CONVOLUTIONAL NEURAL NETWORK FOR IMAGE CLASSIFICATION USING TENSORFLOW

UCS742 Deep Learning

Anshul Aggarwal
101410007

Sreeparna Deb
101410053

Vidushi Sharma
101410062

Arun Verma
101590003

Sahil Chauhan
101590005

CML

Submitted to: **Dr. Sushma Jain**

November 2017

***Abstract* -** Within recent years, convolutional neural networks (CNN) have experienced a comeback and are currently dominating the field of computer vision. In contrast to previously used methods, they allow for full data-based end-to-end training without the need to manually engineer features. On the downside, huge amounts of annotated data are needed to train a network successfully. Furthermore, the architectural design of these networks and other hyperparameters are sometimes hard to choose, especially when predicting high-resolution outputs from images. In this project, we show how networks can be trained without manually labelled data by using surrogate tasks, strong augmentation and synthetically generated training data. We demonstrate how generic image features for classification and matching can be learned unsupervised-ly by augmenting random images to form surrogate classes. All of this is done by using synthetically generated training data which we rendered and processed using a custom version of the Tensor Flow.

Finally, comparisons with the results in various parameter variations are presented. Experimental results demonstrate that the used model achieves a notable improvement in terms of both quantitative and qualitative classification.

1. Introduction

1.1 Neural Networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones. This is true of ANNs as well.

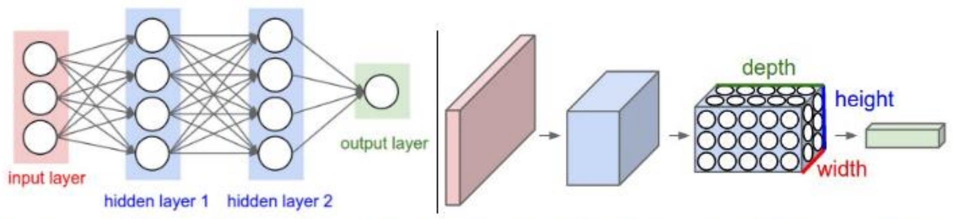
1.2 Convolutional Neural Networks

Convolutional Neural Networks are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function on the last (fully-connected) layer.

Regular Neural Nets receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the “output layer” and in classification settings it represents the class scores.

Regular Neural Nets don’t scale well to full images. Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. As we will soon see, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output will reduce the full image into a single vector of class scores, arranged along the depth dimension. Here is a visualization:



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Figure 1 A Convolutional Neural Network vs a regular Neural Network

1.3 Characteristics Of CNN

1.3.1 Sparse Connectivity

CNNs exploit spatially-local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. In other words, the inputs of hidden units in layer m are from a subset of units in layer m-1, units that have spatially contiguous receptive fields. We can illustrate this graphically as follows:

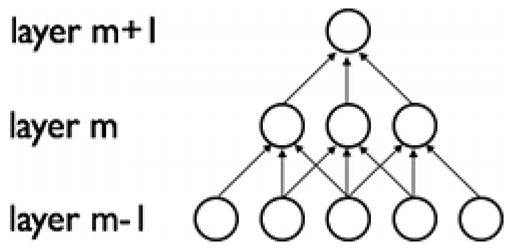


Figure 2 Layering in CNNs

Imagine that layer m-1 is the input retina. In the above figure, units in layer m have receptive fields of width 3 in the input retina and are thus only connected to 3 adjacent neurons in the retina layer. Units in layer m+1 have a similar connectivity with the layer below. We say that their receptive field with respect to the layer below is also 3, but their receptive field with respect to the input is larger (5). Each unit is unresponsive to variations outside of its receptive field with respect to the retina. The architecture thus ensures that the learnt “filters” produce the strongest response to a spatially local input pattern.

However, as shown above, stacking many such layers leads to (non-linear) “filters” that become increasingly “global” (i.e. responsive to a larger region of pixel space). For example, the unit in hidden layer m+1 can encode a non-linear feature of width 5 (in terms of pixel space).

1.3.2 Shared Weights

In addition, in CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map.

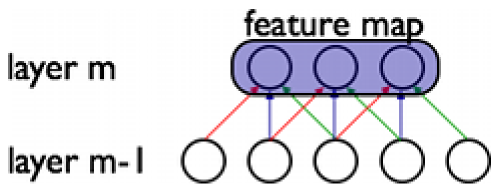


Figure 3 Feature maps in CNNs

In the above figure, we show 3 hidden units belonging to the same feature map. Weights of the same colour are shared—constrained to be identical. Gradient descent can still be used to learn such shared parameters, with only a small change to the original algorithm. The gradient of a shared weight is simply the sum of the gradients of the parameters being shared.

Replicating units in this way allows for features to be detected regardless of their position in the visual field. Additionally, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learnt. The constraints on the model enable CNNs to achieve better generalization on vision problems.

1.4 Overview

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/ReLU/POOL are by far the most popular)
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/FC do, ReLU/POOL don’t)
- Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, ReLU doesn’t)

1.4.1 Limitations of Neural Nets

Some limitations of using neural networks are:

- Need a large dataset
- Because you need a large dataset, training time is usually significantly high

- The scale of a net's weights (and of the weight updates) is very important for performance. When the features are of the same type (pixels, word counts, etc), this is not a problem. However, when the features are heterogeneous--like in many Kaggle datasets - your weights and updates will all be on different scales (so you need to standardize your inputs in some way).
- Parameters are hard to interpret - although there is progress being made.
- Hyper-parameter tuning is non-trivial

2. Literature Survey

In the area of Computer Vision, Convolutional networks have recently enjoyed a great success in large-scale image and video recognition. Convolutional neural networks (CNN) have been successfully applied in many fields of image processing, such as deblurring, denoising, image restoration and furthermore, Image Classification. In contrast to previously used methods, they allow for full data-based end-to-end training without the need to manually engineer features. On the downside, huge amounts of annotated data are needed to train a network successfully. Furthermore, the architectural design of these networks and other hyperparameters are sometimes hard to choose, especially when predicting high-resolution outputs from images. [1]

There has been lots of work done on Image Classification using CNN, but we limit the discussion to only the most relevant recent contributions in this section. In particular, the very recent ImageNet classification challenges have been dominated by CNN-based methods. The convolutional networks based on the structure of Krizhevsky et al. [2] typically contain five convolutional layers, followed by two fully-connected layers of size 4096 neurons, and the output layer. And indeed, the shortcomings of small image datasets have been widely recognized. [3]

Recent publications suggest that unsupervised pre-training of deep, hierarchical neural networks improves supervised pattern classification which lead to Deep Convolutional Neural Network. [4]

But training them requires weeks, months, even years on CPUs. High data transfer latency prevents multi-threading and multi-CPU code from saving the situation. In recent years, however, fast parallel neural net code for graphics cards (GPUs) has overcome this problem. Carefully designed GPU code for image classification can be up to two orders of magnitude faster than its CPU counterpart. [5] Adding to Convolutionlism, Multiscale hierarchical convolutional networks are also structured convolutional networks that can be used for Image classification where layers are indexed by progressively higher dimensional attributes, which are learned from training data. Another line of improvements in Convolution Neural Networks dealt with training and testing the networks densely over the whole image and over multiple scale. [6]

However, Convolution Nets when used for Image Classification can provide efficient processing speed using open source libraries such as Tensor Flow which provides an extensive environment for parallel processing. With ConvNets becoming more of a commodity in the computer vision field, a number of attempts have been made to improve the original architecture of Krizhevsky et al. (2012) [2] in a bid to achieve better accuracy. The architectural design of these networks and other hyperparameters are sometimes hard to choose, especially when predicting high-resolution outputs from images. In particular ones implemented by convolutional neural networks, have led to good progress on many learning problems. However, the learned representations are hard to analyse and interpret, even when they are extracted from visual data. [7] In this project, However, we have used the standard architecture of ConvNets which has been previously successfully used especially with local features and tried to observe the results of such a powerful classification Model. [8]

3. Dataset

The MNIST database [9] is a set of 60,000 handwritten digits, with labels to tell the digit the image represents. The dataset is further divided into two – a training set containing 50,000 images, and a test set containing 10,000 images. Each image is grayscale, so the colour depth is only 1 byte, i.e. the number of channels is one. Each image is of the size 28 x 28 pixels. The digits in the images have been normalized for size in the dataset, and further centered, so that any pre-processing is not necessary before training the model.

Since the images depict digits, there are 10 different categories in which they are classified, i.e. 0-9. The digits in the images were written by 250 different writers, and no writer has images in both the training and the test set. The total size of the dataset is around 11 MB, compressed.

4. Implementing CNN using TensorFlow

The neural network has a total of four interconnected layers. The first two layers, including the input layer, are convolutional layers, and the last two ones, including the output layer, are fully connected layers. Convolution filters are initially chosen at random, and are later optimized to generate the best prediction.

4.1 Convolutional Layer 1

It takes the 28 x 28 pixel images as input and extracts features. The convolution filters are 5 x 5 pixels in size, and the number of filters used in this layer is 16. The layer uses max pooling, using the 'SAME' padding, and the extracted feature-images are down-sampled to 14 x 14 pixels. For each input image, 16 new feature-images are created by this layer, corresponding to the 16 filters used.

4.2 Convolutional Layer 2

It takes each of the 16 feature-images from first layer, and applies another round of convolution on each of these images with 36 convolution filters, each of size 5 x 5 pixels. Again, max pooling is used, and the resultant feature-images are down-sampled to 7 x 7 pixels. This layer generates 36 such feature-images, one for each filter used, per input feature-image. Hence the total number of feature-images generated by this layer corresponding to a single input image is 16 x 36 = 576.

4.3 Flattening

This action takes the result of the convolutional layer 2 and ‘flattens’ it into a single long vector of length 7 x 7 x 36 = 1764.

4.4 Fully Connected Layers

The first fully-connected layer contains 128 neurons, which take the flattened 1764-element-long feature vector as input. These neurons then feed information to the second fully-connected layer, containing 10 neurons, each corresponding to one of the 10 classes in which the images can be classified. This layer generates an output on the scale of 0-1 for each neuron, and the prediction is the neuron that gives the result closest to 1.

5. Results

Table 1 Prediction accuracy for different levels of optimization

# Optimization Iterations	Accuracy
0	8.23%
10	22.51%
100	64.96%
1000	93.81%
10000	98.84%

Table 1 lists the accuracy achieved on the test set (10,000 images) at different number of optimization iterations. The prediction accuracy increased significantly with increasing number of optimization iterations. Figure 4 shows the growth of model accuracy over training set as the number of iterations grow.

As the accuracy grows, only the obscure images, with not a very clear indication of the actual number, fail to give the correct result. Table 2 depicts some of these cases, for incorrect results after 1,000 and 10,000 optimization iterations. These results are generated at the end of every execution.

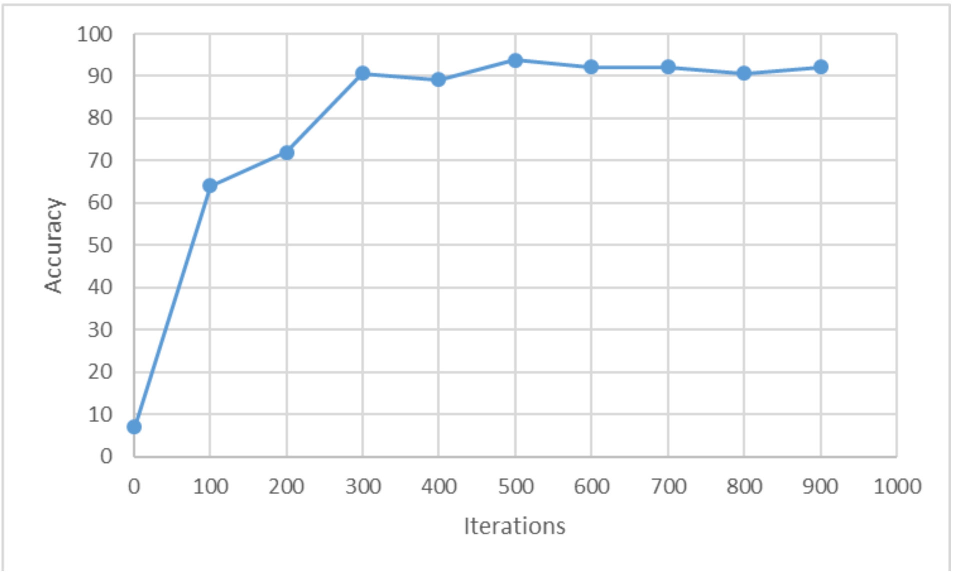
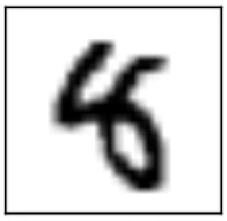
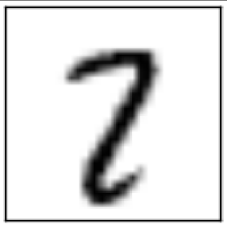
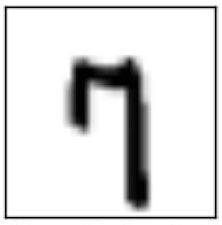
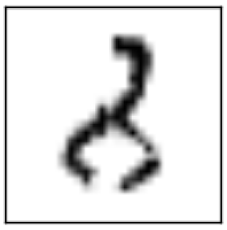
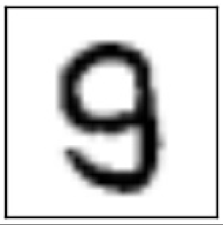
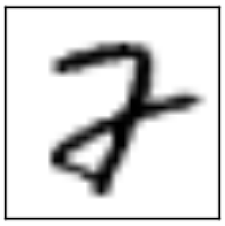
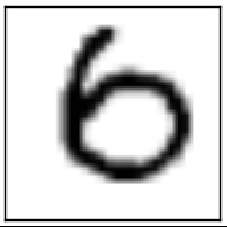
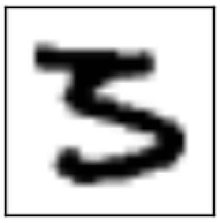


Figure 4 Growth curve of model accuracy

The final accuracy and dataset statistics are automatically written to the results.txt file, generated by the program.

Table 2 Erroneous predictions

Errors after 1,000 iterations		Errors after 10,000 iterations	
	Actual: 8		Actual: 2
	Predicted: 4		Predicted: 7
	Actual: 7		Actual: 8
	Predicted: 9		Predicted: 2
	Actual: 9		Actual: 2
	Predicted: 8		Predicted: 7
	Actual: 6		Actual: 3
	Predicted: 0		Predicted: 5

6. Conclusion

Our results show that a Convolutional Neural Network is capable of achieving record-breaking results on a highly challenging dataset using purely supervised learning in Image Classification. It is notable that our network's performance degrades if a single convolutional layer is removed. For example, removing any of the middle layers results in a loss of about 2% for the top-1 performance of the network. So, the depth really is important for achieving our results.

To simplify our experiments, we have obtained enough computational power to significantly increase the size of the network without obtaining a corresponding increase in the amount of labelled data. Thus far, our results have improved as we have made our network larger and trained it longer, but we still have many orders of magnitude to go in order to match the infero-temporal pathway of the human visual system.

Ultimately, we would like to use Convolutional nets on images where the structure provides very helpful information for classification or obvious in static images of digits.

References

[1] F. Philipp, Convolutional networks to relate images, January 2016.

[2] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In NIPS, 2012.

[3] N. Pinto, D.D. Cox, and J.J. DiCarlo. Why is real-world visual object recognition hard? PLoS computational biology, 4(1):e27, 2008.

[4] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In Neural Information Processing Systems, 2007.

[5] R. Uetz and S. Behnke. Large-scale object recognition with CUDA-accelerated hierarchical neuralnetworks. In IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS), 2009.

[6] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In proceedings of ICLR, 2014.

[7] D. Alexey, B. Thomas, Inverting Convolutional Networks with Convolutional Networks, 2015.

[8] K. Markus, L. T. Jorma, Convolutional Network Features for Scene Recognition, 22nd International Conference on Multimedia, 2014.

[9] Y. LeCun, C. Cortes and C.J.C. Burges, The MNIST Database of handwritten digits, 2007.