UPPSALA
UNIVERSITET

Scale-invariant CNNs

Implementation and performance analysis of different approaches

Ruoqi Zhang & Wei Peng

Project in Computational Science: Report

Jan 2020

PROJECT REPORT

Department of Information Technology

Abstract

Convolutional neural networks(CNNs) have achieved excellent results on image classification tasks on large datasets like ImageNet [1], MS COCO [2] and Open Images [3]. The success of CNNs is from their ability to learn complex patterns by feature maps in the layers. But CNNs have no mechanism to fit different scales. Here, we implement and evaluate four different networks and compare their performance with two baselines on three different datasets: MNIST, FMNIST and OralCancer. In order to test the ability of scale-invariance, we use unscaled dataset as training data and scaled dataset as test data. The results show that standard CNN with adequate data augmentation has the best performance. Except that, significant improvements can be observed using locally scale-invariant convolutional neural networks (SI-ConvNet) compared with standard CNN without data augmentation method. Scale steerable filters for locally scale-invariant convolutional neural networks (SS-CNN) replace the kernels in the SI-ConvNet to avoid interpolation artifacts but they need more time to converge and the performance is not better than that of SI-ConvNet. We also add blur pool for SI-ConvNet and SS-CNN in order to improve the accuracy on OralCancer. Blur pool is simple to implement and does not increase the training time, however the results did not show any improvement.

Contents

1 Introduction

Convolutional neural networks(CNNs) have achieved excellent results on image classification tasks on large datasets like ImageNet [1], MS COCO [2] and the Open Images [3]. The success of CNNs is related to their ability to learn complex patterns by feature maps in the layers. But CNNs have no mechanism to fit different scales. So, to detect simple patterns at different scales, they have to learn different filters to obtain feature maps for different scales. Unfortunately, it is not easy to do so. The networks need more filters, which means the number of parameters increases and this makes it harder to train.

In this project, we implement four different scale-invariant networks and compare their performance with two baselines. Baseline1 is a standard CNN and Baseline2 is a standard CNN with data augmentation. The four different networks, which incorporate scale invariance property in the network architecture in different ways, are SI-ConvNet [4], SS-CNN [5], Antialised-SIConvNet [4][6] and Antialised-SSCNN [5][6]. The SI-ConvNet applies filters to multiple scaled versions of the input in each convolution layer, and obtains a scale-invariant representation through max-pooling over scales. The SS-CNN is constructed based on SI-ConvNet, but replaces the filters with a linear combination of the proposed scale steerable basis filters. Then the replaced filters are steered to different sizes to convolve with the input, and the responses are max-pooled over scales. The Antialised-SIConvNet is designed with an aim to integrate classic anti-aliasing method into the downsampling strategies of SI-ConvNet, and similar operations are performed for Antialised-SSCNN.

We then evaluate the methods on three datasets: MNIST-Scale, FMNIST-Scale and OralCancer-Scale. The datasets are scaled versions of MNIST [7], FMNIST [8] and OralCancer [9]. Also, in order to test the ability of scale-invariance, we use unscaled dataset as training data and scaled dataset as test data. This is different from the experiments, done in the original papers and, thus, we observe different results. The results show that standard CNN with adequate data augmentation have the best performance. Except that, significant improvements can be observed using SI-ConvNet compared to Baseline1. As for SS-CNN, given more epochs, it can perform better than Baseline1 but still not as good as SI-ConvNet. The two networks integrated with anti-aliasing methods do not show obvious difference either in accuracy or training time compared with the same architectures without anti-aliasing.

The rest of the paper is organized as follows. In Section 2, the scale-invariant networks used in this project are briefly introduced, and the principles of how they work are also illustrated. In Section 3, the datasets used to evaluate the networks are presented with some examples shown, and the process of how the scaled versions of datasets are produced is also explained. In Section 4, the design of the experiment is introduced and the results are represented. The discussion of the results is included in Section 5, and finally conclusion and future work are discussed in Section 6.

2 Background

2.1 Equivariance and Invariance

2.1.1 Shift-equivariance and shift-invariance

A function $\widetilde{F}$ is shift-equivariant if the shifting operation applied to the input is equally added on the output:

$$\widetilde{F}(\text{Shift}_{\Delta h,\Delta w}(X)) = \text{Shift}_{\Delta h,\Delta w}(\widetilde{F}(X)) \quad \text{for all } (\Delta h, \Delta w). \tag{1}$$

In other words, if a function $\widetilde{F}$ is shift-equivariant, the result of first shifting the input X with height Δh and width Δw then applying $\widetilde{F}$ will be the same as first applying $\widetilde{F}$ on the input X then shifting with height Δh and width Δw.

A function $\widetilde{F}$ is shift-invariant if the shifting operation applied to the input does not influence the output:

$$\widetilde{F}(\text{Shift}_{\Delta h,\Delta w}(X)) = \widetilde{F}(X) \quad \text{for all } (\Delta h, \Delta w). \tag{2}$$

In the continuous case, convolution is shift-invariant by definition, which ensures that the filter responses do not depend on the position. This property is, however, only approximated in the discrete case, i.e., when CNNs are applied to discrete image functions, as discussed by Zhang [6].

2.1.2 Scale-equivariance and scale-invariance

Similarly, a function $\widetilde{F}$ is scale-equivariant if scaling the input X with a scale factor s equally scales the output:

$$\widetilde{F}(\text{Scale}_s(X)) = \text{Scale}_s(\widetilde{F}(X)) \quad \text{for all } s. \tag{3}$$

A function $\widetilde{F}$ is scale-invariant if scaling the input X results in an identical representation:

$$\widetilde{F}(\text{Scale}_s(X)) = \widetilde{F}(X) \quad \text{for all } s. \tag{4}$$

The common solution to achieve scale-invariance is through data augmentation, where scaling transformations are applied to the existing training set with an aim to capture scale variability expected in the test set and incorporate it in the training set.

2.2 Locally Scale-Invariant Convolutional Neural Networks

In [4], the authors propose a scale-invariant convolutional neural network (SI-ConvNet) that allows CNNs to learn discriminative features without increasing the number of parameters.

To make the CNN learn patterns at multiple scales, the authors build the scale-invariance at the layer level based on the standard CNN. Figure 1 shows the layer comparison of a CNN and the proposed SI-ConvNet. In a scale-invariant convolution layer, first the input is scaled to different sizes, and then each of the scaled input is convolved with one same filter producing feature maps at different scales. To normalize the feature maps, each of them is rescaled to a canonical size. Finally, the rescaled feature maps are aligned and max-pooled,

and only the maximum responses at each spatial location are preserved. In this way, a locally scale-invariant representation is produced and sent to the next layer in the same size as the output of a standard convolution layer.

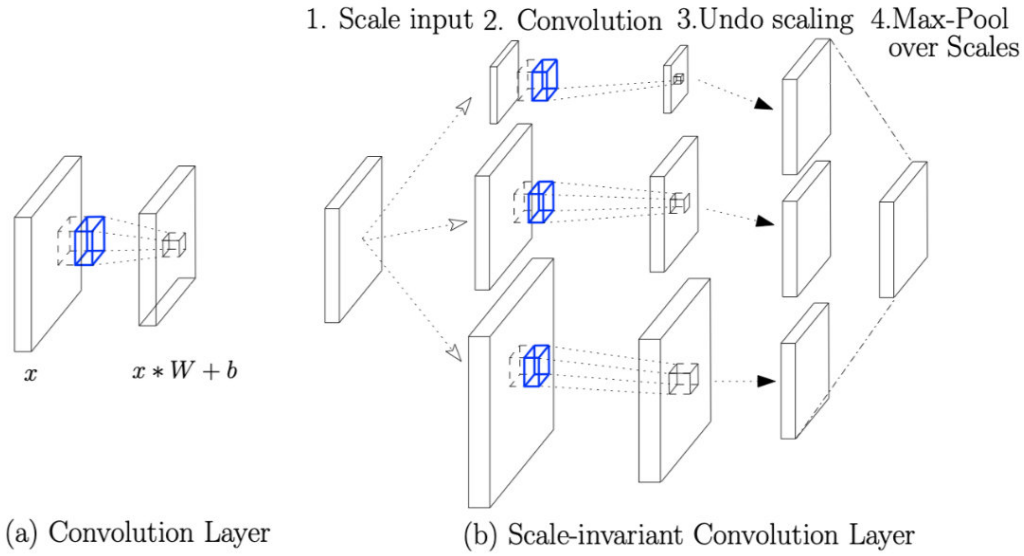


Figure 1: Side-by-side comparison of the structure of a convolution layer and the proposed scale-invariant convolution layer [4].

In the training process, scaling the input to n different sizes is analogous to increasing the number of feature maps by n times, because convolving the n scaled input images with a single filter is similar to convolving a single input image with n filters of different sizes. Thus, the SI-ConvNet can be trained without adding more parameters.

2.3 Scale Steerable Filters for Locally Scale-Invariant Convolutional Neural Networks

In [5], Ghosh and Gupta propose a scale-steerable filter basis for the locally scale-invariant CNN [4] based on the rotation steerable filters. They also introduce a scale-steered kernel through the linear combination of the scale-steerable basis filters. By replacing the kernels with scale-steered kernels, they improve the scale-invariance of CNNs.

2.3.1 Scale-steerable filters

Based on the rotation steerable filters in the form of circular harmonics [10], Ghosh and Gupta [5] construct the scale-steerable filters in the form, $W(r, \phi) = \Phi(\phi)F(\log r)/r^m$. They are expressed in polar coordinates. $\Phi(\phi)$ is of Gaussian form, with mean of ϕ_j and standard deviation of σ_ϕ. $F(\log r)$ is a complex function of unit norm, $e^{i(k(\log r)+\beta)}$, with filter order k and phase β. Thus, the proposed mathematical form of a scale-steerable basis filter is

$$S^{kj}(r, \phi) = \frac{1}{r^m}(K(\phi, \phi_j) + K(\phi, \phi_j + \pi))e^{i(k(\log r)+\beta)}, \tag{5}$$

where $K(\phi, \phi_j) = e^{-d(\phi,\phi_j)^2/2\sigma_\phi^2}$, with $d(\phi, \phi_j)$ equal to the distance between the two angles ϕ and ϕ_j. Examples of scale-steerable basis filters constructed from Equation (5) are shown in Figure 2.

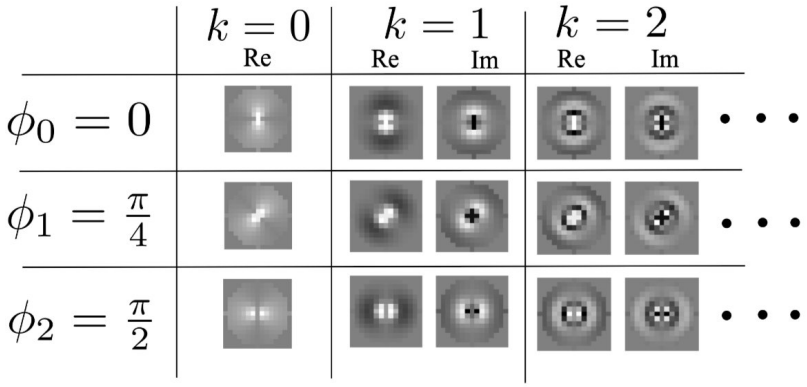


Figure 2: Scale-steerable basis filters for selected orientations and filter orders and $m = 1$ [5].

In order to prove the scale steerability of the basis filters, Ghosh and Gupta [5] propose and prove the following theorem:

Theorem 1 *Given a circular input patch $I(a)$ within a larger image, which is defined within the x, y range of $0 \leq \sqrt{x^2 + y^2} \leq a$. Let $I^s(a)$ denote the same patch when the image was scaled around the centre of the patch by a factor of s. We then have*

$$[I^s(a) \star S^{kj}(a)] = s^{m-2} e^{-i(k \log s)} [I(as) \star S^{kj}(as)], \tag{6}$$

where $\star$ is the cross-correlation operator.

From Equation (6) we can find that, if we want to compute the cross-correlation between the scaled image I^s and the filter S^{kj}, the result will be the same to compute the cross-correlation between the original image I and the filter S^{kj}, multiplied by some complex coefficients.

With the scale-steerable basis filters S^{kj}, Ghosh and Gupta [5] construct a filter W through a linear combination of S^{kj}, s.t. $W = \sum_{k,j} c_{k,j} S^{k,j}$, where $c_{k,j} \in \mathbb{C}$. Due to the linear combination, the scale steerability is preserved in W, and we can steer the scale of $W(a)$ on the range of radii a by a scale factor s:

$$W^s(as) = s^{m-2} \sum_k e^{-ik \log s} \big(\sum_j c_{kj} S^{kj}(as) \big). \tag{7}$$

Note that only the real part of $W^s(as)$ is used in the network.

2.3.2 Scale-invariant CNNs with scale steered weights

The scale-invariant CNN with scale steerable filters is described as Scale-Steered CNN (SS-CNN), and a proposed scale-invariant layer is shown in Figure 3. Firstly the scale-steerable basis S^{kj} is linearly combined as filter W and then W is scaled to different sizes using

Equation (7). Each of the scaled filters is convolved with the same input and generates different feature maps. Finally through max-pooling over scales, only the maximum response at each spatial location are preserved and sent to the next layer as input. In the whole training process, the network only learns the coefficients c_{kj}.

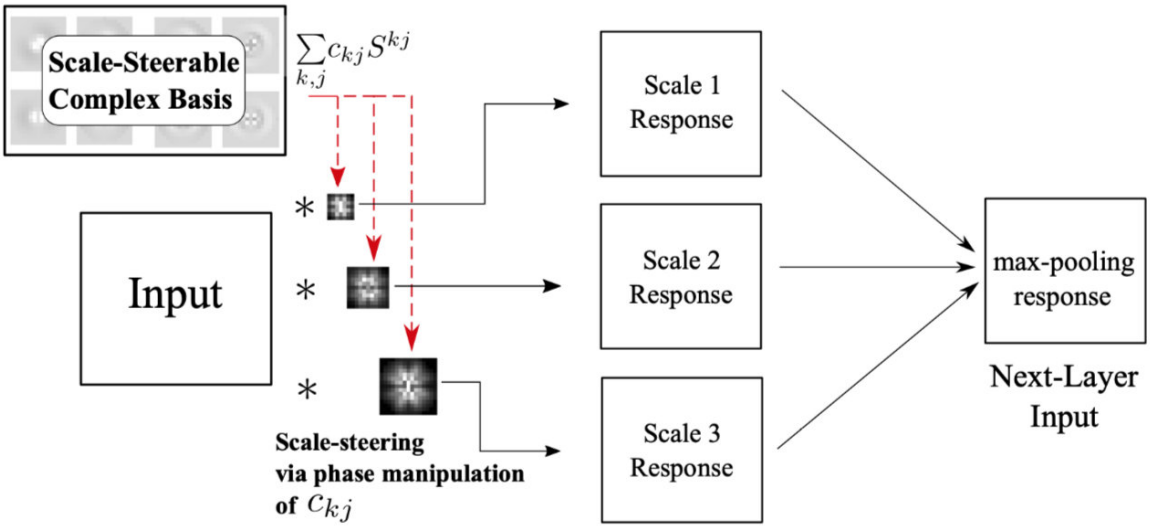


Figure 3: The proposed scale-invariant layer with scale-steered filters [5].

2.4 Anti-aliased Convolutional Neural Networks

Modern convolutional neural networks are not shift-invariant as a small shift in the input images may lead to a totally different output. This is because of the downsampling operations, for example, pooling methods and convolution operation. In [6], Zhang offers a method reconciling antialiasing with existing downsampling methods to improve the shift-invariance of CNNs.

MaxPool→MaxBlurPool The max-pooling of kernel size k and stride s can be divided into two functions sequentially as shown in Figure 4. The input is max-pooled densely with kernel size k, and then the results are subsampled from the immediate feature map with stride s. The max-pooling operation can be written as the following equation:

$$\text{MaxPool}_{k,s} = \text{Subsample}_s \circ \text{Max}_k, \tag{8}$$

where $\circ$ is a composition of two functions.

But actually the max operation in the first step preserves the shift-invariance as it is operated in a dense sliding window way. It is the subsequent subsampling that loses the shift-invariance. Zhang [6] proposes to add an anti-aliasing filter with blur kernel to reduce aliasing. As shown in Figure 5, after the max operation densely applied to the input, the feature map is convolved with a $m \times m$ kernel to low-pass filter the intermediate signal and then subsample from the result. In this way, the shift-equivariance is better preserved. The combination of blurring

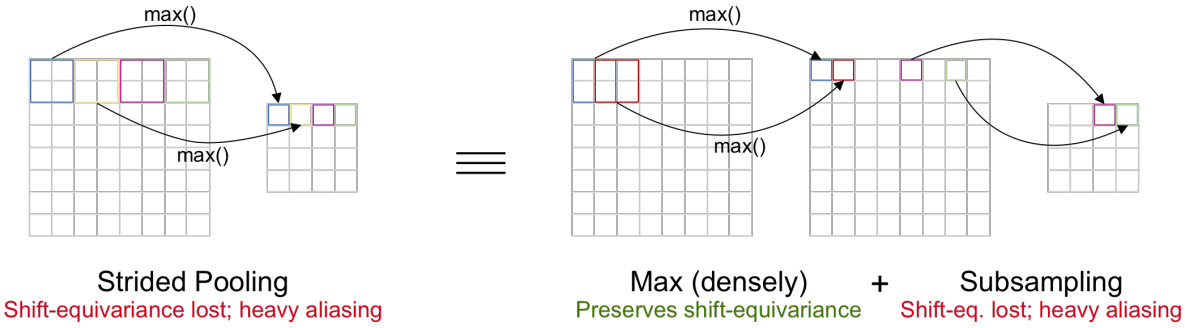


Figure 4: Max-Pooling and its equivariant interpretation [6].

and subsampling is denoted as $\text{BlurPool}_{m,s}$, and the proposed max-pooling method can be written as:

$$\begin{aligned} \text{MaxPool}_{k,s} &\rightarrow \text{Subsample}_s \circ \text{Blur}_m \circ \text{Max}_k \\ &= \text{BlurPool}_{m,s} \circ \text{Max}_k. \end{aligned} \tag{9}$$

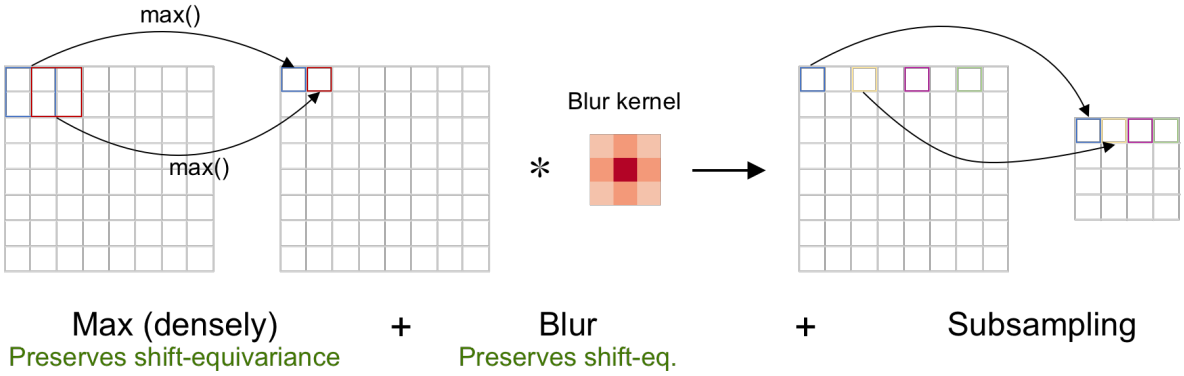


Figure 5: Anti-aliased max-pooling [6].

StridedConv→ConvBlurPool Strided-convolutions suffer from the same problem as max-pooling. Therefore, the same method is used to fix this:

$$\text{Relu} \circ \text{Conv}_{k,s} \rightarrow \text{BlurPool}_{m,s} \circ \text{Relu} \circ \text{Conv}_{k,1}. \tag{10}$$

AveragePool→BlurPool Average-pooling is replaced by BlurPool:

$$\text{AvgPool}_{k,s} \rightarrow \text{BlurPool}_{m,s}. \tag{11}$$

Filter selection The blur kernel can be chosen for different sizes. In [6], Zhang tests the size m from 2 to 5 with increasing smoothing. The filters are produced in two steps. The first step is to calculate the outer product of 3 types of vectors with themselves: Rectangle-2 $[1, 1]$, Triangle-3 $[1, 2, 1]$ and Binomial-5 $[1, 4, 6, 4, 1]$. The second step is normalization. In our

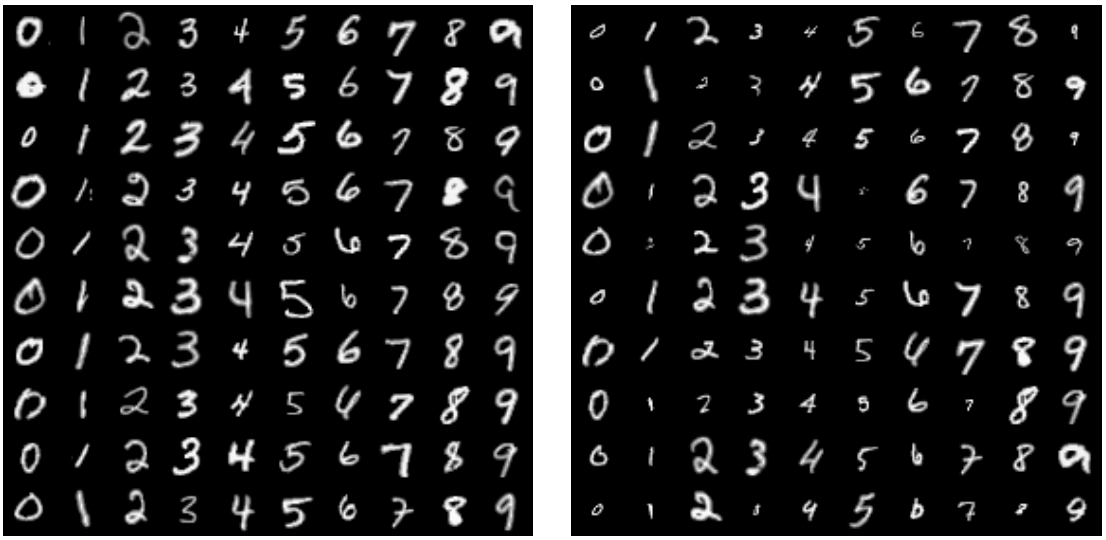
experiment, Triangle-3 vector is selected, and the corresponding blur kernel looks as follows:

$$\begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix}.$$

3 Datasets

3.1 MNIST and MNIST-Scale

MNIST [7] is a handwritten digit classification dataset and has been used as a benchmark to evaluate many classification algorithms. As shown in Figure 6a, all the images are 28×28 gray-scale, with labels in 0-9 corresponding to the digit 0-9. In order to evaluate the scale-invariance of the networks, each image is randomly scaled by a factor $s \in (0.3, 1)$, producing the dataset MNIST-Scale. Figure 6b shows some examples of MNIST-Scale.

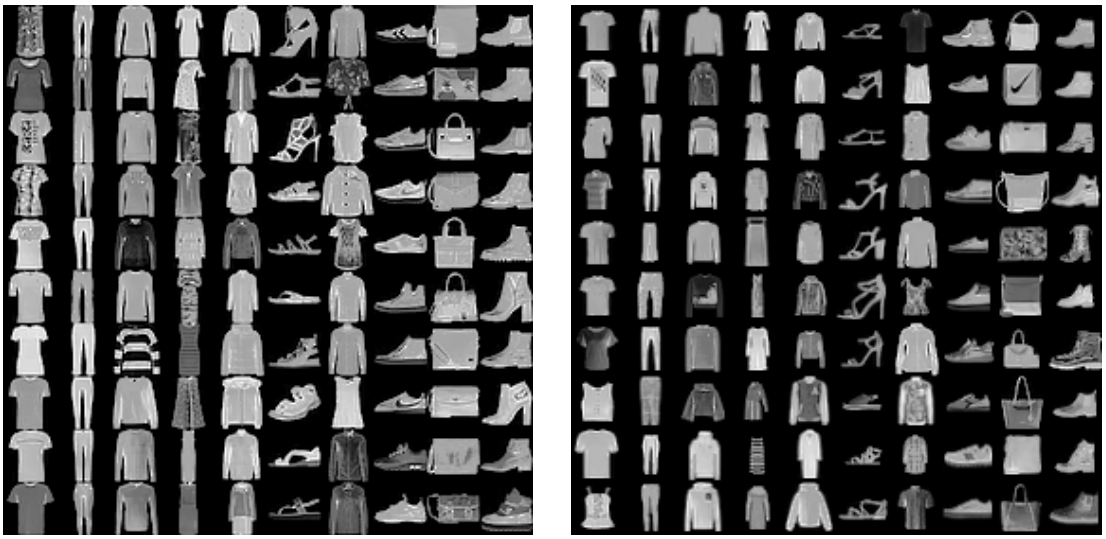


(a) MNIST (b) MNIST-Scale

Figure 6: Datasets: MNIST and MNIST-Scale.

3.2 FMNIST and FMNIST-Scale

Fashion-MNIST [8] (FMNIST) is used as a drop-in replacement of MNIST. It is more complex, and can represent the modern computer vision tasks better than MNIST. As shown in Figure 7a, it consists of 28×28 gray-scale images, and each image is assigned to a label in 0-9, corresponding to the 10 classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot. Similar to MNIST-Scale, the scale transformation with factor $s \in (0.7, 1)$ is applied on FMNIST to create FMNIST-Scale dataset. Figure 7b shows some examples of FMNIST-Scale.

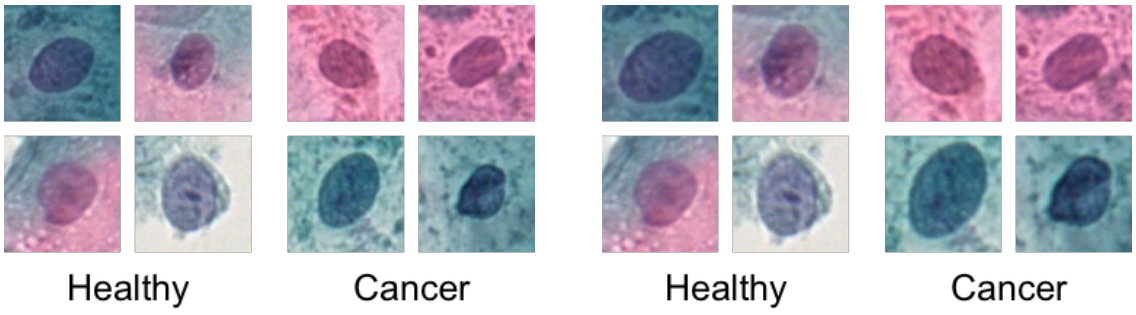


(a) FMNIST (b) FMNIST-Scale

Figure 7: Datasets: FMNIST and FMNIST-Scale.

3.3 OralCancer and OralCancer-Scale

OralCancer [9] contains images of oral cells collected from patients with oral cancer and healthy people. All the images are of size $80 \times 80 \times 3$, with labels of healthy or cancer. Figure 8a shows some examples of each class. Note that all the cells from patients with cancer are labelled as cancer cells. We choose to zoom in the cells to create OralCancer-Scale dataset because the background is not simply black or white, and scaling the cells to smaller size will lead to images with inconsistent background affecting the network training. So, we use a scale factor bigger than 1, which is opposite in MNIST and FMNIST. As shown in Figure 8b, all the images are scaled by a factor $s \in (1, 1.3)$.



(a) OralCancer (b) OralCancer-Scale

Figure 8: Datasets: OralCancer and OralCancer-Scale

4 Experiment

All the experiments are run on an Intel core-i9 server with a GeForce RTX 2080 Ti graphics processing card. Our implementation code for the experiments is available at `https://github.`

com/wsgdrfz/Scale-invariant-CNNs.

4.1 Experiment Design

We find that most of the network evaluations are performed with only the scaled dataset for both training and test [4][5]. With the purpose to replicate their work, we follow the papers and build the networks: Standard-CNN (works as baseline), SI-ConvNet, SS-CNN, and Antialiased-SS-CNN (combination of the anti-aliased CNN with SS-CNN). Consistent to the experiments in [5], all the networks are with the architecture of three convolutional layers and two fully connected layers, and the networks are trained and tested on MNIST-Scale dataset. We also change the training size from 1k to 10k in order to evaluate the scalability of the networks, as we are interested in the relationship between the training size and the performance of the networks. The test size is fixed to 50k. After training for 300 epochs, the average test accuracy over 6 splits with respect to different training size is shown in Figure 9.

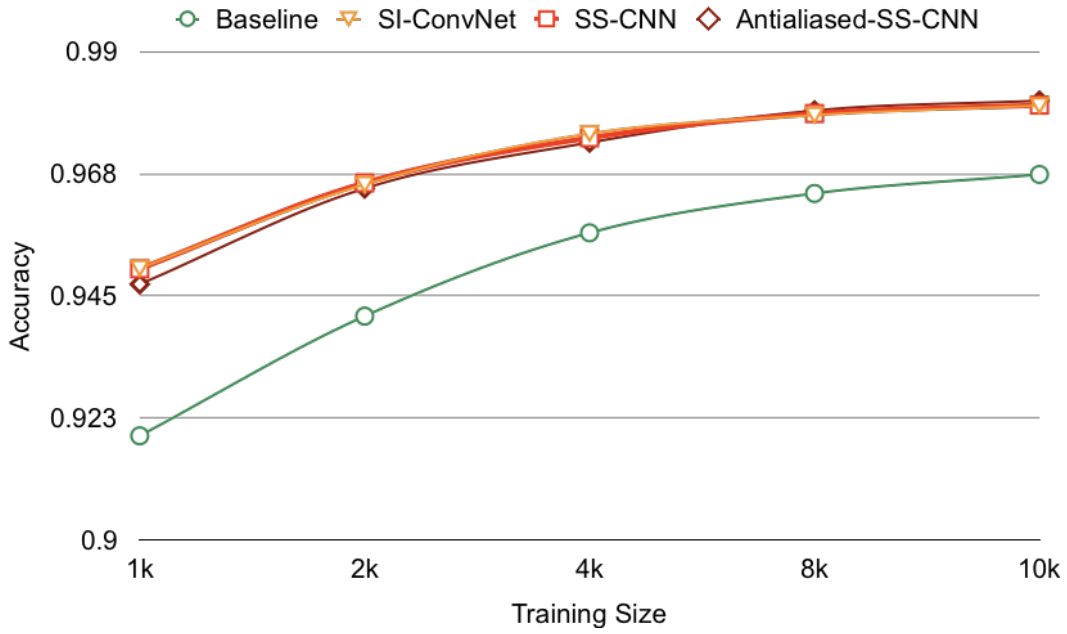


Figure 9: Average test accuracy on MNIST-Scale of different networks.

From Figure 9, all the networks perform quite well with lowest accuracy over 91% and the performance is better when the training size increases. But it is really hard to distinguish the performance of the networks except Baseline. We hypothesize that training on the scaled dataset is affected by data augmentation to a certain degree. In order to deal with variations in the test set which may not be captured in the training set, data augmentation adapts method of increasing the training data by generating data expected to be in the test set, whereas the observed approach here is to equip the network architecture to compensate for that potential deficiency in the training data. We are interested in whether the scale-invariant networks can solve more realistic tasks by their designed properties, rather than by the ideal training set. Thus, we decide to redesign the experiment.

We believe that it is possible to better evaluate the scale-invariance abilities of the networks if they are trained on the original dataset and tested on the unseen scaled dataset. In this case the classification tasks become more difficult and the variance of performance between the networks will be more obvious. Therefore, we have the following experiment settings:

- For MNIST and FMNIST, we have four networks tested on the scaled datasets for comparison. Standard-CNN as Baseline1, SI-ConvNet, SS-CNN, and standard-CNN with data augmentation as Baseline2. Except the convolution layers are replaced by corresponding scale-invariant layers, all the networks share the same architecture and hyper-parameters. The selected architecture consists of three convolutional layers with the number of feature maps of 30,60,90 and kernel size of 11×11, and each convolutional layer is followed by max-pooling, ReLu and batch normalization. The subsequent layers are two fully connected layers, one dropout layer and one soft-max logistic regression layer. Except Baseline2, the other three networks are trained on the original MNIST and FMNIST, with training size from 2k to 10k. Baseline2 adds the same amount of scaled dataset, MNIST-Scale and FMNIST-Scale, into the training set, leading to the double training size. All the networks are trained for 30 epochs and then tested on 10k scaled data. The process is repeated over 6 splits and the training time, training accuracy and test accuracy are recorded.

- For OralCancer, the networks settings are similar but more complex. Here we have six networks: standard-CNN as Baseline1, standard-CNN with data augmentation as Baseline2, SI-ConvNet, SS-CNN, SI-ConvNet combined with anti-aliased CNN as Antialiased-SIConvNet, SS-CNN combined with anti-aliased CNN as Antialiased-SSCNN. Analogous to MNIST & FMNIST, the architecture consists of five convolutional layers with the number of feature maps of 30,60,90,120,150 and kernel size of 11×11. Others remains the same as in MNIST & FMNIST. Except Baseline2, the training sets of the other five networks are randomly selected from the original OralCancer training set, with training size from 10k to 50k. For Baseline2, the same amount of scaled images from the training set of OralCancer-Scale are added to the training data, leading to the training size from 20k to 100k. All the networks are trained for 10 epochs and then tested on 10k data from the test set of OralCancer-Scale. In order to reduce anomalous results, the process is repeated for 6 splits with different random seed for the generation of training and test data. The training time, training accuracy and test accuracy are recorded.

4.2 Experimental Results

Here we only analyze the average results over six splits and the detailed tables with results can be found in Section 6.

4.2.1 MNIST

As shown in Figure 10, it is obvious that Baseline2 has the highest accuracy rate of about 95%, and it performs much better than others. The results of SI-ConvNet and SS-CNN are basically the same. Baseline1 performs the worst. Also, with the large increase in training size, accuracy does not change significantly for all the four networks.

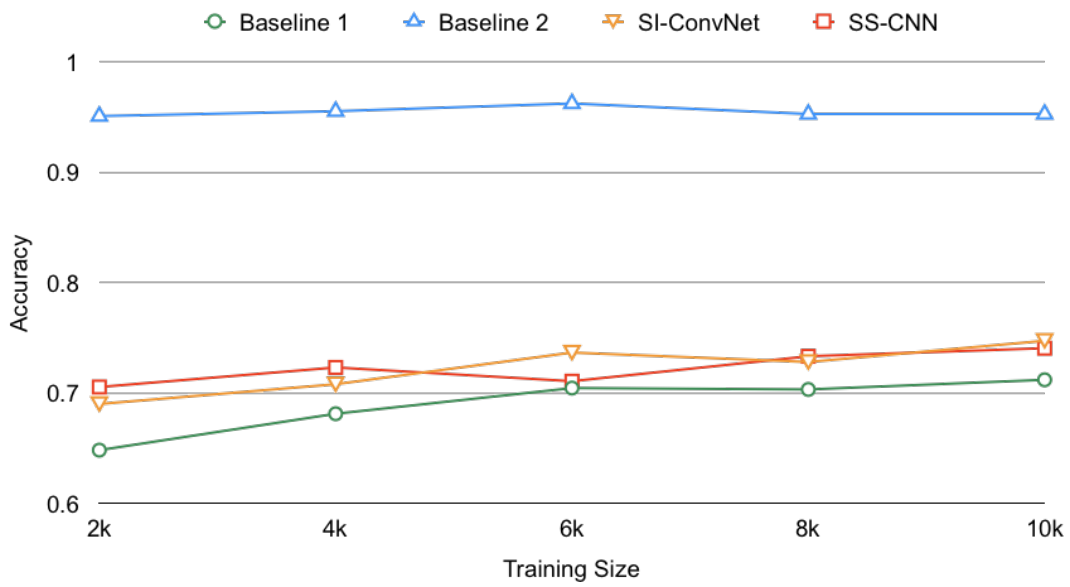


Figure 10: Results: Average test accuracy on MNIST-Scale over 6 splits. Note that the training size of Baseline2 is as twice as others.

4.2.2 FMNIST

From Figure 11 we see that the results of Baseline2 are still the best for FMNIST with the accuracy around 85%. It does not increase with the increase in training size. The gap between Baseline2 and other networks is narrowed compared with MNIST. The second, third and fourth places are SI-ConvNet, Baseline1 and SS-CNN. Among them, the accuracies of SI-ConvNet and Baseline1 grow a lot when training size increases from 8k to 10k. In contrast, SS-CNN performs the worse, and it even decreases a lot when training size increases from 8k to 10k. We assume this is because SS-CNN does not converge within 30 epochs. In Table 8, the training accuracy of SS-CNN is only around 0.85, while that of others are bigger than 95%. To validate our hypothesis, we add one more experiment with 50 epochs for SS-CNN.

As shown in Table 9, the training accuracy of SS-CNN with 50 epochs increases to the same level to that of other networks with 30 epochs, but the test accuracy is not as good as expected. From Figure 12, the performance of SS-CNN with 50 epochs is still slightly lower than that of SI-ConvNet with 30 epochs, which means SS-CNN does not show superiority even at the sacrifice of more epochs and time.

4.2.3 Oral Cancer

As shown in Figure 13, the performance of the six networks is almost identical. It's worth noting that the range of y-axis is from 0.56 to 0.62, which is not a large range. Anomalous results appear more often to Baseline1 and Antialiased-SIConvNet. From Table 10, the training accuracy of Baseline1 is only around 30% on split 3 and split 5 with training size of 30k, leading to the test accuracy lower than 50% for the two splits. Similar anomaly is also observed for Antialiased-SIConvNet with training size of 10k on split 1 and split 5, as shown in Table 13. The anomalous results pull down the average performance of the two networks

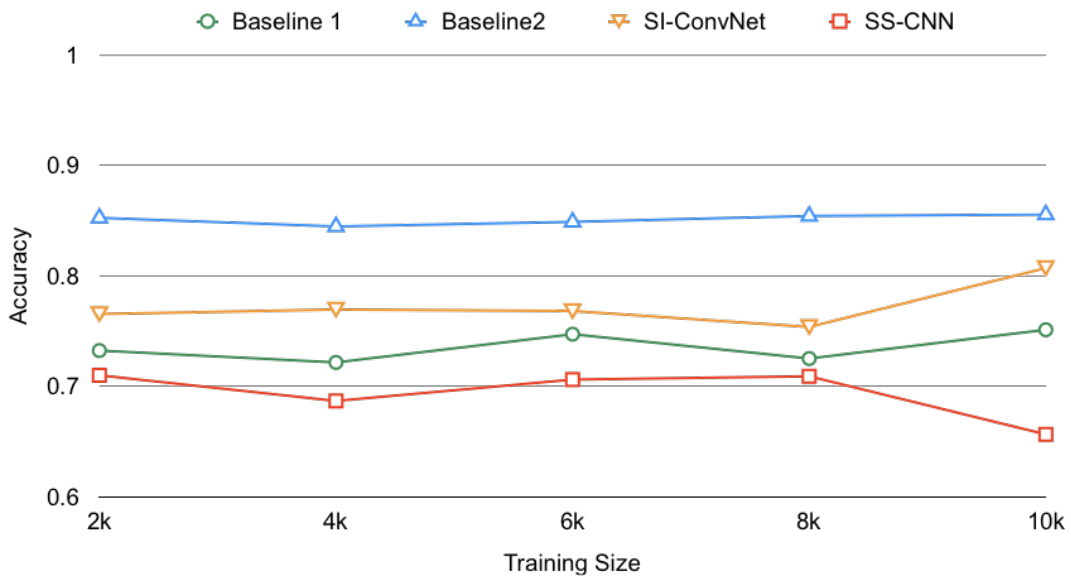


Figure 11: Results: Average test accuracy on FMNIST-Scale over 6 splits. Note that the training size of Baseline2 is as twice as others.

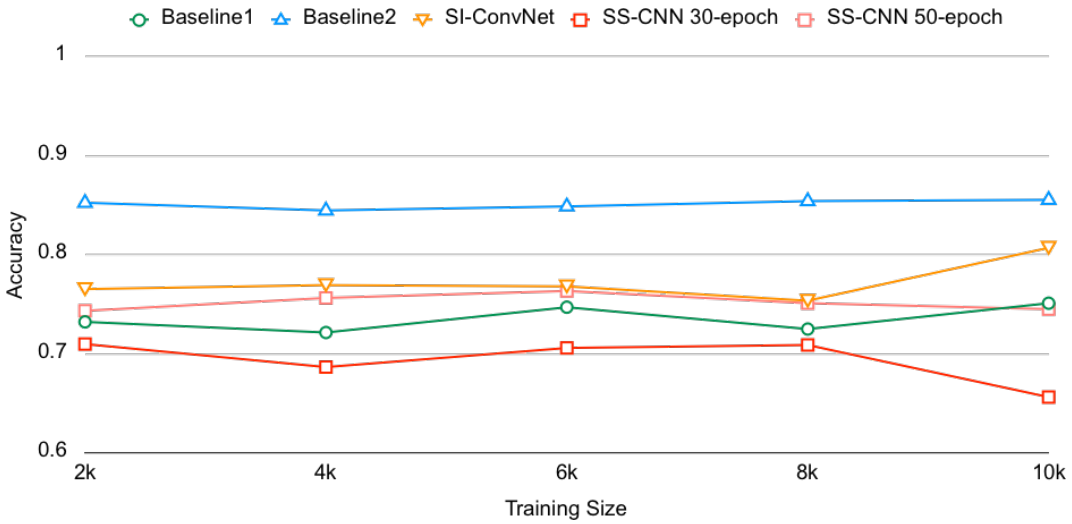


Figure 12: Result: Average test accuracy on FMNIST-Scale over 6 splits. Note that the training size of Baseline2 is as twice as others.

and cause the instability.

From Figure 13 and Figure 14, it can be seen that adding BlurPool to the network does not increase the training time nor improve the accuracy, but may cause instability. We assume that the reason of the not obvious improvement is because the number of downsampling operations in the networks are small, with only three max-pooling operations. Even in [6], the increase for ResNet18 in accuracy after adding BlurPool is only about 2%. Therefore, BlurPool is not recommended here for small networks. For the same training size, the time that SS-CNN requires is one and a half times of that SI-ConvNet requires, but the accuracy is

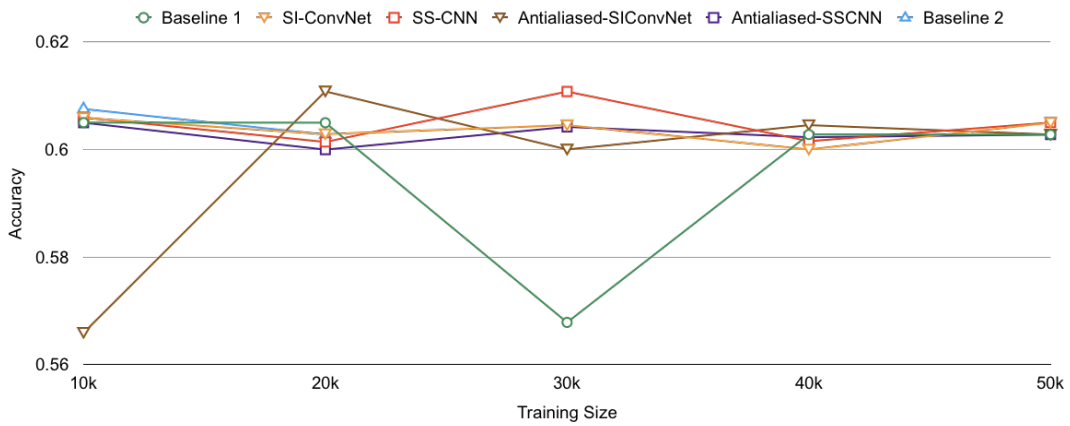


Figure 13: Results: Average test accuracy on OralCancer over 6 splits. Note that the training size of Baseline2 is as twice as others. Baseline2 only have data for 10k (20k) and 20k (40k) here due to limited time.

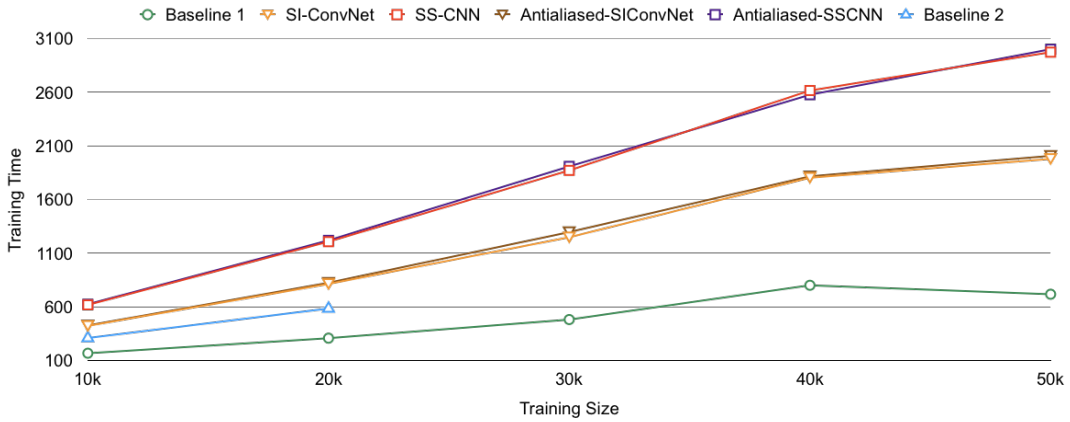


Figure 14: Results: Average training time on OralCancer over 6 splits. Note that the training size of Baseline2 is as twice as others. Baseline2 only have data for 10k (20k) and 20k (40k) here due to limited time.

basically the same, so SS-CNN is more time-consuming for this task.

5 Discussion

Input Scaling vs Filter Scaling (SI-ConvNet vs SS-CNN) SI-ConvNet uses the most direct and easiest method, that is to scale the input both smaller and larger compared with the original size. After resizing feature maps to the same size, the outputs are the maximum response at each location. This method is effective and very suitable for our experiments because the training set are images with the same scaling while test set are images with different scaling. But this method reshapes the images or feature maps twice, and it is easy to produce interpolation artifacts. SS-CNN only steers the filters to different sizes without any interpolation operations. But the interpolation artifacts only effect little on the test accuracy considering the difference between performance of the two networks is little.

Data Augmentation vs Input Scaling (Baseline2 vs SI-ConvNet)) The data augmentation technique used for our experiment is to scale each image in training set with a random factor (each dataset has a specific scale range). From the results, the pattern learned by a standard CNN with data augmentation is closer to the test set than the pattern learned by SI-ConvNet through input scaling. But the gap narrows when the tasks become more complicated (from MNIST to FMNIST). Scale-invariant networks are promising to catch up or even exceed data augmentation in practical classification problems.

Shift-invariance In [6], the BlurPool is very simple to use and it improves the accuracy and generalization of different testbeds, for example, VGG[11], ResNet[12] and DenseNet[13]. But in our network, there are only three downsampling operations, that is three max-pooling layers. Thus, the network itself does not lose much feature of shift-invariance, and in consequence the improvement here is not obvious.

6 Conclusion and Future Work

In this study, we implement and compare the performance of different approaches that allow scale-invariant feature learning and representation in convolutional neural networks. Our baselines are standard CNNs and standard CNNs with data augmentation. It's worth noting that our training set is the images at the same scaling while test set are designed to have different scales which is different to the original papers, in order to better evaluate the scale-invariance ability of the networks. And our results are quite different too. SI-ConvNet [4] is the method that is comprehensive and simple to implement, and it captures the features at different scales through multiple scaled inputs. It is also the best networks except Baseline2 using data augmentation method in our experiment. The scale-steerable filters of SS-CNN [5] help CNNs to have a higher degree of transformative weight sharing and this makes SS-CNN a very promising network for difficult tasks although it need more epochs or time to converge. We also try to integrate low-pass filtering to anti-alias here for SI-ConvNet and SS-CNN to add the ability of shift-equivariance. Unfortunately, we observe no surprising boost in accuracy but decrease in robustness which is quite different from the original paper. We assume that is because all the networks only containing three downsampling layers (max-pooling) therefore the shift-invariance is not lost too much.

Overall, data augmentation in training dataset is the most effective way. With limited GPUs and time, SI-ConvNet would be a good choice. Interesting directions for future work include more detailed experiments to find how the networks work. For example, visualize the intermediate feature maps in different layer of network to check the multi-scale features. Finding the solution for filling the background of scale-inword image can also be an option. Now for MNIST and FMNIST we assume the background is black and we didn't scale inward images of OralCancer dataset. Besides, training with early stopping can be one solution for the non-monotonic performance of the networks with increase in the training size. The same number of epochs for different training size in this experiment probably affects the results by under fitting at small training size and over fitting at large training size.

References

[1] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, June 2009.

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

[3] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv preprint arXiv:1811.00982*, 2018.

[4] Kanazawa Angjoo and Sharma Abhishek. Locally scale-invariant convolutional neural networks. *arXiv preprint arXiv:1412.5104*, 2014.

[5] Rohan Ghosh and Anupam K Gupta. Scale steerable filters for locally scale-invariant convolutional neural networks. *arXiv preprint arXiv:1906.03861*, 2019.

[6] Richard Zhang. Making convolutional networks shift-invariant again. In *International Conference on Machine Learning (ICML)*, 2019.

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[8] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[9] Jiahao Lu, Nataša Sladoje, Christina Runow Stark, Eva Darai Ramqvist, Jan-Michaél Hirsch, and Joakim Lindblad. A deep learning based pipeline for efficient oral cancer screening on whole slide images. *arXiv preprint arXiv:1910.10549*, 2019.

[10] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Harmonic networks: Deep translation and rotation equivariance. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5028–5037, 2017.

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

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[13] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Appendix

Table 1: MNIST-Scale: Standard-CNN(Baseline 1), epochs=30

		2k	4k	6k	8k	10k
Split 0	training time	95.142	109.588	124.559	138.675	152.637
	training	0.9945	1.0	1.0	0.99925	0.9996
	test	0.4006	0.6255	0.705	0.6851	0.7113
Split 1	training time	95.255	109.436	123.857	138.399	152.035
	training	1.0	0.9995	0.999667	1.0	0.9994
	test	0.6615	0.6987	0.7155	0.7234	0.7023
Split 2	training time	95.738	109.172	123.518	138.705	151.749
	training	1.0	1.0	1.0	0.9995	0.9995
	test	0.6792	0.6614	0.705	0.7223	0.7121
Split 3	training time	94.812	109.494	123.751	138.404	153.106
	training	1.0	0.999	1.0	0.999	0.9998
	test	0.6652	0.687	0.7022	0.6989	0.7254
Split 4	training time	94.984	109.405	123.694	138.461	152.209
	training	1.0	0.9985	1.0	1.0	0.9996
	test	0.6422	0.6976	0.6555	0.7075	0.7231
Split 5	training time	95.050	109.420	124.183	138.303	152.417
	training	1.0	1.0	1.0	1.0	0.9987
	test	0.625	0.6798	0.7067	0.6754	0.6928
Average	training time	95.10775	109.43875	123.87125	138.48475	152.3245
	training	1	0.999625	1	0.999688	0.999525
	test	**0.648475**	**0.68145**	**0.704725**	**0.70345**	**0.7122**

Table 2: MNIST-Scale: Standard-CNN(Baseline 2), epochs=30

		4k	8k	12k	16k	20k
Split 0	training time	149.294	178.460	211.155	239.864	275.006
	training	0.99975	0.998875	0.99325	0.973313	0.9829
	test	0.9503	0.9641	0.9637	0.9503	0.9592
Split 1	training time	158.688	177.428	211.508	245.362	271.241
	training	1.0	0.99875	0.989167	0.985	0.9867
	test	0.9625	0.9675	0.9614	0.9625	0.9692
Split 2	training time	150.007	179.102	210.861	240.160	268.200
	training	0.99725	0.994375	0.991917	0.9715	0.96635
	test	0.9453	0.9527	0.9609	0.9509	0.9285
Split 3	training time	150.693	178.399	215.638	239.349	271.662
	training	1.0	0.98975	0.993583	0.980875	0.98255
	test	0.9561	0.9438	0.9642	0.9563	0.9588
Split 4	training time	148.911	177.685	210.593	240.320	268.302
	training	0.998	0.99725	0.988583	0.983	0.97115
	test	0.9527	0.9615	0.9569	0.9545	0.9452
Split 5	training time	149.624	178.294	209.587	239.228	271.482
	training	0.997	0.9895	0.99125	0.967375	0.97425
	test	0.9372	0.9426	0.9647	0.936	0.9488
Average	training time	149.9045	178.2095	211.02925	239.92325	270.67175
	training	0.99875	0.995031	0.991396	0.977172	0.977713
	test	**0.9511**	**0.955525**	**0.96255**	**0.953**	**0.953**

Table 3: MNIST-Scale: SI-ConvNet, epochs=30

		2k	4k	6k	8k	10k
Split 0	training time	426.193	735.863	1040.916	1346.358	1633.629
	training	0.923	1.0	1.0	1.0	0.9992
	test	0.4562	0.6921	0.7456	0.7021	0.7564
Split 1	training time	428.292	736.730	1038.050	1342.193	1647.501
	training	1.0	1.0	0.999167	0.999875	0.9996
	test	0.7152	0.7111	0.6857	0.7221	0.7429
Split 2	training time	426.272	733.781	1037.405	1341.546	1645.858
	training	1.0	1.0	1.0	0.9995	0.9998
	test	0.699	0.7441	0.7426	0.7425	0.7589
Split 3	training time	430.315	735.477	1037.850	1343.879	1648.142
	training	1.0	1.0	0.999667	0.999375	0.9996
	test	0.679	0.6681	0.7703	0.7282	0.7732
Split 4	training time	424.821	734.845	1036.647	1343.466	1646.699
	training	1.0	0.99975	0.999833	1.0	0.9988
	test	0.6944	0.7151	0.7317	0.7205	0.7319
Split 5	training time	430.252	731.479	1042.514	1343.013	1647.869
	training	1.0	0.99975	1.0	0.999875	0.9987
	test	0.6888	0.7145	0.7276	0.7609	0.7229
Average	training time	427.75225	734.9915	1038.55525	1343.13775	1646.98175
	training	1	0.999938	0.999875	0.999813	0.9993
	test	**0.6903**	**0.7082**	**0.736875**	**0.728325**	**0.747525**

Table 4: MNIST-Scale: SS-CNN, epochs=30

		2k	4k	6k	8k	10k
Split 0	training time	226.164	340.792	441.312	541.922	651.177
	training	0.933	0.9995	0.999167	0.99875	0.9955
	test	0.4786	0.7227	0.7349	0.7449	0.7571
Split 1	training time	229.124	334.070	441.907	541.421	647.883
	training	1.0	0.9995	0.995	0.9975	0.9975
	test	0.72	0.708	0.7124	0.7119	0.7134
Split 2	training time	226.252	333.353	443.156	545.089	649.657
	training	0.998	0.99975	0.9995	0.997375	0.9964
	test	0.7124	0.7485	0.7209	0.6961	0.7556
Split 3	training time	226.124	331.673	442.819	548.447	648.391
	training	1.0	0.99925	0.9985	0.9985	0.9853
	test	0.6537	0.7404	0.7064	0.7535	0.6608
Split 4	training time	226.269	332.111	442.841	544.210	646.957
	training	1.0	0.999	0.9975	0.996375	0.9977
	test	0.7432	0.7225	0.6872	0.7396	0.7455
Split 5	training time	226.350	337.715	438.834	544.038	650.369
	training	1.0	1.0	0.999167	0.997625	0.9947
	test	0.7366	0.6923	0.704	0.7377	0.7487
Average	training time	226.25875	334.31225	442.21975	543.81475	649.075
	training	0.9995	0.9995	0.998583	0.99775	0.996025
	test	**0.705675**	**0.7234**	**0.710925**	**0.733525**	**0.7408**

Table 5: FMNIST-Scale: Standard-CNN(Baseline 1), epochs=30

		2k	4k	6k	8k	10k
Split 0	training time	268.162	284.919	318.993	344.008	362.756
	training	0.9865	0.98375	0.9805	0.98	0.96
	test	0.7034	0.7152	0.7462	0.7282	0.7547
Split 1	training time	269.875	280.317	316.986	346.138	371.632
	training	1.0	0.98575	0.991167	0.956625	0.9617
	test	0.7618	0.761	0.7872	0.7118	0.7562
Split 2	training time	271.532	285.401	323.993	344.820	360.649
	training	0.993	0.998	0.979333	0.9755	0.9703
	test	0.6922	0.7568	0.706	0.7502	0.7631
Split 3	training time	276.179	283.555	321.246	339.873	357.763
	training	0.996	0.9325	0.995833	0.980125	0.9632
	test	0.7102	0.5825	0.7386	0.759	0.7159
Split 4	training time	271.148	285.693	314.884	338.797	359.458
	training	0.9985	0.99575	0.9735	0.964375	0.9773
	test	0.7639	0.7739	0.712	0.7176	0.7473
Split 5	training time	274.472	289.395	317.975	334.714	364.723
	training	1.0	0.998	0.983667	0.961125	0.9618
	test	0.7616	0.7393	0.792	0.683	0.769
Average	training time	271.894667	284.88	319.012833	341.391667	362.830167
	training	0.995667	0.982292	0.984	0.969625	0.965717
	test	**0.732183**	**0.72145**	**0.747**	**0.724967**	**0.751033**

Table 6: FMNIST-Scale: Standard-CNN(Baseline 2), epochs=30

		4k	8k	12k	16k	20k
Split 0	training time	155.697	178.567	211.100	245.327	272.472
	training	0.9795	0.94875	0.930417	0.887375	0.9043
	test	0.8651	0.8598	0.8773	0.8441	0.8806
Split 1	training time	150.329	177.895	209.836	240.294	271.467
	training	0.9765	0.914875	0.907167	0.8981875	0.87045
	test	0.8454	0.834	0.8507	0.8654	0.846
Split 2	training time	149.630	178.233	216.988	240.262	268.552
	training	0.98625	0.929375	0.928917	0.9034375	0.90415
	test	0.8504	0.8447	0.86	0.8556	0.8614
Split 3	training time	149.592	176.941	210.264	239.126	271.387
	training	0.9945	0.917625	0.890583	0.882875	0.865
	test	0.8581	0.8351	0.8389	0.8384	0.8453
Split 4	training time	149.007	178.619	210.024	239.593	268.566
	training	0.97825	0.94575	0.888333	0.917375	0.88315
	test	0.8556	0.843	0.8205	0.8706	0.8547
Split 5	training time	149.788	178.719	209.070	239.636	270.784
	training	0.98825	0.953375	0.915417	0.902375	0.89485
	test	0.8373	0.8556	0.8453	0.8511	0.8593
Average	training time	149.83475	178.3285	210.306	239.94625	270.551
	training	0.983063	0.935375	0.910521	0.897844	0.88815
	test	**0.852375**	**0.8446**	**0.848725**	**0.85405**	**0.85535**

Table 7: FMNIST-Scale: SI-ConvNet, epochs=30

		2k	4k	6k	8k	10k
Split 0	training time	368.074	487.033	628.986	756.271	1029.080
	training	0.943	0.9825	0.968833	0.958375	0.9282
	test	0.6948	0.7816	0.7825	0.7505	0.8063
Split 1	training time	363.155	488.651	627.083	754.194	1013.374
	training	0.97	0.9785	0.981667	0.93925	0.9451
	test	0.7686	0.7615	0.7752	0.76	0.9451
Split 2	training time	369.243	488.132	636.018	750.251	880.595
	training	1.0	0.99	0.974	0.9635	0.9027
	test	0.7852	0.7454	0.7832	0.749	0.7662
Split 3	training time	371.423	488.928	628.737	750.411	876.455
	training	0.999	0.989	0.97	0.97225	0.9456
	test	0.7759	0.7755	0.7353	0.7541	0.7538
Split 4	training time	362.387	486.089	623.922	752.269	879.170
	training	1.0	0.9905	0.9815	0.942375	0.9471
	test	0.7843	0.7718	0.7756	0.7112	0.7978
Split 5	training time	369.081	485.961	611.296	753.668	872.336
	training	0.998	0.9975	0.993833	0.98375	0.9674
	test	0.7828	0.78	0.7562	0.7965	0.7725
Average	training time	367.227167	487.465667	626.007	752.844	925.168333
	training	0.985	0.988	0.9783056	0.959917	0.93935
	test	**0.765267**	**0.7693**	**0.768**	**0.75355**	**0.80695**

Table 8: FMNIST-Scale: SS-CNN, epochs=30

		2k	4k	6k	8k	10k
Split 0	training time	365.390	473.281	611.236	715.541	826.784
	training	0.7355	0.904	0.867833	0.878875	0.8377
	test	0.6617	0.7667	0.687	0.6988	0.6496
Split 1	training time	373.874	476.682	608.118	714.985	886.549
	training	0.767	0.884	0.843	0.894625	0.8245
	test	0.6346	0.7461	0.6919	0.7127	0.671
Split 2	training time	370.705	476.473	595.429	716.143	821.481
	training	0.928	0.81525	0.896333	0.898125	0.7708
	test	0.745	0.5821	0.7775	0.7367	0.593
Split 3	training time	356.427	472.439	599.584	714.149	821.911
	training	0.926	0.825	0.82	0.884625	0.8103
	test	0.7414	0.591	0.6524	0.7238	0.6066
Split 4	training time	365.223	474.715	594.581	717.827	855.627
	training	0.8685	0.90325	0.862167	0.868125	0.8591
	test	0.7586	0.7171	0.6786	0.7211	0.6909
Split 5	training time	371.412	474.278	587.673	717.328	835.653
	training	0.9255	0.88875	0.8255	0.867875	0.8893
	test	0.7175	0.7165	0.748	0.6602	0.7267
Average	training time	367.171833	474.644667	599.436833	715.9955	841.334167
	training	0.858417	0.870042	0.852472	0.882042	0.83195
	test	**0.7098**	**0.686583**	**0.7059**	**0.708883**	**0.6563**

Table 9: FMNIST-Scale: SS-CNN, epochs=50

		2k	4k	6k	8k	10k
Split 0	training time	310.670	484.474	659.103	834.024	1005.363
	training	0.928	0.987	0.968333	0.917875	0.9562
	test	0.6988	0.7949	0.7694	0.7081	0.7418
Split 1	training time	310.959	484.401	659.517	832.840	1007.711
	training	0.9845	0.98	0.919667	0.94475	0.9336
	test	0.7776	0.764	0.7818	0.7797	0.7421
Split 2	training time	310.993	484.581	658.927	833.209	1006.552
	training	0.993	0.98825	0.953333	0.939125	0.9252
	test	0.7311	0.7435	0.7498	0.7521	0.7107
Split 3	training time	310.741	484.881	658.986	832.710	1006.849
	training	0.9985	0.97925	0.914667	0.933875	0.9347
	test	0.7548	0.7648	0.7467	0.7765	0.7765
Split 4	training time	310.870	484.781	658.736	834.008	1007.109
	training	0.9865	0.9855	0.937833	0.945625	0.8703
	test	0.7402	0.7654	0.7546	0.7794	0.7313
Split 5	training time	311.352	484.621	659.125	833.223	1006.846
	training	0.9975	0.9885	0.952833	0.955875	0.9415
	test	0.7579	0.7055	0.7779	0.7106	0.7679
Average	training time	310.930833	484.623167	659.065667	833.335667	1006.738333
	training	0.981333	0.98475	0.941111	0.939521	0.926917
	test	**0.7434**	**0.75635**	**0.763367**	**0.751067**	**0.74505**

Table 10: OralCancer-Scale: Standard-CNN(Baseline 1), epochs=10

		10k	20k	30k	40k	50k
Split 0	training time	190.191	305.882	446.300	1007.052	713.877
	training	0.7215	0.71595	0.715133	0.7142	0.71504
	test	0.607	0.607	0.607	0.607	0.607
Split 1	training time	169.607	310.728	449.570	792.370	721.895
	training	0.7137	0.73245	0.711567	0.711675	0.71224
	test	0.6008	0.611	0.6008	0.6008	0.6008
Split 2	training time	167.045	307.264	447.834	797.274	714.525
	training	0.707	0.7097	0.711067	0.71125	0.71246
	test	0.6007	0.6007	0.6007	0.6007	0.6007
Split 3	training time	169.632	308.008	449.801	811.174	720.330
	training	0.7556	0.7098	0.289933	0.7119	0.71314
	test	0.6406	0.5959	0.4079	0.5959	0.5959
Split 4	training time	167.879	311.029	583.129	807.047	718.025
	training	0.7058	0.7091	0.711	0.71335	0.7147
	test	0.6026	0.6026	0.6026	0.6026	0.6026
Split 5	training time	167.928	310.131	816.513	587.976	721.111
	training	0.7147	0.71435	0.3113	0.7159	0.71632
	test	0.6096	0.6096	0.4673	0.6096	0.6096
Average	training time	168.7615	309.03275	482.5835	801.96625	718.49775
	training	0.714225	0.71245	0.611233	0.712781	0.713835
	test	**0.605**	**0.604975**	**0.56785**	**0.602775**	**0.602775**

Table 11: OralCancer-Scale: SI-ConvNet, epochs=10

		10k	20k	30k	40k	50k
Split 0	training time	441.137	813.011	1206.097	1960.244	1978.224
	training	0.7215	0.71595	0.715133	0.7142	0.71504
	test	0.607	0.607	0.607	0.607	0.607
Split 1	training time	423.699	816.390	1211.750	1796.212	1988.791
	training	0.7318	0.7131	0.711567	0.711675	0.49086
	test	0.6131	0.6008	0.6008	0.6008	0.6757
Split 2	training time	423.913	811.499	1211.583	1797.645	1972.464
	training	0.707	0.7097	0.711067	0.71125	0.71246
	test	0.6007	0.6007	0.6007	0.6007	0.6007
Split 3	training time	424.244	816.884	1205.091	1817.141	1989.915
	training	0.7162	0.7098	0.710633	0.7119	0.71314
	test	0.5978	0.5959	0.5959	0.5959	0.5959
Split 4	training time	421.597	812.453	1368.251	1805.582	1974.030
	training	0.7058	0.7091	0.4888	0.71335	0.7147
	test	0.6026	0.6029	0.682	0.6026	0.6026
Split 5	training time	422.764	816.482	1535.258	1600.925	1977.998
	training	0.7188	0.71435	0.715267	0.381825	0.71632
	test	0.6136	0.6096	0.6096	0.5931	0.6096
Average	training time	423.655	814.584	1249.42025	1804.145	1979.76075
	training	0.715875	0.711738	0.7121	0.712044	0.713835
	test	**0.60585**	**0.60285**	**0.604525**	**0.6**	**0.604975**

Table 12: OralCancer-Scale: SS-CNN, epochs=10

		10k	20k	30k	40k	50k
Split 0	training time	629.207	1206.823	1797.130	2620.347	2967.679
	training	0.7215	0.28405	0.715133	0.7142	0.71504
	test	0.607	0.393	0.607	0.607	0.607
Split 1	training time	618.981	1208.119	1799.127	2611.689	2972.679
	training	0.7285	0.7131	0.711567	0.72415	0.71224
	test	0.6097	0.6008	0.6008	0.6087	0.6008
Split 2	training time	620.018	1207.594	1802.444	2613.277	2972.914
	training	0.7077	0.7097	0.748467	0.71125	0.58218
	test	0.6046	0.6007	0.6327	0.6007	0.6511
Split 3	training time	619.387	1212.843	1800.283	2631.140	2980.723
	training	0.7162	0.7144	0.710633	0.7119	0.42352
	test	0.5959	0.6015	0.5959	0.5959	0.3042
Split 4	training time	619.421	1207.923	2087.730	2622.263	2973.650
	training	0.7058	0.7091	0.711	0.71335	0.7147
	test	0.6026	0.6026	0.6026	0.6026	0.6026
Split 5	training time	618.460	1210.100	2182.020	2393.361	2977.325
	training	0.7147	0.71435	0.755433	0.344725	0.71632
	test	0.6096	0.6096	0.6438	0.5279	0.6096
Average	training time	619.45175	1208.434	1872.396	2616.894	2974.142
	training	0.715025	0.711563	0.721542	0.712675	0.68104
	test	**0.60595**	**0.6014**	**0.610775**	**0.60155**	**0.605**

Table 13: OralCancer-Scale: Antialiased-SIConvNet, epochs=10

		10k	20k	30k	40k	50k
Split 0	training time	426.295	822.862	1219.535	1822.042	1994.075
	training	0.7295	0.7472	0.715133	0.7142	0.71504
	test	0.6141	0.6391	0.607	0.607	0.607
Split 1	training time	428.116	829.092	1224.414	1819.404	2012.590
	training	0.287	0.7131	0.711567	0.711675	0.71224
	test	0.4014	0.6008	0.6008	0.6008	0.6008
Split 2	training time	427.845	825.258	1224.863	1819.374	2000.565
	training	0.707	0.7097	0.711067	0.71125	0.71246
	test	0.6007	0.6007	0.6007	0.6007	0.6007
Split 3	training time	427.671	825.501	1221.805	1830.955	2019.690
	training	0.7162	0.7098	0.710633	0.7119	0.71314
	test	0.5959	0.5959	0.5959	0.5959	0.5959
Split 4	training time	427.079	822.911	1519.959	1806.814	2005.133
	training	0.7058	0.7091	0.711	0.73645	0.7147
	test	0.6026	0.6026	0.6026	0.6205	0.6026
Split 5	training time	427.564	827.989	1566.681	1616.256	2014.166
	training	0.3136	0.7542	0.6873	0.7159	0.71632
	test	0.4645	0.6451	0.5421	0.6096	0.6096
Average	training time	427.53975	825.41475	1297.76025	1816.9085	2008.1135
	training	0.61065	0.71995	0.711067	0.713419	0.713835
	test	**0.565925**	**0.6108**	**0.6**	**0.604525**	**0.602775**

Table 14: OralCancer-Scale: Antialiased-SSCNN, epochs=10

		10k	20k	30k	40k	50k
Split 0	training time	624.381	1219.556	1811.700	2624.714	2988.673
	training	0.7215	0.71595	0.767767	0.7142	0.71504
	test	0.607	0.607	0.6194	0.607	0.607
Split 1	training time	626.136	1221.786	1816.186	2631.083	3003.408
	training	0.62	0.7289	0.711567	0.288325	0.71224
	test	0.6982	0.6587	0.6008	0.3992	0.6008
Split 2	training time	625.946	1219.217	1818.662	2640.099	3003.499
	training	0.707	0.62375	0.711067	0.71125	0.71246
	test	0.6007	0.4015	0.6007	0.6007	0.6007
Split 3	training time	625.523	1221.530	1836.373	2644.048	3001.036
	training	0.7162	0.7098	0.710633	0.7119	0.71314
	test	0.5959	0.5959	0.5959	0.5959	0.5959
Split 4	training time	624.979	1216.846	2167.596	2414.513	3002.050
	training	0.7058	0.74565	0.739033	0.714125	0.71464
	test	0.6026	0.632	0.6488	0.6055	0.6026
Split 5	training time	625.077	1222.270	2197.721	2409.355	3001.924
	training	0.7147	0.6981	0.376367	0.7159	0.71662
	test	0.6096	0.565	0.56	0.6096	0.6119
Average	training time	625.38125	1220.52225	1909.70425	2577.60225	3002.1045
	training	0.710925	0.713188	0.718075	0.712869	0.71382
	test	**0.604975**	**0.599975**	**0.6042**	**0.602275**	**0.602775**