MLP Coursework 2: Exploring Convolutional Networks

s1837504

Abstract

As is the case with many neural network hyperparameters, the number of kernels in layers of convolutional neural nets is usually determined ad-hoc. This paper explores the performance gains of a common industry practice, increasing the number of kernels in deeper layers, in the context of EMNIST, with regards to max-pooling, striding, and dilation. Our experiments indicate that all three approaches perform almost equally better, with up to 0.95% increase in test set accuracy.

1. Introduction

It is common practice in convolutional neural networks to increase the number of kernels in deeper layers of the model, with some of the influential networks which used that architecture being *LeNet-5* (Lecun et al., 1998), *AlexNet* (Krizhevsky et al., 2012), and *VGGNet* (Simonyan & Zisserman, 2014). However, although there has been some theoretical work trying to define the appropriate amount of kernels in each consecutive layer (Chu & Krzyżak, 2014), it is usually determined through trial and error (Simard et al., 2003). This paper explores the effects of doubling the number of kernels after each layer, both in test accuracy and training speed, for 3 different dimensionality reduction approaches, namely max-pooling, striding, and dilation, using the balanced EMNIST dataset (Cohen et al., 2017). EMNIST is an extension of the MNSIT handwritten digits, including 131,600 greyscale images of size 28×28, which represent digits and letters. The images are split into 47 classes and we use 100,000 of them as our training set and the rest are equally divided to form our validation and test sets (s1837504, 2018). In section 2, we introduce convolutional networks and max-pooling layers and discuss a possible implementation approach. In section 3, we explain further the three dimensionality reduction approaches we applied, and in section 4, we describe our experiment's methodology and results. Finally, section 5 includes a discussion of the results which indicate that all three approaches benefit from the increased number of kernels with striding convolutions displaying the least amount of test accuracy gains (0.54%). In addition, we identify that using dilations is the most computationally heavy of the three and adding more kernels greatly increases its training time.

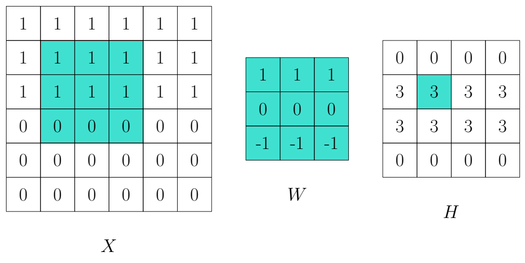


Figure 1. Horizontal edge detection kernel

2. Implementing convolutional networks

2.1. Convolutional networks and pooling layers

Suppose we want to build a fully connected neural network in order to classify greyscale images of dimensions 100×100. One neuron in our first hidden layer will thus have 100*100=10,000 weights. That number is then multiplied by the number of neurons in the layer to calculate the total number of weights in the first hidden layer. Considering that we will have more than one hidden layer, and that we may have a coloured image ($3 \times$ dimensions), the number of weights can quickly get out of hand slowing our network's training speed. Convolutional networks solve this problem by taking advantage of the spatial structure of images. Each convolutional layer has a number of kernels W, which detect the same features in every part of the image X, and produce a feature maps H. Figure 1 demonstrates a kernel which detects horizontal edges in a greyscale image. The number of parameters is reduced since the weights are shared between all the points, and the output from a 3×3 kernel is given by $H_{k,l} = \sum_{i=k}^{k+2} \sum_{j=l}^{k+2} W_{i,j} X_{i,j} + b$, where b is the bias.

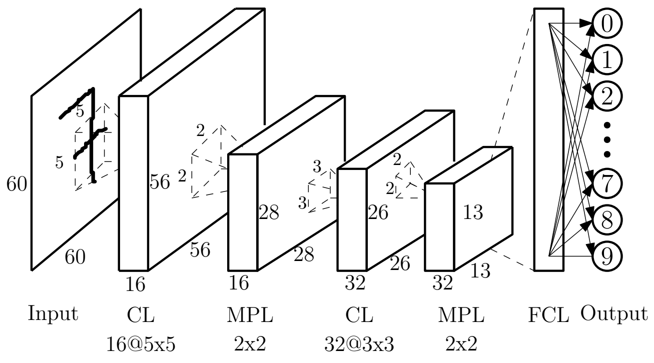


Figure 2. Convolutional neural network

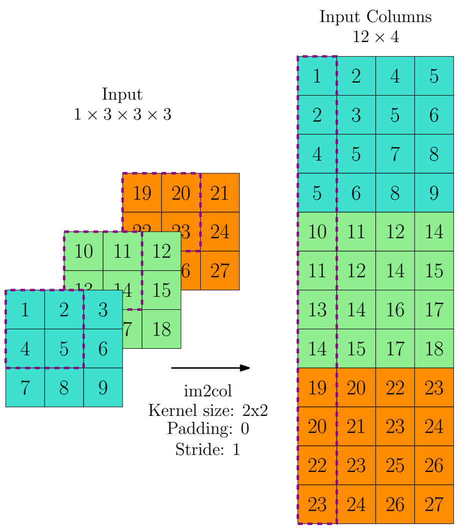


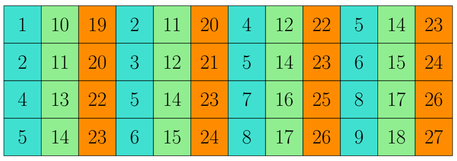
Figure 3. Output of im2col for 1 element with 3 channels

In order to further reduce the parameters, apart from the convolutional layers, we often implement intermediate pooling layers which down-sample each feature map H. We can think of a 2×2 max-pooling layer as a window which goes over H, usually with no overlap, and picks the highest value of the 4 observable. For example, a 2×2 max-pooling layer applied to H from figure 1 would output a 2×2 matrix with all its values equal to 3. A convolutional network with 2 convolutional layers followed by max-pooling, a fully connected layer, and the output, is displayed in figure 2, the softmax layer before the output is omitted.

2.2. Implementation

The most obvious approach to implement a convolutional or pooling layer is to loop over all possible windows, for each kernel, for all elements in the mini-batch, and perform the appropriate operation. However, looping is inefficient and we can eliminate the two loops required to traverse the two dimensions of the windows by explicitly performing the convolution operation using `scipy.signal.convolve2d` from the SciPy library (Jones et al., 2001). In order to eliminate the remaining loops for the kernels and for the mini-batch elements we can perform the entire operation in one matrix multiplication. To achieve that we first have to reshape the 4D input into 2D using a function called `im2col` (Hu, 2018). As it is shown in figure 3, the drawback of this approach is that elements in the input columns matrix are repeated so our memory footprint increases (Lai et al., 2018).

To complete the forward propagation for the convolutional layer we have to apply the same function to our kernels, transpose them, and multiply the two 2D matrices. Finally, the result has to be reshaped to the correct dimensions of (batch size × number of kernels × input height - kernel height + 1 × input width - kernel width + 1). For the backward propagation we perform the reverse transformations



1	10	19	2	11	20	4	12	22	5	14	23
2	11	20	3	12	21	5	14	23	6	15	24
4	13	22	5	14	23	7	16	25	8	17	26
5	14	23	6	15	24	8	17	26	9	18	27

Figure 4. Reshape of input columns for forward propagation in max-pooling layer

and to obtain the array of gradients with respect to the inputs we apply the complementary function of `im2col`, named `col2im`, which transforms from 2D to 4D.

For the max-pooling layer, after using `im2col` in our inputs as in figure 3, we further reshape in order for each column to contain the elements of each possible window as shown in figure 4. We are now able to get the maximum value from each column and reshape to the correct dimensions of (batch size × number of kernels × (input height - size)/stride + 1 × (input width - size)/stride + 1). During this procedure we save the indexes of the maximum values since they are to be used in the backward propagation, in order to place the gradients with respect to the outputs in the cells that had the maximum value.

3. Context in convolutional networks

In this section we present three possible ways to manipulate the context over which our kernels operate. Our goal is to identify how increasing the number of kernels in deeper layers affects each of the following approaches.

3.1. Pooling layers

Pooling layers were introduced in subsection 2.1 as a method to reduce the number of parameters in convolutional networks. Apart from the speed gains, reducing the amount of parameters also decreases our chances of overfitting. In addition, since the exact location of the features is lost in the pooling layer, minor translations of the image will have a lesser effect in our accuracy (Goodfellow et al., 2016).

Pooling is performed in each feature map separately and the pooling layer has only two hyper-parameters and no learnable weights. The first hyper-parameter controls the size of the window and the second, called stride, controls how far along each axis the window moves. In practice there are two commonly used combinations, either using a window size of 2 with equal stride, or window size of 3 with stride 2 (Karpathy). Increasing the window size even more, although it reduces the computational time, removes too much information.

Apart from max-pooling which was explained in 2.1, there are other techniques in order to extract data from the selected cells. *L2* pooling calculates the L^2-norm of each window by taking the square root of the sum of squares of the elements, and simply taking the average is another

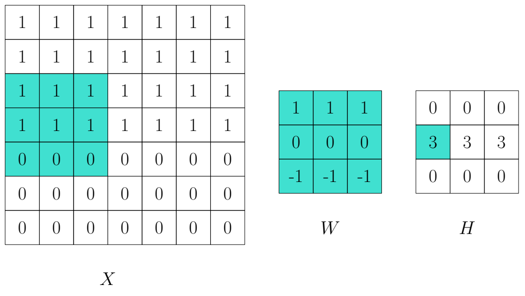


Figure 5. Convolutional layer with stride set to 2

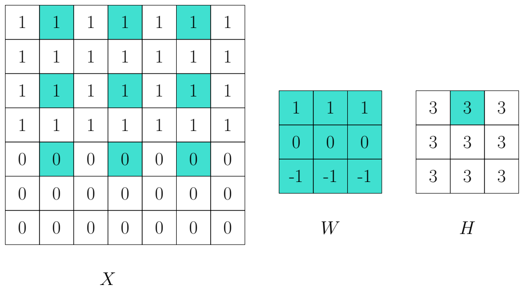


Figure 6. Convolutional layer with dilation set to 2

approach. However, max-pooling is prefered since it has been shown to work better in practice (Karpathy).

3.2. Striding

Striding can also be used in order to down-sample our image by moving the kernel filter more than one position along the axis. As is the case with pooling, striding also reduces the number of parameters producing similar effects. In fact, Springenberg et al. (2014) managed to eliminate the pooling layers by increasing the stride with no reduction on accuracy. However, when increasing the stride we have to ensure that the dimensions align correctly so that the kernel does not extend outside our input. For example, in figure 1 we are unable to implement stride equal to 2 since the input dimensions should have been either 5×5 or 7×7, as it is shown in figure 5. In general, the remainder of $\frac{X-W}{S}$, where X is the input size, W the kernel size, and S the stride size, must be zero. If we choose to include zero padding, i.e. enclosing our input with zeros in order to keep the spatial dimensions of the output equal to the input's, then the term $2P$ must be added to the numerator, P being the amount of padding.

3.3. Dilation

The most recent technique of the three is dilation which increases the context of the kernel by inserting gaps between its cells. A dilation value of 1 results in a continuous kernel, a value of 2 in a kernel where each cell has distance 2, etc. Yu & Koltun (2015) exponentially increased the receptive field of the kernels by increasing the dilation in each consecutive layer while the number of parameters increased linearly. As a result, the deeper layers of the network are able to gather knowledge in a larger context. Figure 6 displays a convolutional layer with dilation set to two and unit stride. As we observe, the feature map indicates that a horizontal edge exists in our image but with no knowledge regarding its location. That knowledge was present using strides (figure 5) but would also not be present if we applied a 2×2 max-pooling layer to the feature map of figure 1.

4. Experiments

In order to identify the effect of doubling the number of kernels in each layer, we first have to establish our baseline using a constant number of kernels. For all our experiments we utilise the Pytorch experiment framework, provided in coursework 2 (Renals, 2018), which acts as a wrapper for the Pytorch library (Paszke et al., 2017). The framework creates n convolutional layers, followed by a ReLU activation function and, if specified, one of the following dimensionality reduction layers: strided convolution, dilated convolution, max-pooling, or average pooling. Each dimensionality reduction layer is also followed by a ReLU layer and, before the final fully connected layer, we have an adaptive average pooling layer which, as stated in the comments of the code, ensures that the output of the convolutional layers is 2×2 in order to allow for better comparisons. We choose not to experiment with average pooling since it is similar enough to max-pooling to justify not wasting computational resources on it. Although, in general, dilated layers do not reduce dimensionality (Antoniou et al., 2018), we mentioned them as such because this is the way they are treated in the framework. Each run was repeated three times with different seeds of 0, 1, and 2.

The only variable which changes between the baseline and the rest of the experiments is the number of kernels so the hyper-parameters stated in the following paragraphs were used in all instances. The batch size was left to the default size of 100, since the coursework specification (2018) states that the default parameters are reasonable. The input dimensions are the given dimensions for EMNIST, i.e. 28×28 images with 1 channel (greyscale), and we trained for 100 epochs. We are not concerned about overfitting since the provided framework automatically tests with the best validation model. The number of convolutional layers, not counting dimensionality reduction layers, was chosen to be 4 since we will be doubling the amount of kernels in each layer, therefore increasing our computational costs, and we want the training procedure to complete in a reasonable amount of time. In addition, 4 layers are enough since Nielsen (2015) used only 2 convolutional layers in his network to classify MNIST images. Finally, the weight decay for Adam was left at the default value of 10^{-5}.

Regarding the hyper-parameters of the convolutional layers,

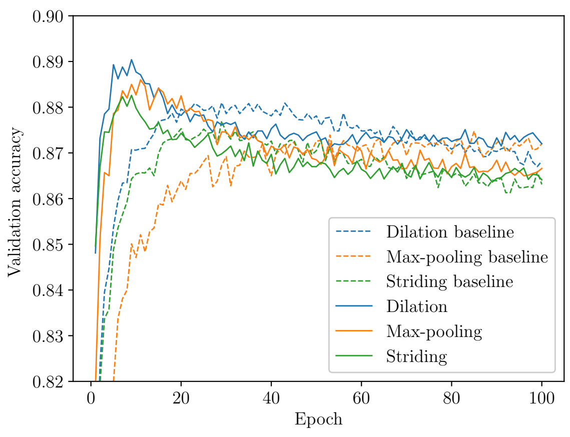


Figure 7. Average validation accuracy with respect to epochs for the baselines and the increasing number of kernels models

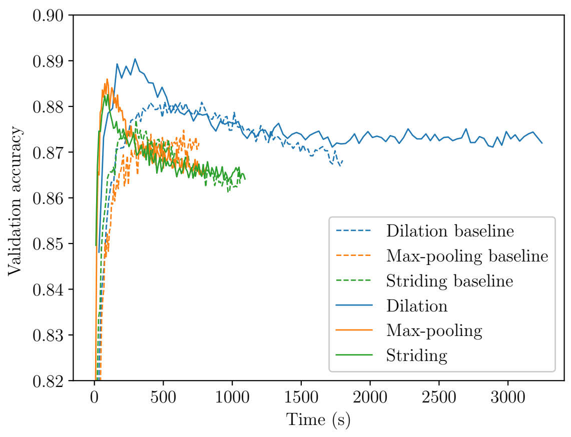


Figure 8. Average validation accuracy with respect to training time for the baselines and the increasing number of kernels models

we used a small kernel size of 3, as suggested by Karpathy, and we also used zero padding of one, as mentioned by the same author, in order for the convolutional layer to preserve the input size. The stride and the padding were chosen to be one, leaving the down-sampling for the intermediate dimensionality redaction layers. The striding for the strided convolution was left to the default value of 2 and the dilation for the dilated convolution layer starts from 2 and increases by 1 after each layer. According to Antoniou (2018), the large dilation rate was used to reduce the dimensions more rapidly. The max-pooling size and stride were both left to the default value of 2 as it is one of the two recommendations from Karpathy we mentioned in subsection 3.1.

The initial number of kernels is chosen to be 16 because after doubling for each layer, with four layers, a larger number would greatly impede our training speed. The number is low considering that *AlexNet* starts with 96 kernels in the first layer, increasing up to 384 (Krizhevsky et al., 2012), and *VGGNet* starts with 64 and goes up to 512 (Simonyan & Zisserman, 2014). We also tried to start with 64 kernels but the training time was too much. In addition, an older network, *LeNet-5*, used only two convolutional layers, with 6 and 16 kernels each (Lecun et al., 1998), so, since they increased only by 10, there must have been in improvement even for such a small increase in the number of kernels. Therefore, the effect of doubling the kernels ought to be visible even if we start from 16.

To create our baseline we build three models with 16 kernels in every layer, each model with one of the following dimensionality reduction techniques: max-pooling, striding, and dilation, and repeat the training procedure three times with the seeds we mentioned before. To perform the experiment with doubling the number of kernels, we change only line 119 in `model_architectures.py` from `out_channels = self.num_filters` to `out_channels = self.num_filters*(2**i)`. As a result, the first layer will have 16 kernels, the second 32, the third 64, and the fourth 128. Additional code is also added in `experiment_builder.py` in order to record the training time per epoch.

After running the experiments, the results from the three different seeds for each model are averaged by a separate script. We also calculate the standard error of the mean, which is given by $\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$, where σ is the standard deviation of the results, and n is the population size, in our case 3, since we have three different seeds. In the results in table 1 we report the average of the 3 runs and the error bars are the standard error of the mean. Figures 7 and 8 display the validation accuracy for our 3 models, and the 3 corresponding baselines, with respect to the number of epochs and time.

5. Discussion

Regarding the validation accuracy with respect to the number of epochs in figure 7, we observe that all 3 dimensionality reduction approaches benefit almost equally from continuously doubling the number of kernels. Each of the baseline models achieves about 1% less accuracy than their "deeper" counterparts. While the baseline models require ~20 epochs to reach their maximum accuracy, the enhanced models require ~10 and after that point they start overfitting. In figure 8 it is evident that dilation takes the heaviest toll with regards to training time since the time required to complete 100 epochs almost doubles. Although the number of parameters in all 3 approaches is equal, we can conclude that performing convolution with dilated filters is less efficient. However, this should not be a discouraging factor when considering using the doubling kernels model since the problem can be eliminated by using early stopping. The other two approaches present almost zero changes in their training times, with max-pooling performing the fastest.

The test results in table 1 present a similar story. All 3 dimensionality reduction approaches benefit from increasing the number of kernels, with striding displaying the least

Model	Test Accuracy (%)	Test Error (10^{-2})	Training time (s)
Max-pooling baseline	86.90 ± 0.17	39.82 ± 0.34	758 ± 12
Max-pooling	87.73 ± 0.12	35.95 ± 0.42	809 ± 18
Striding baseline	86.92 ± 0.02	39.60 ± 0.47	1056 ± 49
Striding	87.39 ± 0.11	39.75 ± 0.86	1092 ± 60
Dilation baseline	87.31 ± 0.11	37.66 ± 0.38	1800 ± 16
Dilation	88.06 ± 0.05	34.22 ± 0.23	3246 ± 06

Table 1. Average test results ± standard error of the mean

improvement in terms of test accuracy (0.54%), and max-pooling the highest (0.95%). The best test accuracy of 88.06% is achieved using dilation which also presents high precision (SEM of 0.05%).

The performance increase comes from the fact that adding kernels in deeper layers of the model enables the model to learn a higher volume of global, abstract, structures. On the other hand, in the first layers, the model detects local structures, for example edges, so increasing the filters in those layers does not yield the same benefits, e.g. there is little gain from detecting more than a dozen of edges (horizontal, vertical, diagonal, etc.). Simard et al. (2003), in their convolutional network which classified MNIST digits, found that using more than five kernels in the first layer did not improve performance while the upper bound for the second layer was 100.

6. Conclusions

Our experiments validate that the common architecture of adding more kernels in deeper layers increases performance both in terms of test set accuracy and training speed. We also found that using striding as a dimensionality reduction approach yields the lowest gains of 0.5% test accuracy. On the contrary, max-pooling and dilation presented slightly higher gains of ~0.8%. However, the time required for dilation almost doubled while for the other two, the increase was less visible. Therefore, we suggest that early stopping is implemented, especially if using dilation, but also because all approaches reach their maximum validation accuracy in about 10 epochs, which is almost half of what the baseline models need, and there is no need to waste computational resources. Future research may also experiment with setting the number of kernels for the constant baseline equal to our upper bound of 128 in order to compare accuracy and speed on the upper end of the spectrum. Another aspect which could be explored is to experiment with different increase ratios apart from the implemented doubling approach.

References

Machine Learning Practical 2018/19: Coursework 2, 2018. URL http://www.inf.ed.ac.uk/teaching/courses/mlp/2018-19/mlp_cw2_2018-19.pdf.

Antoniou, Antreas. Piazza: INFR 11132, November 2018. URL https://piazza.com/class/jls4v23lvrw70i?cid=520.

Antoniou, Antreas, Słowik, Agnieszka, Crowley, Elliot J., and Storkey, Amos. Dilated DenseNets for Relational Reasoning. *arXiv:1811.00410 [cs, stat]*, November 2018. URL http://arxiv.org/abs/1811.00410. arXiv: 1811.00410.

Chu, Joseph Lin and Krzyżak, Adam. Analysis of Feature Maps Selection in Supervised Learning Using Convolutional Neural Networks. In Sokolova, Marina and van Beek, Peter (eds.), *Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pp. 59–70. Springer International Publishing, 2014. ISBN 978-3-319-06483-3.

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

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016.

Hu, Jesse. huyouare/CS231n/im2col.py Github repo, November 2018. URL https://github.com/huyouare/CS231n. original-date: 2015-02-17T01:28:43Z.

Jones, Eric, Oliphant, Travis, Peterson, Pearu, and others. *SciPy: Open source scientific tools for Python*. 2001. URL http://www.scipy.org/.

Karpathy, Andrej. CS231n Convolutional Neural Networks for Visual Recognition. URL http://cs231n.github.io/neural-networks-3/.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pp. 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

Lai, Liangzhen, Suda, Naveen, and Chandra, Vikas. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. *arXiv:1801.06601 [cs]*, January 2018. URL http://arxiv.org/abs/1801.06601. arXiv: 1801.06601.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Nielsen, Michael A. Neural Networks and Deep Learning. 2015. URL http://neuralnetworksanddeeplearning.com.

Paszke, Adam, Gross, Sam, Chintala, Soumith, Chanan, Gregory, Yang, Edward, DeVito, Zachary, Lin, Zeming, Desmaison, Alban, Antiga, Luca, and Lerer, Adam. Automatic differentiation in PyTorch. October 2017. URL https://openreview.net/forum?id=BJJsrmfCZ.

Renals, Steve. MLP 2018-19, 2018. URL http://www.inf.ed.ac.uk/teaching/courses/mlp/index-2018.html.

s1837504. *MLP Coursework 1*. University of Edinburgh, 2018.

Simard, Patrice Y., Steinkraus, Dave, and Platt, John C. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2*, ICDAR '03, pp. 958–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 978-0-7695-1960-9. URL http://dl.acm.org/citation.cfm?id=938980.939477.

Simonyan, Karen and Zisserman, Andrew. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, September 2014. URL http://arxiv.org/abs/1409.1556. arXiv: 1409.1556.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin. Striving for Simplicity: The All Convolutional Net. *arXiv:1412.6806 [cs]*, December 2014. URL http://arxiv.org/abs/1412.6806. arXiv: 1412.6806.

Yu, Fisher and Koltun, Vladlen. Multi-Scale Context Aggregation by Dilated Convolutions. *arXiv:1511.07122 [cs]*, November 2015. URL http://arxiv.org/abs/1511.07122. arXiv: 1511.07122.