MLP Coursework 2: Exploring Convolutional Networks

s1818503

Abstract

Convolutional neural networks are a very strong framework for many complex tasks including image classification. Figuring out how to maximize their performance could help research across many different sectors that now utilize deep learning in various ways. An integral part of CNN architectures are layers that perform dimensionality reduction. There are many different methods to achieve that, each with different strengths and weaknesses. We performed experiments using different methods while varying the number of free parameters to determine the utility of each one. No method was proven to be superior in all respects and the most suitable should be chosen based on the task and resources available.

1. Introduction

Convolutional neural networks (CNNs) have proven to be very effective for image classification tasks and significantly outperform algorithms utilising carefully designed filters by computer vision experts or simpler deep learning methods like a fully connected multilayer perceptron (MLP).

The main mechanism in CNNs is the operation of convolution. The input is convolved with filters whose parameters can be learned by gradient optimisation. This turned out to be an extremely powerful idea. Convolutions force the network to analyze a neighbourhood around each pixel and thus preserve the spatial information of the image efficiently in contrast to a fully connected MLP. Additionally, letting the network to form its own filters surpassed human capabilities and imagination. Also, the fact that the filters share weights across the whole input significantly reduces the number of necessary free parameters when compared to a fully connected MLP on a similar task.

One of the main reasons for the success of deep learning systems in classification problems is their ability to learn more abstract representations of the input data. In a fully connected MLP this happens when reducing the number of hidden units inside a layer with the output layer having representations for entire classes.

Dimensionality reduction is similarly very important for CNNs and there many ways of implementing it, for example, max or average pooling, using convolutions with bigger strides and finally dilated convolutions. All these methods operate significantly differently but share a common goal of dimensionality reduction. It is not clear which method is superior for image classification and that is what we are to explore in this coursework.

We start by implementing a convolutional and max pooling layer and continue by running experiments across different architectures and dimensionality reduction methods in order to determine their utility. We use a model very similar to the standard architecture of CNNs for image classification (Figure 1) by alternating convolutional layers with dimensionality reduction layers.

Our experiments were run on the EMNIST dataset (Cohen et al.) described in detail in our previous coursework. Our training set consisted of 100000 examples and our validation and test sets of 15800 each.

2. Implementing convolutional networks

Implementation of the convolutional and max pooling layers was done using 4D tensors and the correlate2d function from **scipy**. The input tensor consisted of a batch of images. The first dimension corresponded to the index of the example in the batch, the second to the input channels (eg different colour channels for an RGB image) and the final two were the standard dimensions of the 2D array. A demonstration of 3D input corresponding to one batch example is found in figure 2.

The forward propagation of the convolutional layer for a 4D input tensor H^{l-1} and a 4D kernel tensor W can be concisely summarised by the following equation using python like indexing:

$$\mathbf{H}^l[i, j, :, :] = \sum_{k=0}^{C_{in}-1} (\mathbf{H}^{l-1}[i, k, :, :] \otimes \mathbf{W}[j, k, :, :]) + \mathbf{b}[j]$$

where $\otimes$ represents the cross-correlation operation and **b** is a vector of biases.

For the backpropagation, we have to calculate both the gradients with respect to the inputs and the parameters. Given a 4D tensor $\frac{\partial E}{\partial H^l}$ representing the gradients with respect to the outputs we can calculate the gradients with respect to the kernel weights:

$$\frac{\partial E}{\partial \mathbf{W}}[i, j, :, :] = \sum_{n=0}^{N-1} (\mathbf{H^{l-1}}[n, j, :, :] \otimes \frac{\partial E}{\partial \mathbf{H}^l}[n, i, :, :])$$

The gradient with respect to the biases is just a sum of $\frac{\partial E}{\partial \mathbf{H}^l}$ across the first, third and fourth dimensions.

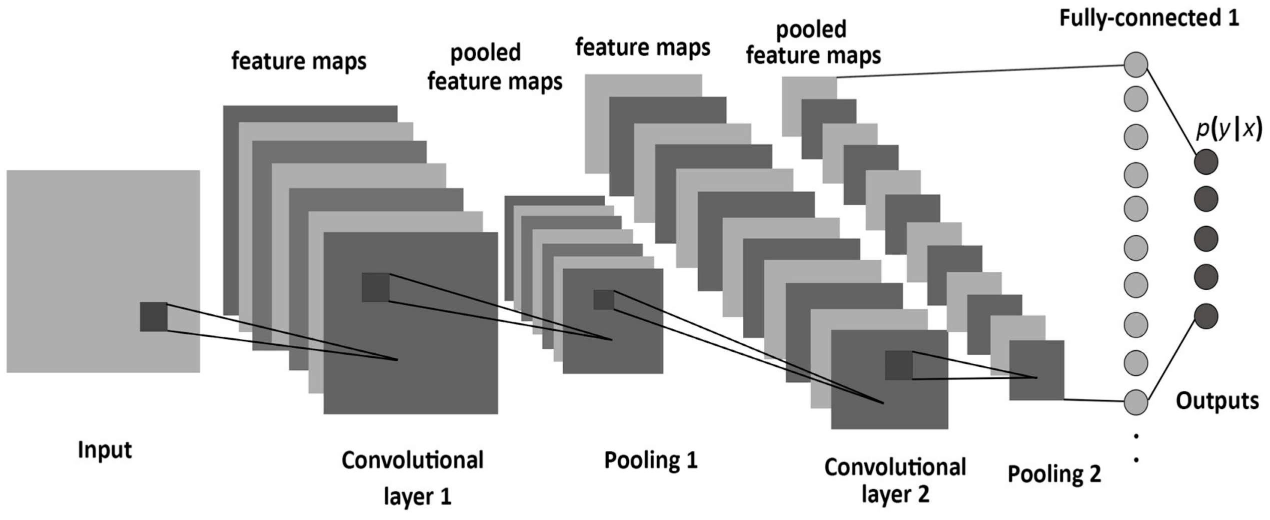

Figure 1. Typical architecture of a convolutional neural network. The input is processed through a convolutional layer then a pooling layer performs dimensionality reduction. Picture by (Albelwi & Mahmood, 2017).

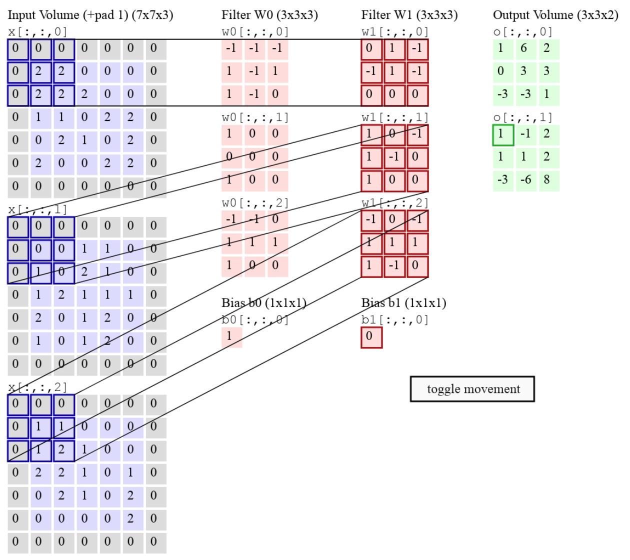

Figure 2. Demonstration of a convolution operation for three input channels. Figure by cs231n course.

To calculate the gradients with respect to inputs we first padded $\frac{\partial E}{\partial H^l}$ with zeros and flipped the kernels in the 3rd and 4th dimension. Then we implemented the following equation:

$$\frac{\partial E}{\partial \mathbf{H}^{l-1}}[i, j, :, :] = \sum_{k=0}^{C_{out}-1} \left(\frac{\partial E}{\partial \mathbf{H}^{l}}[i, k, :, :]_{pad} \otimes \mathbf{W}[k, j, :, :]_{flipped}\right)$$

This implementation of the convolutional layer is not as efficient as using the im2col function which unfolds the 4D tensor to 2D array and does the entire convolution operation for a batch using a single matrix multiplication. It does however require significantly less memory. It is worth noting that the best of both worlds could be possible by methods such those presented by (Anderson et al.). We also added assertions to check whether the output of the convolutional layer would have a valid shape.

The max pooling layer was also implemented for 4D tensors. For a 2D array, the max pooling is done by sliding a window over it and returning the value of the biggest entry inside this window. Thus in the 4D case, the first two dimensions remain the same and max pooling is applied to each batch example and input channel. We only implemented max pooling for non-overlapping windows as that is how they are most commonly used for deep learning.

Backpropagation was done by multiplying the gradients with respect to outputs with a mask consisting of ones in the positions of the maximum elements and zeros elsewhere. In the case of draws, our implementation would assign gradients to all elements the maximum value but this should happen very rarely in convolutional neural networks and thus we did not normalise them by the number of draws to save computation time.

3. Context in convolutional networks

Dimensionality reduction is necessary for CNNs to learn abstract features as was mentioned in the introduction. By lowering the dimensions in deeper layers a neuron is accessing information from a bigger part of the original input. This part of the image that the neuron has access to is called its receptive field.

One very common method of performing dimensionality reduction is average pooling. This is very similar to max pooling which was described in the previous section but instead of the maximum in the window, the average of the entries is returned. Recently max pooling is usually the

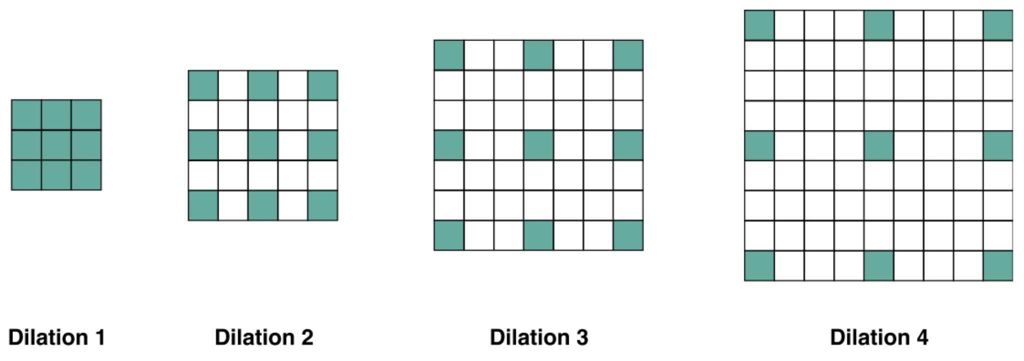

Figure 3. Graphical representation of dilated convolutions. Figure by (Antoniou et al.).

favoured method out of the two but its superiority is not clear. Great results can be achieved with average pooling as seen in (Jarrett et al.). It seems like both average and max pooling are effective in specific situations and have different advantages and disadvantages as illustrated in (Yu et al.) where a stochastic combination of both methods was used with great success.

Another option is to just use standard convolutional layers but with bigger strides. The term stride refers to the number of pixels the kernel is moved during a convolution. Usually, convolutional layers can maintain the same dimension as their input by using a stride of 1 and some padding depending on the size of the kernel. Bigger strides reduce dimensionality without performing any pooling and have been shown to perform just as good as max pooling in image recognition by (Springenberg et al.).

A new method that has started proving itself for in many different applications is that of dilated convolutions. This changes the way that convolutions are calculated. Instead of taking direct neighbours of each pixel the kernel skips some pixels depending on a dilation parameter. This is illustrated graphically in figure 3.

Dilated convolutions have been applied with great success to audio generation in (Van Den Oord et al.) where it is argued that they can substitute a deeper network architecture due to their rapidly increasing receptive field. Another impressive application was for semantic segmentation (Yu & Koltun) where they outperformed other state-of-the-art methods. Finally, in (Antoniou et al.) they were used for relational reasoning, a task in which traditional CNN architectures with pooling layers struggle.

All these methods have different strengths and weaknesses and it is unclear which one would perform better for a simple image classification task. Does the superiority of a dimensionality reduction method depend on other attributes of the network or is there a clear best performer? What happens when the performance of the network is bottlenecked by restricting the number of free parameters? Does any method succeed in learning a more economical representation of the input data? Is the use of max pooling as

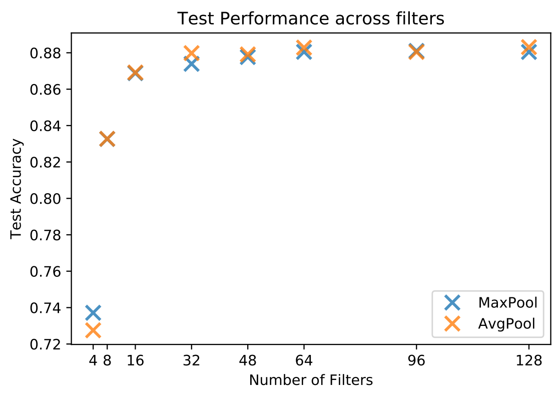

Figure 4. Test accuracy of the CNN model using the max and average pooling dimensionality reduction methods across varying number of filters in the convolutional layers.

appropriate as suggested by its popularity?

4. Experiments

In order to determine the utility of each dimensionality reduction method, we ran experiments on the same CNN architecture for all of them for image classification on EMNIST. We controlled the number of free parameters by adjusting how many filters there were in each convolutional layer.

The basic design of (Kingma & Lei Ba) the CNN we used for the experiments was the following. The network consisted of four convolutional layers with a rectified linear unit (RELU) activation function to break linearities as suggested in (Jarrett et al.). After each RELU we added a layer that performed dimensionality reduction. The final dimensionality reduction layer was followed by an adaptive average pooling layer to enforce correct dimensions and finally a softmax output with a cross-entropy loss function. All of our filters were 3×3 and padding of 1 was added to the input of each convolutional layer to keep the dimensions the same. We used the same number of filters in every

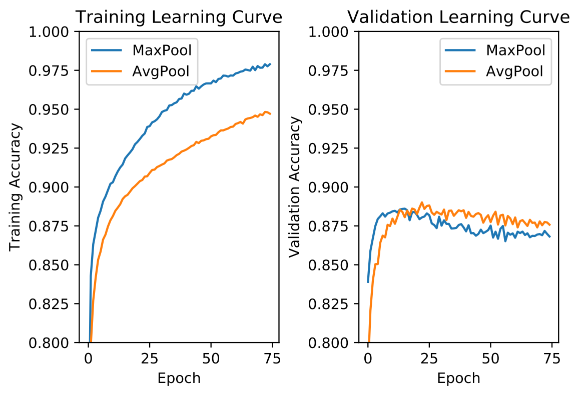

Figure 5. Learning curves for the CNN model using max and average pooling dimensionality reduction methods with 64 filters in each convolutional layer.

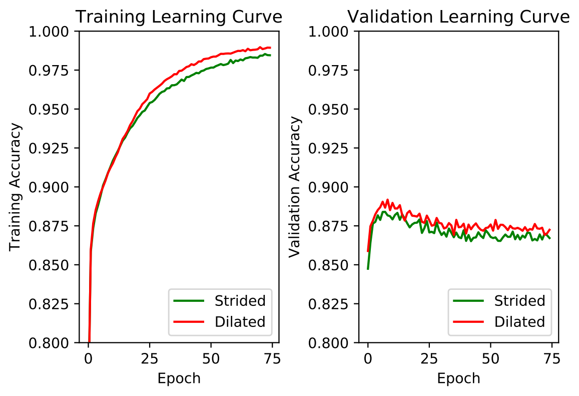

Figure 6. Learning curves for the CNN model using strided and dilated convolutions for dimensionality reduction with 64 filters in each convolutional layer.

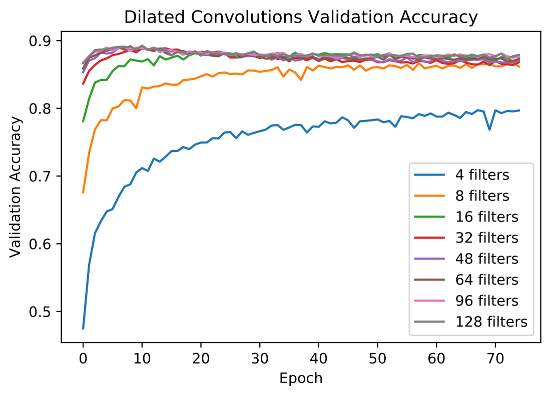

Figure 7. Validation set learning curve for the dilated convolutions method for varying numbers of filters.

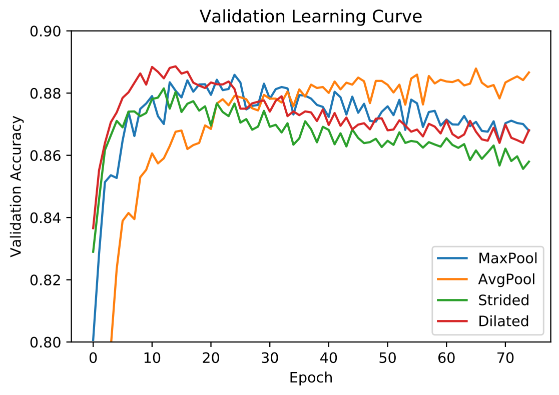

Figure 8. Validation set learning curve for all methods for an architecture with 32 filters in each convolutional layer. This showcases the overfitting tendencies of all methods apart from average pooling.

convolutional layer.

We trained each model using Adam (Kingma & Lei Ba) with weight decay (Loshchilov & Hutter) with a learning rate of 10^{-3}, default momenta and decay coefficient of 10^{-5}. This selection of hyperparameters is suggested as reasonable in the literature. No scheduler for the learning rate was used to keep the design of the experiment simpler. We allocated a budget of 75 epochs for each experiment. After the experiment was done the model corresponding to the epoch with the highest performance was selected and evaluated on the test set.

Our first experiments were done using max and average pooling with 4,8,16,32,48,64,96 and 128 kernels. We used non-overlapping pooling with a 2×2 pooling window. We included such small numbers of kernels to restrict the number of free parameters and accentuate the differences between methods. Given enough freedom and training budget the overall performance might not vary substantially between methods, especially in simple tasks. We stopped increasing the number of kernels after 128 because performance gains had saturated.

It is worth noting that the CNN with pooling managed match the performance of the fully connected MLP from our previous coursework only using 8 filters per layer with a test accuracy of 83.2% with max pooling.

Significant difference in the performance of the two pooling methods was observed only at the extremely limited experiment using 4 filters. In the rest of the experiments average pooling tended to score slightly higher than max pooling but the difference was not significant, ie less than 0.5% accuracy in the test set as can be seen in figure 4. What is quite remarkable though is the fact that average pooling managed to achieve good general performance with a consistently lower accuracy in the training set across all configurations of filters. Learning curves were all similar to figure 5. Average pooling is generally slower to train but it managed achieve better performance with the allocated budget. The best test performances for both methods

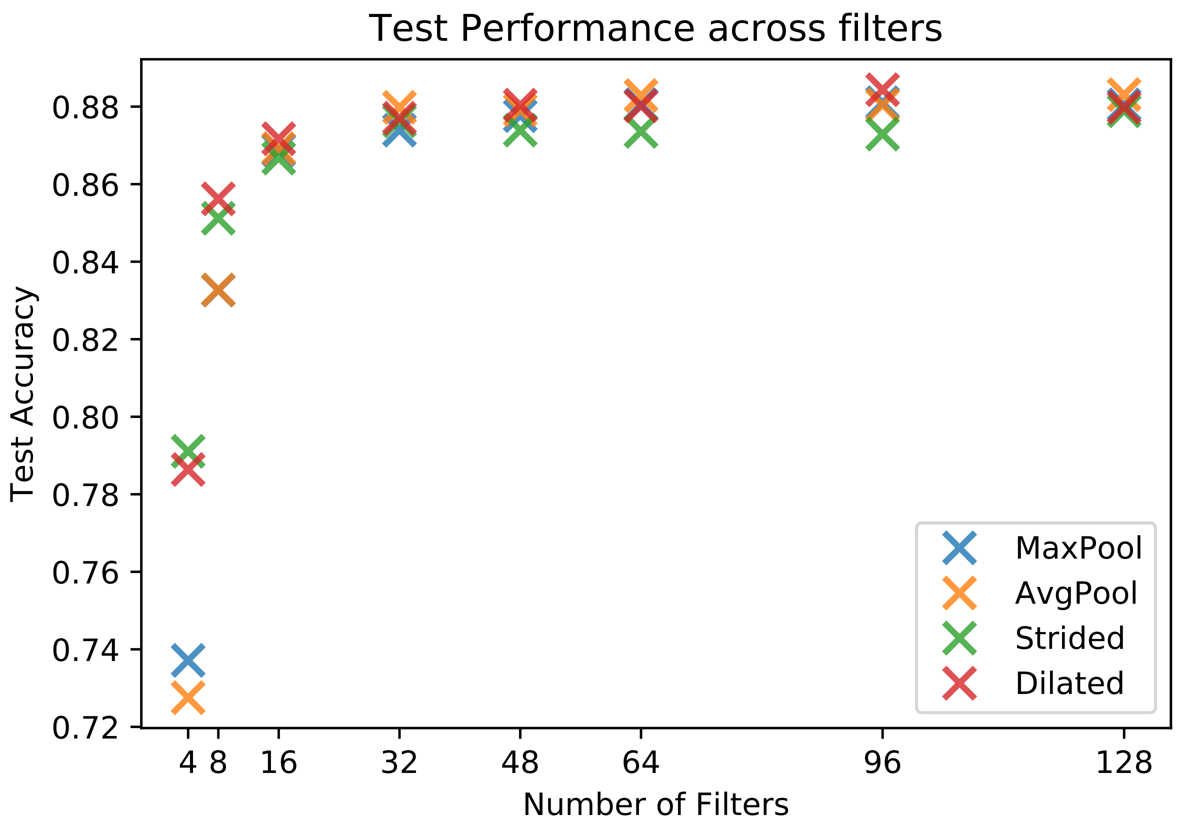

Figure 9. Test accuracy of the CNN model using across all the different reduction methods across varying number of filters in the convolutional layers.

were achieved with a high number of filters. Max pooling achieved 88.1% accuracy with 96 filters and average pooling 88.3% with 126 filters.

Next, we performed experiments with bigger strides. The same architecture was used with the standard main convolutional layers with RELU activations and stride 1. In between those we added additional convolutional layers with stride 2 kernel size 3×3 and padding 1. Thus this architecture consisted of 8 convolutional layers in total. The aforementioned selection of kernel numbers was repeated.

Strided convolutions significantly outperformed both pooling methods when each convolutional layer consisted of less than 16 filters. They surpassed the performance of the fully connected MLP when given just 8 filters with a test accuracy of 85.1%. This method achieved higher performance faster than pooling but the peak was generally very close for more than 32 filters. Its best overall test accuracy was 87.9% with 126 filters.

Finally, we tried dilated convolutions by inserting a convolutional layer with dilation between the standard ones. We used a dilation of $i + 2$ where i is an index for the dilation layer, eg the first layer corresponded to $i = 0$. By increasing the dilation parameter we rapidly enlarged the receptive field of each neuron in deeper layers.

This method had very similar convergence behaviour to the strided convolutions across all filter configurations, an example learning curve for 64 filters can be seen in figure 6. It did however tend generalize better with a peak test accuracy of 88.4% with 96 filters. A learning curve on the validation set for all choices of filters is included in figure 7 where the saturation in performance gains is evident for high numbers of filters. Respective learning curves for the rest of the methods were very similar. Although very high accuracy was reached in a small number of epochs training time was substantially longer due to a slow implementation of dilated convolutions in pytorch. A full comparison of all method performances on the test set is included in figure 9.

5. Discussion

The first thing that is evident from our results is that there is no overall best method. Strided convolutions were the best performer when free parameters were restricted while simultaneously being the worst for a large number of filters. On the contrary, average pooling started with the worst performance for 4 filters and ended up with the highest one

	Best Val Accuracy	Test Accuracy	Number of Filters
Average Pooling	**89.1%**	**88.3%**	**128**
Max Pooling	**89.0%**	**88.1%**	**96**
Strided Conv	**88.9%**	**87.8%**	**128**
Dilated Conv	**89.1%**	**88.4%**	**96**

Table 1. Best performance of each dimensionality reduction method.

for 126 filters.

Big differences between methods were only observed for a small number of filters. All four methods performed similarly for a high number of filters with strided convolutions having consistently slightly lower accuracy. While the pooling methods failed to learn as descriptive filters as the convolutional methods they did not struggle at all for more realistic configurations with a higher number of filters. It is, however, worth keeping in mind that strided and dilated convolutions do learn a more economical representation as evidenced by their high performance in the few filters regime.

Dilated convolutions are a very strong method that performed really good for all configurations of filters while also getting the best overall performance. However, the slight gain in accuracy is not enough to justify the training time required due to their slow implementation in pytorch. Perhaps their superiority is not as dramatic as in (Van Den Oord et al.), (Yu & Koltun) or (Antoniou et al.) due to the simplicity of our task. The contextual information obtained by dilated convolutions seems to be especially strong for relational reasoning but only marginally better for handwritten character classification.

One of the most striking differences, consistent for all configurations of filters, was the significantly lower performance of the average pooling method in the training set. However, this did not at all hinder its performance in the validation set or its generalisation ability. Convergence was slower but the quality of the extrema found was just as good as the rest of the methods which found smaller minima in the training set.

It is probably due to this fact that average pooling was the method least prone to overfitting. An example learning curve in the validation set can be seen in figure 8 where all methods apart from average pooling start to overfit after about epoch 15 while it continues to improve performance. It very useful to know whether a method can be used for regularisation and our results imply that average pooling can do just that. Its slower convergence is definitely an issue that may explain why it has fallen out of favour.

Max pooling while not excelling at any regime proves to be a decent dimensionality reduction method. Training is quite fast and the final performance is very close to average pooling and dilated convolutions. It also reaches peak performance in a small number of epochs. These facts probably justify its ubiquitous use in CNN architectures.

In contrast to the findings of (Springenberg et al.) strided convolutions produced generally unimpressive results although their task and architecture were similar to ours. The results, however, were not catastrophic and should not discourage the use of strided convolutions as they are also quite fast to train.

6. Conclusions

In conclusion, no method was proven to be superior in for every filter configuration. Average pooling was a very strong performer but needed more epochs to find an optimum. Dilated convolutions look very promising but they are limited due to their slow implementation and the simplicity of our task. Max pooling achieved decent performance and was fast to train. Strided convolutions were the most underwhelming method, with the worst performances for realistic configurations with many free parameters. A saturation in performance gains was observed for all methods after about 64 filters in each convolutional layer but best accuracies were achieved for 96 or 128 filters.

It is evident that no matter the dimensionality reduction method CNNs are a very strong framework for object recognition that completely outclasses a fully connected MLP. More research in their behaviour could lead to even greater performance gains. Future investigation could include stochastic combinations of different dimensionality reduction methods similar to (Yu et al.). An exploration of different hyperparameter settings could also shine more light on the maximum capabilities of each method. It is also worth similarly comparing the performance of each method in different tasks like segmentation or relational reasoning in order to more accurately identify their strengths and weaknesses.

References

Albelwi, Saleh and Mahmood, Ausif. A Framework for Designing the Architectures of Deep Convolutional Neural Networks. *Entropy*, 19(6), 2017. ISSN 1099-4300. doi: 10.3390/e19060242. URL http://www.mdpi.com/1099-4300/19/6/242.

Anderson, Andrew, Vasudevan, Aravind, Keane, Cormac, and Gregg, David. Low-memory GEMM-based convolution algorithms for deep neural networks. Technical report. URL https://arxiv.org/pdf/1709.03395.pdf.

Antoniou, Antreas, Słowik, Agnieszka, Crowley, Elliot J, and Storkey, Amos. Dilated DenseNets for Relational Reasoning. Technical report. URL https://arxiv.org/pdf/1811.00410.pdf.

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and Van Schaik, AndrÃĨ'. EMNIST: an extension of MNIST to handwritten letters. Technical report. URL https://www.westernsydney.edu.au/bens/home/.

Jarrett, Kevin, Kavukcuoglu, Koray, Ranzato, Aurelio, and LeCun, Yann. What is the Best Multi-Stage Architecture for Object Recognition? Technical report. URL http://yann.lecun.com/exdb/publis/pdf/jarrett-iccv-09.pdf.

Kingma, Diederik P and Lei Ba, Jimmy. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. Technical report. URL https://arxiv.org/pdf/1412.6980.pdf.

Loshchilov, Ilya and Hutter, Frank. Fixing Weight Decay Regularization in Adam. Technical report. URL https://github.com/loshchil/AdamW-and-SGDW.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin. STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET. Technical report. URL https://arxiv.org/pdf/1412.6806.pdf.

Van Den Oord, AÃď'ron, Dieleman, Sander, Zen, Heiga, Simonyan, Karen, Vinyals, Oriol, Graves, Alex, Kalchbrenner, Nal, Senior, Andrew, and Kavukcuoglu, Koray. WAVENET: A GENERATIVE MODEL FOR RAW AUDIO. Technical report. URL https://arxiv.org/pdf/1609.03499.pdf.

Yu, Dingjun, Wang, Hanli, Chen, Peiqiu, and Wei, Zhihua. LNCS 8818 - Mixed Pooling for Convolutional Neural Networks. doi: 10.1007/978-3-319-11740-9{_}34. URL https://pdfs.semanticscholar.org/de66/4f22dd4c7b4c15ac4a52513004aee55765ff.pdf.

Yu, Fisher and Koltun, Vladlen. MULTI-SCALE CONTEXT AGGREGATION BY DILATED CONVOLUTIONS. Technical report. URL https://arxiv.org/pdf/1511.07122.pdf.