MLP Coursework 2: Exploring Convolutional Networks

s1877727

Abstract

Convolutional Neural Networks seem to be the state-of-the-art architectures for image recognition. Their main contribution is the hierarchical way they represent information. This is achieved by progressively analyzing the whole image from local context to an overall holistic view. This is mainly done by dimensionality reduction. Dimensionality reduction is a very important aspect of CNNs since it allows CNNs to extract only the information needed for classification, while at the same time keeps the overall size manageable. Dimensionality reduction methods, besides the typical convolution, have been introduced to further enhance this characteristic of CNNs namely max pooling, average pooling, strided convolution and dilated convolution. In this work we experimented on the balanced EMNIST database, to test which of the methods performs better in terms of generalization. We found out average pooling to the best. We also showed that dilated convolution can perform almost as well with half resources, making it the best choice when simplicity and low resources is the goal.

1. Introduction

The fully-connected deep neural networks (DNNs) have managed to achieve outstanding performance in many tasks. However when applied to image classification tasks they did not scale with the same accuracy. DNNs heavily rely on the assumptions that the input's features are independent and that a neuron should take into account all neurons of the previous layer. This approach did not allow local attributes to be discovered effectively and also failed to acknowledge the dependence between adjacent pixels. On top of that, since every neuron had it's own weight and bias vector the network's free parameters grew exponentially on the image's size leading to overfitting. If we also take into account that an image has multiple channels (RGB model) then DNNs are too naive and complex for this task.

Convolutional Neural Networks (CNNs) were introduced to address this problem. CNNs share most of their attributes with the DNNs, that is their neurons on each layers are connected with neurons on the previous layers, they have trainable weights and biases and minimize a loss function by fitting their parameters. The most common architecture is a series of convolutional layers followed by a non-linearity and a dimensionality reduction layer, called *pooling layer*.

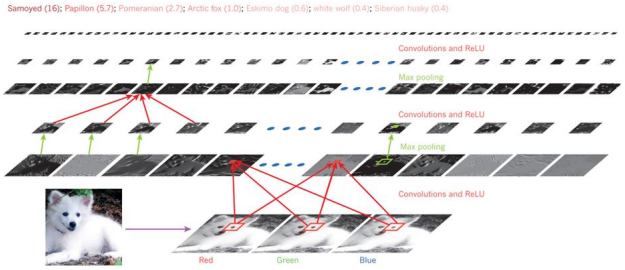

Figure 1. The general architecture of CNNs when for object classification. Convolution layer is followed by a non-linearity and then by a pooling layer, also called *subsampling layer*. After the feature extraction layers, a fully connected layer is used producing the output which is then used to decide the class of the input.

Finally, the output is a fully connected layer. This is illustrated above at *Figure 1* . In a convolutional layer a neuron is connected only to a proportion of the neurons found on the previous layers and in particular to a small window of the image. This window is called *receptive field* of the neuron. A convolutional network convolutes this window with another window of the same size called *kernel*. This kernel and the biases of each neuron constitute the trainable parameters of the layer. Sliding this window around the image and convoluting it with the kernel outputs an activation map called *feature map*. Another important aspect of the convolutional layer is *weight sharing*, which simply means that neurons of the same layer share the kernel. Since all images are convoluted with the same kernel, kernels are also called filters. The intuition behind this idea is that we want to extract similar patterns across the image such as texture, color, edges etc.(Goodfellow et al., 2016). As we will see more in *Chapter 4*, CNNs mainly differ from the DNNs in the way the extract the features, that is in a structured, hierarchical way combining detailed and different regions of the image. The first convolutional layers extract local and detailed information from the image, and following layers combine these features, augmenting the receptive fields, resulting in a much wider and but more abstract features.

Usually a dimensionality reduction layer is added. By doing so networks manage to reduce the trainable parameters of the networks, and thus avoid overfitiing, while still manage to preserve their hierarchical approach. Pooling layers provide a summary of a particular feature map region. There are multiple techniques to do so, with the most usual ones being max and average pooling. Although not a separate layer, *strided* and *dilated* convolutions, which just modify

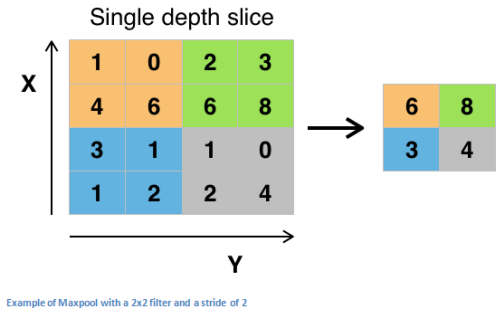

Figure 2. Example of a max pooling layer. The maximum element is selected from each highlighted area

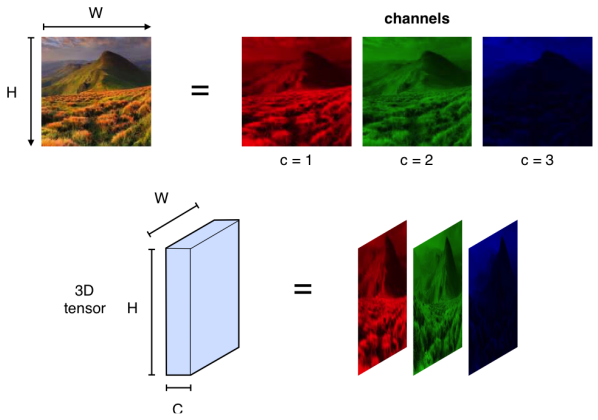

Figure 3. Images are now 3-d tensors. Besides of height and width they also have depth, which is measured in the number of channels they have. RGB model defines the number of channels to 3.

the convolution operation in-place, are also considered an effective dimensionality reduction technique, and we will analyze them too.

In this work tested the classification accuracy of different CNNs architectures to determine which generalizes better. As an extension we also tested whether there is a simple model capable of achieving good results in every configuration. Even though we found out that average pooling achieved best results, dilated convolution had almost identical performance using half the filters, making it the best alternative when time and resources are more important.

All our architectures were tested on the EMNIST balanced database. The balanced EMNIST data set is thoroughly described in our previous *coursework*. In all our experiments we used a batch size equal to 100 and all our systems were trained exactly for 80 epochs. We split our data into a training, a validation and test set consisting of 100000, 15800 and 15800 data respectively.

2. Implementing convolutional networks

In this section we explain the idea and our implementation of the basic components of a convolutional network, namely the convolution and pooling layer. First of all, a single input is not anymore a single vector but a 3-dimensional tensor. Since it is an image, the two dimension are the height In_h and width In_w. We also introduce a third dimension, the *depth* C_{in}. Depth is defined as the number of different channels an image might have, e.g RGB color model. Taking also into account the batch size N, the inputs now become a 4-dimensional tensor $In[N, C_{in}, In_h, In_w]$.

All these channels must be convoluted with a corresponding *kernel*, producing a single feature map. Since every channel must be convoluted with a kernel, kernels are also a 4-dimensional tensor with height W_h, width W_w and the same number of channels as the input. Lastly, the depth of the kernel defines the number of output channels C_{out} resulting in the kernel being $W[C_{out}, C_{in}, W_h, W_w]$. To define the output volume of the layer, we will introduce three more hyper-parameters namely stride, zero-padding and dilation. *Stride S* is the step the corresponding convolution window takes. *Dilating* the kernel by D means inserting D-1 spaces between the weights of the kernel (Chen et al.) and finally zero-padding P, meaning padding the border of the image P times with zeros.

The output is again a tensor $\text{Out}[N, C_{out}, Out_h, Out_w]$. Having defined the above, a single feature map of the output has size of:

$$Out_h = (In_h - W_h - 2P)/S + 1$$

$$Out_w = (In_w - W_w - 2P)/S + 1$$

As in every layer, we have to define the forward and backward passing, necessary for the network's training. For the convolution operation we used, the *correlate2d*() function of the scipy library which correlates two given matrices (Jones et al.). The equation for the forward passing is:

$$Out[i, j] = \sum_{d=1}^{C_{out}} In[i, d] \otimes W[j, d] + b[j]$$

By inspecting the equations we can conclude that the backward passing will also be a convolution. To compute the gradients in a convolution layer at depth l we have to calculate the $\frac{\partial E}{\partial H_l}$, $\frac{\partial E}{\partial w_l}$ and $\frac{\partial E}{\partial b_l}$. If $\frac{\partial E}{\partial H_{l+1}}$ is the error with respect to the next layer then we can define:

$$\partial E/\partial w_l[i, j] = \sum_{d=1}^{N} In[d, j] \otimes \frac{\partial E}{\partial H_{l+1}}[d, i]$$

$$\partial E/\partial b_l[i] = \sum_{d=1}^{N} \sum_{h=1}^{In_h} \sum_{w=1}^{In_w} \frac{\partial E}{\partial H_{l+1}}[d, i, h, w]$$

To calculate the $\frac{\partial E}{\partial H_l}$ we need to zero-pad the border of $\frac{\partial E}{\partial H_{l+1}}$ and convolve it with the kernel, concluding in the following equation:

$$\partial E/\partial H_l[i,j] = \sum_{d=1}^{C_{out}} \frac{\partial E}{\partial H_{l+1}}[i,d]_{pad} \otimes W[d,j]$$

Looping through the batch and applying *correlate2d*() is a memory-efficient approach to implement the convolutional layer but it is not the fastest. Serializing the input and the kernels, using a function called *im2col*(), and then applying matrix multiplication can be much faster. That said this approach is memory-heavy since it has to serialize the whole mini-batch and also has lot of repeated elements. We implemented the convolution layer using the looping approach.

Pooling layer is implemented in a analogous way as it also has 4-dimensional *In*, *W* and *Out* tensors as parameters. The only difference is that, instead of convolution it applies the non-linear max function to the "scanning" window. Pooling is able to achieve both non-linear activation and dimensionality reduction without adding any trainable parameters to the network. For the forward passing, the output's dimension are defined similarly as in the convolution layer but this time the kernel has a size of $W[C_{in}, C_{out}, 2, 2]$. Although overlapping pooling has be utilized before with benefits in reducing overfiting (Belmont & Gibbs, 2004), in our implementation, pooling is non-overlapping having $S = 2$. For the backward passing only the maximum value of each region must be multiplied by the corresponding $\frac{\partial E}{\partial H_l}$. If $m = \max(a, b)$ then the gradients are given by the following formula:

$$\partial E/\partial x = \begin{cases} 1, & \text{if x} = m, \\ 0, & \text{otheriwse} \end{cases}$$

3. Context in convolutional networks

In this section we will analyze how convolutional network model context in general and then we will analyze some of the most common techniques of dimensionality reduction; Average pooling, max pooling, dilated and strided convolution. After analyzing each of them, we will try to highlight the differences between them, their behavior and their ability to generalize.

Convolutional Networks mainly differ from the common Multi Layer Perceptons (MLP) in the way they model the context. In particular the typical pipeline is alternating multiple convolutional and pooling layers and only in the final layers having fully connected layers. In most cases a non-linearity is used between the convolutional and the pooling layer, to further enhance the network. This is highlighted in *Figure 1*.

The main function of these multi-stage networks is to initially extract local information about image at a high-resolution from different regions (most of the time, the regions are overlapping) of the image. This is done on the layers closer to the input. As depth increases the networks combine them into more complex features, but with lower resolution. In terms of the perceptive field of a neuron, it increases exponentially with the depth resulting in a more holistic view of the image. The trade-off is that it is not as detailed as the receptive field of preceding neurons. (Miao et al., 2014).

Adding multiple convolutional layers is a valid approach for this hierarchical representation, with the main drawback being increasing the number of trainable parameters and thus risking possible overfiting. An alternative approach is the convolutional layers to be followed by pooling layers which serve a similar purpose but do not add any new parameters. They summarize the extracted local patterns of non-overlapping regions and compress them even more, discarding non essential information. Besides broadening the context, these layers also make the feature map more manageable.

The most common pooling function is the *max* function, which only selects the maximum element of a region, rejecting all others. Summarizing a region by its maximum elements is an aggressive choice and distributions exist that max-pooling doesn't describe well. An alternative is the average pooling, which instead of picking just one element, computes the average of a selected region. Doing so, it achieves invariance to small distortions, smooths out possible outliers and discards less information. Even though this might be considered a more suitable option, both techniques have their own drawbacks and examples can be constructed to highlight them (Miao et al., 2014).

Although pooling layers are present in many existing CNNs architectures, there have been efforts to simplify the existed architectures by removing the need of pooling layers.It has been shown that using convolutional layers with increased stride can achieve state-of-the-art performance in classification tasks (Springenberg et al., 2014). This simple but effective modification can reduce the feature's map dimensions by just allowing less overlapping area between different receptive fields. Inflating the kernel's receptive field using dilation is also deployed in many resent works with promising results in classification, segmentation and sound processing (Wei et al., 2018) , (Antoniou & Crowley), (Oord et al., 2016). The main difference of the dilated convolution is that it increases the receptive field of the kernel in-place without sacrificing spatial information or resolution. This means that dilated convolution is a much more aggressive approach overcoming the need for very deep networks (Yu et al., 2017). Although in this work we treat dilated and strided convolution as separate layers, in reality they can be integrated inside an existing convolutional layer, by just adjusting the convolution operation that takes place. These results show that great performance can be achieved without pooling layers since slightly modifying the original convolution has a similar effect. This simplicity could also allow better understanding of the nature of CNNs.

Choosing the right procedure to reduce dimensionality is only aspect of CNNs. Multiple hyper-parameters

determine how good a CNN behaves. The number of feature maps at a given layer can be used to compensate the loss of resolution and greater depth can extract more high-level information. The size of the kernels decides how much resolution is sacrificed per level, and how much spatial information is being considered.

This diversity of methods and parameters naturally raises the question on whether there is a single best architecture in terms of generalizing better and if yes under which configuration. More interestingly, is there a relatively simple architecture which can achieve close to best performance with others, more complex ones? To answer these questions we experimented on different dimensionality reduction methods and networks configurations. In the next Chapter we explain in more details the network architecture and how each approach was evaluated.

4. Experiments

In this section we experiment on the different dimensionality reduction approaches. Our proposed network is a flexible network model in which we can tune the number of adjacent convolutional layers, the number of filters and the image's height and width. For simplicity we kept the number of filters same across the network and we kept the dimensions of each input image as defined in the balanced EMNIST database (28x28).

Our convolutional layer integrates the non-linear map and the dimensionality reduction layer. Even though there are multiple choices of non-linear activation functions (e.g relu, tanh), we chose the RElu since it is alleged to work better in a CNNs environment. (Jarrett et al., 2009). Additionally, it ensures valid positive feature maps to all layers. If we define the number of filter as C_{out}, ($C_{in} = C_{out}$) then we can define our "generic convolutional layer" is as follows:

1. Convolutional Layer
2. Non Linear Layer
   - In all cases it a RELU.
3. Dimensionality Reduction Layer
   - Max pooling
   - Average Pooling
   - Dilated Convolution
   - Strided Convolution

Max and average pooling both used kernels of size $[C_{in}, C_{out}, 2, 2]$, stride $S = 1$ and padding $P = 1$. All convolutional layers share the same kernel size $[C_{in}, C_{out}, W_h, W_w]$ and have padding of P=1. Striding layer slides the window with stride S=2. Given the depth l of a layer, dilated convolution applies a dilation $D + 2$, meaning that dilation increases linearly with respect to l. After the convolutional layers we have adjusted an adaptive average pool layer to flatten out the images. The final layer is a fully connected affine layer followed by a softmax responsible for choosing the correct class. Since the outputs are probabilities the most appropriate choice of a loss function is the *cross entropy* function. Since CNNs are prone to overfit we used Adam with weight decay as a regularizing factor. Our choice is being confirmed to have satisfactory results on the generalization of CNNs (Zhou & Feng, 2018). We initialized the Adam with $\beta_1 = 0.9, \beta_2 = 0.99$, learning rate $\alpha = 10^{-3}$ and weight decay coefficient $c = 10^{-5}$.

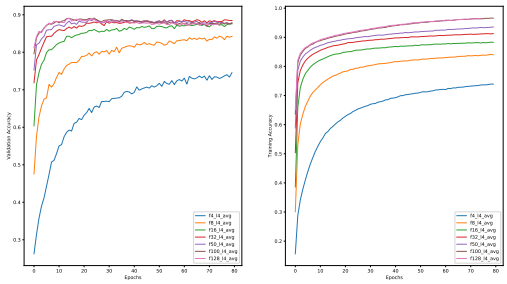

Figure 4. Increasing the number of filters improves the accuracy of the network on the training (right) and validation (left) set. The magnitude of increase decreases more drastically after a certain number for the validation set.

The first hyperparameter we tested to see how it determines the generalization behavior of the network was the number of filters. To do we kept the depth of the network stable at 4 layers and we experimented on filter sizes of $4, 8, 16, 32, 50, 100$ and finally 128.

As mentioned before, more feature maps compensate the loss of resolution and spatial information in deeper layers. Increasing the number of filters makes the network more flexible. That naturally leads to an increased training accuracy. Training set performance kept getting better with more filters. Although a similar behavior would be expected in the validation and test error we found out that as the numbers of the filters grew the effect was less substantial, remarkably faster. This behavior is illustrated in *Figure 4* and *Figure 5*. Even worse, in dilated, strided and max pooling methods the test accuracy decreased when we increased the filters from 100 to 128. This indicates that each method has an upper bound in the number of filters before it starts to overfit. *Table 2* shows this effect. Another very interesting result is the difference in performance of the pooling and the convolutional methods when the number of feature maps is low. This is because both strided and dilated convolutions train the network with increased resolution and spatial information between layers. This means that they don't actually need many feature maps to sustain information. Of course information is quickly regained with more filters resulting in very similar results in all methods, as illustrated in *Figure 6*

To test the methods more thoroughly we tested how the depth affects the accuracy of the algorithms. Due to implementation limitations, our research was only limited in

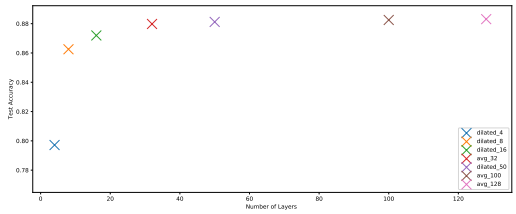

Figure 5. Increasing the number of filters improves the generalization of the network as long as the number of filters if logical

Filters	Average Pool	Dilated	Strided	Max Pool
4	73.6	79.7	79.5	74.1
8	83.5	86.3	85.8	83.5
16	87.1	87.2	87.1	86.9
32	88.0	87.9	86.9	87.7
50	88.2	88.3	87.3	87.6
100	88.4	88.2	88.1	88.05
128	88.4	88.0	87.9	88.04

Table 1. Test accuracy of all methods for different number of filters. As before, too many filters cause the test accuracy to decrease. Average pooling scored the highest score of 88.4%

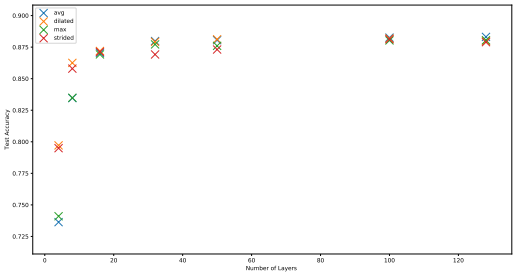

Figure 6. Dilated and strided convolutions are substantially better when few filters are available. When many filters are applied all methods reach similar accuracy

Filters	8 Layers	4 Layers
4	76.8	74.1
8	84.4	83.5
16	86.5	86.9
32	87.3	87.7
50	87.8	87.6
100	87.7	88.05

Table 2. Doubling the number of layers cannot outperform adding more filters

testing the max and average pooling and strided convolution. Dilated convolution could not be combined with 8 layers since the reduction was too aggressive. For architectures greater than 8 layers, all methods were not able to run. Also average pooling and strided convolution were not able to train resulting in test accuracy of 2% (which is 1/47, simply meaning a random guess). The only method with a reasonable performance was average pooling. As before more filters resulted in similar behavior as analyzed before. It is worth mentioning that for few filters the network improved the generalization error but the 4-layers architecture was able to achieve a better test error overall for 100 and 128 layers. This is shown on *Figure 7*. This result raises the question whether it is better to to increase the filters or layers and if there is a balance between those two hyper-parameters.

Even though these experiments indicate that less depth is better we must note that the size of the images of the EMNIST database pose a limitation to the complexity of the network. If we had larger and more complex images the results might not have been the same.

This motivated us to investigate some intermediate-size architecture and changing the size of the kernel. We implemented a 5-layer architecture which managed to actually train with most of the methods (only dilated failed) and achieved better initial results when fewer filters where used. For example with 8 filters max pooling achieved 85.1% accuracy and average pooling achieved 84.3%. Unfortunately strided convolution was worse across all numbers of feature maps. Increasing the size of the kernel and setting the depth of network to 4, was found too aggressive for most methods (strided, dilated and average pooling) and was not able to run on our architectures. Interestingly, we didn't get better overall results for max pooling. The 3-layer architecture also did not give us any overall improvements being too shallow to provide enough representation.

One important aspect of our experiments was memory and time requirements the different models had. Training for example with 256 filters took more than 3 hours, making these experiments prohibitively expensive. Also, dilated convolution was far the slowest of them in all cases, while max pooling was the fastest. Of course, the depth of the network had a similar effect, increasing substantially the training time. Finally, more complex networks (both in terms of depth and width) have more parameters and can become memory-heavy.

The results highlighted that there wasn't a single best architecture which outperformed the rest by much. Even though average pooling managed to score the best results, it did so by 0.01% and with the expense of doubling the number of filters compared to dilated convolution. Also it was shown that for this relatively simple data set increasing the filters seemed like a better choice. Finally, we showed that dilated and strided convolution are able to achieve substantially better results when the resources are scarce.

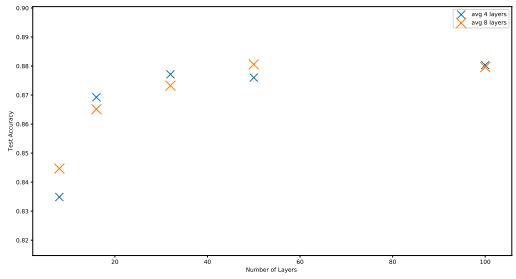

Figure 7. Test accuracy of Max Pooling for 4 and 8 layers. Doubling the layers, didn't manage to highly improve the test accuracy

5. Discussion

The experiments highlighted some key differences of the tested methods, and the required configurations under which they can excel. Overall, the best results were acquired for a 4-layer architecture. The best method was average pooling. This is an interesting finding, firstly indicating that average pooling is better than max pooling when a more generalized computation instead of extreme one is needed. In our case, we are more interested in finding general features of a number or letter (like if a pattern is found in a generic region), rather than a very specific one (like width of a line or the exact position where this line appears). This is why we speculate average pooling performed better. Furthermore, average pooling managed to outperform the dilated convolution by 0.01%, having though to double the number of filters. Our explanation to that is that average pooling loses a lot of resolution whereas dilated convolution doesn't. Thus, if more filters are applied the average pooling makes up for that lost resolution, while the dilated convolution stops gaining any new information.

This discovery also underlines the trade-off between simplicity and accuracy. Even though dilated and strided reduction methods did achieve not best results, they managed to get close-to-best accuracy while being the simplest in terms of network modifications, achieving 88.3% 88.1% respectively over the best 88.4%. Even though we had an extra layer for these two methods, they could be easily integrated to a single convolutional layer meaning no actual modification is needed to an existing architecture. The only change would be to modify how the convolution is performed. Strided convolution can also be considered the simplest method combining fast training time and low memory requirements. That means these methods are more appropriate if resources are limited since they reduce the overall complexity of the network and get satisfactory results.

Increasing the depth didn't have drastic differences in accuracy and even caused our algorithms to diverge. Doubling the number of layers, didn't manage to have analogous increase to the network's performance. Even worse, it was outperformed by the shallower network when more filters where used. Even the addition of one layer (5 generic convolutional layers in total) did not improve the accuracy. Decreasing depth too much, in our case 3 , also lowered the accuracy indicating that the optimum for EMNIST database is 4. That means that in our case width (spatial information, and increased resolution) is more important than depth (higher level representation, more holistic view).

All of our architectures didn't manage to achieve more than 88.4% test accuracy even though the training accuracy kept getting better, which could be a sign of poor feature representation or overfitting. Since no drastic regularization was used, e.g dropout, we think overfitting is most likely explanation. We should note here that the depth of a network is data dependent. Since the EMNIST data are relatively simple, great depth or width is unnecessary. This also applies when increasing the size of kernels as we saw that wider kernels led to more overfitting. That said, if were training on larger-scale data set we'd expect a different behavior, with more complex networks performing better (Chung et al., 2018).

6. Conclusions

In this work we tried to answer two important questions: if a single architecture can be found which can perform better in the classification task. Secondly we tried to see whether a simple architecture exists which achieves close-to-best accuracy. We found average pooling to be the best architecture, achieving 88.4% accuracy. We also found that dilated convolution can achieve close to best accuracy 88.3% with half the filters. Strided convolution also achieved good results 88.1% while being the simplest and the fastest of these 3 methods. Our findings indicate that the scale of the data set is the main factor of the corresponding best architecture's complexity. If simplicity is the most important factor then dilated convolution seems to be the best choice, reducing a network's both width and depth with minimal classification error increase. Furthermore, since dilated convolution has been reported to perform better for small images (Springenberg et al., 2014), running experiments more than once could help get more concrete results canceling out possible computer rounding errors and noise.

Future work includes reporting error bars for each of the methods, to get more statistically stable results. An interesting modification would be placing more filters at layers deeper in the architecture instead of having the same number across the whole network. The motivation behind this idea is that since high resolution is already high at lower levels and lower in deeper approach, resources can be used more effectively. Lastly, testing the accuracy of different architectures on data sets of different scale and size would help to solidify our knowledge on the overall performance of the different dimensionality reduction methods.

References

Antoniou, Antreas and Crowley, Elliot J. Dilated DenseNets for Relational Reasoning.

Belmont, John W. and Gibbs, Richard A. Genome-wide linkage disequilibrium and haplotype maps. *American Journal of PharmacoGenomics*, 4(4):253–262, 2004. ISSN 11752203. doi: 10.2165/00129785-200404040-00005.

Chen, Liang-chieh, Papandreou, George, Member, Senior, Kokkinos, Iasonas, Murphy, Kevin, and Yuille, Alan L. DeepLab : Semantic Image Segmentation with Deep Convolutional Nets , Atrous Convolution , and Fully Connected CRFs. pp. 1–14.

Chung, Clement, Patel, Shital, Lee, Rosetta, Fu, Lily, Reilly, Sean, Ho, Tuyet, Lionetti, Jason, George, Michael D., and Taylor, Pam. Implementation of an integrated computerized prescriber order-entry system for chemotherapy in a multisite safety-net health system. *American Journal of Health-System Pharmacy*, 75(6):398–406, 2018. ISSN 15352900. doi: 10.2146/ajhp170251.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618, 9780262035613.

Jarrett, Kevin, Kavukcuoglu, Koray, Ranzato, Marc'Aurelio, and LeCun, Yann. What is the best multi-stage architecture for object recognition? *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2146–2153, 2009. ISSN 1550-5499. doi: 10.1109/ICCV.2009.5459469.

Jones, Eric, Oliphant, Travis, Peterson, Pearu, and others. {SciPy}: Open source scientific tools for {Python}. URL http://www.scipy.org/.

Miao, Duoqian, Pedrycz, Witold, Ślezak, Dominik, Peters, Georg, Hu, Qinghua, and Wang, Ruizhi. Rough Sets and Knowledge Technology: 9th International Conference, RSKT 2014 Shanghai, China, October 24âĂŞ26, 2014 Proceedings. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8818:364–375, 2014. ISSN 16113349. doi: 10.1007/978-3-319-11740-9.

Oord, Aaron van den, Dieleman, Sander, Zen, Heiga, Simonyan, Karen, Vinyals, Oriol, Graves, Alex, Kalchbrenner, Nal, Senior, Andrew, and Kavukcuoglu, Koray. WaveNet: A Generative Model for Raw Audio. pp. 1–15, 2016. ISSN 0899-7667. doi: 10.1109/ICASSP.2009.4960364. URL http://arxiv.org/abs/1609.03499.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin. Striving for Simplicity: The All Convolutional Net. pp. 1–14, 2014. ISSN 02548704. doi: 10.1163/{_}q3{_}SIM{_}00374. URL http://arxiv.org/abs/1412.6806.

Wei, Yunchao, Xiao, Huaxin, Shi, Honghui, Jie, Zequn, Feng, Jiashi, and Huang, Thomas S. Revisiting Dilated Convolution: A Simple Approach for Weakly- and Semi-Supervised Semantic Segmentation. 1, 2018. ISSN 0940-9602. doi: 10.1016/j.aanat.2005.04.012. URL http://arxiv.org/abs/1805.04574.

Yu, Fisher, Koltun, Vladlen, and Funkhouser, Thomas. Dilated residual networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:636–644, 2017. doi: 10.1109/CVPR.2017.75.

Zhou, Pan and Feng, Jiashi. Understanding Generalization and Optimization Performance of Deep CNNs. 2018. ISSN 1938-7228. URL http://arxiv.org/abs/1805.10767.