**1. Definition**

**Large-scale Identification of Multiple Digits**

**From Real-World Images with Deep Convolutional Networks**

Ritchie Ng

National University of Singapore

ritchieng@u.nus.edu

**1.1 Overview**

This project explores how deep convolutional neural networks (ConvNets) can be used to effectively identify a series of digits from real-world images that are obtained from “The Street View House Numbers (SVHN) Dataset” (Netzer, Wang and Coates). ConvNets have evolved dramatically every year since the inception of the ImageNet Challenge in 2010.

A proverbial ConvNet structure is the “LeNet-5” that has relatively few layers of convolutions, poolings, and full connections (LeCun, Bottou and Bengio). Subsequently, with the advent of the ImageNet Challenge, we are experiencing a gradual trend towards deeper ConvNets with more layers and higher accuracy such as AlexNet (Krizhevsky, Sutskever and Hinton), ZFNet (Zeiler and Fergus), VGGNet (Simonyan and Zisserman), GoogLeNet (Szegedy, Liu and Jia), and the ResNet (He, Zhang and Ren) being the latest state-of-the-art implementation of ConvNets.

To this point, I used VGGNet’s (Szegedy, Liu and Jia) framework as a base where I made modifications to the fully connected layers to suit our problem of identifying multiple digits and I optimized the hyperparameters to determine my optimal model for identifying multiple digits from real-world images.

**1.2 Problem Statement**

I am attempting to predict a series of numbers given an image of house numbers from the SVHN dataset. An important thing to take note is that instead of the standard identification of numbers, as with the MNIST dataset, I now need to correctly detect the numbers and the sequence of numbers.

**1.3 Metric**

***Loss***

***Accuracy***

**2. Analysis**

2.1 Data Exploration

2.2 Exploratory Visualization

2.3 Algorithms and Techniques

2.4 Benchmark

3. Methodology

3.1 Data Pre-processing

**3.2 Implementation**

***Programming Language and Libraries***

Our programming language of choice is Python 2.7 and we will be using Keras with the backend as TensorFlow to build our deep ConvNets.

***Checkpoints***

Due to the nature of the size of this network and the possibility of anyone reproducing the results lacking a strong configuration of multiple high-end GPUs, I ran the ConvNet using checkpoints where I saved the best ConvNet model and its weights. This ensures that anyone, including myself, can pick up the training and continue for more epochs. The weights are saved to HDF5 files and the models are saved into JSON files where their respective file names will be given in the following paragraphs.

***ConvNet Topology: Trial 1***

In this first trial, I started off with a simple model comprising the following layers. For the purpose of simplicity, every convolution and fully connection layer has a Rectified Linear Unit (ReLU) activation layer except the output layers where they use sigmoid activation instead.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input Layer**  32 x 32 x 1  *Images with 32 x 32 dimensions with 1 channel (grey scale)* | | | |
| **Convolution**  Filters: 32  Receptive fields: 3 x 3  Stride: 2  Padding: Same | | | |
| **Rectified Linear Unit (ReLU) Activation** | | | |
| **Convolution**  Filters: 32  Receptive fields: 3 x 3  Stride: 2  Padding: Same | | | |
| **ReLU Activation** | | | |
| **Dropout**  Probability = 0.5 | | | |
| **Fully Connected (FC) Layer**  Nodes: 1024 | | | |
| **FC**  Nodes: 10 | **FC**  Nodes: 10 | **FC**  Nodes: 10 | **FC**  Nodes: 10 |
| **Sigmoid Activation** | **Sigmoid Activation** | **Sigmoid Activation** | **Sigmoid Activation** |

***Weights and Models***

The weights and models are saved to trial\_1\_weights.h5 and trial\_1\_model.json respectively.

The model was trained for 10 epochs on the training set of 188, 602 images.

***Machine Specifications: Trial 1***

The models’ computations are accelerated with a CUDA-enabled GPU, NVDIA’s GeForce GT 750M with 2GB of GDDR5, on a local machine with 16GB of RAM. Hence, there was sufficient RAM to ensure there was no bottleneck in our data transfer from our main memory (RAM) to our GPU’s memory (VRAM) for numerical computation with the GPU’s 384 CUDA Cores.

**3.3 Refinement**

***ConvNet Topology: Trial 2***

I adopted VGGNet’s (Szegedy, Liu and Jia) topology. In particular, I chose the ConvNet configuration with 16 weight layers also called VGG16 and added 15 more weight layers (5 digits x 3 full-connected layers) to predict up to 5 digits. Without this modification, we are unable to predict 5 digits.

|  |
| --- |
| **Input Layer** |
| **ConvNet**  Filters: 64  Receptive fields: 3 x 3 |
| **ConvNet**  Filters: 64  Receptive fields: 3 x 3  Stride: 1 |
| **Max Pooling**  Filter: 2 x 2  Stride: 2 |
| **ConvNet**  Filters: 128  Receptive fields: 3 x 3  Stride: 1 |
| **ConvNet**  Filters: 128  Receptive fields: 3 x 3  Stride: 1 |
| **Max Pooling**  Filter: 2 x 2  Stride: 2 |
| **ConvNet**  Filters: 256  Receptive fields: 3 x 3  Stride: 1 |
| **ConvNet**  Filters: 256  Receptive fields: 3 x 3  Stride: 1 |
| **ConvNet**  Filters: 256  Receptive fields: 3 x 3  Stride: 1 |
| **Max Pooling**  Filter: 2 x 2  Stride: 2 |
| **ConvNet**  Filters: 512  Receptive fields: 3 x 3  Stride: 1 |
| **ConvNet**  Filters: 512  Receptive fields: 3 x 3  Stride: 1 |
| **ConvNet**  Filters: 512  Receptive fields: 3 x 3  Stride: 1 |
| **Max Pooling**  Filter: 2 x 2  Stride: 2 |
| **ConvNet**  Filters: 512  Receptive fields: 3 x 3  Stride: 1 |
| **ConvNet**  Filters: 512  Receptive fields: 3 x 3  Stride: 1 |
| **ConvNet**  Filters: 512  Receptive fields: 3 x 3  Stride: 1 |
| **Max Pooling**  Filter: 2 x 2  Stride: 2 |
| **5 x Fully Connected Layer**  Nodes: 4096 |
| **5 x Fully Connected Layer**  Nodes: 4096 |
| **5 x Fully Connected Layer**  Nodes: 1024 |
| **Softmax** |

4. Results

4.1 Model Evaluation and Validation

4.2 Justification

5.0 Conclusion

5.1 Free-Form Visualization

5.2 Reflection

5.3 Improvement: ResNet