MLP Coursework 2: Exploring Convolutional Networks

s1866666
November 2018

Abstract

An important aspect of convolutional network is its ability to relate information in the spatial structure of the data, and how the network processes spatial information is determined by its context. Using successive conventional convolutional layers in a network is a way of modelling context, however, is not efficient as the size of such network tends to be huge. In this report we will look into four approaches of broadening the context of convolutional networks while keeping the size of the network small, which are max pooling, average pooling, strided convolution and dilated convolution. We will explore the advantages and disadvantages of them in terms of efficiency and classification accuracy on reduced EMNIST dataset, as well as the effects of adding filters to a network. The experiments are carried out and show that average pooling leads to more accurate classification than max pooling, and all-convolutional networks may not perform better than pooling networks even with more parameters. The experiments also shows that the improvement of adding more filters diminishes exponentially as the number of filters gets large.

1. Introduction

Convolutional neural networks (ConvNet) (Lecun et al., 1998) are designed to process spatial or temporal information of the inputs. It learns high-level features like objects in image, and sentences in speech audio by incorporating the lower-level features built layer-by-layer from the bottom of the networks where it recognizes local features like edges in image and phoneme in speech audio.

The essential building block of the ConvNets is convolutional (Conv) layers. It works so that each neuron is only fed with a limited number of inputs, often based on spatial relations of these inputs. The parameters are contained in structures called filters, which are convoluted with the inputs during the feed forward to produce feature maps. The spatial features are learned due to weights sharing, that is, all the inputs shares the weights on a filter and output a single feature map. And because of this, the number of parameters in ConvNet can be dramatically reduced compared with a fully-connected network. For example, suppose there are $64 \times 26 \times 26 = 43,264$ neurons in a hidden layer and $64 \times 24 \times 24 = 36,864$ neurons in the next layer. Fully-connecting the two layers requires $43,264 \times 36,864 \approx 1.5$ billion parameters, whereas connecting them using 3×3 filters only requires $64 \times 3 \times 3 = 576$ parameters.

The keys to building a ConvNet which is capable of fitting the data considerably well while using small amount of parameters involves context modelling. The context of a filter describes the scale and breadth of the information this filter is able to capture. The goal of this report is to investigate some approaches of context modelling in a ConvNet, which are max pooling, average pooling, strided convolution and dilated convolution. In particular We will explore the difference between max pooling and average pooling layers, as well as the advantages and disadvantages of strided and dilated convolutional layers compared with pooling layers. In addition, we will also look into the effects of changing the number of filters on the learning curves of the training process.

Section 2 introduces the implementation of Conv layer and max pooling layer in detail. Section 3 briefly interprets the idea of context modelling and presents the concepts of average pooling, strided convolution and dilated convolution. Section 4 covers the experiments on the these dimension reduction layers, as well as on changing the number of filters. Section 5 presents the analysis on the outcomes of the experiments. Section 6 summarizes the report and provides further questions related to the work.

All the experiments are carried out on EMNIST dataset (Cohen et al., 2017), a well-labeled balanced dataset including images of handwritten letters and digits. A reduced dataset including 47 classes, instead of 62 classes (26 upper case letters, 26 lower case letters and 10 digits) is used in our experiments. This is because there are 15 letters which have small distinctions between their upper case and lower case, and thus their upper- and lower-case labels are merged. These letters are: C,I,J,K,L,M,O,P,S,U,V,W,X,Y,Z. The training set contains 100,000 samples. The validation set and test set each contains 15,800 samples. Every sample consists of 784 float numbers representing the pixel values of a 28 by 28 image.

2. Implementing convolutional networks

2.1. convolutional layer

In practice, the input to a Conv layer is usually represented by a 4-dimensional tensor whose dimensionality is $B \times I \times H_1 \times W_1$, where B is batch size, I is number of input channels, H_1 is input height, and W_1 is input width. The filters are represented by a 4-dimensional tensor as well, whose dimensionality is $O \times I \times K \times K$, where K is kernel height and width. A Conv layer takes a input tensor and a kernel tensor and outputs a 4-d tensor whose dimensionality is $B \times O \times H_2 \times W_2$, where H_2 and W_2 are output height and width.

The key to implementing a Conv layer is how the convolutions are implemented. Two approaches to it will be discussed in this report: (1) explicitly using the SciPy convolution function `scipy.signal.convolve2d` and (2) Serialization using `im2col` and its reverse `col2im`.

Convolve2d. This implementation requires loop through the 4-d tensors and apply `convolve2d` to the corresponding inputs and kernels in forward propagation or gradients in back propagation. The loops can produce large amount of overhead, and thus slow down the algorithm significantly when processing huge tensors. However, the algorithm does not require any extra memory during the forward or backward propagation.

Serialization. Using `col2im`, a 4-d input tensor can be converted into a matrix, where each column represents the receptive field of a certain neuron. After converting the input and flattening the kernel into matrices, the convolution operation is just a single matrix multiplication of these two matrices and thus much more computationally efficient than many smaller multiplications in the case of `convolve2d` implementation. The product of the matrix multiplication is then converted back to the 4-d tensor using `col2im`. Same philosophy applies to back propagation. The drawback of this implementation is that it requires extra memory which is potentially extensive, because the matrix contains a huge amount of repeated values, as most of the values in the original input tensor can be included in the receptive field of multiple neurons and thus has to appear in multiple columns.

One can choose either method depending on the situation. Assuming stride 1, and padding 0, the two methods both run at $O(BIOH_2W_2K^2)$ time complexity, however, serialization could save the overhead of looping and hence significantly outperform the other in terms of speed. In the contrary, serialization requires $(IK^2) * (BH_2W_2)$ more space to store the converted input matrix, and thus worse than the other in terms of memory usage.

I personally implemented the first approach in this assignment. The average running time over 7 trials of forward propagating an input with size of $100 \times 64 \times 26 \times 26$ is 14.6 ± 1.29 s (mean ± 1 std. dev.)

2.2. Max pooling layer

Pooling layers are commonly inserted in between successive Conv layers in a network to broaden the context of filters. It also reduces the dimensionality of the features, thus effectively moderating overfitting. In particular, a max pooling (MAX) layer takes the max value over an area of inputs on a single channel, and output it to the next layer.

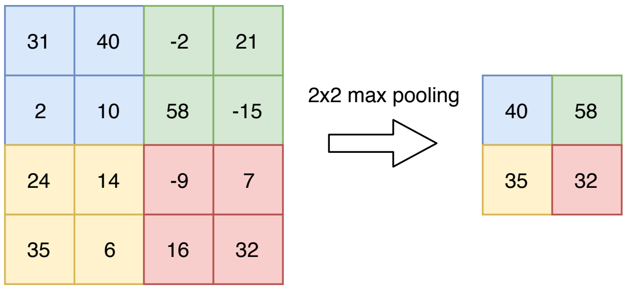

Figure 1. A 2x2 max pooling layer with stride of 2 applies on the 4x4 input matrix on the left and outputs the matrix on the right

For example, a 2×2 max pooling layer with a stride of 2 as illustrated in 1 outputs the max of every 4 input values in shape of 2×2. This can be realized using the same methods for convolution discussed in section 2.1. In this assignment I implemented it using a series of loops to perform the max operation on each pooling region, and then put the max values together into the output tensor. A mask for every pooling region indicating the position of the max value can be cached in forward propagation and used in back propagation, as the gradient of outputs w.r.t inputs is 1 for those max values taken to the next layer and 0 for others. The average running time over 7 trials of forward propagating an input with size of $100 \times 64 \times 26 \times 26$ is 23.2 ± 1.279 s (mean ± 1 std. dev.)

3. Context in convolutional networks

A powerful aspect of ConvNets is its ability to process spatial or temporal relations in the information. The information at different scale can be captured by the filters with different level of context. Specifically, the layers close to outputs have broader context than the layers close to inputs, therefore the layers close to inputs work on local features, and the layers close to outputs work on higher-level features. Successively implementing multiple Conv layers broadens the context of kernels layer-by-layer gradually, however, this is neither efficient as it reduces the dimensionality of feature maps too slow, nor effective as it requires too many parameters that it runs a huge risk of overfitting. Max pooling is one way of solving this problem as it systematically diminishes the size of feature maps and thus broadening the context of the following layers efficiently. There are other ways of broadening the context in ConvNets while keeping the number of parameters in check:

Average pooling (AVG). Average pooling is similar to max pooling. It takes the average of the inputs in the forward propagation instead of max, and thus discarding less information than max pooling does. In back propagation, the gradient w.r.t inputs which are in the same pooling region are always the same. e.g. If the filter size is 3×3, then the gridient w.r.t output of each input is $1/9$.

Strided convolution (STR). Stride is the scanning step size of the filters. Instead of using stride 1 in default

Conv layers, larger stride reduces the size of feature maps by a larger amount and increases the context of filters. The height and width of a feature map are computed by $H_2 = (H_1 - K + 2P)/S + 1$ and $W_2 = (W_1 - K + 2P)/S + 1$, where S is stride and P is padding. Some has argued that max pooling can simply be replaced by strided convolution without loss in accuracy on several image recognition benchmarks (Springenberg et al.).

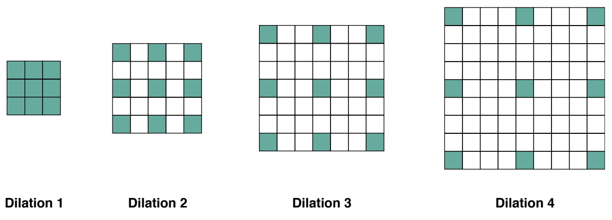

Figure 2. Kernels for dilated convolutions with successively increasing receptive field sizes. Figure copied from (Antoniou et al., 2018)

Dilated convolution (DIL). Illustrated in figure 2, dilation involves increasing the distance between the filtered pixels on the kernels (Yu & Koltun, 2016). It exponentially expands the context of kernels layer-by-layer, while keeping the number of parameters on a kernel unchanged. The height and width of a feature map are computed by $H_2 = (H_1 - K - (K - 1)(D - 1) + 2P)/S + 1$ and $W_2 = (W_1 - K - (K - 1)(D - 1) + 2P)/S + 1$, where D is the dilation rate. The application of dilated convolutions includes relational reasoning in image processing (Antoniou et al., 2018), as well as audio/speech processing (van den Oord et al., 2016).

These types of layers are often combined in an architecture depending on the task, nevertheless, it is important to understand the capability of the layers respectively. Therefore we want to compare the performance of the models implementing different types of these layers on an EMNIST classification task in terms of accuracy and training speed. In particular, we want to figure out whether average pooling leads to higher classification accuracy than max pooling, and which of the two needs less epochs to reach its best validation accuracy.

The STR layer and DIL layer are in essence generalizations of convolutional layers, and thus the networks implementing them without pooling are all-convolutional networks (Springenberg et al.). It is natural to expect that all-convolutional networks would outperform the pooling networks as there are more parameters involved. However this may not hold when it comes to a task as complicated as image classification. Hence we will compare the performance of all-convolutional networks and pooling networks in terms of classification accuracy and training speed, so as to augment the argument in (Springenberg et al.).

Another aspect of context in a ConvNet is its quantity, which can be determined by the number of filters. In general, the more layers, the more information likely to be learned. It is hence worth investigating the amount of contribution of increasing the number of filters to the classification accuracy and what effects the number of filters has on the learning curves of a network.

4. Experiments

4.1. Dimensionality reduction layers

To fully explore the effect of dimensionality reduction layers, we adopt an architecture in which multiple layers of the same type are implemented. Specifically, we set up four networks, each of which uses one type of dimensionality reduction layer. Every network consists of 4 successive Conv layers with kernel size 3×3, stride 1 and padding 1, each of which is followed by a relu unit (Glorot et al., 2011) and a dimensionality reduction layer. In addition, every STR and DIL layer is followed by a relu unit. An adaptive average pooling layer is then added to ensure the output to the next layer always is the size of 2×2. Finally, we feed the output to a fully-connected logit linear layer to obtain the final likelihood of each class. The structure of the networks is illustrated in figure 3.

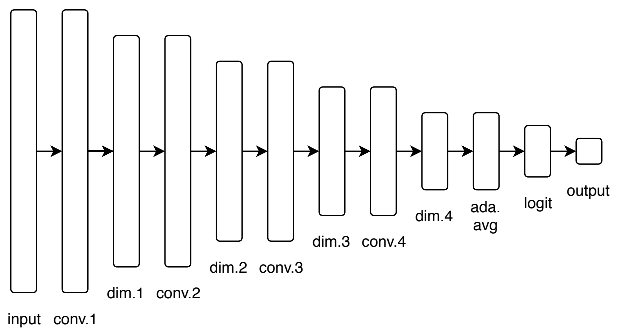

Figure 3. The general architecture of the ConvNets used in the experiments. The rectangles represent layers, and arrows representing forward data flow. The text below each layer indicates the layer type, where 'conv.' denotes convolutional layers, 'dim.' denotes dimensionality reduction layers, and 'ada. avg' denotes adaptive average pooling layers. Relu units are not shown in the figure as their positions may vary in particular networks

To compare the outcomes of these networks, we want to make sure the variables in different networks are identical. Hence we use a pooling size of 3×3 and padding of 1 for both MAX and AVG layer. As with STR and DIL layers, there are more hyperparameters involved, which make the direct comparison between them difficult. Ideally, a grid search could be carried out to optimize the hyperparameters. For this experiment, however, we can just use reasonable settings for STR and DIL layers, as our goal is to get a general idea of how STR and DIL layers perform compared with the pooling layers. We use stride of 2, kernel size of 3×3 and padding of 0 for all STR layers. For DIL layers, the dilation rate is set so that it gets incremented by 1 per layer, starting from 2 for the first layer. All the networks are set to have 64 filters per layer.

Shown in the table 1 is the size of neurons in each dimensionality reduction layer and total number of parameters for each network. We are using EMNIST data set, so the image size of inputs are 28×28 For each type of network,

NETWORK	D1	D3	D3	D4	# PARAMETERS
MAX NET	15	8	5	3	123,200
AVG NET	15	8	5	3	123,200
STR NET	14	7	4	2	270,656
DIL NET	26	22	16	8	270,656

Table 1. The size of each dimensionality reduction layer and total number of parameters for each network. For the layer sizes, only the last dimension, i.e. W_2, is shown for easy comparison. The column label 'D1' denotes the first dimensionality reduction layer.

we run 100 epochs to train the parameters using ADAM (Kingma & Ba, 2015) with weight decay coefficient 1e-05. The batch size is 100, and the number of input channel is 1. We also use a random number for the seed of the random number generator, so that the results generalize well.

Shown in the figure 4 is the learning curves of these four networks in this run. we can clearly see that DIL network gives the highest validation accuracy. In terms of efficiency, AVG network is the slowest, as it is the last to reach its peak validation accuracy. However, AVG network is also more resistant to overfitting, since its validation error are much lower than the others' as the training continues on.

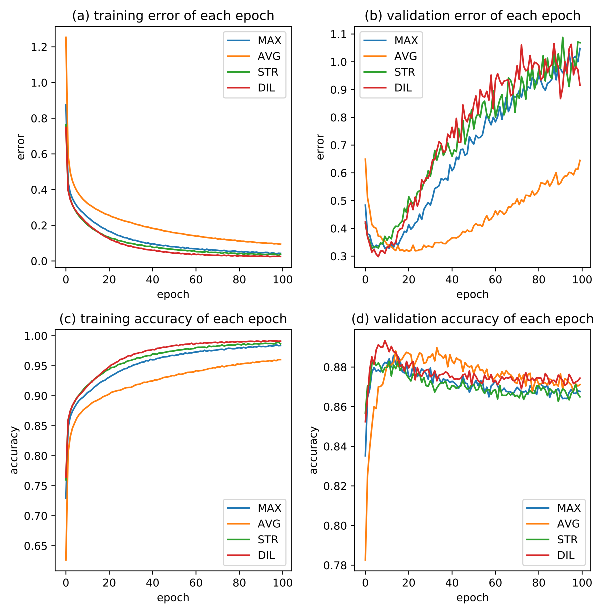

Figure 4. The learning curves in 100 epochs of the four networks using different dimensionality reduction layers, where each plot shows one type of learning curve for simple comparison: (a) training error, (b) validation error, (c) training accuracy, and (d) validation accuracy. The results are taken from a single run for each network. The networks are initialized with different seed.

To compare their performance qualitatively. we repeat each experiment 10 more times, and take the average of their highest validation accuracy as well as its epoch number (1-based). The epoch budget is adjusted to 50 for each run to save computational resources, since according to the learning curves 50 epochs should be enough for all the networks to reach their best. The best models are then tested using the test set to obtain the average test accuracy for each network. In addition, we compute their average runtime on a NVIDA K80 GPU per epoch. Table 2 shows the results of the experiment, and figure 5 presents a more intuitive comparison of their performance.

LAYER	EPOCH	TEST ACCURACY	RUNTIME
MAX	9.73 ± 0.55	87.87% ± 0.050%	17.08s
AVG	25.82 ± 1.23	88.05% ± 0.053%	17.42s
STR	8.09 ± 0.92	87.51% ± 0.060%	20.51s
DIL	9.91 ± 0.51	88.24% ± 0.061%	60.92s

Table 2. The epoch number of the best model and its test accuracy for the networks using different dimensionality reduction layers, as well as their runtime per epoch on a K80 GPU. The results shown in every row are averaged over 11 trial. The epoch and accuracy are reported in the format of 'mean ± 1 standard deviation'.

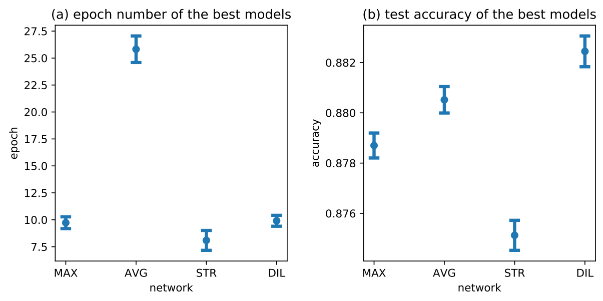

Figure 5. The results in table 2 plotted in error bars. The errors shown are 1 standard deviation.

4.2. Number of filters

We now turn to look at the effect of changing the number of filters in the MAX networks. We use the same architecture of the MAX network as is in section 4.1 and run 100 epochs for each network with number of filters 16, 32, 64, 128, and 256 respectively. The numbers are picked on the logarithmic scale so as to cover a wide range of values, and thus to make the results more comprehensive and general. Still, we use ADAM learning rule with weight decay coefficient 1e-05 to optimize the parameters. The batch size is 100, and the number of input channel is 1. A random number is used to seed the random number generator.

Shown in the figure 6 is the learning curves of these four networks in this run. we can see that in general the more filters in the network, the higher the best validation accuracy achieved. In particular, the training error of the network with 16 filters continually goes down though very slowly, and seems still have not reached its lowest after 100 epochs. This is a strong sign of underfitting, in that the representing power of the filters is not adequate to explain the complexity of the inputs. Therefore, we will drop this case where the number of filters is 16 in the following context.

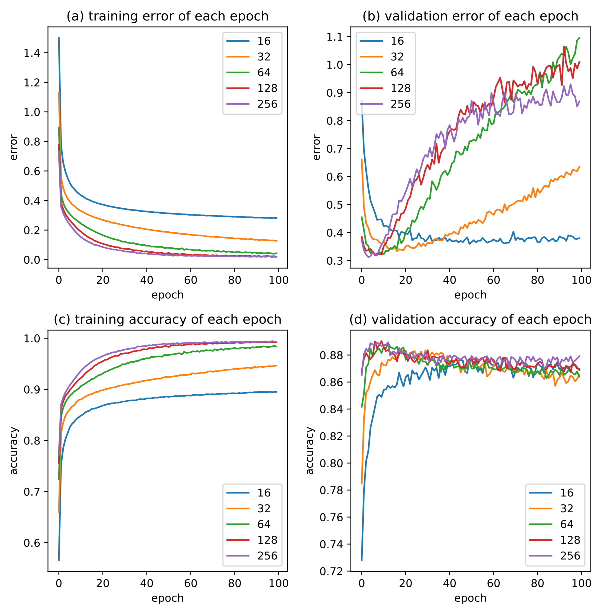

Figure 6. The learning curves in 100 epochs of the five networks using different number of filters, where each plot shows one type of learning curve for simple comparison: (a) training error, (b) validation error, (c) training accuracy, and (d) validation accuracy. The results are taken from a single run for each network. The networks are initialized with different seed. initialization.

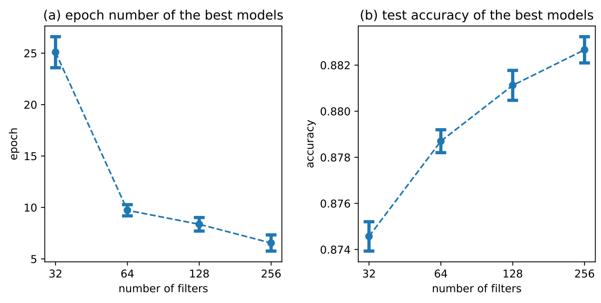

Figure 7. The results in table 3 plotted in error bars. The errors shown are 1 standard deviation. Note the x axis is plotted in log scale.

Similar to the procedure in section 4.1, we repeat each experiment 10 more times, and take the average of their highest validation accuracy as well as its epoch number (1-based). The epoch budget is adjusted accordingly so that each network can reach its best while not being overtrained for too many epochs. Table 3 shows the results of the experiment, and figure 7 presents a more intuitive comparison of their performance.

NUMBER OF FILTERS	EPOCH	TEST ACCURACY
32	25.09 ± 1.51	88.46% ± 0.064%
64	9.73 ± 0.55	87.87% ± 0.050%
128	8.36 ± 0.66	88.11% ± 0.065%
256	6.55 ± 0.79	88.27% ± 0.057%

Table 3. The epoch number of the best model and its test accuracy for the networks using different number of filters. The results shown in every row are averaged over 11 trials, and are reported in the format of 'mean ± 1 standard deviation'.

5. Discussion

5.1. Max pooling vs Average pooling

From figure 5 we see that under the exactly same conditions, using average pooling layer gives a slightly higher test accuracy than max pooling layer. This is because max pooling reduces the input size simply by discarding, in our 2×2 case, 75% of the data, and therefore loses the information which could be useful in the task. On the other hand, although average pooling gives better accuracy, it takes almost 3 times more epochs for the network with average pooling to train than max pooling in average (table 2). In terms of the learning curves (figure 4), we see that average pooling is more stable, in that the validation accuracy does not drop dramatically as max pooling after reaching its peak. This is because it takes all the inputs into account and thus making the previous layer adjust more smoothly in back propagation than max pooling, which only extracts the extreme values. Considering these facts, we can conclude that average pooling has no significant advantage over max pooling. Therefore two pooling layers are interchangeable depending on the application.

Beyond these conventional pooling techniques, there are also upgraded versions of them. A mixed pooling layer which generalizes pooling operation was proposed in (Accivatti et al., 2006). This max-average pooling combines the efficiency of max pooling and the high performance of average pooling by taking the weighted sum of the two operations, and has been shown to outperform the conventional max pooling in terms of accuracy. Proposed more recently in (Graham, 2015), the fractional max pooling overcomes the limitation of the rapid reduction by allowing non-integer pooling size, and has been shown to improve the state-of-art for CIFAR-100 without using dropout.

5.2. Convolution vs Pooling

In our experiments, we see that for this particular setting, the dilated Conv network gives the highest test accuracy among the models, and only uses almost as few epochs as max pooling network to train. This result can be partially explained by the fact that although having the same size of kernels, the dilated convolutional network has a much larger size (input size in layers) and more than 2 times of parameters than pooling layers (table 1). Moreover, the breadth of its context is another important factor. The dilation allows the filters to focus on the spatial information in a larger scale, and thus making it likely to extract more useful information from the inputs. The drawback of it is that it trains too slowly in terms of runtime (table 2), because of its large size.

The strided convolutional network, on the contrary, is outperformed by both pooling networks on test accuracy, even with more parameters, although it uses the least number of epochs to reach its best model. In (Springenberg et al.), it is argued that the max pooling can be replaced with strided convolutional network without the loss in accuracy. That said, our experiment shows that this may not be true, and the result of this replacement is dependent on the architecture and the specific task to perform. It is worth investigating in depth how to determine whether to use strided convolution in place of max pooling in future work.

In general, we can conclude that the all-convolutional networks requires more memory than pooling networks, and dilated convolution is preferred over strided convolution for high accuracy.

5.3. Number of filters on max pooling

From figure 7, we see that increasing the number of filters improves the accuracy and shortens the training epochs as expected. Specifically, it can be noticed that increasing the number of filters has a larger effect when there are few number of filters than many, that is, the accuracy has an approximate linear relationship with the log of number of filters. More precisely speaking, increasing the number of filters causes a linear growth in memory requirement and runtime, while only a logarithmic logarithmic in accuracy. This clearly suggests that allocating more computational resources by adding filters to the network when there are already many will not cause any significant improvement of the performance.

6. Conclusions

In this report we reviewed four approaches of broadening the context of convolutional networks, which are max pooling, average pooling, strided convolution and dilated convolution. Two of them are pooling methods, which does not require extra parameters. We have shown that average pooling yields a higher classification accuracy of the two on reduced EMNIST dataset while taking more epochs in training stage. As with all-convolutional networks, we discovered that dilated convolution outperforms the pooling layers, whereas strided convolution does not, meaning that all-convolutional networks does not always perform better than networks that use pooling. We have also shown that increasing the number of filters does improve the accuracy of the max pooling network. However, the effect of adding more filters diminishes exponentially as the number of filters gets large. Future work is suggested to aim on exploring the conditions which determines the performance of strided convolution.

References

Accivatti, C, CC, Harro, and KE, Bothner. Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree. Technical Report 1, 2006. URL http://search.ebscohost.com/login.aspx?direct=true{&}db=cin20{&}AN=106323365{&}site=ehost-live.

Antoniou, Antreas, Slowik, Agnieszka, Crowley, Elliot J., and Storkey, Amos. Dilated DenseNets for relational reasoning. *arXiv:1811.00410*, 2018. URL https://arxiv.org/abs/1811.00410.

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

Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Deep Sparse Rectifier Neural Networks. Technical report, 2011. URL http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

Graham, Benjamin. Fractional Max-Pooling. Technical report, 2015. URL https://arxiv.org/pdf/1412.6071.pdf.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. In *ICML*, 2015. URL https://arxiv.org/abs/1412.6980.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. ISSN 00189219. doi: 10.1109/5.726791. URL http://ieeexplore.ieee.org/document/726791/.

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

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. WaveNet: A generative model for raw audio. *arXiv:1609.03499*, 2016. URL https://arxiv.org/abs/1609.03499.

Yu, Fisher and Koltun, Vladlen. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016. URL https://arxiv.org/abs/1511.07122.