MLP Coursework 2: Exploring Convolutional Networks

s1786991

Abstract

This report aims to understand the different learning behaviors of convolutional networks using different methods to broaden the context. We start by exploring the implementation of convolutional layers and max-pooling layers in the framework provided using matrix multiplication, or correlation. Then we conduct a series of experiments to study the learning curves and test accuracy of convolutional models under the different setting. We find that same hyper-parameters have a different effect on a convolutional network.

1. Introduction

Convolutional Networks have been motivating the flourishing of deep learning literature since the great success of AlexNet (Krizhevsky et al., 2012) on ImageNet classification challenge (Russakovsky et al., 2014). They also benefit many other computer vision applications as objects detection (Girshick, 2015), semantic segmentation (Long et al., 2015), and images synthesis (Kingma & Welling, 2013), etc. This report aims to form a solid understanding of convolutional networks by focusing on the classification task of Balanced EMNIST dataset (Cohen et al., 2017) with 47 classes (digits and letters), whose training set has 100,000 images, validation set and test set both have 15,800 images. When used for classification, a convolutional network works as a feature extractor, who processes an inputs images into a latent feature representation. Then fully connected layers are applied upon the latent representation to obtain a prediction of the class an image belongs to (more details in section 2).

In order to strengthen the understanding of convolutional networks: 1) the idea of convolutional layers and pooling layers in a convolutional network; 2) how convolutional networks broaden context (receptive field) to extract feature representations, this report explores their implementation from scratch and the effects of different ways used to enlarge their context extracted.

A convolutional layer could be implemented by convolution or correlation, which are stemmed from signal processing. A correlation could be achieved by flipping the sign of the variable in convolution. A convolutional network normally consists of multiple convolutional layers and pooling layers, where convolutional layers extract feature representation and pooling layers reduce the spatial dimension of the feature representation. We implement a convolutional layer and pooling layer within the framework provided from scratch.

One core concept in convolutional networks is the receptive field, referring to a region of an input image over which a hidden unit covers, also could be understood as the context. The receptive field of a convolutional model directly relates to the number of convolutional layers stacked in this model, the kernel size of each convolutional layer, and the operation the model used to reduce the spatial information. We look into three common operations to broaden the receptive field in this report – pooling (LeCun et al., 2015), strided convolution (Springenberg et al., 2014), and dilated convolution (Yu & Koltun, 2015). Also, we study the effect of kernel size, the number of kernels, and the number of convolutional layers for different operations, which directly relates to the receptive field of a model. We show that for convolutional models using different operations, these parameter settings have different effects.

In a nutshell, this report is organized as follows: we introduce the idea of convolutional layer and pooling layer, and how we implement them in section 2; in section 3, we study the context of a convolution model and distinguish the different operations used to broaden it; we then conduct a series of experiments in **??** to study the effect of different operations and parameters in convolutional models; we interpret the results in **??** and section 5 ; we then conclude the report in section 6.

2. Implementing convolutional networks

A convolutional layer usually has three main parameters: kernel size, stride, and padding size. Convolutional networks use kernels, which have a smaller size than input, interacting sparsely with units in next layer, whereas in multi-layer perceptron, all input units in one layer contribute to an output unit in next layer. Such sparse connectivity allows the network to efficiently describe complicated interactions between many variables by constructing from simple sparse interactions. Kernels slide over inputs to obtain output feature maps by a stride defined in the convolutional layer. In order to construct more complicated interaction and capture long-dependency from more elements while avoiding shrinking the output feature map, padding is introduced into a convolutional layer, where zeros are added surrounded the inputs.

A pooling function helps the model learn relatively invariant features to be more robust to small variance in input by replacing the output feature map at a certain location with a summary statistic of the nearby outputs. A pooling layer reduces the spatial dimension of the output feature maps from a convolutional layer hence reduce the computation required, which also has parameters of kernel size and stride. However, a pooling layer damages the spatial relationship when its kernel size equals to its stride, where a non-overlap pooling is conducted.

We implement the convolutional layer and pooling layer using matrix multiplication rather than loop over the sliding windows and take convolution with kernels for efficiency. For forward propagation, we first transform the 4D mini-batch input into a matrix, whose column represents an unrolled sliding window from the inputs and row represents an element in this window that would be multiplied with corresponding weight in a kernel. We then transform the kernels accordingly and conduct matrix multiplication between the transformed kernels matrix and transformed input matrix to get the output of a 2D matrix form. Finally, the 2D output could be reshaped into 4D to get output feature maps. For back propagation, we use the same trick as in forward propagation, transforming kernels and gradients w.r.t output into a 2D matrix, performing matrix multiplication, transforming reversely to obtain the gradient w.r.t input in this layer or multiplying with transformed input matrix to obtain the gradient w.r.t parameters.

The implementation of max pooling layer is similar to convolutional layer. For forward propagation, we transform the 4D input tensor into a 2D matrix, filter the matrix by its maximum value across columns, which corresponds to the maximum value within a sliding window, and record their indices to get a vector, finally unroll this vector to get the output. For backward propagation, we transform the gradient w.r.t output into a vector, expand it according to the indices recorded to a 2D matrix, reversely transform it to gradient w.r.t. input or multiply with transformed inputs to get gradient w.r.t parameters.

Although matrix multiplication is more efficient than for loop, this method requires careful manipulation of the index.

3. Context in convolutional networks

Convolutional networks outperform multi-layer perceptrons due to their capability of exploiting context over an image. The context is extracted by convolutional kernels, whose weights could be learned to conduct edge detection of any angle and shape. A convolutional network, stacking these kernels with non-linearity imposed in between, is capable of obtaining a receptive field as large as input images by extracting low-level features at shallower layers, such as edges, and high-level features at deeper layers, such as item shape (Zeiler & Fergus, 2014). The receptive field of a model could be computed by:

$$r_n = r_{n-1} + (k_n - 1) \prod_{i=1}^{n-1} s_i \qquad (1)$$

where r_n represents the receptive field of a convolutional network at *nth* layer, k_n and s_n represents the kernel size and stride at *nth* layer respectively. From eq. (1), one could understand that the receptive field of a model is directly related to kernel size and number of convolutional layers stacked. The stride size at each layer or the downsampling operation conducted after one convolutional layer could also enlarge the receptive field.

One of the most common downsampling operations is pooling. A pooling layer decreases the spatial dimension of feature maps by only keeping the maximum value (max pooling) or the average value (average pooling) within the kernel, hence increasing receptive field. Although max-pooling layers are used in many work (LeCun et al., 2015; Krizhevsky et al., 2012) by alternating it with convolutional layers, the reason why it helps the network achieve human-level performance stays unclear and poorly explained (Springenberg et al., 2014). Also, non-overlapping pooling layers discard spatial relationship to keep most invariant features while reducing the spatial dimension. Another downsampling operation is strided-convolution. Springenberg et al. (2014) show convolutional layers with a stride of 2 could be used to replace pooling layers with the benefits of learning to down-sample rather than deterministic down-sampling. Models consist of strided-convolutional layers are simple and could achieve start-of-the-art results as the ones using pooling layers, though they contain more parameters because of the usage of extra convolutional layers.

While down-sampling operations may help improve model performance in classification tasks, they result in deleterious effect to tasks as dense prediction and semantic segmentation that require pixel-level information, due to loss of resolution through processing. Although it's possible to involve a series of up-sampling operation to recover the lost resolution after down-sampling, as in U-Net (Ronneberger et al., 2015), the number of down-sampling or up-sampling operation needed for full recovery still remains difficult to decide. Under such circumstances, we may expect to increase the receptive field by stacking more convolutional layers and using larger kernel size, which, with no doubt, brings a considerable increase in the number of parameters into a model. However, Yu & Koltun (2015) propose dilated convolution for more fine-grained semantic segmentation results without increasing the number of parameters in a network, where the receptive field is expanded by dilating the kernel size at each layer and the feature map dimension is kept.

Aiming to fully understand these operations (max pooling, average pooling, strided convolution, and dilated convolution), and how context is extracted by convolutional layers, we address the following problems through extensive experiments in next section:

1. What is the difference between these 4 methods in terms of learning behavior? (section 4.1)
2. How does the number of filters in each layer effect the

performance of each operation? (section 4.2)

3. How does the number of layers affect the performance for each operation and receptive field? (section 4.3)

4. How does kernel size affect the performance for each operation and receptive field? (section 4.4)

4. Experiments

In this section, we conduct diverse experiments to address the above questions. We use a baseline convolutional network with 4 convolutional layers, 64 filters and a kernel size of 3 for each layer. Max pooling layers with a size of 2, stride of 2 and padding of 1 are used after each convolutional layer in the baseline model. For easier comparison, we use adaptive pooling to ensure output of final layer before the fully-connected layer is always (2, 2) spatially, and we only use 1 fully-connected layer with 47 units, whose output directly feeds into the softmax activation. We use Adam optimize with a default learning rate of $1e-3, \beta_1$ of $0.9, \beta_2$ of 0.999 ,and a weight decay coefficient of $1e-5$ to train all the models in this section. We train all the models with a batch size of 100 for 30 epochs. All the models in this section are trained for 3 times with 3 independent random seeds to reduce the stochastic when conducting the comparison.

4.1. Difference of 4 methods

To compare the different learning behaviors of these methods, we replace the max-pooling layers in the baseline model with average pooling layers, strided convolutional layers, and dilated convolutional layers. The average pooling layers have the same parameters as max-pooling layers. The strided convolutional layers also have 64 filters and a kernel size of 3, but the stride is set to 2. The dilation for dilated convolutional layers is 2, 3, 4, 5 respectively. We set the same padding for all the dilated convolutional layers, hence all the feature maps are of the same size for dilated convolutional models. We train these 4 models according to the hyper-parameter setting above and plot the learning curves. The learning curves are computed using the statistics from 3 independent training with error band.

From training time curves in fig. 1, it's obvious that, for down-sampling operations, strided convolution outperforms both max pooling and average pooling. Although average pooling has the worst training performance for both accuracy and robustness, it illustrates a descent capability of generalization, where its validation curve is sticking to its training curve before overfitting and a smaller gap between them after the model overfits training data. Max pooling outperforms average pooling in terms of training curves due to the fact that it only utilizes the locally maximum values rather the average value over a region, where max pooling model learns the most invariant features of training data. While max pooling model becomes more robust to training data, its validation accuracy drops dramatically due to overfitting. Because the usage of extra convolutional layer enables the model to learn to down-sample and extract invariant features, we can also observe that strided convolution and dilated convolution are pretty robust with quite a small error band of training curves. The dilated convolutional model has comparable performance with max pooling model, yet has the best generalization performance and best test accuracy (see table 1).

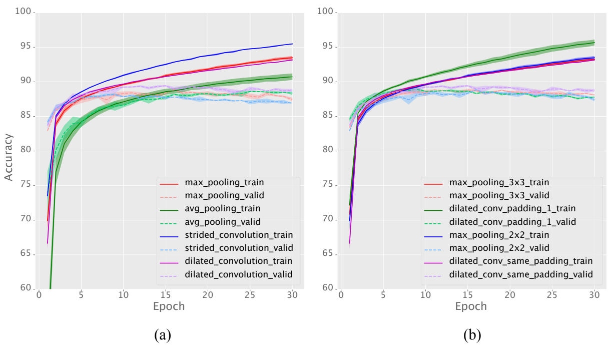

Figure 1. (a) Learning curves for different operations. (b) Learning curves for comparison of non-overlapping max pooling with overlap max pooling, dilated convolution with a padding of 1 and same padding.

We also calculate the receptive field for each method using eq. (1), and count the number of parameters for each model. Then we compute the best test accuracy (obtained using models from early stopping) for each model. From table 1, we can observe that strided convolutional model has the greatest receptive field which is larger than the input image dimension, compared to other models, yet its test accuracy is similar to pooling models, where also have receptive fields larger than input dimension. Strided convolutional model and dilated convolutional model have same parameter number, which is twice more than pooling models. Out of these 4 methods, the dilated convolutional model has the best test accuracy with the smallest receptive field compared to others, yet its training curve is similar to max pooling. This may result from we exploit the same padding for dilated convolution. As the dilation rate increasing, the padding needed also increase dramatically. Therefore, dilated convolutional, which use the same padding, incorporates much invalid information from padding, resulting in a longer training time. Also, dilated convolution does not down-sample feature maps, especially when using the same padding, it may preserve useful information that is discarded by down-sampling operations, which may help improve the test accuracy.

We also study the effect of non-overlapping pooling (pooling using a kernel size of 3) and the padding in dilated convolutional models. We train a max pooling model with a kernel size of 3 rather than 2, and a dilated convolutional models with a padding size of 1 rather than using the same padding. We plot the learning curves in fig. 1. From table 1 and fig. 1, we can observe that, overlapped max pooling improve both training and test performance without introducing more parameters, which might result from overlapped max pooling preserve the spatial relationship as in strided convolution (also using a kernel size of 3 and a stride of 2), which is discarded by non-overlapping pooling. Also, dilated convolution using a padding of 1 can achieve better training performance than dilated convolution using the same padding, but the latter have better test accuracy, which reflects the fact that down-sampling, appearing in dilated convolution using a padding of 1, helps improve the training performance of models, but weaken the capability of generalization.

Methods	R.F.	# Params	Test Acc.
Max Pool. 2×2	46	123,503	87.86 ± 0.12 %
Max Pool. 3×3	61	123,503	88.02 ± 0.09 %
Avg. Pool. 2×2	46	123,503	87.85 ± 0.10 %
Strided Conv.	61	271,215	87.56 ± 0.08 %
Dilated Conv.(same pad.)	37	271,215	88.74 ± 0.10 %
Dilated Conv.(pad. 1)	37	271,215	88.09 ± 0.17 %

Table 1. Receptive field, number of parameters, and test accuracy of different methods.

Methods	# Filters	# Params	Test Acc.(%)
Max Pool.	16	10,175	86.20 ± 0.37 %
	32	34,127	87.60 ± 0.32 %
	48	71,903	87.71 ± 0.35 %
	64	123,503	87.86 ± 0.12 %
Avg. Pool.	16	10,175	84.04 ± 0.53 %
	32	34,127	86.72 ± 0.30 %
	48	71,903	87.78 ± 0.03 %
	64	123,503	87.85 ± 0.10 %
Stided Conv.	16	19,455	86.68 ± 0.21 %
	32	71,119	86.92 ± 0.14 %
	48	155,039	87.34 ± 0.08 %
	64	271,215	87.56 ± 0.08 %
Dilated Conv.	16	19,455	87.97 ± 0.19 %
	32	71,119	88.49 ± 0.17 %
	48	155,039	88.72 ± 0.13 %
	64	271,215	88.74 ± 0.10 %

Table 2. Receptive field, number of parameters and test accuracy of models using different number of kernels under different operation settings.

4.2. Effects of filters

In this subsection, we study how filters affect the performance of different methods. We vary the number of kernels for each of the models in section 4.1 from 16, 32, 48, and 64. We also plot the learning curves and compute the parameters and test accuracy. The receptive field has no relationship with filter numbers, hence they are the same for each method in table 1.

The learning curves in fig. 2 demonstrate that, although an increase in filters always leads to better training performance within our selection range, the improvement of training performance has been decreasing, where the gaps between the training curves becomes smaller, which means there exists a point that blindly increasing the number of filters does not help improve the training performance but only increase overfitting. We can also observe generalization of these models from the stickiness of validation curves to training curves. Average pooling models are of better generalization, as we conclude in section 4.1, where the validation curves stick to training curves when filters are 16, 32, and when filters are 48, 64, the gaps between validation curves and training curves are small. For dilated convolutional models, 16 filters are enough to yield better results than other models with more filters (see table 2).

4.3. Effects of layers

We research the effects of the number of layers in this subsection. As in section 4.2, we vary the number of layers for each model using different operations as 2, 3, 4, and 5. Stacking more layers results in a larger receptive field. We plot learning curves in fig. 3 and compute test accuracy, and receptive field as before in table 3.

Adding extra layers also introduces more parameters into the model, and hence brings more overfitting. Also, the improvement of training performance tends to saturate as more layers are adding to the model. Except for average pooling model of 2 layers, increasing the number of layers does not produce a huge improvement for the models, and we can notice that for models that have layers of 3 have a receptive field that is spatially similar to input. Larger receptive field than input dimension incorporate redundant information from padding, which is invalid and inefficient, therefore have little improvement on the model but even make the model overfit on training data because of the increase of parameters. Noteworthy is that, for strided convolution, a model with layers of 5 performs worse than the model with layers of 4. This may result from too many down-sampling operations conducted by convolutional layers of a stride of 2, which makes the spatial dimension of the output feature map before the adaptive layer smaller than (2, 2). Therefore, for down-sampling operations such as pooling and strided convolution, we need to take care of the number of down-sample operations in a model, where too many down-sampling may discard too many useful information, results in worse performance of a model. If we use 6 layers for pooling models, similar results would appear. Also, for the dilated convolutional model, the model that has layers of 5 and model that has layers of 4 have similar training and validation performance because of the usage of the same padding, which means the model preserves all information rather than only using the most invariant and running the risk of over down-sampling and lose information.

4.4. Effect of filter size

Finally, we consider another factor that affects the receptive field – the kernel size of convolutional layers. The size of a kernel directly determines how large a context the model is looking for from the last layer. We train the models using different kernel size from 3, 5, 7 under different

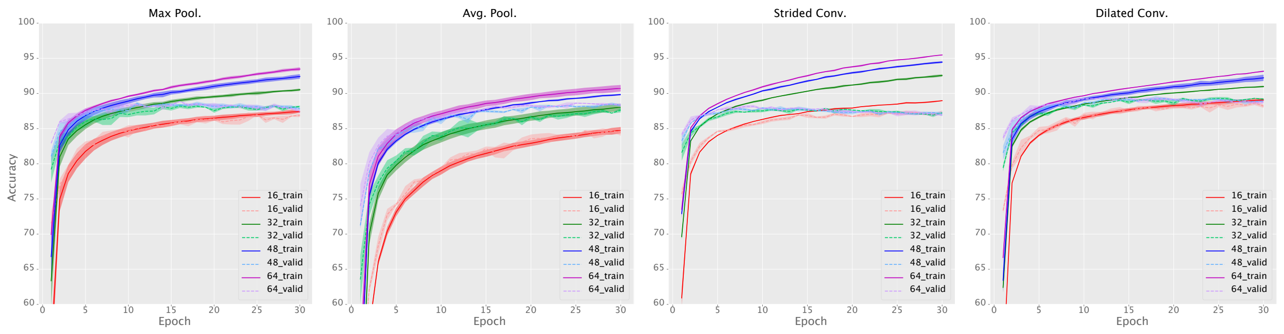

Figure 2. Learning curves of different methods using different number of filters from 16 to 64.

Methods	# Layers	R.F.	# Params	Test Acc.
Max Pool.	2	10	49,647	86.76 ± 0.30 %
	3	22	86,575	87.81 ± 0.33 %
	4	46	123,503	87.86 ± 0.12 %
	5	94	160,431	87.82 ± 0.11 %
Avg. Pool.	2	10	49,647	86.14 ± 0.15 %
	3	22	86,575	87.80 ± 0.05 %
	4	46	123,503	87.85 ± 0.10 %
	5	94	160,431	86.60 ± 0.17 %
Strided Conv.	2	13	123,503	87.47 ± 0.16 %
	3	29	197,359	87.09 ± 0.13 %
	4	61	271,215	87.56 ± 0.08 %
	5	125	345,071	87.03 ± 0.01 %
Dilated Conv.	2	15	123,503	88.22 ± 0.19 %
	3	25	197,359	88.68 ± 0.12 %
	4	37	271,215	88.75 ± 0.09 %
	5	51	345,071	88.51 ± 0.04 %

Table 3. Receptive field, number of parameters and test accuracy of models using different number of layers under different operation settings.

operations. We also vary the padding size accordingly to avoid the output feature maps shrink too much. A larger kernel size results in a larger receptive field, but also significantly increase the number of parameters in the model (see table 4).

For max pooling models, an increase of kernel size also introduce more stochastic into the model, where the error band of learning curves broadens. Although increasing the kernel from 3 to 5 improve the training performance of the model, from 5 to 7 results in similar training, validation, and test performance table 4, which demonstrates that a kernel size of 5 may help capture the most invariant features from a larger context, but kernels of size 7 or larger captures similar features as kernels of size 5. For average pooling models, a larger kernel size always leads to better anytime performance within our selection range, which differs from max-pooling layers. This also reflects the inherent difference between max pooling and average pooling, where the latter utilize the average over the context but the former only keeps the maximum.

For strided convolution, kernels of size 5 and kernels of size 3 have similar training performance and validation performance, which reflects the fact that strided convolutional models learn to extract useful features rather than fix behavior as in pooling operations. Therefore, varying the kernel size within a reasonable range could yield comparable results. However, kernels of size 5 outperform kernels of size 7 for strided convolutional models. Same results appear in dilated convolutional models. The stacking of small kernels has regularization effect on using only one larger kernel (Simonyan & Zisserman, 2014), with far fewer parameters, forcing larger kernels to have a decomposition using smaller kernels, therefore, producing better results.

Methods	Kernel	R.F.	# Params	Test Acc.(%)
Max Pool.	3	46	123,503	87.85 ± 0.12 %
	5	76	321,135	88.05 ± 0.04 %
	7	106	617,583	87.71 ± 0.24 %
Avg. Pool.	3	46	123,503	87.85 ± 0.10 %
	5	76	321,135	88.37 ± 0.38 %
	7	106	617,583	88.51 ± 0.20 %
Stided Conv.	3	61	271,215	87.47 ± 0.04 %
	5	121	730,991	87.65 ± 0.13 %
	7	181	1,420,655	87.47 ± 0.15 %
Dilated Conv.	3	37	271,215	88.22 ± 0.13 %
	5	73	730,991	88.41 ± 0.18 %
	7	109	1,420,655	87.88 ± 0.17 %

Table 4. Receptive field, number of parameters and test accuracy of models using different kernel sizes under different operation settings.

5. Discussion

We summarize the interpretation of section 4 in this section. Convolutional models using different methods to broaden the context differs in terms of learning behaviors. For different methods, the parameters setting in a convolutional model also has different effects.

Max pooling models, learning the most invariant features by exploiting the maximum value within a region, are robust to small variances in input images. Due to this characteristic,

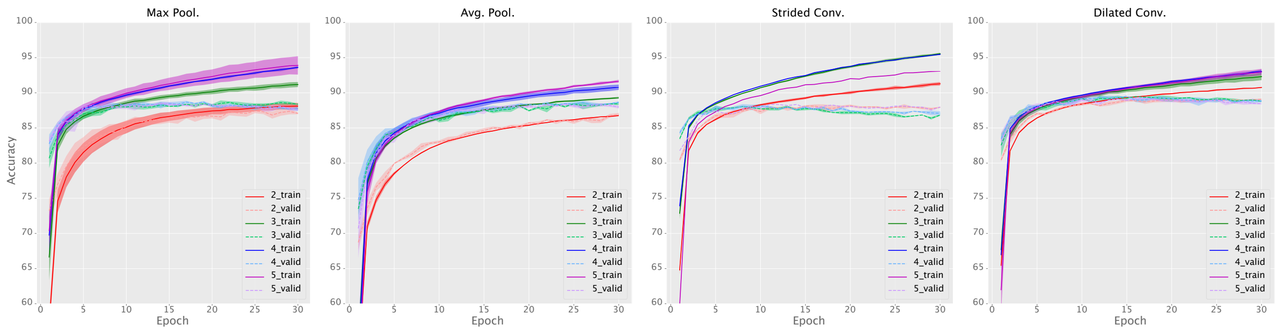

Figure 3. Learning curves of different methods using different number of layers from 2 to 5.

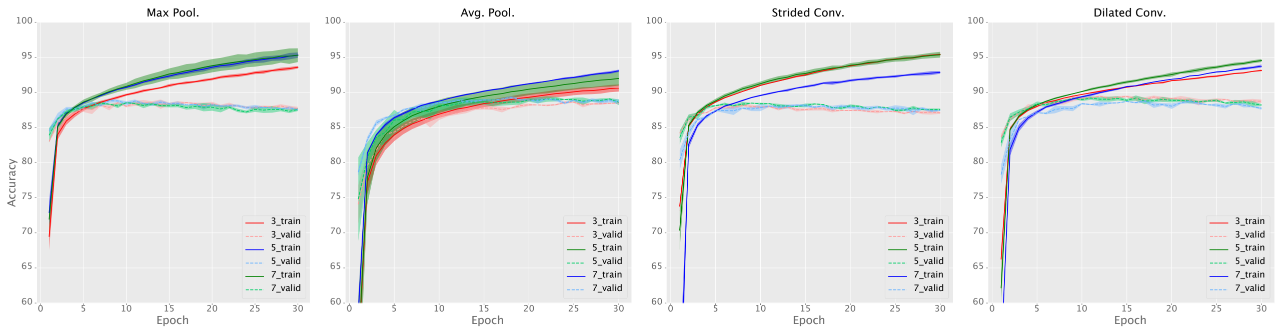

Figure 4. Learning curves of different methods using different kernel sizes from 3, 5 and 7.

larger kernels may help max pooling models extract more useful features. Contrast to max pooling, average pooling models utilize the average value over a region, hence demonstrating a better capability of generalization on test data. Also, larger kernels improve the model performance.

Strided convolutional models and dilated convolutional models use extra convolutional layers to broaden context. Larger kernels would facilitate training, but an improvement of training performance saturates while kernels become larger due to learning of convolutional layers and the regularization effect of stacking small kernels (Simonyan & Zisserman, 2014). For dilated convolutional models, Wang et al. (2018) has demonstrated when designing dilation rate for dilated convolutional layers, one needs to pay attention to set reasonable dilation rate to avoid the gridding effect. Otherwise, the context extracted would be separated by elements unused, which is undesirable and deleterious to semantics segmentation tasks. Also, dilation convolution utilizes long-ranged information which might be irrelevant. Long-ranged information might be helpful for segmentation of larger objects but also leads to a loss of information of smaller objects.

6. Conclusions

From the implementation of convolutional models and experiments of them, we have learned that down-sampling operations facilitate learning by preserving the most useful and invariant features, but also hurt generalization/test accuracy of a model due to the loss of resolution. Strided convolution learns to downsample, further improving the training performance of a model and making it more robust. Dilated convolution is a way to broaden the receptive field of a model without introducing down-samplings. Different operations are suitable for different tasks. One need to build a convolutional network referring to the task, where downsampling helps the classification and dilated convolution which can preserve full resolution performs better for segmentation. Sometimes, one need also combine strided convolution and dilated convolution to achieve the best results.

References

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.

Girshick, Ross. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

LeCun, Yann et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, pp. 20, 2015.

Long, Jonathan, Shelhamer, Evan, and Darrell, Trevor. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

Ronneberger, Olaf, Fischer, Philipp, and Brox, Thomas. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael S., Berg, Alexander C., and Li, Fei-Fei. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. URL http://arxiv.org/abs/1409.0575.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Wang, Panqu, Chen, Pengfei, Yuan, Ye, Liu, Ding, Huang, Zehua, Hou, Xiaodi, and Cottrell, Garrison. Understanding convolution for semantic segmentation. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1451–1460. IEEE, 2018.

Yu, Fisher and Koltun, Vladlen. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

Zeiler, Matthew D and Fergus, Rob. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.