COMP551 - MiniProject 3

John McGowan, Mahad Khan, Benjamin MacLellan

Abstract

Image classification is a ubiquitous machine learning task, in which the contents of an image are predicted by a learned model – most commonly using convolutional neural networks. A common benchmark for comparing results of image classification techniques is the MNIST handwritten digits dataset, where each image contains a digit 0-9, written by thousands of different individuals. In this report we design, implement, and analyze multiple convolutional neural networks (CNNs) to predict which digit is the largest in an example image from a modified version of the MNIST dataset. We use custom-built variations of VGGNet and GoogLeNet using PyTorch as well as image preprocessing techniques to build a number of models. The model which performed the best was a variant of VGGNet with a depth of 7 layers, and had a test accuracy of 91% (on 30% of the Kaggle test data).

1 Introduction

Image classification is a broad field within machine learning – covering computer vision, medical image classification, machine translation, and much more. A number of exciting technological advances rely on image classification – such as self-driving cars and medical diagnoses with smartphones. Image classification is not a new topic, with research and models being developed for many decades – but it has been within the past 7 years that interest, funding and applications have exploded.

The MNIST dataset contains handwritten numerals from 0 to 9, written by a large number of individuals. The MNIST dataset and other collections of images such as ImageNet Large Scale Visual Recognition Competition (ILSVRC) have been used in competitions over the years which have seen advances in image recognition from AlexNet [1] to GoogLeNet [2]. The dataset considered in this project is a modified version of the canonical MNIST set, combining multiple digits into a single image, along with background noise. This adds an extra layer of difficulty to the task, as the model must not only recognize the digit, but also select which one has the largest bounding box.

In this project, neural network models are used to classify the images in this modified MNIST dataset. First, in Sections 2 & 3 we review different models which have become commonly used in image classification tasks, and recent work specifically pertaining to the MNIST dataset. We then implement and present results for multiple CNN architectures, motivated by previous works in Sections 4 & 5. Our models are inspired by VGGNet and GoogLeNEt and we use 3×3 convolutional filters and skip connections. All models are implemented via PyTorch in Python [3]. Our best performing model draws inspiration from VGGNet [4] and obtained 91.4% accuracy on our validation tests and 91.0% on the Kaggle competition (using 30% of the test images).

2 Related Work

The widespread use of deep neural networks for image classification began in 2012 with the results of, the now named, AlexNET – which drastically outperformed competitors in the ImageNet challenge to classify images in 1000 categories using a convolution neural network (CNN) [1]. AlexNET made use of 5 convolutional layers, 3 fully connected layers, Rectified Linear Units (ReLU) nonlinearity, and a softmax activation for the final layer. The use of ReLU provided the advantage of not requiring normalization to prevent saturating [5]. The model also used dropout regularization [6] so that the net learns only the most salient features of the image. Following this seminal work, a number of demonstrations of deep CNNs advanced the bar for image classification accuracy, reaching super-human performance [7].

Simonyan et. al investigated the effect depth has on the accuracy of convolutional neural networks [4]. Simple three by three (3×3) filters are used in all convolutional layers followed by three fully connected layers and again, a softmax activation. Different architectures with increasing depth are examined and it is shown that better accuracy is obtained with increasing depth. Their best performing model is now widely known as VGGNet – and highly popular due to the open-source nature and it's use in transfer learning to new, but related, problems.

As deeper models provide greater accuracy, they come with a greater demand in time and computing power. Significant work has done to increase the efficiency of convolutional networks. In 2015 ResNET won the ILSVRC challenge with a convolutional network $8\times$ deeper than VGGNet – but with lower complexity. This was accomplished by applying residual learning to every few stacked layers – in which the output of multiple preceding layers are fed into the layer in question [7]. DenseNet took this a step further by adding connections between all layers with the same feature size[8].

Inception nets, such as GoogLeNet [2], were also an important milestone in the development of convolutional neural networks. Instead of naively increasing the depth of networks and number of connections, "inception" layers are engineered with more care and placed throughout the network. In each inception layer, multiple convolutions of different sizes are performed before feeding the concatenated outputs to the next layer. This is designed to solve the problem that salient features may vary in size from one image to the next. GoogleNet also reduced complexity by average pooling the output of the last inception layer before the fully connected layer. GoogLeNet was the winner of ILSVRC 2014.

It is important to note that deep learning is not the only avenue recently pursued for similar classification tasks. Majumder *et al.* utilize elastic matching, a technique which has been employed in facial recognition, to the MNIST dataset. Although the model performs with an accuracy of $\sim 96\%$, it falls short of the benchmark by standard deep learning techniques [9].

One of the most interesting techniques for classifying digits in the standard MNIST dataset implements machine learning techniques via a diffractive deep neural network [10]. The learning is accomplished in a similar manner to traditional neural networks (via error back-propagation), but with the activation functions and weighting performed through optical diffraction (an physical process, not on a digital computer). Once the network was trained, it operated at low-power and high-speed (speed of light). The model achieved an accuracy of 93.39% on the MNIST digit dataset – which, although lower than more traditional approaches, is promising for optical machine learning.

3 Dataset

The dataset is a modified version of the famous National Institute of Standards and Technology (NIST) Handwritten Digit dataset. The original dataset contains handwritten digits from 0 to 9, from 3600 writers [11]. The modified version incorporates multiple digits into one image, but with varying sizes. The task, therefore, is to predict the digit with the largest bounding box. The training data contains 40000 training and 10000 testing examples.

Three image pre-processing methods were tested with the aim to improve the generalization and accuracy of the model(s). The first method used image processing techniques[12] to calculate the contours of the image and the corresponding bounding boxes. The largest bounding box is selected and copied to the center of blank image, such that the images fed to the model are zero everywhere except for the center region, where the largest bounding box of the original image exists (Method 1). In the second method, all contours and bounding boxes are calculated (as in Method 1), but rather than copying the bounding box to a null image, the whole image except for the contour with the largest bounding box is masked out. Thus, the images fed to the model (ideally) contain non-zero values within the contour of the digit with the largest bounding box (Method 2). The final method, Method 3, increases the number of training examples by rotating the original images by $\pm 90^\circ$, in order to add invariance to rotation in the model (see Figure A.1).

4 Models

For all models, we used a standard set of hyper-parameters during development. We used a 90%-10% training-validation split of the training examples, and a learning rate of $\alpha = 0.001$ (incrementally reduced until the models converged, and kept for future models as it continued to perform well). We, generally, used 5 epochs with a batch size of 50, and all training/validation was shuffled prior to the learning. For the loss function, we used cross-entropy

$$L_{y'}(y) \equiv \sum_i y'_i \log(y_i) \tag{1}$$

where i is the class, y_i and y'_i are the predicted and true probabilities, respectively. We used the Adam optimizer for gradient descent of our networks [13]. All models were implemented, optimized, and tested in PyTorch [3].

A number of CNN architectures were tested on the dataset, inspiring by a number of famous CNNs from the literature. Here we outline three different models, and the results are presented in Section 5.

Our best performing model (`Net1`) was based loosely off a VGGNet model from [4]. Here at each layer, as in VGG net, 3x3 convolutions are used with padding of 1. Max pool with kernel size 2 and stride 2 is performed at certain layers. Each layer is normalized using batch-norm [14]. After 7 convolutional layers, we implement 2 fully connected layers and one layer of softmax activation. This is outlined in Figure 1.

Based on the work of [7] a similar model was tested adding shortcut connections before the maxpooling layer (`Net2`). For example, instead of maxpooling just operating on output from layer 4, the output of layer 3 and layer 4 are concatenated and maxpooled before being fed into layer 5. The same thing is done for layers 5 and 6. Note that in [7] shortcut layers are added to a convolutional network with 32 convolutional layers rather than our simple model with 7 convolutional layers.

Inspired by GoogLeNet, a third model with three inception layers is also tested (`Net3`). Two normal convolution layers with kernel sizes 7 and 1 are followed by a max pooling layer and three inception layers. Each inception layer has convolutions of size 1×1, 3×3, and 5×5. The output of the 1x1 convolutions are fed into the 3×3 and 5×5 convolutions as well as the output. Another 1x1 convolution fed by a max pool is also fed to the output. After the output a maxpool is performed before the next inception layer, or in the case of the last inception layer, a fully connected layer and softmax activation function. This architecture is illustrated in Figure 2.

5 Results

Initially, using a simple CNN based on VGGNet, we tested the three pre-processing techniques to determine if they improve our model's accuracy. The results are tabulated in Table 1. We see that the preprocessing techniques we used do not improve our models – but rather, for the case of Methods 2 and 3, hinder it. The reason for Method 2 to drastically reduce the performance is likely that the preprocessing eliminates important information, as not every image is processed perfectly. Method 1, while keeping more information, does a task that the CNNs do implicitly – thus it adds no benefit. Method 3 is the only preprocessing technique which does not reduce the information (rather it increases it by artificially tripling the number of training examples) and performs a service the CNNs cannot. However, the reason it likely worsened the performance is because it adds in ambiguity between classes. While in the original dataset, all the digits are rotated only slightly from vertical, in general, by adding larger rotations some digits lose the structure which is imperative in numerals. This is best considered by the example of '6' and '9' – with rotations, the CNN may confuse a rotated '6' with a '9'

Our best performing model (`Net1`), shown in Figure 1) was our best performing model – with an validation accuracy of 91.4%. We experimented with a couple different kernel sizes (7×7 and 5×5, instead

Best Performing Net
conv3 - 16
maxpool
conv3 - 32
maxpool
conv3 - 64 conv3 - 64
maxpool
conv3 - 128 conv3 - 128
maxpool
conv3 - 256
maxpool
fc-1024
fc-10
softmax

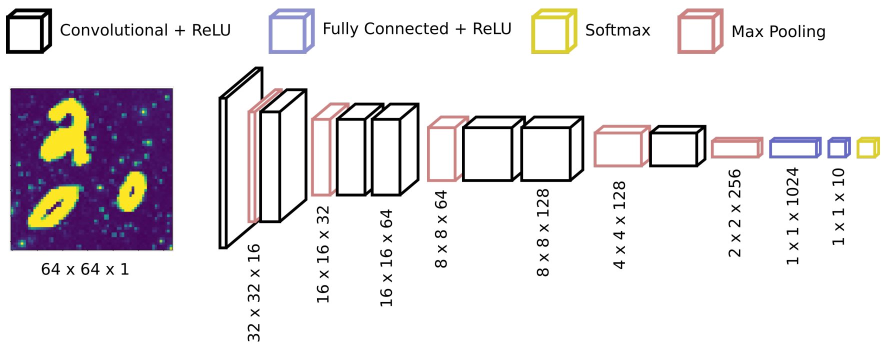

Figure 1: Model `Net1`, based off of [4]. Each convolutional layer is denoted conv[filter size] - [number of channels]. ReLU and batch-norm is used at each convolutional layer, and

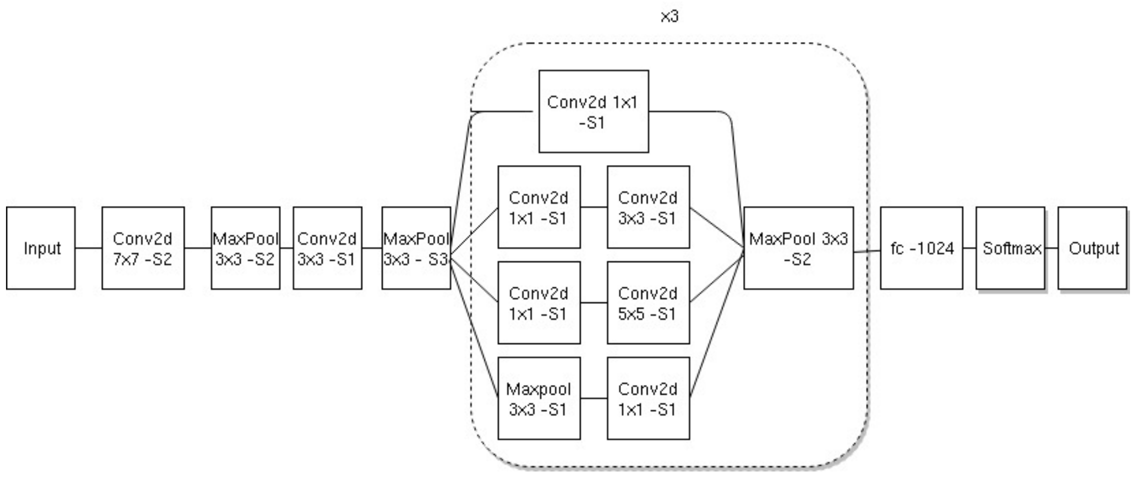

Figure 2: Model `Net3`, including 3 inception layers – inspired by GoogLeNet. S indicates stride length.

of 3×3) and adding more layers. However, the model as described above, with accuracy of 91.4% was our top-performer. We also tested the model using ELU nonlinearity instead of ReLU, but saw a drop in performance of, on average, 4-5%. The training loss of this model is demonstrated in Figure 3, and the first-layer kernels are in Figure 4.

With the changes to the model and creating `Net2`, there was little effect on the validation accuracy, scoring 90.05% – a decrease in performance.

Our final model, `Net3`, based on GoogLeNet, performed much worse. The accuracy of this model on the validation set was between 50% and 60%, depending on the hyperparameters. With the addition of just one more inception layer, this became too demanding to train given our computational resource constraints. GoogLeNet makes use of nine inception layers. So implementing anything close to the true GoogLeNet is not feasible for us.

Method	Pre-processing technique	Validation accuracy
–	No pre-processing	91.4 %
Method 1	Isolate and center largest bounding box	91.4%
Method 2	Masking contour with largest bounding box	83.5%
Method 3	Increase number of examples, rotate images ($\pm 90°$)	88.1%

Table 1: Results of the three pre-processing techniques described in Section 3 with the accuracy measured while using the same model (based off of the VGGNet structure). We see that these pre-processing techniques do not improve accuracy with this model architecture, but instead hinder it.

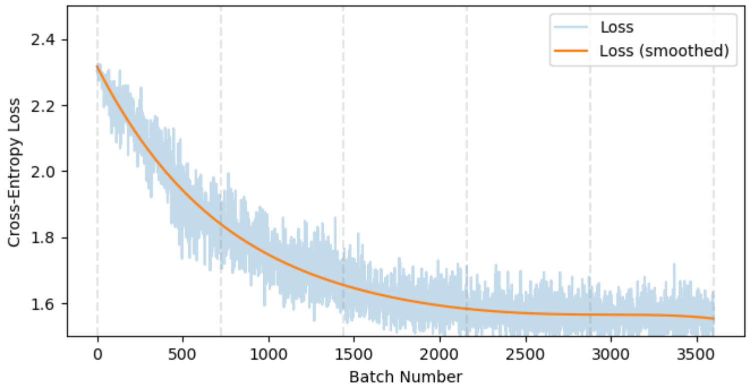

Figure 3: Cross-entropy loss over the training process for 5 epochs in mini-batches of 50 examples (3600 training steps with a 90%-10% training-validation split) with model `Net1`. Vertical grey lines indicate epochs. We see that our model converges in about 4 epochs in this example.

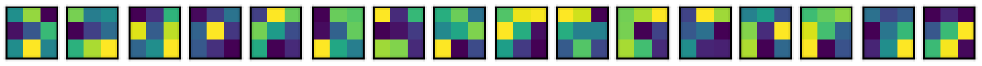

Figure 4: The learned kernels weights in the first layer of the best-performing model, `Net1`. This layer is 16 channels, each 3×3, with a stride and padding of 1 pixel each.

6 Discussion and Conclusion

Based on the results presented in the Section 5, it is clear that the preprocessing techniques implemented are detrimental to the task and models at hand. Our best performing model (`Net1`) was based off of VGGNet and contained seven convolutional layers (3x3 convolutions, padding of 1, batch normalization and max pooling), two fully connected layers, and a softmax activation function at the end. In attempt to improve this model, skip connections are added, but these did not improve the model. Our best performance was 91.4% on the validation set and 91.0% on the accessible Kaggle test images. Models which include skip connections, such as ResNet, use them to reduce complexity on very deep models. Our model which used skip connections (`Net2`) is relatively shallow – thus the benefit which has been demonstrated in literature did not apply here, and performed worse than the more straight-forward `Net1`. A simple model of an inception network is implemented, but due to resource constraints with training, it is not deep enough to outperform our simpler model(s), and performed quite poorly. We predict that if we were able to deepen the network and have training converge in a realistic time frame, this model would perform better. These results points to future work and improvements via developing deeper models with more inception layers or simple convolutional layers with skip connections – however more computational resources are needed for this. We could also ensemble multiple, simple models together – something which was not explored here but we predict could provide better accuracy without largely increasing the resources required.

7 Statement of Contributions

Benjamin MacLellan Pre-processing tests/analysis, first draft of simple model, literature review, writeup

John McGowan Final models, literature review, writeup

Mahad Khan Literature Review, Resnet

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[3] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[5] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. *Proc. 27th International Conference on Machine Learning*, 2010.

[6] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[8] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2016.

[9] Sagnik Majumder, C. von der Malsburg, Aashish Richhariya, and Surekha Bhanot. Handwritten digit recognition by elastic matching, 2018.

[10] Xing Lin, Yair Rivenson, Nezih T Yardimci, Muhammed Veli, Yi Luo, Mona Jarrahi, and Aydogan Ozcan. All-optical machine learning using diffractive deep neural networks. *Science (New York, N.Y.)*, page eaat8084, jul 2018.

[11] Yann LeCun, Corinna Cortes, and Christopher Burges. The mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist`, 2018. Accessed: 2019-03-15.

[12] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

Appendix A Preprocessing Methods

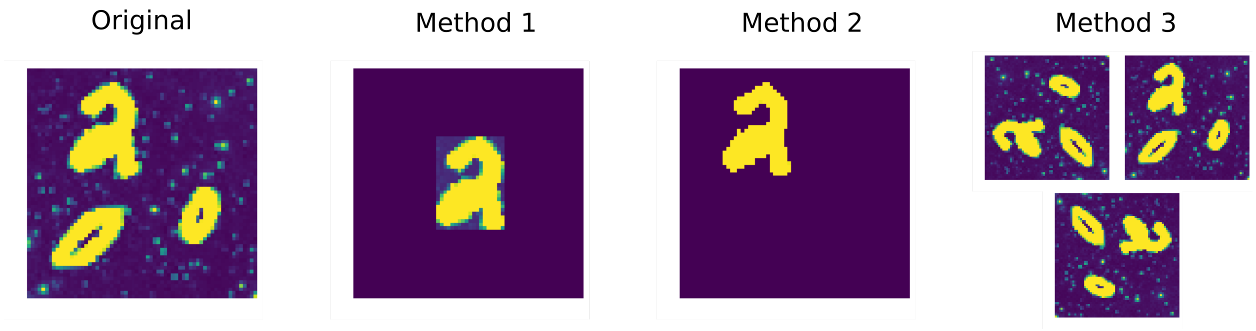

Figure A.1: Preprocessing techniques tested. These methods worsed the accuracy of the models.