Exploring Convolutional Networks

Stylianos Milisavljevic

Abstract

Convolutional Neural Networks (CNN) are considered state-of-the-art for analyzing visual imagery. However for machine learning beginners can seem intimidating. First, CNN's two main components: Convolutional Layers and Pooling Layers, are explained in an implementation centric context, that also discusses underlying memory and efficiency issues. Then context expanding methods: striding, dilating and pooling are explored more deeply. Their hyperparameters are investigated, together with other questions researched. Using the EMNIST dataset we train and evaluate a number of different models. Convolution with none of the context expanding methods, performs at much slower than the rest, indicating that the methods are indeed helpful for CNN. The number of filters affecting the performance of each method directly. The Average Pooling seems to outperform Max pooling unlike most of the latest trends. Striding has the lowest performance when compared to the other methods. Dilation generally outperforms the other trained models. Finally, a new pooling layer that combines max and average pooling was introduced and shows promising results.

1. Introduction

Since the introduction of Convolutional Neural Networks (CNN) by Lecun et al., CNNs have demonstrated state-of-the-art performance on tasks that analyze visual imagery. After Krizhevsky et al. used them in *AlexNet* to beat the ImageNet image classification challenge, there is widespread applications of CNNs that continue to advance in a spectacular rate (Szegedy et al., 2014) (Simonyan & Zisserman, 2014) (He et al., 2015) (Kruthiventi et al., 2017). Unlike CNNs, fully connected-networks struggle with image classification due to the relatively bigger number of parameters that have to be learned and because the spatial structure of the input image is ignored (topological information is not taken into account).

Convolutional neural networks therefore are essential for any machine learning engineer's arsenal. This article's objective is two-fold.

First, to explain the implementation of convolutional networks and discuss efficiency of different approaches. Learning to use CNNs for the first time is generally an intimidating affair. Unlike fully-connected layers, the output shape of a convolutional layer can be affected by a number of parameters: input shape, kernel shape, padding and strides. Manually implementing a convolutional network, is a valuable learning experience that can also help understand the underlying efficiency issues.

Second, to provide further insight into the context broadening techniques of convolutional networks by exploring their interactions with different parameters and architecture choices. Pooling, stride and dilation are all techniques used to provide invariance to small translations of the input as well as incorporate information at different scales.

Some research questions were raised and experimentation was done. How is the number of filters affecting these techniques, how well does a system with none of these techniques compared and how does the hyperparameter of each affect them, are all questions research here.

Furthermore, a new pooling layer that combines max and average pooling was introduced and shows promising results

My experimental setup uses the EMNIST (Extended MNIST) Balanced dataset (Cohen et al., 2017). EMNIST is comprised of centred *digit* and *letter* images of dimensions 28×28. For this environment I've used 47 out of the 62 (10 digits, 26 lower case letters, and 26 upper case letters) potential classes in the data set. The 15 letters that are missing is because it's difficult to differentiate between their upper and lower case counterparts thus the two versions were merged. For the purpose of reproducing the environment of the experiments the 15 letters are: C, I, J, K, L, M, O, P, S, U, V, W, X, Y, Z.

The dataset was split into a training set used for learning with size 100000, validation set used to tune the hyperparameters with size 15800, and finally a testing set with size 15800, used to evaluate the trained model. Modern neural networks typically use mini-batch Stochastic Gradient Descent (SGD) (a middle-way between online SGD and batch gradient descent), because they yield more stable and reliable training (Masters & Luschi, 2018). For this environment I've used a batch size of $n = 100$ for all three training, validation and testing sets. I have experimented using smaller batch sizes as suggested by Masters & Luschi, $n = 32$, however the training run-time per epoch increased significantly, making it a worse choice because of the limited computational resources and time.

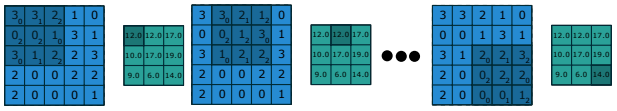

Figure 1. Convolutional layer inputs (blue) and outputs (green), with *kernel* size $F = 3$ and stride $s = 1$ shown in the shaded areas. (Dumoulin & Visin, 2016)

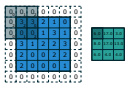

Figure 2. Padded convolution $p = 1$ (Dumoulin & Visin, 2016)

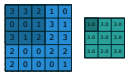

Figure 3. Maxpool layer with *pool* size $F = 3$ (Dumoulin & Visin, 2016)

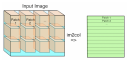

Figure 4. Input image into column form example (Warden, 2015)

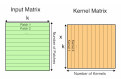

Figure 5. Matrix multiplication example (Warden, 2015)

2. Implementing convolutional networks

Convolutional neural networks are comprised of an input layer, an output layer, and a number of hidden layers. In a CNN the hidden layers are usually Convolutional layers, Pooling layers, Fully-Connected layers and Normalization layers (Lecun et al., 1998) (Krizhevsky et al.) (He et al., 2015).

Convolutional layers the heart of CNNs, are a set of learnable filters. Usually compared to the mammalian visual cortex (Cireşan et al., 2011) (Lecun et al., 1998), the filters have a receptive field that analyzes the image. The filter is a (conv kernel) usually small in term of width and height (kernel size $F \times F$, where usually $F \in \{3, 5, 7\}$ (Lecun et al., 1998) (Krizhevsky et al.) (He et al., 2015)), resulting in a sparse transformation. Only a few input units contribute to a given output unit. Each filter however slides (convolves) across the width and height of the input. The sliding is controlled by the stride parameter which is the number of pixels the filter moves. A method to increase the size of the feature map is to add zero padding to the edges of the image. For simplicity in the implementation only stride $s = 1$ is considered e.g. the filter will jump 1 pixel at a time, and no padding $p = 0$.

At each location, the dot product (Equation 1) between each element of the kernel and the input element it overlaps is calculated and a two-dimensional feature map is produced as shown in Figure 1.

$$h_{ij} = \sum_{k=1}^{F} \sum_{l=1}^{F} w_{k,l} x_{k+i-1,l+j-1} + b \tag{1}$$

where i and j are the coordinates of the feature map and b the bias.

Note that filters can be stacked together along the depth axis, for example: if an image has 3 colour channels it would need depth 3. The convolution helps contain the number of learnable parameters as the weights are shared between the different depths and has a regularization effect (introduces inductive bias towards local correlations).

For the calculation of the backward propagation, the gradient of the error surface E with respect to the parameters H^{l-1} (layer l with output shape $M \times N$) $\frac{\partial E}{\partial H^{l-1}}$ is calculated by a convolution operation but with rotated filters. Then the gradient of the error surface E with respect to the parameters W^l (layer l) can be calculated by:

$$\frac{\partial E}{\partial w^l_{r,s}} = \sum_{m=1}^{M^l} \sum_{n=1}^{N^l} \frac{\partial E}{\partial h^l_{m,n}} h^{l-1}_{r+m-1,s+n-1} \tag{2}$$

As noted before, *Pooling layers* are part of the CNN architecture. Usually used between Convolutional layers, their purpose is to reduce the spatial size of the representation. Thus reduces the number of parameters and computation for the network as well as controls overfitting by increasing the context (Section 3). A Pooling layer uses a function like Max (Figure 3) or Average on every input channel and down-samples it according to its pool size p (filter). For example a Max Pooling layer will discard 75% of the input activations with $p = 2$ and $s = 2$, because the pool of shape 2×2 will take only maximum number over the 4 values for the output. For backpropagation there is no gradient with respect to non maximum values, since changing them slightly does not affect the output. Thus the gradient from the next layer is passed back to only the neuron that achieved the max, while other neurons get zero gradient.

When implementing the filters for the Convolutional layer and the Pooling layer, there are different methods of approach. The straight forward but inefficient way is to do an element wise convolution.

A more efficient approach uses "*Serialization*" which arranges the data in a way that the convolution output can be achieved by matrix multiplication (Figure 5). Each convolution is converted into a column (Figure 4) using im2col (mat). If the stride is less than the kernel size, the produced columns include a lot of data duplication, thus expanding the memory requirements considerably.

However, General Matrix to Matrix Multiplication (GEMM) (BLA, 2002) which is also used in Fully-Connected layers, has been highly optimized (Ciregan et al., 2012) and benefits from very regular pattern of memory access. Large matrix to matrix multiplications are better than element wise convolution, because there are significantly less irregular memory accesses. Using *Serialization* is very important for the efficiency of CNNs as over 95% of the GPU time and 89% of the CPU time in AlexNet (Krizhevsky et al.) is spend on the Convolutional and Fully-Connected layers (Jia, 2014). Further efficiency has been achieved if it's combined with the use of a GPU where the serialized data is in a GPU tensor form (Cireşan et al., 2011).

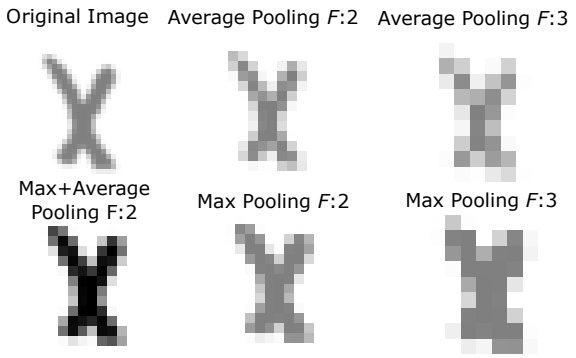

Figure 6. Different Pooling layer outputs visualized from a sample image (EMNIST Dataset). The intensity of black corresponds to the weights of the image.

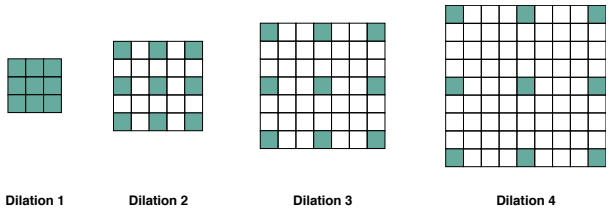

Figure 7. Kernels for dilated convolutions with successively increasing receptive field sizes. (Antoniou et al., 2018)

3. Context in convolutional networks

One feature of CNNs that is often overlooked in the ability to integrate information at different scales. The construction of the feature maps can be considered hierarchical (Zeiler & Fergus, 2013). Layers closes to the input are dealing in a more local context, learning shapes like corners and other edge/colour junctions. As the layers get closer to the output more complex textures are captured (e.g. mesh patters), each unit is trained over much larger regions of the image.

Expansion of context, while keeping a small kernel size for the filters, can be attributed to several dimensionality reduction techniques.

Discussed in Section 2, *Pooling layers* are used to increase the receptive field (Luo et al., 2017) by non-linearly downsampling the input (Figure 6). There are several functions that implement pooling, but in this paper only Max Pooling (Nagi et al., 2011) which is the most used these days and Average Pooling (Lin et al., 2013) are considered. A research question, concerning only pooling that is being experimented, is how Max Pooling compares against Average Pooling. Does this trend of switching from Average Pooling to Max Pooling justified? In addition, I experiment with the effect of summing the Max and Average Pooling, creating a new Max+Average Pooling layer. The intention is to question whether this new layer is able to improve the classification by capturing the edges that the Max Pooling layer (Nagi et al., 2011) tends to extract while "smoothening" the output by also adding the Average pooling value.

Also discussed in the Implementation Section 2, *stride* is broadening the context captured by the CNN. Stride reduces the size of feature maps, because of the different shifting distances of filters over the image (Figure 1). Thus, after each Convolutional layer the feature map is decreased and the receptive field is increased. A research question asked here, is how does different strides compare to pooling. Can stride replace pooling? (Springenberg et al., 2014)

Finally, there's Dilation (Yu & Koltun, 2015), a fairly new advancement in the CNN architecture. Not only used in computer vision (Yu & Koltun, 2015) and its relational relational reasoning (Antoniou et al., 2018), but also in speech synthesis (van den Oord et al., 2016). Dilation expands the context, by altering the filters of a Convolutional layer. It adds spaces between each cell of the filter (Figure 7). Thus, the dilation hyperparameter is added to the Convolutional layers to control the number of spaces between each cell. The more important point is that the architecture is based on the fact that dilated convolutions support exponential expansion of the receptive field without loss of coverage and resolution. The key difference from polling and strided convolutions is that they reduce the resolution. The number of parameters is not affected by the spaces between them thus remains constant with increasing dilation. How does a constant dilation compare to increasing dilations, is one of the questions investigated here.

Having detailed each of the context broadening methods, some research questions that can be asked for all three are: Is a CNN with no pooling as effective? How does the number of filter influence each method? How does the number of layers influence them?

4. Experiments

Having asked a number of research questions for context broadening techniques detailed in Section 3, experiments had to be designed and executed to test the hypotheses. Here it should be noted that the experiments were constructed so that where possible only one component / hyperparameter is varying at a time, so that the effect of the component-change is more clear. All models were trained for 100 epochs, and as indicated in Section 1, were trained and tested on the EMNIST dataset (Cohen et al., 2017). The best model for each architecture was picked using the validation set accuracy and then the testing set accuracy was used to test how well the model generalized. It should be mentioned that each experiment could have been executed with multiple random seeds to minimize "randomness", however due to the limited computation power and time, the experiments have been executed once. The GPU used for was one Nvidia K80, thus the training time detailed in some experiments should vary in different systems.

For the purpose of reproducing the experiments, here are the default hyperparameters that unless stated, remain the same between experiments.

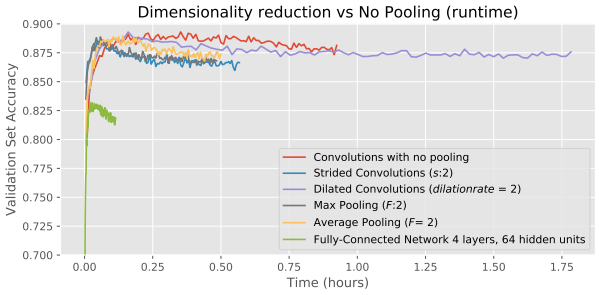

Figure 8. Runtime comparison of different architectures

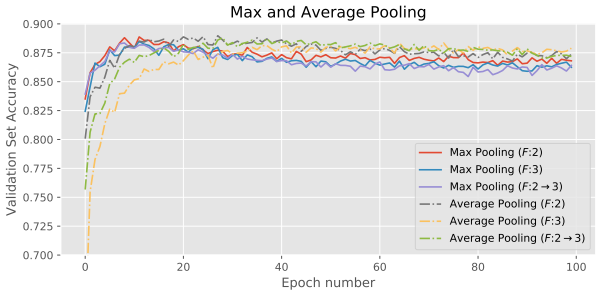

Figure 9. Max and Average Pooling with varying pool size

Number of layers = 4
Number of filters = 64
Kernel size $F = 3$
Pool size $F = 2$
Stride $s = 2$
Dilation = i+2 (where i is the layer depth)
Padding = 1

Regarding the architecture, the input is a 28x28x1 image. Each subsequent layer starts with a Convolutional layer (*CONV*), followed by ReLU (*RELU*) activation function, followed by the dimensionality reduction layer (*CONV* then *RELU* for stride and dilation, or for Pooling layers their appropriate Max or Average *POOL* layer). Finishing at a Fully-Connected Softmax output layer.

Started with question of how does *no pooling* (neither dimensionality reduction method used) convolutions compare to dimensionality reduction methods. As a baseline, a Fully-Connected Network with 4 layers and 64 hidden units was trained for comparison. The results of the experiment are detailed in Table 1, while Figure 8 gives the validation accuracy over the runtime. For the interpretation of the results of this and the following experiments, see Section 5.

After, an experiment that investigates whether *stride* can replace pooling, as suggested by (Springenberg et al., 2014). Springenberg et al. proposes that increasing the stride in Convolutional layers can outperform pooling layers. The hyperparameter varied here is the stride, and the values tested were 2,3 and 4. These values were chosen because they are the closest to the kernel-size, thus tests for overlapping, non-overlapping and some cell skipping while moving the kernel. Table 2 shows the effect of stride on the performance and the runtime of training.

How does the *number of filters* affect each dimensionality reduction approach? This experiment varies the number of filters to evaluate its effect on each method. The values used for the number of filters are 16, 32, 64 and 128. Additionally an increasing number of filters 16 $\rightarrow$ 32 $\rightarrow$ 64 $\rightarrow$ 128 (doubling each deeper layer) was added inspired by the hypothesis that less filters are needed for the shallower layers that work in at a local context and more filters are needed for the deeper broader context layers (Zeiler & Fergus, 2013).

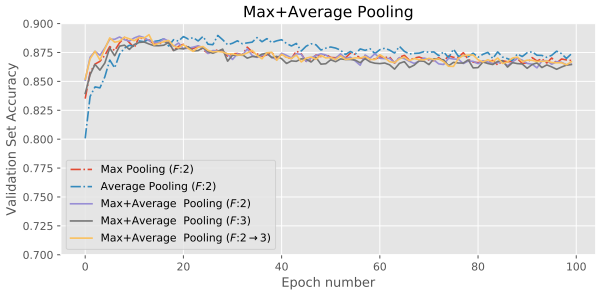

Figure 10. Max+Average Pooling with varying pool size and the best performing Max and Average Pooling models for comparison

Table 3 show a summary of the measured results.

Next an experiment was designed to test whether *Max Pooling* leads to more accurate classification than *Average Pooling* as suggested by Scherer et al.. The pool size (F) was the variable for this experiment. The pool sizes tested were 2, 3 and starting a pool size of 2 for the first to layer then changing to a pool size 3 for the 2 last layers. No higher pool size was used because the input images are already small (28×28), a lot of information will be lost. The increasing pool size was motivated by the hierarchical construction of the feature maps (Zeiler & Fergus, 2013). Figure 9 and Table 4 shows the results.

Then as mentioned in the Context Section 3, the *Max+Average Pooling* layer was introduced. The aim of this experiment is to evaluate its effectiveness in image classification compared to the conventional pooling layers. Similar to the Max and Average Pooling layers comparison, the variable hyperparameter was the pool size. The resulting measurements can be found in Figure 10 and Table 4.

Finally, *dilation* is considered. Different dilation rates can have different effects on the neural network and this is what this experiment is intended to investigate. The dilation rates considered are $i + 1$, $i + 2$, $i + 3$ and constant dilation of 2 (where i is the layer depth). Note that for $i + 3$, padding 2 and stride 2 have to be used to have valid output shapes. Increasing the dilation rate, reduces the output layer size faster (Yu & Koltun, 2015). Table 5 shows the results of

Architecture	Test Set Accuracy	Best model epoch	Average Epoch Runtime (s)	Runtime until best (s)
Fully-Connected	0.816	17	4	70
No Pooling	0.883	38	33	1269
Stride	0.877	9	20	185
Dilation	0.883	9	64	576
Max Pooling	0.880	12	17	209
Average Pooling	0.881	28	18	504

Table 1. No Pooling architecture compared with networks that use context extension methods and a baseline Fully-Connected Network

Stride	Testing Set Acc.	Runtime to best model
2	0.877	185
3	0.874	203
4	0.867	222

Table 2. Stride variation

the experiment in detail.

5. Discussion

What follows is an analysis of the results from the experiments detailed above.

First we consider the experiment evaluating *no pooling*. From Table 1, we can conclude that all of the CNNs outperformed the baseline model with a significant margin, this was expected for the reasons outlined in the introduction (Section 1). No pooling has the same testing accuracy **0.883** (3 d.p.) as the *dilated* convolution. However dilated convolution trained the best model in less than the half runtime required by no pooling. Because there is no dimensionality reduction, the parameters (weights) that the no pooling architecture has to train is significantly larger than dilation and the rest of the architectures resulting in a slower convergence to an optimal model. Figure 8 gives a more insight on the runtime for each architecture. Notice that the Fully-Connected Network is much faster. The reason is that Fully-Connected Layers require less computation because they use the highly optimized matrix multiplication directly, more details on Implementation Section 2.

Considering the experiment that tests the *stride* hyperparameter. When stride is 3, the filter slides in a non-overlapping manner as the kernel size is 3. Some data loss occurs when stride is 4, because its bigger than the kernel size, and this explains the worst performance of the tested architectures. When stride is compared with both Max and Average Pooling with $F : 2$, the pooling methods of dimensionality reduction are considerably better with testing accuracies **0.880** and **0.881** respectively, while the best stride model $s : 2$ achieves an accuracy of **0.877**. Unlike Springenberg et al., the results suggest that stride alone is an inferior context broadening method. Springenberg et al. states that the dataset has to be large enough so that the network is learning all the necessary invariances with just convolutional layers. Our dataset EMNIST has double the training images that CIFAR-10(Krizhevsky & Hinton, 2009) used by Springenberg et al. which suggests that the different conclusion relies on the different (much deeper with some different layers) architecture of the neural network .

When examining the effect that the *number of filters* have on each dimensionality reduction method, there's strong evidence that adding more filters improves the produced model (Table 3). However increasing the filters come as a cost, runtime increases significantly. Increasing the filters with each deeper layer proved to be less ideal for achieving maximum accuracy, however it achieves results close to the second best 64 filters, but in about half the time in most the cases. Intuitively the more filters the CNN has, the more features (e.g. edges, lines, object parts) it can potentially learn. The increase in training time is due to the increase in the number of learnable parameters from the added feature maps.

Comparing *Max* and *Average Pooling* from Figure 9, we can see that generally Max Pooling converges in less epochs to the optimal model while the Average Pooling takes more epochs. This can be interpreted by the fact that Max Pooling is a more "aggressive" down-sampling method (Nagi et al., 2011) that detects the edges of an image, while Average Pooling is more "smooth" (Figure 6), thus not extracting the more important features fast. Measurements from Table 4, support this observation, as the average epoch runtime is similar but the epochs that take to reach the optimal model are more than double for Average Pooling. Considering the classification accuracy there is no clear advantage of either one. Max Pooling has a better accuracy for $F : 3$, but worse for $F : 3$ & $2 \rightarrow 3$. The overall best model was achieved by Average Pooling with $F : 2$. The results, suggest Average Pooling tends to be better for the current dataset and architecture. This doesn't refute the conclusions of Scherer et al. however. EMNIST dataset is much different than the more complex Caltech (Fei-Fei et al., 2004) and NORB (LeCun et al., 2004) (bigger images with multiple channels) datasets used and edge detection is likely more important than the smooth down-sampling Average Pooling provides for EMNIST.

The *Max+Average Pooling layer* showed promising results. From Figure 10 we can see that it follows the trend of Max Pooling, while being partially better. However when an increasing pooling size is used ($F : 2 \rightarrow 3$) it outperforms every other pooling layer and size. While the other systems did not improve from the pool increase, the introduced layer proceed it's best results. A potential justification for these results is that it takes the edge detection from Max Pooling

Architecture	Best Performing Model			Second Best Performing Model		
	# of filters	Testing Acc.	Runtime to model	# of filters	Testing Acc.	Runtime to model
Stride	128	0.878	303	64	0.877	185
Dilation	128	0.884	1583	64	0.883	576
Max Pooling	128	0.880	270	64	0.880	209
Average Pooling	128	0.882	834	64	0.881	504

Table 3. Varying number of filters for each dimensionality reduction architecture

Pooling Architecture	Pool size F	Testing Set Accuracy	Epoch of best model	Average Epoch Runtime (s)
Max	2	0.880	12	17
	3	0.877	12	11
	$2 \rightarrow 3$	0.874	9	17
Average	2	0.881	28	18
	3	0.874	65	11
	$2 \rightarrow 3$	0.878	29	17
Max+Average	2	0.880	12	20
	3	0.877	13	14
	$2 \rightarrow 3$	0.883	14	21

Table 4. Pooling experiments measurements

Dilation	Test Set Acc.	Runtime to best model
i+1	0.885	640
i+2	0.883	576
i+3	0.886	741
2	0.886	1426

Table 5. Varying Dilation

(Nagi et al., 2011) but its down-sampling "aggressiveness" is toned down by Average Pooling.

Finally, we consider the varying *dilation* rate. The results cannot conclude that when increasing the dilation rate, there is an increase in the performance because the dilation rate $i + 2$ performed the worst but was not the lowest rate tested. It is worth noting that with dilation rate i+3, this is the best performing system of all the experiments with accuracy **0.886**. Compared to the other context broadening methods, dilated convolutions support exponential expansion of the receptive field without loss of resolution or coverage (Yu & Koltun, 2015). This provides a larger receptive field with the same computation and memory costs while also preserving resolution. Constant dilation 2 has the same accuracy but took twice the runtime to reach an optimal model. The dimensionality with constant dilation doesn't reduce as fast which then has an increased number of learnable parameters.

6. Conclusions

The number of filters can be considered a trade off between runtime and accuracy. More filters higher accuracy but also higher runtime, not necessarily at the same rate. As an extension to this experiment, more filters can be added to determine when the accuracy stops to improve but the runtime keeps increasing.

Results from the stride experiments suggest that it performs worse than pooling, however there is opposite evidence (Springenberg et al., 2014).

Average Pooling outperformed Max Pooling for two of the three different cases tested in the experiments. As there is literature (Scherer et al., 2010) suggesting otherwise, it reasonable to say that one isn't superior to the other, but depends on the individual case. A possible extension experiment is to testing if overlapping pooling windows (Scherer et al., 2010) can improve the performance.

The introduced Max+Average Pooling layer showed promising results, outperforming the other pooling layers. To further evaluate its effectiveness in visual classification, more experiments are suggested using some of the more complex datasets (Fei-Fei et al., 2004), (LeCun et al., 2004) and different CNN configurations.

Dilation rate need more experiments to deduce its effect on performance using different datasets that have bigger images and allow for higher dilation rates. Concluding, dilation generally has improved performance over the other methods.

References

Rearrange image blocks into columns. URL https://www.mathworks.com/help/images/ref/im2col.html.

An updated set of basic linear algebra subprograms (blas). *ACM Trans. Math. Softw.*, 28(2):135–151, June 2002. ISSN 0098-3500. doi: 10.1145/567806.567807. URL http://doi.acm.org/10.1145/567806.567807.

Antoniou, Antreas, Słowik, Agnieszka, Crowley, Elliot J, and Storkey, Amos. Dilated densenets for relational reasoning. *arXiv preprint arXiv:1811.00410*, 2018.

Cireşan, Dan C., Meier, Ueli, Masci, Jonathan, Gam-

bardella, Luca M., and Schmidhuber, Jürgen. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pp. 1237–1242. AAAI Press, 2011. ISBN 978-1-57735-514-4. doi: 10.5591/978-1-57735-516-8/IJCAI11-210. URL http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-210.

Ciregan, D., Meier, U., and Schmidhuber, J. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3642–3649, June 2012. doi: 10.1109/CVPR.2012.6248110.

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373, 2017. URL http://arxiv.org/abs/1702.05373.

Dumoulin, Vincent and Visin, Francesco. A guide to convolution arithmetic for deep learning, 2016. URL http://arxiv.org/abs/1603.07285. cite arxiv:1603.07285.

Fei-Fei, Li, Fergus, R., and Perona, P. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 Conference on Computer Vision and Pattern Recognition Workshop*, pp. 178–178, June 2004. doi: 10.1109/CVPR.2004.383.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Jia, Yangqing. *Learning Semantic Image Representations at a Large Scale*. PhD thesis, EECS Department, University of California, Berkeley, May 2014. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-93.html.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 2012.

Kruthiventi, S. S. S., Ayush, K., and Babu, R. V. Deepfix: A fully convolutional neural network for predicting human eye fixations. *IEEE Transactions on Image Processing*, 26(9):4446–4456, Sept 2017. ISSN 1057-7149. doi: 10.1109/TIP.2017.2710620.

LeCun, Y., Huang, Fu Jie, and Bottou, L. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pp. II–104 Vol.2, June 2004.

Lecun, Yann, Bottou, LÃl'on, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pp. 2278–2324, 1998.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *CoRR*, abs/1312.4400, 2013. URL http://arxiv.org/abs/1312.4400.

Luo, Wenjie, Li, Yujia, Urtasun, Raquel, and Zemel, Richard S. Understanding the effective receptive field in deep convolutional neural networks. *CoRR*, abs/1701.04128, 2017. URL http://arxiv.org/abs/1701.04128.

Masters, Dominic and Luschi, Carlo. Revisiting small batch training for deep neural networks. *CoRR*, abs/1804.07612, 2018. URL http://arxiv.org/abs/1804.07612.

Nagi, J., Ducatelle, F., Caro, G. A. Di, CireÅ§an, D., Meier, U., Giusti, A., Nagi, F., Schmidhuber, J., and Gambardella, L. M. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pp. 342–347, Nov 2011. doi: 10.1109/ICSIPA.2011.6144164.

Scherer, Dominik, Müller, Andreas, and Behnke, Sven. Evaluation of pooling operations in convolutional architectures for object recognition. In Diamantaras, Konstantinos, Duch, Wlodek, and Iliadis, Lazaros S. (eds.), *Artificial Neural Networks – ICANN 2010*, pp. 92–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin A. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014. URL http://arxiv.org/abs/1412.6806.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott E., Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

van den Oord, Aäron, Dieleman, Sander, Zen, Heiga, Simonyan, Karen, Vinyals, Oriol, Graves, Alex, Kalchbrenner, Nal, Senior, Andrew W., and Kavukcuoglu, Koray. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL http://arxiv.org/abs/1609.03499.

Warden, Pete. Why gemm is at the heart of deep learning, Apr 2015. URL https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/.

Yu, Fisher and Koltun, Vladlen. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015. URL http://arxiv.org/abs/1511.07122.

Zeiler, Matthew D. and Fergus, Rob. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL http://arxiv.org/abs/1311.2901.