Image classification using Network in Network(NiN) architecture

Aditya Kant Sharma (SID: 470375615), James Macdonald (SID: 307135187)

Abstract
Convolution neural network (CNN, or ConvNet) are a special kind of neural networks specially designed for recognizing patterns from pixels of images. In this case study we implement convolution neural network using **Network in Network(NiN)** architecture. The architectire used is inspired by the **inception module of GoogLeNet** where we use collaborative pooling of parallel convolution networks .

Keywords
Network in Network(NiN) — Inception architecture — Convolution Network — Pooling Layers — Deep Learning — ReLU Activation — SGD — Dropout — Softmax — Batch Normalization — Weight Normalization — l2 Regularisation

The University of Sydney, Australia

Contents

1. Introduction

Image classification is an important application of ConvNets. There are various architectures to design a ConvNet. The purpose of this case study is to experiment and establish an understanding of various techniques. Convolution layers and pooling layers are basic building blocks of a ConvNet. And its is crucial to understand the reason behind choosing the shape and sequence of these layers.
We have used NiN architecture where we have built parallel ConvNets and used a concatenation layer to collaborate the learning of different layers. These small NiNs are further stacked vertically with layers of NiNs. With this study we wish to demonstrate that parallel networks learn different featurs of the images and collaborating these networks outperform the traditional approach of vertically stacking convolution and pooling layers.
ConvNets can face issues with training. Training on the whole data simultaneously creates a large computational overhead for little performance gain. Training on a single observation can produce unpredictable results that do not generalise. Additionally, neural networks are prone to exploding coefficients and can suffer from the zero-gradient trap with the chosen ReLU activation. In order to address these challenges, the following features will be deployed within the neural network:

- Dropout
- Batch normalization
- Weight Normalization
- Adam optimiser
- l2 Regularisation

Since this is a multiclass problem with 62 unique classes, a Softmax activation has been chosen for the output layer, allowing for the calculation and comparison of class probabilities.

1.1 Data

There are 62 classes in this dataset. The dataset has been split into training set and validation set, where the training set has 37,882 images of alphabets and numbers; while the validation set has 6,262 images. The images are of size 128*128 pixels with 3 channels.

2. Techniques

2.1 Convolution Layer

Convolution neural networks are special networks which take advantage of the fact the input consists of images. In contrast

to regular neural networks ConvNets have neurons arranged in three dimensions(i.e. height,width and depth of the image). Here depths is also referred as the number of colour channels in the image.
Figure 1 gives shows that convolution layers can be seen as hidden layer in neural networks while the output layer is a vector of size Number_Of_Classes arranged along depth dimension.

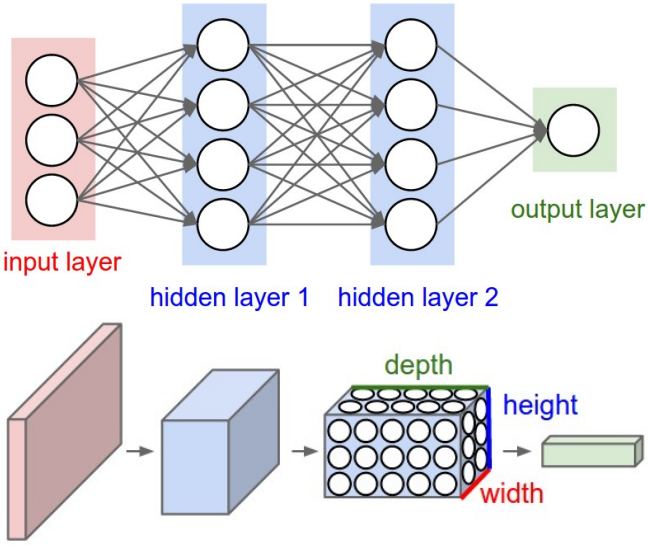

Figure 1. Visualisation of ConvNet

2.1.1 Building blocks of ConvNet

As mentioned earlier we have build ConvNet using NiN architecture inspired by GoogLeNet . The three building blocks of the network are: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer**(similar to regular neural networks). Details about the layers:

- Convolutional Layer: Computes the output of filter applied to the local regions of the image. The filter as shown in figure 2 is used to extract spatial features of the image. The efficiency of feature extraction depends upon **depth**(depth of the input to the conv layer), **stride**(number of pixels by which we slide filter over the input matrix) and **zero padding**(zero padding is done along the border of the image to control the size of feature map). Since convolution is a linear operation, **ReLU activation** is often used to learn non-linearity in the data.
- Pooling layer: Spatial pooling(aka downsampling) reduces the dimension of each feature map while preserving the most important information. Various methods like max, average sum etc. can be used to perform spatial pooling. In ConvNets pooling layer is stacked over conv layer to reduce the length*width of the output while preserving the features learnt by different filters. Pooling layer makes the input representation smaller and manageable. Another most important feature of pooling layer is that it makes the network invariant of the the small distortions and variations in the input image.
- Fully connected layer: FC layers are used to compute the scores of classes. As the name suggests it is similar to normal neural networks have all neurons connected to output of previous layers. The output of FC layers is a vector of shape [1x1xNumber_Of_Classes].

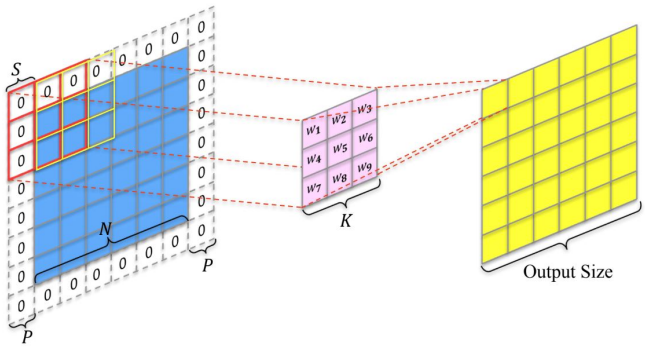

Figure 2. Convolutional Layer

2.2 Network Architecture

We have used **Inception architecture** to build the ConvNet in this experiment. Filter size plays a very important role in the performance in ConvNet. But how do we decide if we should use a filter of size 3x3 or 5x5 etc. Rather we create **parallel convolutions** and concatenate the feature maps before feeding it to the next layer. The use of competitive activation units in ConvNet utilises the fact that each smaller ConvNet is capable of doing simpler tasks. Using multiple layers divide the input feature space into a number of regions which is exponentially proportional to the number of layers. Since sub-networks are trained with the samples which fall into one of the regions and as a result it becomes specialised in one set of features. Using parallel layers avoid overfitting as we concatenate the filter outputs and utilise collaborative learning of all sub networks. If the next layer is again an inception module, the feature maps are passed through a mix of convolutions. This way the network is able to learn different features with the help of filters of different sizes and pick the best one. Having smaller convolutions help in learning local features while larger convolutions help in learning high abstracted features.
Figure 3 shows the inception module used for the first layer. We have used a variety of convolutions: 1x1, 3x3, 5x5 convolutions along with a 3x3 max pooling layer. Since larger convolutions use more computations, we have ised **1x1 convolutions for dimensionality reduction** of the feature maps. After dimension reduction the feature map is passed to larger convolutions thus keeping the computations lower.

The output of the convolutions is passed to a batch normalisation layer which improves stability and performance of the network. Similar to the concept of normalizing the input to the network this method normalizes the input to every layer so

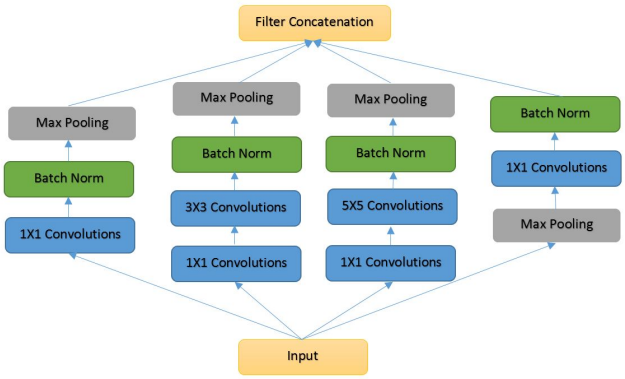

Figure 3. First Inception Layer

that they have a mean output of zero and standard deviation of one. By normalizing the input before applying the activations it reduces the amount by which the hidden units output shifts. It adds few more steps to the calculation of feed-forward and back-propagation , which slows down the network. But due to normalization of inputs the network converges faster which improves overall speed of the network.each scale and also pre-conditions the model.

Finally multi scale filter outputs, weighted bu batch normalization layer are combined in the **concatenation layer** before passing on to the next layer. We have used two inception layers to learn the low level features of the image. As shown in figure 4 further it was stacked with two convolutional layers to recognise higher level features in the image. As the input is passed via multiple conv-layers the networks learns more complex features and increases the prediction accuracy.

Using conv-layers the networks is able to learn both low level features and high level abstractions in the image. We use fully connected layers at the end of the network and outputs the N dimensional vector where N is the number of classes to be predicted. We have used **softmax** to find the class which is best suited to the input image. Fully connected layers connect to the out of the previous layer(which is activation map of the high level features) and finds the features which are most suitable for a particular class. FC layer uses the most correlated features for a particular class and learns weights so that we get correct probability for the different classes when it performs product of weights and output of previous layer.

3. Experiments and results

An experiment was run to determine the optimal settings for a number of hyperparameters. These hypereparameters are network architecture, regularization, iteration, image size and batch size.

For the first hyperparameter, sophisticated architectures outperformed basic architectures. The introduction of multiple convolutional layers, as well as layer 'inception', raised out-of-sample performance considerably. The use of parallelized convolutions of different sizes meant that different streams could concentrate on extracting different features, and this

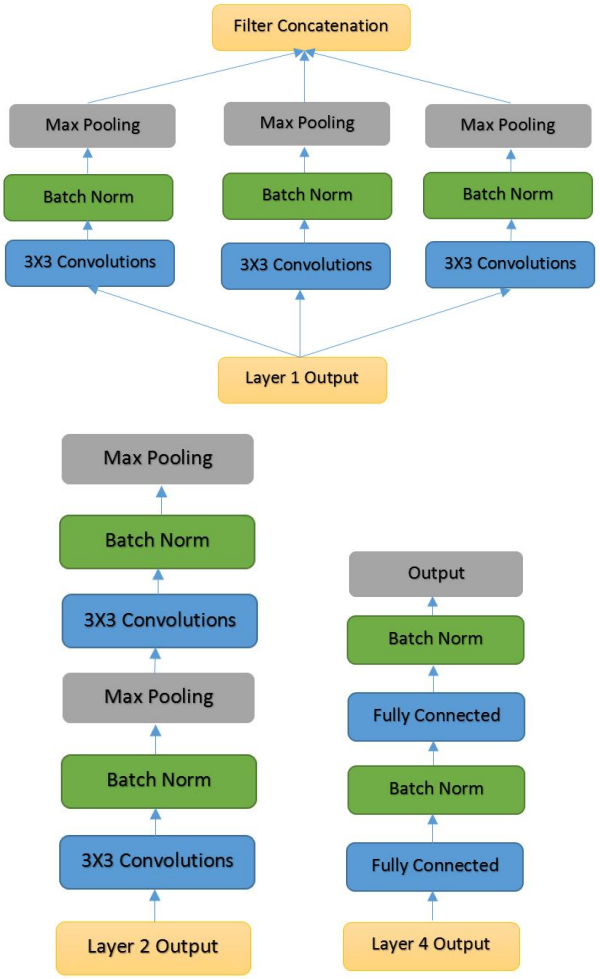

Figure 4. Second Inception Layer, convolution layers and fully connected layers

capability was reflected in better performance.

The second hyperparameter, regularization, quickly became necessary as the network architecture exploded the count of trainable variables. In order to prevent these from overfitting, three regularization devices were used. Firstly, the model weights were penalized using L2 regularization. This suppressed extreme coefficient values in both the convolutional and fully-connected layers. L2 was eventually chosen as L1 produced sparsity, which reduced weights to zero. Secondly, batch normalization was employed in all convolutional and fully-connected layers. Batch-normalization was applied before activation functions, and was useful in preventing exploding or vanashing gradients. Lastly, a slight dropout probability was applied to the first fully-connected layer. All of these techniques improved out-of-sample performance when training accuracy approached the ceiling.

The third, fourth and fifth hyperparameters were set by trial and error. Iteration count was set at 10,000, image size was 32 by 32 pixels, and batch size was 128. Images of 32-by-32 were initially selected as those values are commonly found

in other machine learning datasets such as SVHN. Testing image size at 64-by-64 pixels resulted in a four-fold increase in training times, which was unacceptable. Additional training iterations did not improve out-of-sample performance, and the training process was already long at 100 minutes. Batch size was trialled at 64, 128 and 256. The first size did not guarantee all classes would appear often enough in a batch, and the last value resulted in overly long training times.
Once the hyperparameters were established, the final version of the network was trained. The results achieved are explored below.

3.1 Primary Metrics

Out of sample accuracy was 86.94%. Precision was 88.12%, Recall was 86.94% and F1 score was 86.73%. All results were calculated on the test set and multiclass averaging was performed according to class weights, which were all equal. In multi-class problems it is always harder to interpret these metrics, since there is no clear null case. This is especially true given that all classes are of the same frequency, since weighting in this instance equates to a flat average. In reality results may differ, as some letters are more common than others.
This was a strong result but considerably lower than in-sample performance, which was 95.04%. While regularization was employed to prevent overfitting, there was still a considerable gap between in-sample performance and out-of-sample performance.

Metric	Test Score	Train Score
Precision	88.12%	95.04%
Recall	86.94%	94.33%
F1 score	86.73%	94.17%
Accuracy	86.94%	94.33%

3.2 Extensive analysis

3.2.1 Confusion Matrix

The confusion matrix is an effective visualization of model performance. From the visualization provided in figure 5, some interesting observations emerge. Firstly, while the model is very accurate there are some classes where it is quite weak. For example, for the first class recall was 69% and precision was 79%, both far below average. The reason for this is that the first class is the number ”0”, and is very similar to the letters ”O” and ”o”. Particularly with the diversity of fonts in the dataset, the model is having difficulty discriminating between similar-looking characters. This difficulty extends to cases - there is a striking correlation in the confusion matrix as classes are misclassified with a class exactly 26 steps away. The reason for this is confusion between upper case and lower case characters - this trend stops abruptly when classes cross over into numbers. Letters like ”X”:”x”, and ”Z”:”z” in particular suffered from this problem.

Both these errors show the importance of context - a human reader would not make the mistake between capital ”O” and zero largely due to whether it’s in a context of numbers or letters. Unfortunately context is not something available to this model, so there is a performance ceiling. The confusion matrix (figure 5) of the network’s performance on the validation data suggests that the model struggles to recognise classes 6 and 4 - there are more incorrect predictions of data in these classes than correct ones. Most other classes, however, are predicted well and this reflects the high accuracy scores explored in the previous section.

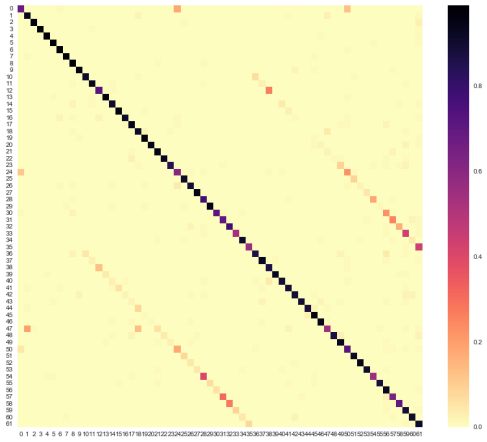

Figure 5. Confusion Matrix

3.2.2 ROC and ROC-AUC

The Response Operator Characteristics curve can be generalised to multiclass problems via a similar weighting method to the primary metrics discussed above. Weighting the AUC of individual ROCs gives an aggregated ROC AUC value of 98.15% This is much higher than the primary metrics given above, and reflects the relative uncertainty of some of the model’s predictions, where second and third place have been given a high probability too. This stems from the ambiguity in the label between similar cases and figures discussed above. The individual ROC curves are provided in figure 6.

4. Discussion

In terms of deign of the network we can see that the use of collaborative multi-scale convolution produces more accurate results as compared to conventional ConvNets. A better performance of Inception Style model can be attributed to the collaboration of multiple parallel networks. It is important to note that the number of parameters is exponentially related to the number of features in training set. Inception style model has fewer parameters and it trains much faster using the collaborative approach.
In this dataset we have 62 classes and the number of images in training set is 38K. Considering the number of classes a higher number of images to train would help learn the features better. Also we can see a wide variety of fonts is used in the data.

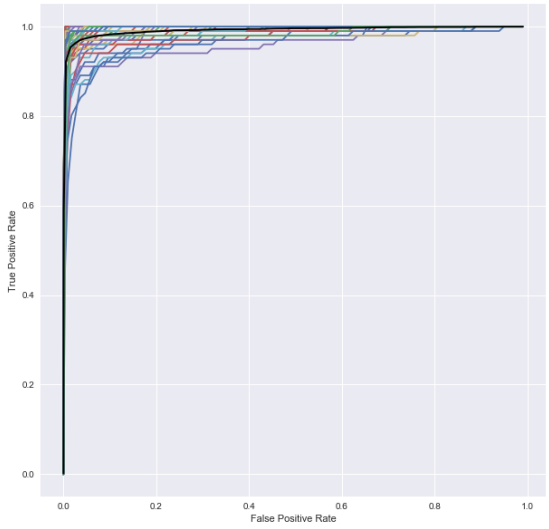

Figure 6. ROC-AUC

This increases the number of features for the model to learn. Apart from learning the shapes of alphabets the ConvNet has to learn the fonts as well. Thus increasing the training data would help in learning the rich input features.
As mentioned earlier there are similarities in the shape of few characters like 0, O, o, 1, l, L, i, I etc. These ambiguities also confuses the model and adversely affects the classification ability of the network.

5. Conclusions

This study has demonstrated a functioning ConvNet with some inception modules. It has performed well on an example dataset and met expectations based on ConvNet's performance on held out data. Batch normalisation also helped in significantly increasing network performance.
The performance of the ConvNet gives concrete evidence that multiple networks are able to divide the data into subsegments and specialise. The biggest advantage of using inception layer is significant decrease in the collaborative learning of different filters. As further study we would like to increase the depth of the networks and use filters to learn high level features to improve image classification of the network.
In most of real world cases we perform image classification with the context of the image. In the case of alphabets, the context is created by the letters around it. For example, zeros are often only distinguishable from 'O' by the presence of other numbers. For further improvement in classification performance we would like to incorporate methods to improve feature map with the help of context and use it resolve ambiguities.

References

[1] Why neural networks
https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/ds12/article1.html

[2] Competitive Multi-scale Convolution
https://arxiv.org/abs/1511.05635

[3] Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree
https://arxiv.org/abs/1509.08985

[4] DEEPLY SUPERVISED NETS
http://vcl.ucsd.edu/~sxie/2014/09/12/dsn-project/

[5] CNNs Architectures
goo.gl/5nxhpG

[6] Inception modules: explained and implemented
https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/

[7] Fractional Max-Pooling
https://arxiv.org/abs/1412.6071

[8] Tensorflow 97% MNIST result tutorial
https://www.tensorflow.org/tutorials/layers

[9] Lecture slides of COMP5329

6. Appendix

6.1 Instructions on how to run code

Following python packages are used for creation of neural network:

- cv2
- sklearn
- tensorflow
- matplotlib
- numpy

We used windows machine to run this neural network. On following configuration it too around 70 minutes to execute the code:

- Operating system: Linux Mint 64-bit OS
- CPU: Intel(R) Core(TM) i5-6600 CPU @ 3.30GHz
- RAM: 16.00 GB
- GPU: Nvidia 1070

Unzip the zip file submitted as part of assignment and follow following steps to execute the code.

6.1.1 Data Setup

Copy training and test data from goo.gl/bY9yLd to and unzip in Input folder.
Run command in **Code/Algorithm** folder: 'python CNN.py'

6.1.2 Output

Output file with predicted labels for test file would be copied in Output folder. Name of the output file with predicted labels: test.txt.