Implementation of Convolutional Neural Network

FRANKFURT UNIVERSIY OF APPLIED SCIENCES, MARCH 2019, AI/ML SOFTWARE ENGINEERING SUPERVISED BY DAMIR DOBRIC

Devan Nair

Matriculation Nr:1274369
Dept.of Information Technology
Frankfurt University of Applied Sciences
Frankfurt,Germany
dnair.stud@fra-uas.de

Taimour Yousuf

Matriculation Nr:1276574
Dept.of Information Technology
Frankfurt University of Applied Sciences
Frankfurt,Germany

***Abstract*— Convolutional network is a multiple layer learning network that uses convolutional filters to extract meaningful features from image to be learnt discriminatively by linear or non-linear regression units in the network. In recent times convolutional networks have gained prominence for unsupervised and supervised learning implementations. We use multiple layers of convolutional ,pooling and rectified linear units to train the neural network to classify digits of handwritten MNIST database.**

Keywords—convolution ,neural network ,digit ,classification.

I. INTRODUCTION

Extracting features out of two-dimensional images is a challenging problem, one that has many applications in field of computer vision. Unlike feature extractions out of multidimensional data points in case of multi-layer perceptron networks, features in images are often bound through geometric information in the image which are often lost when using linear hyperplane separation methods those employed by standard multi-layer logistic regression algorithms. The distinctive features of an image are in most cases curves which cannot be captured through linear feature extraction methods .

In this paper we try to demonstrate how images can be classified into their digit or class labels if we stack multiple numbers of these feature extraction layers on top each other to sequentially extract meaningful geometric information out of the image. We then train a convolutional neural network based on these features which are representative of the geometric non-linearities of the image. We use stochastic gradient descent algorithms to estimate the parameters of the convolutional filters , rectifying and pooling units.

II. LEARNING ARCHITECTURE

In this section we describe our digit recognizer module which is the fundamental unit of the end-to-end system. The MNIST database contains handwritten samples which are 28-by-28 pixels . Different stacked convolutional filters in a layer extract different geometric features from the image .The first convolutional layer has eight, 5-by-5 convolutional filters .It is therefore capable of extracting 8 different nature of features in a single image .For example one of the convolutional filter in the layer may detect a third order polynomial curve in the upper left hand corner while the other filter may be detecting intersections in the center of the image. We only need a 5-by-5 filter [1] because we slide the feature detector across the whole 28-by-28 image .At each point the filter would convolve with the image to give a resulting co-efficient of probability of feature presence at that particular site .Each filter in the layer would be detecting a feature such that at the end of training the network, the image is mapped into a distinct label class by a regression unit .Note how the number of trainable parameters stay constant irrespective of the size of the image .To classify the image into label classes we need to train on $5*5*8 \sim 200$ different weight parameters and 8 bias preferences for the filter layer .In the case of a multi-layer perceptron network one would have been forced to construct a weight array that was as large as the image ,making the number of trainable parameters proportional to the square of image size. Weight storage requirements are therefore of the order

$$\text{For MLP} \cong O(n^2) \quad (1)$$

$$\text{For CNN} \cong \textit{order of filter} \quad (2)$$

Thus the storage requirements for estimation parameters in a convolutional neural network are far lower than that of corresponding multilayer perceptron (MLP) network equivalents .This results in a twofold advantage namely that memory requirements for seemingly larger image dimensions are constrained to filter dimensions in a convolutional neural network (CNN) and also the computation time required for these weight and bias parameter estimation are constrained based on the order of filter sequences that are required for classification.

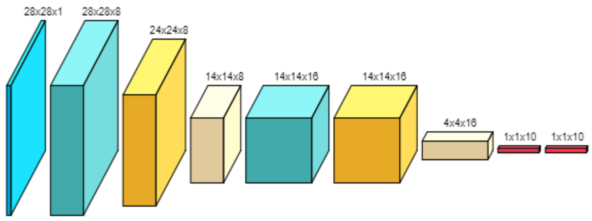

Figure 1: The Convolutional Network Structure

For MNIST digit classification [2] we use multi-layer ,convolutional neural network .Our network has two convolutional layers with n_1 and n_2 filters respectively .The network used for image classification employs $n_1 = 8$ and $n_2 = 16$. We train the network using stochastic gradient descent ,which will estimate the weights of the network weights and bias parameters by first forward propagating the image upstream to the network assuming a randomized initialized parameter assignment , calculating its deviation from the expected class since in supervised learning we know the class to which the image belongs .Then the network propagates the error downstream to the network adjusting the weights and biases such that error in the upstream layer is minimized . In particular if we take the first convolutional layer then with unit stride of the filter with an additional 2 row padding at the top and the bottom, we extract 5-by-5 patches which are contrast normalized to form input vectors in $\mathbb{R}^{25}$ [3].We then use gradient descent to estimate filter coefficients { $x : x \in \mathbb{R}^{64}$ x n_1 & $\delta(error) \sim 0$ }.We compute the response of convolution by forming inner product with the convolutional filter followed by a scalar activation function ,which in our case is a sigmoid activation

$$f(x) = 1/(1 + exp(-\alpha * x)) \qquad (3)$$

where α could be a scalable hyper-parameter but in our implementation we are taking to be 1. Given a 28-by-28 input image ,we compute $f(x)$ for every 5-by-5 window to obtain a 28-by-28-by- n_1 first layer response map. The rectified linear units and the pooling layer then reshape it to a 14-by-14-by-n_1 response which is passed on to next convolutional filter which is a 5-by-5-by- n_2 window. This would result in a 14-by-14-by- n_2.These outputs are then fully connected to a classification layer .We discriminatively train the network by backpropagating the classification error.

III. Implementation And results

The convolutional network was implemented with four major modules namely Layers , Tensors ,Network Architecture and Training Algorithm.

A. Implementation

Layers [4] is an implementation of different types of neural network layers used in convolutional network , namely the *convolutional filters ,rectified linear units ,pooling layer ,fully connected layer* and *Softmax* layer. All of the above classes implements *LayerBase* interface and in case of *Softmax* layer a *LastLayerBase* interface .The logistic regression units implement *IClassification* interface which are responsible for supervised framework of the network training and testing.

The next module is the Tensor libraries [5] which are the most suitable data structures for binding mathematical operations to each dimension of a multidimensional array. So convolution operation which happens across the pixel-space of the image can be totally isolated from the pooling operations namely averaging and maximum-search that span across different filter stack in a layer. Such data structures are more capable of stacking multiple types of layers and using more than one training algorithm for each subsection of the neural network .However in this case we have used a single learning algorithm and used five different kinds of neural network layers.

The Network Architecture module is responsible for binding the neural network and the gradient descent algorithm that is going to be used in the CNN implementation .The given CNN implements a multi-layer stacked network consisting of two convolutional filters, two rectified linear units and pooling layer, two fully connected layer and a SoftMax normalization layer. The network uses a gradient descent trainer to estimate the weights and biases of these layers in the network. *TrainParameters* class is responsible for setting up the hyper-parameters namely the learning rate,momentum,L2 Decay and batch size for training the CNN.

Training module consists of the flavor of the gradient descent trainer used for estimating the parameters of the convolutional network. It extends a basic Training class [4] which defines the forward pass of the input tensor through the network .The specific implementation based on gradient descent is implemented in the backward pass of the *GradientDescentTrainer* . The circular buffer class is used for calculating a moving window average when it comes to estimating the accuracy with which a given batch of images has propagated upstream through the convolutional network.

The training and testing dataset that is used is a csv file of 28-by-28 MNIST handwritten digit database.

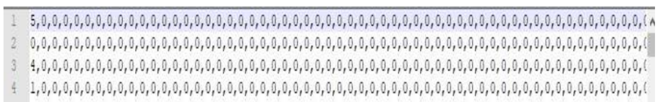

Figure 2: An MNIST dataset example

Figure 3: Part of an MNIST vector for images of digits5 ,0 and 4

The first digit in a row corresponds to the label corresponding to that image. For example if the given 784 columns of a particular row corresponds to image of digit 5 then the first value of the row will be 5 , followed by the first row of the digit, then the second row and so on till all the 28 rows have been flattened out into a single row having a label digit followed by 784 integer values in the range of 0-to-255. So, for example for a case of digit 5 the data could look something

Here convolutional network was implemented using Tensor, Training and Layers libraries from open source repository https://github.com/cbovar/ConvNetSharp .

like 5,0,0,…34,0,55,78,..seven hundred and eighty-five values in a single row.

The Learning API [6] architecture brings all of the modules together in its *IAlgorithm* interface implementation. The *IScore* and *IResult* implementations indicate the convergence criterions, prediction and probability for the trained network namely the mean squared error in training and the probability with which a given label is predicted by the convolutional network. The Learning API architecture provides the method *UseConvolutionalNetwork(numOpClasses,numOrder,BatchSize)* [7].The parameter *numOpClasses* refers to the number of output classes to which the input images have to be mapped to,*numOrder* refers to the square order of the image that is being trained into the network, for example in the case of MNIST datasets that are used which are 28-by-28 pixel square images, the *numOrder* is 28.*BatchSize* refers to the number of images that are learnt by the network in one epoch .Specifying 10 would mean that for a training sample set of 100 images, the network would train with 10 images in the first epoch,10 int the next and so on till the whole training sample set is trained by the network. Another method that Learning API provides is *Run()*,which trains the network with the training data set loaded via *UseActionModule()* method. All the above methods are responsible for training the network with the training data sets. The prediction part is carried out by *Predict(data[][])* method which returns an *IScore* inherited object which has the information about the label corresponding to the digit that was passed into the network for prediction. The argument passed is two-dimensional array which is a flattened version of MNIST image at each row.

B. Project Repository

- GitHub https://github.com/UniversityOfAppliedSciencesFrankfurt/se-dystsys-2018-2019-softwareengineering/tree/ConvolutionalNetwork/MyProject

C. Test Methodologies

In this subsection we present a detailed analysis of results of our convolutional network implementation. We measure loss estimate and confidence of convergent solution in our gradient descent training for MNIST dataset. In addition to that a custom statistically independent sequence of four classes of 6-by-6 images were used to test if the implementation was dynamic to image size variations and if the architecture of the prediction and training evaluations were suitable for any type of labelled image data sets.

First we evaluate our network for MNIST datasets [8]. The nine- layer classification network was trained with images from MNIST database having a total of 2400 samples. Due to concerns of memory we didn't keep separate excel files for each test case ,rather we kept 2400 MNIST images in a single .csv file, which had first label column and then followed by 784 columns corresponding to flattened out MNIST image as explained in Implementation subsection above. In the *UseActionModule()* [6] method of Learning API ,we then passed in different test methods, the number of MNIST image samples that we wanted to train the network with in that test method. So for example in test method MNIST60Samples , the *UseActionModule()* parsed only the first 60 rows of the MNIST csv database and using a default batch size of 20,the training algorithm split those 60 images into 20 images each into 3 separate batches and in the first epoch 20 images were taught and int the next one the next batch of 20 images were taught and so on till the network was trained with 60 images. At the end of each batch training the network was forward passed with the images it was trained with and the percentage of number of correct estimations of image labels were calculated, this was called ***convergence confidence*** and was one of the variables that was stored in IScore implementation which was returned by the Learning API Run() method. The other variable stored int the IScore implementation was the **loss** estimate of training which is the mean square deviation of tested dataset samples from the expected sample labels.

```
[TestMethod]
public void MNIST60Samples()
{
    const String mnistPath = @"Samples\mnist_train3.csv";
    LearningApi api = new LearningApi();
    var path = System.IO.Path.Combine(Directory.GetCurrentDirectory(), mnistPath);
    api.UseCsvDataProvider(path, ',',false);
    api.UseActionModule<object[][], double[][]>((object[][] data, IContext ctx) =>
    {
        List<double[]> res = new List<double[]>();
        double[] features;
        for (int i = 0; i < 60; i++)
        {
            features = new double[data[0].Length];
            for (int j = 0; j < data[0].Length; j++)
            {
                double converted;
                if (double.TryParse((string)data[i][j], out converted))
                    features[j] = converted / 1.0;
```

Figure 4: Training 60 samples of MNIST images in MNIST60Samples test method

Then in the prediction part the label data is removed, and the remaining 784 columns of image data are passed into *Predict*() method which returned a label index which is an integer between 0-to-9 indicating which image was asked to recognize by the trained network and a probability indicating the confidence with which the network is predicting the current result. These two parameters are part of the *IResult* implementation which the Predict() method returns.

```
api.UseConvolutionalNetwork(10);
ConvolutionalNetworkTrainingScore tScore = api.Run() as ConvolutionalNetworkTrainingScore;
Assert.IsTrue(tScore.convergenceConfidence > 0.50);
Assert.IsTrue(tScore.convergenceConfidence > 0.89);

double[][] predictingData = new double[3][];

predictingData[0] = new double[] { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

predictingData[1] = new double[] { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

predictingData[2] = new double[] { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

ConvolutionalNetworkPredictionScore pScore = api.Algorithm.Predict(predictingData, api.Context) as ConvolutionalNetworkPredictionScore;
```

Figure 5: Predicting the labels of an MNIST image passed into a trained network

Based on these criterions the accuracy and prediction results were captured into the testing methodology and two test methods corresponding to training 60 and 2400 samples of MNIST images and predicting random samples within those

Here MNIST dataset csv sample that was used was acquired from https://pjreddie.com/projects/mnist-in-csv/

trained lists were used to check the performance of the trained network. Based on such a method the two test methods had converging parameter estimates for the training datasets at close to 94% over the trained data sample .The mean square errors of prediction(loss) were as low as 0.18 .The network reached a convergent solution

The network was first trained for a small sample of 60 training image sets from the MNIST dataset, with a batch size of 20.This implied that there were three batches of image set iterations per epoch of training. The system converges within 50 iterations to reach mean squared error in class estimation of 0.15. The network was then trained for 2400 samples of the MNIST database with a batch size of 20 images per iteration. The network trained to 94% accuracy in correct estimation of labels of the samples that it was trained with. The convolutional network had converged to a solution point at close to training about 2100 samples. So, the network was almost completely trained within one epoch itself.

D. Results

The results of experiments are shown below

TABLE I. TRAINING WITH A BATCH SIZE OF 20 IMAGE PER ITERATION

Samples Trained	**IScore**	
	Loss	***Confidence***
MNIST 60	0.1550	93%
MNIST 2400	0.085	94%

The unit tests [9] in *LearningApi* implementation of CNN successfully recognized sample training testing samples that were provided to it while training and successfully passed the test cases when 60 MNIST samples were given for training and 2400 MNIST samples were given for training. In addition, the network also learned 4 samples of 6-by-6 images of uncorrelated custom sequence to demonstrate that the implementation was dynamic to changes in image size, provided the size of the image was larger than the convolutional filters which was 5-by-5.

IV. CONCLUSION

The convolutional network was successfully trained with a statistically independent custom test image and MNIST [8] dataset and was successfully predicting labels based on training inputs and the parameter estimations had a converging solution. The network successfully recognized sample training testing samples that were provided to it while training and successfully passed the test cases when 60 MNIST samples were given for training and 2400 MNIST samples were given for training. In addition, the network also learned 4 samples of 6-by-6 images of uncorrelated custom sequence [9] to demonstrate that the implementation was dynamic to changes in image size, provided the size of the image was larger than the convolutional filters which was 5-by-5.

V. REFERENCES

1 L. Honglak, P. Largman and Y. Andrew, "Unsupervised feature learning for audio classification using convolutional deep belief networks," *http://www.robotics.stanford.edu/~ang/papers/nips09-AudioConvolutionalDBN.pdf.*

2 V. Quoc, N. Jiquan, C. Zhenghao, C. Daniel, W. Pang and Y. Andrew, "Tiled convolutional neural networks," *http://www.robotics.stanford.edu/~ang/papers/nips10-TiledConvolutionalNeuralNetworks.pdf.*

3 W. Tao, Y. Andrew, C. Adam and J. David, "End to end recognition with Convolutional Networks," *http://www.robotics.stanford.edu/~ang/papers/ICPR12-TextRecognitionConvNeuralNets.pdf.*

4 B. Cedric, "Convolutional Network Implementation ConvNetSharp," [Online]. Available: https://github.com/cbovar/ConvNetSharp.

5 "Convolutional Network Tensors," [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/se-dystsys-2018-2019-softwareengineering/tree/ConvolutionalNetwork/MyProject/ConvolutionalNetwork/Tensors.

6 "Learning API," [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi.

7 "Convolutional Network Extension Method," [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/se-dystsys-2018-2019-softwareengineering/blob/ConvolutionalNetwork/MyProject/ConvolutionalNetwork/ConvolutionalNetworkExtension.cs.

8 "MNIST CSV," [Online]. Available: https://pjreddie.com/projects/mnist-in-csv/.

9 "Convolutional Network Unit Test," [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/se-dystsys-2018-2019-softwareengineering/blob/ConvolutionalNetwork/MyProject/ConvolutionalNetworkTests/UnitTest1.cs.