Spatial Transformer Network Replication

Jiaxuan Wang

April 2018

1 Introduction

From natural objects to speech signals, symmetries are all around. The most intuitive example is geometric shape. Take a square for example, we can rotate it 90 degrees without changing its pose. More broadly speaking, symmetry is a transformation that preserves the information of an entity's identity. Note that this doesn't mean that under symmetry transformation, the object label is kept constant. It means that under those transformations, we can recover what the object identity is if we undo the transformation. For example, the digit 6 and 9, even though are different numerically, are considered 180 degree rotational symmetry of each other. Depending on whether the symmetry changes an entity's identity, we can further break down the transformation as equivariant (predictable change to object identity under the transformation) or invariant (object identity kept constant under the transformation).

Symmetries are extremely relevant to human learning. One hypothesis for our ability to do one shot learning is that we naturally generalize learned symmetries to new concepts. For example, when we learned about one pose of an object, we assume that its identity is invariant under common transformations such as view point changes or even actions that will deform the object's shape.

Symmetries are also relevant to machine learning. Convolution neural networks (CNN), for example, can be seen as an architecture specifically designed to exploit translational symmetry. The key insight is that small positional changes in features won't affect output label. Other examples exploiting symmetry include [1] [2] [4] and [9].

While CNN is able to handle small positional changes, it fails to capture other invariances such as rotational symmetry and scaling symmetry. However, hard coding these symmetries into a network seems arbitrary and labor intensive as a) not all symmetries are relevant to the task one is solving, and b) the effort spent on symmetry engineering may outweigh the benefit of using a learned model. It would be nice to have a soft way of adding invariance into a network without enforcing it, and then let data choose the invariance it want to keep. The paper I'm reproducing tackles this specific task.

Spatial transformer network (STN) [6], a 2015 NIPS publication by Max Jaderberg and his colleagues at Google Brain, introduces a clever method to learn invariance from image data. Specifically, they uses a neural network to output the transformation parameter per input image so that the transformation is doesn't affect object label and helps the later optimization procedure by bringing the input image to its canonical pose. This method is very clever because the entire process is differentiable. It is also very light weight. It can be understood as a contextual bandit problem where the action is the optimal invariant transformation. Furthermore, unlike regularization based technique, it does not aim to bias you with any symmetry, you have to learn yourself. However, our transformation is fundamentally bounded by the parametrization of the transformation classes we consider. For example, in this replication, we only consider affine transformation as a potential invariance, the network would not learn projective transformation. This restriction focuses the network's optimization procedure to narrow down the search path, which is instrumental to STN's success.

I want to reproduce this paper because learning invariance is crucial for machine learning applications. For computer vision tasks, invariant to view point changes as well as lighting conditions is essential for algorithmic robustness. For health care, identifying invariance in patient's vital signals for a particular disease can lead to actionable insights. Furthermore, learning invariance gives us a principled way to augment the dataset, which is critical for tasks with little data.

In this paper, I implement STN from scratch and replicate two experiments presented in STN. The first experiment verifies STN's ability to turn perturbed input back to its canonical pose. The second experiment generalized STN to perform per feature transformation, that is to transform different layers of input differently. In addition to the reproduction, I give two extensions to STN. The first extension looks at identifying two objects in an image. The second extension applies STN to a large scale clinical dataset, MIMIC3 [5]. The results indicate that STN is able to handle multiple objects in an image only if correct prior knowledge is added in the network (sensitive to the initialization of transformation parameters). In addition, on time series data, STN tends to smooth the time series as an artifact of bilinear interpolation, which may not be desirable when high frequency data are crucial. Overall, I think STN is an elegant solution to learning invariances and the two experiments considered are reproducible.

The paper is organized as follows. Section 2 introduces STN. Section 3 elaborates the replication task. Section 4 breaks down the two reproduced experiments, documents the network used, and compares the replication result to that of the original paper. Section 5 gives two more extension to the current approach, demonstrating potential problems with this model. Section 6 concludes the work and provide a discussion of the lessons learned.

2 Method: Works Being Replicated

STN is used for learning invariances in data. Specifically, this work focuses on object recognition. As a running example, let's consider a classification task where the data consists of images of hand written digits from 0 to 9. As natural in all images, the digits may come in different poses, parametrized by θ (could encode angles, translation, scaling factor, shearing, and etc., relative to its canonical pose). Given this image, STN first uses a network to output $\hat{\theta}$ as a guess of θ, and then it uses bilinear interpolation to apply $\hat{\theta}$ to all channels or the image. For example, if an image consist of RGB channels, the same spatial transformation is applied to each channel per input. After the transformation, a regular neural net is applied on this transformed image to perform the classification task.

The key is that the entire transformation and parameter prediction is end to end trainable. I reproduced two experiments in this paper, both using the MNIST dataset with custom modifications. Although MNIST is considered a toy dataset in computer vision, it is commonly used to verify intuition about an algorithm.

The first experiment illustrates STN's robustness over a CNN with comparable number of parameters. Essentially, the experiment introduces extra rotation, translation, and scaling noise into MNIST. With no extra supervision, STN is able to pass the eye test in correcting noises and it is better in classification performance than CNN on the held out test data. I verified the validity of its claim in Section 4.1.

The second experiment demonstrates STN's ability to handle multiple objects. This is an example presented in the Appendix of the paper to relax its assumption on applying the same transformation to all channels. The input consists of two randomly selected images from MNIST, each independently perturbed by some random transformation, and the task is to predict the sum of those two digits. Since the digits are presented on different channels to the network, the goal is to have the network untangle each image independently. My results agrees with STN's reported numbers. For experimental details, please refer to Section 4.2.

3 Replication Detail

To replicate the work, I used the bilinear interpolation module from $PyTorch$ [10]. The entire pipeline involving data loading, training, validation, testing, and model building are implemented from scratch. I obtained the MNIST dataset using the $PyTorch$ API. For the extension, I used MIMIC3 dataset cleaned by my labmate Jeeheh Oh. All data transformations such as random rotation, scaling, and translation are written with python's PIL package.

This project goes on rather smoothly. The biggest challenge is that hyperparameters used in this paper are not reported. For example, instead of giving the precise number of neurons in a layer, the paper just gives the number of layers used and the total number of parameters in the network. Fortunately, STN seems to be robust enough to any model architecture. In each experiments, I follow the paper's suggestion on the number of network layers, while improvising the architecture based on Alexnet [8], the first popular deep learning model applied on Imagenet [3]. I got comparable result as shown in the paper. The specific hyper parameters I used are detailed in each experiments shown below. To evaluate efforts made in this project as well as to verify my results, please refer to my code repository located at `https://github.com/nathanwang000/eecs692-reproduction`.

4 Replication Experiments

In this section, I detail two sets of experiments reproduced in the paper. For each task, I first describe how the data set is transformed and then provide full details of the training pipeline used. Finally, I give an in depth analysis into the results I got.

4.1 Single Object Recognition

The first set of replication experiments focuses on recognizing distorted MNIST digits. The MNIST handwritten digit database has a training set of 60,000 examples, and a test set of 10,000 examples.

4.1.1 Experiment Setup

Following the original paper, I choose to apply two transformations to the dataset. The first transformation considered is rotation. Here, I uniform randomly sample a rotation angle from -90 degree to 90 degree independently applied on each training instance. The rotation transformation is used as a sanity check to see if STN is able to handle single transformation on a single object dataset. In addition to rotation, I also added in translation and scaling for the second transformation. The procedure is similar. Following the practice of the paper reproduced, I uniform randomly sample a rotation angle from -45 to 45 degree, a translation amount of -0.1 to 0.1 fraction of input width, and a scaling of 0.7 to 1.3 to each input image. This second transformation is abbreviated as RTS.

For each of these two sets of data augmentations, I evaluate its generalization performance on held out test sets with transformations not seen in the training data.

4.1.2 Model Architecture

The STN model used for this task consists of a 4 layer localization network and a 4 layer classification network. Each convolutional layer has a stride of 1 and is followed by maxpooling with stride of 2. The activation function used is ReLU. The first layer converts a 28×28 gray scale image to a 8 channel output through 2d convolution with kernel size of 7. The second layer output 10 channels using 2d convolution with kernel size of 5. The third layer is fully connected with 32 neuron output. The fourth layer is fully connected resulting in 6 output parametrizing the affine transformation. Following the paper, the weight of the fourth layer is initialized to 0 and the bias to $[1, 0, 0, 0, 1, 0]$, effectively biasing towards the identity transformation. This initialization is crucial for STN to function correctly, I will expand on this observation in Section 5.1.

The classification network is similar. The first two layers are convolutional, both with kernel size of 5. I increase the output channel for these two layers from 10 to 20. To avoid overfitting, 2d droupout with 0.5 dropout rate is applied on the second convolutional layer. The third and fourth layer are again fully connected with output neuron of 50 and 10 respectively. Again, 0.5 dropout rate is applied between these layers. The output from the last layer is passed to a softmax function.

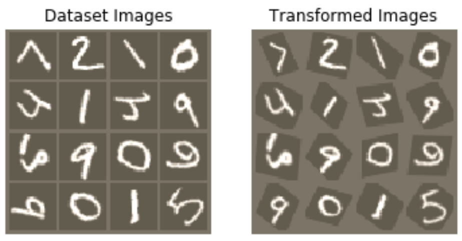
Our baseline method is a CNN, with similar architecture as the classification network. Instead of 20 output channels for the second convolutional layer, I increased it to 25 so that it has the same capacity as STN.

Both networks are trained using cross entropy loss with ADAM [7] optimizer. The learning rate is 0.001. We train the network for 10 epochs.

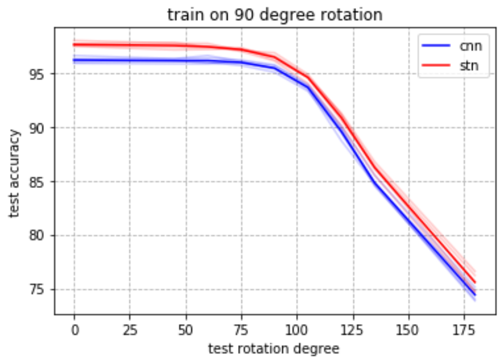
4.1.3 Result Analysis

Figure 4.1.3 gives result for STN training on 90 degree rotation but testing on 120 degree rotation. We observe that without any extra supervision, most transformed images pass the eye test of correcting the digit pose. The most obvious mistakes made by the network is on horizontal digit 4. The reason for that is the image resembles digit 3 which is hard to tell by the model. Beyond qualitative evaluation, we can look at the test performance. Figure 1b shows average test accuracies over 5 runs of the same experiment. It is clear that STN is much more robust compared to CNN in handling rotational changes.

The same analysis is performed on RTS. Note that unlike the rotation only experiment, RTS only trains on rotation angles of 45 degrees. Figure 4.1.3 shows the transformed result of RTS on 120 degree rotation. The mistakes made are similar to the rotation only case (observe that the transformed horizontal digit 4 is very close to digit 3). For most digits, it is able to handle the rotation correctly. Note that even though the training rotation is more constrained, STN is still able to generalize to beyond 75 degrees without much degradation both qualitatively and quantitatively, which is impressive. The same trend holds for both translation and scaling. Figure 4.1.3 and Figure 4.1.3 demonstrate the robustness of STN to translation and scaling beyond data augmentation. My result and observations agree with numbers reported in the paper.

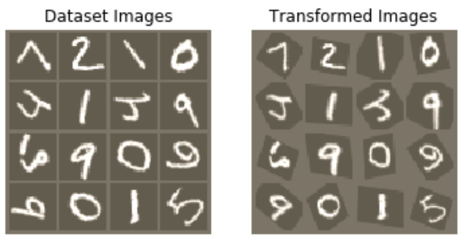


(a)

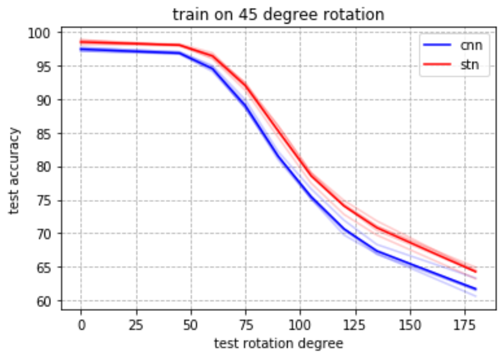


(b)

Figure 1: Rotation transformation examples for STN trained on rotation of 90 degrees. Figure (a) shows the original and transformed image under 120 degree rotation transformation (uniformly random sampled from the negative range to positive range). Note that the image is randomly picked. Figure (f) shows STN dominates CNN when varying rotation.

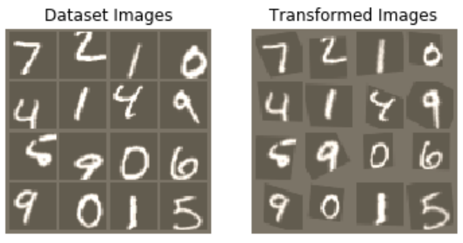


(a)

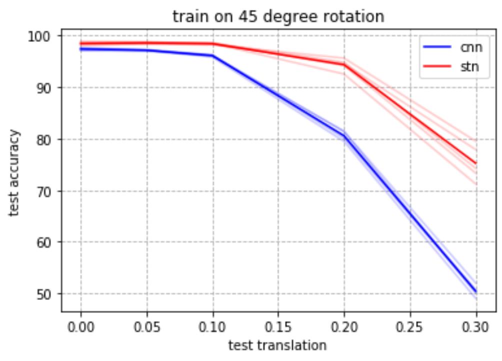


(b)

Figure 2: Rotation transformation examples for STN trained on RTS. Figure (a) shows original and transformed image under 120 degree noise (uniformly random sampled from the negative range to positive range). Figure (f) shows STN dominates CNN in classification accuracy when varying rotation, averaged over 5 runs of the experiments.

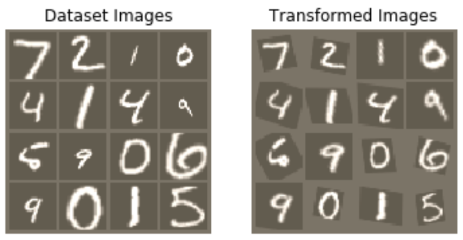


(a)

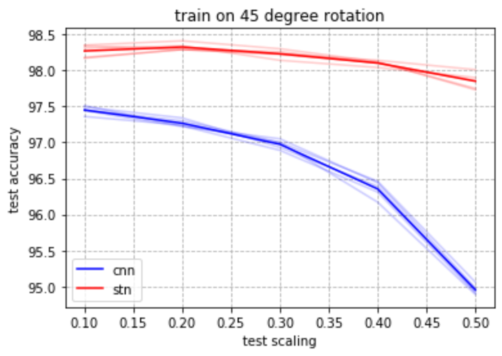


(b)

Figure 3: Translation transformation examples for STN trained on RTS. Figure (a) shows original and transformed image under 0.2 translation fraction of the input width (uniformly random sampled from the negative range to positive range). Figure (f) shows STN dominates CNN in classification accuracy when varying translation, averaged over 5 runs of the experiments.



(a)



(b)

Figure 4: Scaling transformation examples for STN trained on RTS. Figure (a) shows original and transformed image under 0.5 scaling noise (uniformly random sampled from 1 - scaling amount to 1 + scaling amount of the input image width). Figure (f) shows STN dominates CNN in classification accuracy when varying scaling, averaged over 5 runs of the experiments.

4.2 Multiple Objects Different Channels

The second experiment I reproduced extends the idea of focusing on one object to two. The assumption in regular STN is that all input channels are transformed the same way. In this experiment, the author relax the assumption to predict different transformation parameters per channel.

4.2.1 Experiment Setup

As in the first experiment, the authors continue to use MNIST. The task here is given two digit images, independently transformed by RTS, try to predict the summation of those digits. The images are fed into the network as different channels so the network doesn't need to segment those digits.

The authors do not include any detail on how the synthetic dataset is collected and how many training and testing examples there are. In my experiment, I randomly permute the order of both training and testing set. Then within each set of data, I treat two consecutive images as input so that the resulting dataset has one less the number of data than the original. Each pair of consecutive digits after the permutation are concatenated to form a two channel input. Figure 5 gives a pictorial illustration of the pipeline used for this task.

4.2.2 Model Architecture

The training procedure as well as hyperparameter used are the same as in Section 4.1.2. I independently apply the localization network on each channel. Moreover, since now two images are transformed, the input to the classification network also increases in size. I share the first two convolutional layers as before to apply to each image, and then their output are concatenated to feed into fully connected layers. Furthermore, with the sum of digits as target, we are no longer dealing with 10 way classifications, but 19 ways. We just need to make the final layer output 19 neurons.

4.2.3 Result Analysis

The digit addition task is much harder than the previous digit recognition task. The baseline CNN in the paper logs 14.7% error rate while my CNN baseline gives 47.3% error rate, only matching their fully connected network performance (47.7%). I suspect my CNN has been overfitting to the data, so

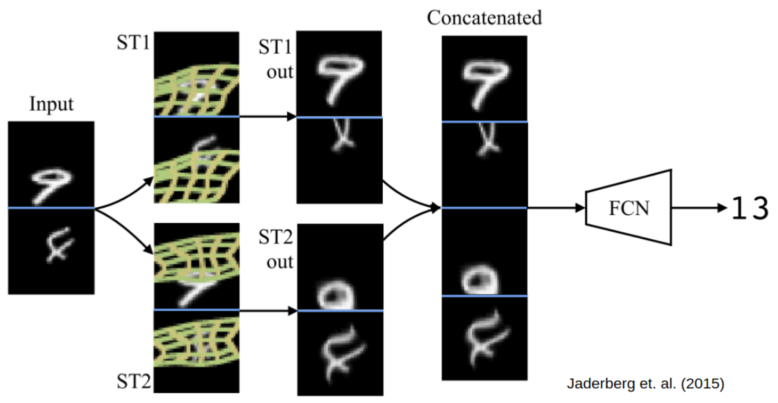


Figure 5: Illustration of two digits addition problem shown in the original paper. The idea is to use two separate STN to look at each digit. In this illustration, the digit 9 is picked up and transformed by the first STN, while the digit 4 is picked up and transformed by the second STN.

I tried smaller architectures but the performance doesn't go up. In addition to network difference, this fail in replication in baseline may also be a result of the particular random seed used for getting this synthetic data. I did not continue investigating the baseline performance because it doesn't affect the main result.

In terms of STN performance, it is comparable with the original paper's value. My STN achieves test error of 6.5%, even better than the reported number of 9.0%. Figure 6 gives the transformed result on a randomly selected batch of data. It should be clear that both digits are reasonably transformed to similar scale and orientation.

5 Extension

After verifying my implementation of STN using the previous two experiments, I extend the work to expose its weakness, namely unable to handle multiple objects in the same image. Moreover, I also applied STN to a time series dataset. The motivation for applying STN to time series data is that unlike images, time series invariances are less known. In this work, we only consider testing contraction, expansion, phase shift, scaling, and up down shift in the

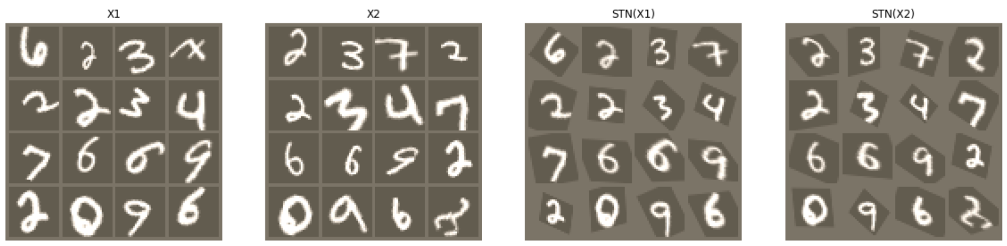


Figure 6: STN is able to correctly transform both input digits back to their canonical poses. The first two figures are the input to the network, while the last two figures are the transformed result. See Section 4.2 for detail.

experiment. I acknowledge that there are much more candidate invariances worth testing such as occlusion and warping, I focus on those transformations as a starting point.

5.1 Multiple Objects in the Same Image Channel

Although the authors claim that STN can handle multiple objects, the experiment done is not convincing enough. While the MNIST addition example deals with two digits in the input, they are actually presented in two channels to the network, which means that an STN easily knows where to focus on. It would be much challenging to reconstruct the canonical poses of the two digits if they are presented in the same image because then STN has to learn how to segment the image into interesting parts.

5.1.1 Experiment Setup

For this part of the extension, I constructed a synthetic data set from MNIST following the same procedure as in Section 4.2 with one caveat. Instead of concatenating two digits into two channels, I merge them into the same channel, forming a 28×56 sized image from two 28×28 sized images. Before merging the two images, each image is independently transformed by random rotation, translation, and scaling with the same procedure as described in Section 4.1.1. An illustration of the task is shown in Figure 7.

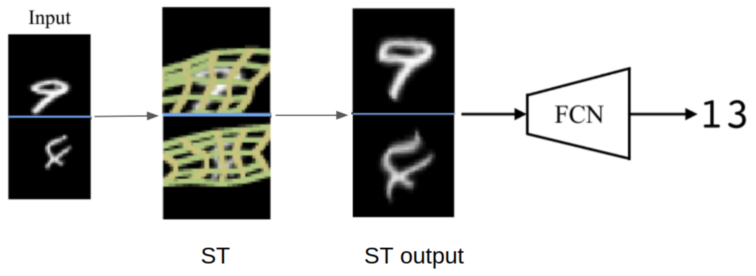


Figure 7: Illustration of the pipeline for two digits addition in the same image problem. As an extension over the second experiment, the input image is no longer given in two channels. Instead, they are concatenated and individually transformed. A single STN has to correctly focus on two digits in order to solve the addition problem.

5.1.2 Model Architecture

The network used in this task is similar to that of the original MNIST addition experiment. One difference is that to accommodate for the changed input size, the first convolutional layer of both the localization network as well as the classification network takes input with one channel. Moreover, to give independent predictions for the two images, the last layer of the localization network output 12 parameters instead of 6 because we have two digits to focus on. In this experiment, we examine STN's sensitivity to different initialization scheme as well as imputation method for points outside input image. We use the same learning rate, optimizer, dropout percentage, and stopping criteria as before. The results are averaged on 5 random runs.

5.1.3 Result Analysis

Recall that STN works by sampling from the input image according to the localizaiton network's output. The STN paper does not specify what value to use if the sampled point is outside of the input image. Here, I try two variants of the imputation, one using the nearest neighbor pixel value (referred to as border throughout the paper), the other using zero. Another loose point of the paper is how does the initialization of bias for the localization network affect model performance. In this section, we consider the original identity initialization (initializing the last layer bias of localiza-

Table 1: Test Accuracy for Two Digits in Same Image Over 5 runs

Method	Initialization	Imputation	Test Accuracy ($\pm$ std)
STN	Left Right	Border	89.73 ± 2.66 %
STN	Left Right	Zero	**91.4** ± 0.51%
STN	Identity	Border	87.10 ± 2.02%
STN	Identity	Zero	87.97 ± 0.97%
STN	Random	Border	36.32 ± 23.16%
STN	Random	Zero	33.55 ± 20.73%
CNN	NA	NA	84.99 ± 2.45 %

tion network to $[1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0]$), the standard spherical gaussian initialization, and the left right initialization (using the first 6 parameters output to look at the left, while using the last 6 parameter output of the localization network to look on the right; the bias that acheives this effect is $[0.5, 0, -0.5, 0, 1, 0, 0.5, 0, -0.5, 0, 1, 0]$).

The result of average 5 run test performance is summarized in Table 1. Note that the imputation and initialization scheme are not applicable to the baseline CNN. In general, random initialization did badly because the resulting transformation does not make sense and the network is easily trapped into local minimum. Curiously, border imputation consistently performs worse than zero imputation for reasonably initialized STN. This result suggest that knowing the boundary of the imputation (that is using zero imputation) helps the classification network narrow in on the interesting part of the transformed image for decision making. Furthermore, we see left right initialization performs better than other initialization scheme. This suggest that prior knowledge can significantly improve classification performance on this task. However, this also begs the question of how much did the network learn versus how much does human helps it learn. The reassuring part is that both identity and left right initialization outperforms CNN. This result consolidate the original paper's claim that STN is more robust, even in two objects setting.

A qualitative evaluation of the different STN variants used can be found in Figure 5.1.3. From (a) to (f), the transformation used are border with identity, border with left right, border with random, zero with identity, zero with left right, and zero with random respectively. The main observation is that left right initialization is able to separate out and transform back each digit reasonably. Combined with its superior test performance, this result agrees with human intuition. The identity transformation completely fails to separate two digits with both imputation schemes, which again suggest that STN is hitting a local minimum in the loss function space. The two random initialization results show that once their original focus is wrong, it is hard to recover back the true transformation.

5.2 Application to Time Series Data

In this task, we shift focus to time series data. While it is important to recover invariance in natural images, it is perhaps even more rewarding to learn invariance in time series as there are less intuition behind what invariance to test.

5.2.1 Experiment Setup

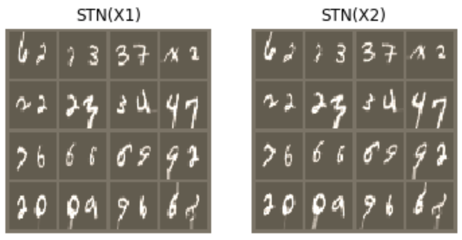
In this extension, I used the MIMIC3 dataset [5] cleaned by my labmate Jeeheh Oh. The data consists of 76 time series with time discretized into 48 timestamps, that is, one input is of size 76×48. For details on this dataset, please refer to [5].

5.2.2 Model Architecture

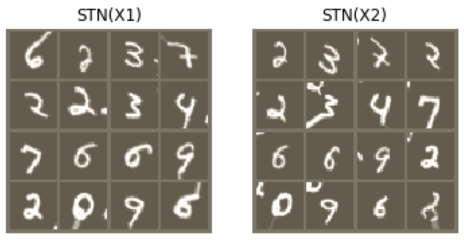
The adaptation from image to time series data is relatively straight forward. We just restrict the affine transformation parameters of STN to be $[1, 0, 0, 0, \theta_{22}, \theta_{23}]$. What this parametrization means is that only contraction ($|\theta_{22}| > 1$), expansion ($|\theta_{22}| < 1$), phase shift (controlled by θ_{23}), and flipping ($\theta_{22}| < 0$) is allowed. Moreover, we allow scaling and vertical shifting by letting spatial transformer output two more values (the slope and bias for a linear transformation). In addition, instead of 2d convolution, we apply 1d convolution with same filter size as in Section 4.1.2. The task is to predict the risk of in hospital mortality (a binary classification task).

5.2.3 Result Analysis

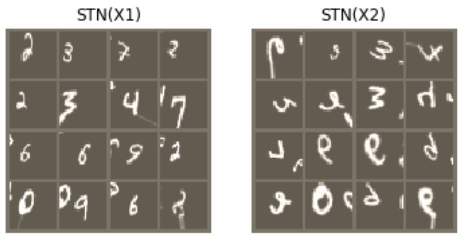
In using medical datasets, it is common to report performacne in terms of area under the receiving operating curve (AUC) because it is more robust against class label imbalance (much more negative examples than positive). We follow the same convention here. The STN used in our model acheives 85.10% AUC,



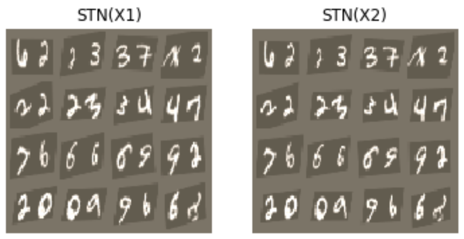
(a)



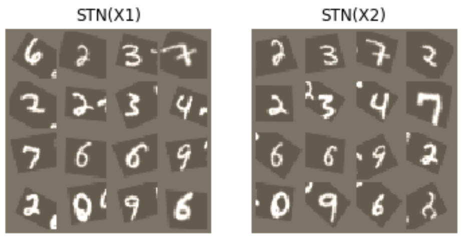
(b)



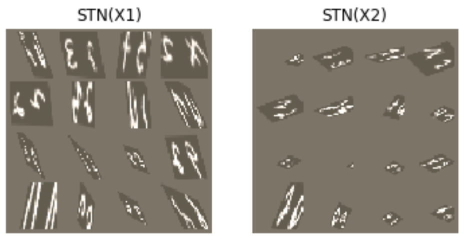
(c)



(d)

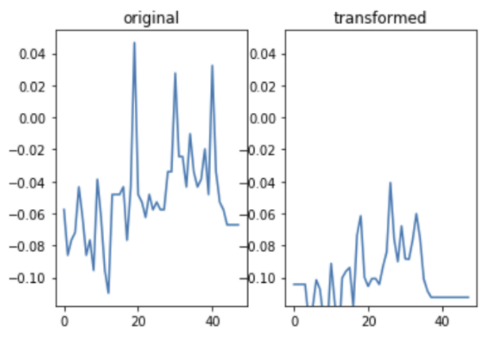


(e)

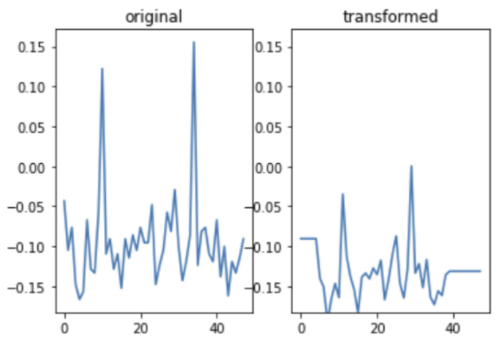


(f)

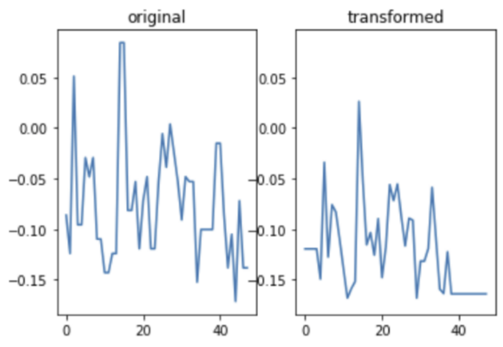
Figure 8: Two digits in the same image: comparing initialization and interpolation methods. From (a) to (f), the transformation used are border with identity, border with left right, border with random, zero with identity, zero with left right, and zero with random.



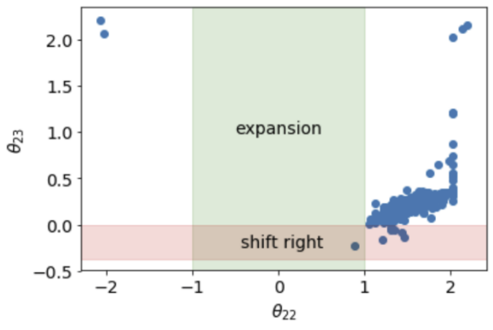
(a)



(b)



(c)



(d)

Figure 9: Figure (a) to (c) shows diastolic blood pressure for three randomly sampled patients. Figure (d) gives qualitative behavior based on two learned parameters transforming the test set: essentially, the network learns to zoom out (contraction instead of expansion) on the signal to smooth the data.

outperforming our baseline CNN (83.48%). The state of art result on this dataset is a long short term memory architecture (LSTM) reported in [5], which achieves 85.40% AUC on the test set. I'm surprised that without much hyperparameter tuning, we are close to that result. While out of scope for this class project, we intend to beat the LSTM baseline as a future work.

Figure 5.2.3 summarizes the qualitative result of the transformation learned. Essentially, the network learns to zoom out a little from the original signal and moves the whole curve down. We suspect this zoom out behavior helps smoothing the signal because of the bilinear interpolation used.

6 Conclusion

In this project, I replicated the results of two experiments from the Spatial Transformer network paper and verified that STN is reproducible. In reimplementing the network, I learned that it is crucial to carefully document the hyperparameters used, especially when you are not planning on releasing your code. Another lesson I learned from this project is that careful weight initialization is key to get your method working. As I shown in the first extension, the initialization method adds prior knowledge into the network. In this sense, STN can be seen as a clever way to regularize a network without introducing regularization constant. In my own research, I will aim to make my work as reproducible as possible by giving enough details.

Doing this project also got me into thinking about the relationship between human knowledge and machine learned representations. The two extremes are human direct coding (rule based systems) versus programs learn from scratch (e.g., alphago zero [11]). The middle of the spectrum include regularization techniques such as weight decay, dropout, and STN. The stronger the regularization, the less demand for data, but also less accurate. It would be nice to see a work that explore the whole spectrum.

For future works, I would like to continue along the line of applying STN to time series data. The main difficulty in this domain adaptation is that we don't know what is the canonical time series for a given dataset. My current thought is to cluster time series based on the learned transformation parameters and find the median time series as the cluster representation. As shown in Figure 9d, it seems like most time series clutter together in the parameter space. I need to spend time on thinking about a good clustering technique to deal with this technical challenge.

References

[1] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International Conference on Machine Learning*, pages 2990–2999, 2016.

[2] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *CoRR, abs/1703.06211*, 1(2):3, 2017.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[4] Josif Grabocka. Invariant features for time-series classification. 2015.

[5] Hrayr Harutyunyan, Hrant Khachatrian, David C Kale, and Aram Galstyan. Multitask learning and benchmarking with clinical time series data. *arXiv preprint arXiv:1703.07771*, 2017.

[6] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[9] Diego Marcos, Michele Volpi, and Devis Tuia. Learning rotation invariant convolutional filters for texture classification. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 2012–2017. IEEE, 2016.

[10] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[11] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.