 2014

```
# Convert RGB to Greyscale
image_size = 32  # Pixel width and height.
pixel_depth = 255.0  # Number of levels per pixel.

def im2gray(image):
    '''Normalize images'''
    image = image.astype(float)
    # Use the Conversion Method in This Paper:
    # [http://www.eyemaginary.com/Rendering/TurnColorsGray.pdf]
    image_gray = np.dot(image, [[0.2989],[0.5870],[0.1140]])
    return image_gray

train_data_c = im2gray(train_data)[:,:,:,:]
test_data_c = im2gray(test_data)[:,:,:,:]
valid_data_c = im2gray(valid_data)[:,:,:,:]

print("train data: ", train_data_c.shape, train_labels.shape)
print("test data: ", test_data_c.shape, test_labels.shape)
print("valid data: ", valid_data_c.shape, valid_labels.shape)
```

```
train data:  (65931, 32, 32, 1) (65931, 10)
test data:  (26032, 32, 32, 1) (26032, 10)
valid data:  (7326, 32, 32, 1) (7326, 10)
```

6. Save preprocessing dataset into pickle file

```
# Create pickle file to save processed data
pickle_file = 'SVHN_single.pickle'

try:
  f = open(pickle_file, 'wb')
  save = {
    'train_data': train_data,
    'train_labels': train_labels,
    'test_data': test_data,
    'test_labels': test_labels
    }
  pickle.dump(save, f, pickle.HIGHEST_PROTOCOL)
  f.close()
except Exception as e:
  print('Unable to save data to', pickle_file, ':', e)
  raise

statinfo = os.stat(pickle_file)
print('Compressed pickle size:', statinfo.st_size)
```

```
Compressed pickle size: 1133720175
```

Implementation

We implemented 3 CNN models and compare the results as follows:

- Model 1: Conv1/Pool1/Conv2/Pool2/FC1/dropout/FC2/softmax CNN architecture with grayscale images

- Model 2: Conv1/Pool1/Conv2/Pool2/FC1/dropout/FC2/softmax CNN architecture with RGB images
- Model 3: Conv1/Pool1/Conv2/Pool2/Conv3/Pool3/FC1/dropout/FC2/softmax CNN architecture with RGB images

Links to Github repos (using Jupyter Notebook):
- Model 1: SVHN+Capstone_single+digit-greyscale.ipynb
- Model 2: SVHN+Capstone_single+digit-RGB.ipynb
- Model 3: SVHN+Capstone_single+digit-RGB-deeper+layer.ipynb

Refinement

To refine our data, I used the following techniques:
- Set 50% dropout to allow more degree of freedom

```
# Dropout
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

- Set exponential decay learning rate

```
# Optimizer
global_step = tf.Variable(0)
learning_rate = tf.train.exponential_decay(1e-4, global_step=global_step, decay_steps=10000, decay_rate=0.97)
```

- Use AdamOptimizer instead of Gradient Descent

```
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y_conv, y_))
train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy, global_step)
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

IV. RESULTS

Model Evaluation and Validation

Model	Test Accuracy
Conv1/Pool1/Conv2/Pool2/FC1/dropout/FC2/softmax CNN architecture with grayscale images	17.04%
Conv1/Pool1/Conv2/Pool2/FC1/dropout/FC2/softmax CNN architecture with RGB images	88.93%
Conv1/Pool1/Conv2/Pool2/Conv3/Pool3/FC1/dropout/FC2/softmax CNN architecture with RGB images	86.90%

Justifications

1. The best model with the highest accuracy is model 2 with 88.93% test accuracy.
2. Surprisingly, the test accuracy for the same CNN architecture using the greyscale images is very low, ie. 17.04%. The low quality of greyscale (1 channel) images could be the root cause of the low test accuracy. The 10 sample images below are unreadable by human and computer.

3. Despite the deeper CNN architecture, model 3's test accuracy of 86.90% is no better than that for model 2. The deeper networks don't necessarily translate into better test result. In fact, we have to wary that deeper networks may cause overfitting problem, where the model is good at predicting training data, but fail the test data.

V. CONCLUSIONS

Reflections

In this project, we examine SVHN dataset and apply convolutional neural network to predict the digits from the images. Researchers have experimented different CNN techniques and achieved ~98% accuracy, which is higher than human's accuracy at ~90%. In this project, we tried 3 different models using SVHN dataset and the best result is 88.93%, which is close to human's level of accuracy. While the result is satisfactory, it's still below the benchmark which apply more sophisticated techniques and powerful GPUs.

In this project, we also learned that deeper networks do not necessarily result in better accuracy. It makes sense because deeper networks will learn training data better, but not test data. Therefore, we may end up with an overfitting model.

Improvements

These are some of the improvements we can think of to solve this problem:

- Due to hardware and computation power limitations, it took a long time to run an epoch of training data. Therefore, we couldn't test more sophisticated models, such as ResNet, GoogLeNet, VGGNet. We can improve the results if we use higher GPU power by removing the memory bottleneck.
- We can also experiment with image preprocessing. In this project, we use im2gray method to smoosh the 3-channel RGB image into 1 channel. There are many other preprocessing techniques that can be applied to improve the result, such as edge detection, Gaussian blurring, rotations and contour selections. Scikit-learn, Numpy, OpenCV, and Matplot libraries have the built-in functions to manipulate the images to make the image more readable by computers.
- Tensorflow library can now be embedded into mobile devices and Raspberry-pi IoT

devices. We can implement our trained model into mobile and IoT devices with camera to detect images and videos and create an AI application using computer vision using real-time data.