Tied Spatial Transformer Networks for Character Recognition

Bogdan-Ionuţ CIRSTEA, Laurence LIKFORMAN-SULEM, Institut Mines-Télécom / Télécom ParisTech, Université Paris-Saclay, Paris, France

ABSTRACT. *This paper reports a new approach applied to convolutional neural networks (CNNs), which uses spatial transformer networks (STNs). It consists in training an architecture which combines a localization CNN and a classification CNN, for which most of the weights are tied, which from here on we will name Tied Spatial Transformer Networks (TSTNs). The localization CNN is used for predicting the best affine transform for the input image, which is then processed according to the predicted parameters and passed through the classification CNN. We have conducted initial experiments on the cluttered MNIST dataset, comparing the TSTN and Spatial Transformer Networks (STN) with untied weights, as well as the classification CNN only. In all these cases, we obtain better results using the TSTN architecture.*

RÉSUMÉ. *Cet article présente une nouvelle approche appliquée aux réseaux de neurones convolutionnels (RNC), qui utilise les réseaux de transformations spatiales (RTS). L'approche consiste à construire une architecture combinant un RNC pour la localisation et un RNC pour la classification. Bien que les deux réseaux soient dédiés à des taches différentes, la majorité de leurs poids sont partagées. Par la suite nous appelons ce type de réseaux réseaux de transformations spatiales liées ou RTSL. Le RNC de localisation est utilisé pour prédire la meilleure transformée affine que l'on peut appliquer sur l'image d'entrée, qui est ensuite traitée selon les paramètres prédits et passée au RNC de classification, qui fournit la prédiction. Nous avons mené des expériences initiales sur l'ensemble de données cluttered-MNIST comprenant des bruits additionnels. Nous comparons le RTSL et un RTS avec poids non liés, ainsi qu'avec le RNC de classification seul. Dans tous les cas, de meilleurs résultats sont obtenus en utilisant l'architecture RTSL proposée.*

KEYWORDS: *Convolutional Neural Network, Deep Learning, Character Recognition,*

MOTS-CLÉS : *Réseau de neurones convolutionnel, Apprentisage Profond, Reconnaissance de caractères,*

1. Introduction

Recently, deep architectures based on convolutional neural networks (CNNs) have obtained state-of-the-art results for several recognition tasks in computer vision, such as handwriting recognition (Graves and Schmidhuber, 2009) (Bluche *et al.*, 2015) and object recognition (Krizhevsky *et al.*, 2012) (Simonyan and Zisserman, 2014) (Russakovsky *et al.*, 2015). In contrast to classical approaches, deep learning systems learn features in an automatic fashion, avoiding the time-consuming task of hand-crafting them.

In this article we use a variation on STNs (Spatial Transformer Networks), deep learning architectures that couple two CNNs using a Spatial Transformer module, which was first introduced by (Jaderberg *et al.*, 2015). The first one, the so-called *localization* CNN, is used to perform a spatial transformation on the input image. This spatially transformed image is then used by the second CNN, the *classification* CNN, to perform the class prediction.

The advantage of using such coupled architectures (transformation/localization coupled to perform classification) is that the weights of the localization network are learned during the training phase, along with those of the classification network, so that images (of characters in our case) can be transformed and classified in an unified way. Since localization and classification are different tasks, the weights of the localization and classification networks generally differ. We propose here to tie the weights of both networks in order to obtain a regularization effect which can improve performance. We apply our approach to the recognition of noisy characters, using the MNIST-cluttered database, derived from the MNIST database but including additional noises.

2. Common elements

In this section, we present the common elements used between the classification CNN, the untied STN, and the TSTN. For simplicity and to make the comparison of the results easier, we have tried to keep as many elements as possible (convolutional architectures, activation function, optimization procedure, regularization) identical or as similar as possible between the three neural network architectures.

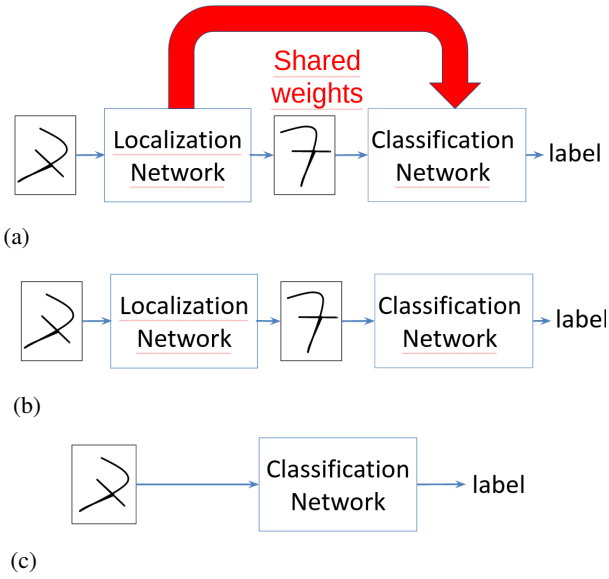

Figure 1 – TSTN, untied STN and CNN architectures

2.1. *Convolutional architectures*

We perform experiments using two different architectures for the classification and localization CNNs, the second one being more complex (and more expressive) than the first. These architectures follow many of the guidelines from (Simonyan and Zisserman, 2014), the system which has obtained the best single-model performance in the ILSVRC 2014 object classification competition (Russakovsky *et al.*, 2015). Each of the convolutional layers contains filters of size 3x3. 'Same' mode convolution (which allows the size of the output of each convolutional map to be equal to the input) is used in all the layers. The subsampling layers are always 2x2 max-pooling with stride 2, which results in subsampling both image height and width by 2. We double the number of convolutional maps between each consecutive convolutional layer, from 32 in the first convolutional layer to 64 then 128 in the last one, drawing once more inspiration from the guidelines outlined in (Simonyan and Zisserman, 2014).

The fully connected parts are identical between the two CNN architectures described in Tables 1 and 2, but different between the localization and classification CNNs. The convolutional parts are also very similar; the difference is that for Architecture 2 we use 2 operations of convolution followed by nonlinearity instead of a single one, followed by the pooling layer, for each of the 3 convolutional layers, making Architecture 2 more powerful and expressive. In both architectures, we remove the last pooling layer of the localization CNN. The motivation for this architectural

choice is to keep more of the spatial information that the fully connected layer of the localization CNN might need in order to output the estimated parameters of the affine transformation. Notice also that since the max-pooling layers don't contain any trainable parameters, this means that the classification and localization CNNs still have the same (tied) weights. For both architectures, the last layer of the localization CNN is smaller than that of the classification CNN, imitating the design of the architectures used in (Sønderby *et al.*, 2015) and (Sønderby, 2015b).

STNs can be useful for character recognition because they are able to disentangle the characters' pose and deformation from their shape. In noisy images such as those of the cluttered MNIST dataset, STNs can also crop out and scale-normalize the appropriate image region for the classification task. The Spatial Transformer module is differentiable, so it can be easily inserted into existing convolutional architectures without having to change the optimization algorithm. It can also be interpreted as a differentiable attention mechanism, whose output is conditioned on individual data samples (input images). The appropriate output for each data sample (i.e. its class) is learned during the training / optimization phase; when integrated into CNN architectures used for recognition, the Spatial Transformer requires no extra supervision signal.

Generally, the transformation performed on the input image using the Spatial Transformer can include scaling, cropping, rotations, as well as non-rigid deformations (Jaderberg *et al.*, 2015). In all of our architectures, we only use the affine transform Spatial Transformer, introduced in (Jaderberg *et al.*, 2015) and also used in (Sønderby *et al.*, 2015). It allows cropping, translation, rotation, scale, and skew to be applied to the input image and only requires 6 parameters to be estimated by the localization network. The implementation we use, introduced and described in (Sønderby *et al.*, 2015), also allows the use of a subsampling factor d, by which both the height and the width of the image are scaled, after applying the affine transform. To make the models simpler and help simplify comparisons between different models (including between CNNs and STNs), we use $d = 1$ in all of our experiments involving STNs. By performing affine transforms, architectures integrating the Spatial Transformer module can both select relevant portions of an image (a form of attention) and transform those regions into a pose which can simplify the recognition task for the following layers of the architecture. For all the technical details about Spatial Transformers, we refer the reader to the relevant publication (Jaderberg *et al.*, 2015).

2.2. *Activation functions and parameter initialization*

As activation function, we use in all of our experiments the Rectified linear unit (ReLU) (Glorot *et al.*, 2011), one of the most successful activation functions in deep learning.

For simplicity, we initialize all the weight matrices using a method similar to the one proposed in (He *et al.*, 2015) and which is also used in (Sønderby, 2015b). Weight

matrices are initialized using values drawn from uniform distributions with mean 0 and standard deviation $\sqrt{\frac{2}{n_i}}$, where n_i is the number of inputs to the neuron, while the biases are initialized to 0. The procedure, presented in detail in (He *et al.*, 2015), helps avoid instability in gradients during the training phase, as well as instability of inputs to each layer during inference. The affine transform layer parameters are initialized to the identity transform, like in (Sønderby, 2015b). We refer you to (He *et al.*, 2015) for the full mathematical treatment.

2.3. *Loss function and optimization*

The loss function we minimize is the negative log-likelihood of the conditional distribution of the correct label given the input. Each one of the systems we experiment with represents a differentiable function. To optimize each of them, we can use stochastic gradient descent (SGD), applied to neural networks using the backpropagation algorithm on mini-batches of multiple examples (computation over mini-batches is much more efficient than over single samples when using GPUs).

We use as optimization algorithm a variant of Adam (Kingma and Ba, 2014), which uses the gradients computed through backpropagation. Adam is a state of the art algorithm developed for improving the performance of stochastic gradient descent (SGD) on stochastic non-stationary loss functions and sparse gradients. It introduces a few additional hyper-parameters, all described in detail in (Kingma and Ba, 2014); we have kept all these hyper-parameters fixed, using the values recommended in (Kingma and Ba, 2014). The only parameter we vary is the learning rate, which we multiply by a fixed amount after a fixed number of epochs. We have used the same values as in (Sønderby, 2015b) and multiplied the learning rate by 0.7 after each 20 epochs.

2.4. *Regularization*

Dropout (Srivastava *et al.*, 2014) is a regularization method which allows for much bigger networks to be trained without overfitting, by randomly dropping units from the neural network during training with probability p sampled from a Bernoulli distribution. During validation and testing, the units are no longer dropped, but their activations are scaled deterministically by the same probability p. We use $p = 0.5$ dropout in the fully connected layer of the classification CNN only, because it is the layer with the largest number of parameters, where regularization is potentially most useful. We don't use dropout in the fully connected layer of the localization network. The intuition behind this choice is that dropout, as a noisy form of regularization, tends to remove some of the spatial information present in the activation maps. This spatial information might be more valuable for the localization task (as compared to the classification one), so we prefer to preserve more of it (at the risk of overfitting) by not using dropout in the fully connected layer of the localization network.

Localization CNN	**Classification CNN**
Convolutional Layer 1	**Convolutional Layer 1**
conv: 3 x 3 same, stride 1, 32 feature maps	conv: 3 x 3 same, stride 1, 32 feature maps
ReLU	ReLU
2 x 2 max pooling, stride 2	2 x 2 max pooling, stride 2
Convolutional Layer 2	**Convolutional Layer 2**
conv: 3 x 3 same, stride 1, 64 feature maps	conv: 3 x 3 same, stride 1, 64 feature maps
ReLU	ReLU
2 x 2 max pooling, stride 2	2 x 2 max pooling, stride 2
Convolutional Layer 3	**Convolutional Layer 3**
conv: 3 x 3 same, stride 1, 128 feature maps	conv: 3 x 3 same, stride 1, 128 feature maps
ReLU	ReLU
	2 x 2 max pooling, stride 2
flatten	flatten
Fully Connected Layer	**Fully Connected Layer**
200 neurons linear layer	625 neurons linear layer
ReLU	ReLU
	0.5 dropout
Affine Transform Layer	**Softmax Layer**
6 parameters	10-class softmax

Table 1 – Architecture 1: localization CNN (left) and classification CNN (right).

Localization CNN	**Classification CNN**
Convolutional Layer 1	**Convolutional Layer 1**
conv: 3 x 3 same, stride 1, 32 feature maps	conv: 3 x 3 same, stride 1, 32 feature maps
ReLU	ReLU
conv: 3 x 3 same, stride 1, 32 feature maps	conv: 3 x 3 same, stride 1, 32 feature maps
ReLU	ReLU
2 x 2 max pooling, stride 2	2 x 2 max pooling, stride 2
Convolutional Layer 2	**Convolutional Layer 2**
conv: 3 x 3 same, stride 1, 64 feature maps	conv: 3 x 3 same, stride 1, 64 feature maps
ReLU	ReLU
conv: 3 x 3 same, stride 1, 64 feature maps	conv: 3 x 3 same, stride 1, 64 feature maps
ReLU	ReLU
2 x 2 max pooling, stride 2	2 x 2 max pooling, stride 2
Convolutional Layer 3	**Convolutional Layer 3**
conv: 3 x 3 same, stride 1, 128 feature maps	conv: 3 x 3 same, stride 1, 128 feature maps
ReLU	ReLU
conv: 3 x 3 same, stride 1, 128 feature maps	conv: 3 x 3 same, stride 1, 128 feature maps
ReLU	ReLU
	2 x 2 max pooling, stride 2
flatten	flatten
Fully Connected Layer	**Fully Connected Layer**
200 neurons linear layer	625 neurons linear layer
ReLU	ReLU
	0.5 dropout
Affine Transform Layer	**Softmax Layer**
6 parameters	10-class softmax

Table 2 – Architecture 2: localization CNN (left) and classification CNN (right).

3. Experiments

As initial experiments, we apply TSTNs to the recognition of noisy handwritten digits. We conduct recognition experiments on the cluttered MNIST database (Sønderby, 2015a) which is derived from the MNIST database (Yann LeCun, 2015) but includes additional noise and transformations, which makes it harder to classify: the digits are distorted using random translation, scale, rotation, and clutter. We show some images samples from cluttered MNIST in Figure 2.

For our implementation, we use Theano (Bergstra *et al.*, 2010) (Bastien *et al.*, 2012) and Lasagne (Dieleman *et al.*, 2015) and rely heavily on the following implementation (Sønderby, 2015c) and example (Sønderby, 2015b) of Spatial Transformer Networks.

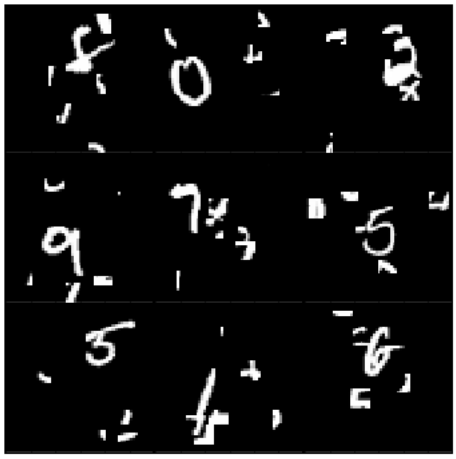

Figure 2 – Cluttered MNIST dataset image samples

3.1. *CNN and STN comparison*

We compare in Table 3 the proposed TSTN network (tied weights) against the untied STN and the classification CNN alone, using two different architectures (architecture 2 being more complex) on the cluttered MNIST dataset. We list here the tested networks:

1) Classification CNN only, Architecture 1

2) Untied STN, Architecture 1

3) TSTN, Architecture 1

4) Classification CNN only, Architecture 2

5) Untied STN, Architecture 2

6) TSTN, Architecture 2

We have stopped the training of each one of the networks manually, after checking that the training accuracy is (almost) perfect. In a single case, the untied STN architecture 2, we haven't been able to get the training procedure to converge. To highlight that the number of epochs of training is comparable and that we have allowed each network a reasonable amount of training to achieve good performance, we also show in Table 3 the number of training epochs and best training accuracy (achieved up to the maximum training epoch). For each architecture and training procedure, we choose the parameters for which the best validation accuracy is obtained for testing. The results are shown in the last column of Table 3.

As can be seen in the results table, the TSTN obtains significantly better performance than both the classification CNN and the untied STN, for both architectures 1 and 2.

Model	Maximum training epochs	Best training error (%)	Validation error (%)	Test error (%)
Architecture 1, classification CNN only	145	0	4.39	4.44
Architecture 1, untied STN	179	0	4.38	4.33
Architecture 1, TSTN	196	0.002	3.02	3.14
Architecture 2, classification CNN only	95	0	1.82	2.18
Architecture 2, TSTN	231	0	1.3	1.74

Table 3 – Results on the cluttered MNIST database for different achitectures and training schemes.

4. Discussion

In this section we provide a short possible interpretation of the prelinary results of our experiments with TSTNs.

First, the parameters learned by the localization CNN can be interpreted as encoding the knowledge of how to transform training samples (the transform being conditioned on each individual sample) so as to simplify the task of recognition for the following layers of the entire network. By tying the parameters of the classification and localization CNNs, the TSTN is constrained to become good at both tasks (of transforming the input image and classifying the transformed image). From this perspective, tying the parameters of the classification and localization CNNs can be interpreted as a form of regularization. We have performed additional preliminary

experiments which would seem to support this hypothesis; those experiments will be presented in a future publication.

An advantage of tying parameters in the TSTNs is that we no longer have to do a separate hyper-parameter search for the localization CNN. A disadvantage of STNs (both untied and tied), however, is that the computational time needed to use them is approximately twice the computation cost of the classification CNN alone. From this perspective, using training procedures where we combine training the TSTN and classification CNN alternately and only use the classification CNN at test time is interesting. This can be interpreted as a curriculum form of learning and is one of the experiments we are currently pursuing to validate our hypothesis of the tying the parameters of the classification and localization CNNs as a form of regularization.

5. Conclusion

In this article we have proposed a new way of using Spatial Transformer Network architectures, by tying the parameters between the localization and classification networks, which we call Tied Spatial Transformer Networks (TSTNs). A first extension of our work should be performing more experiments and analyzing the representations learned by the TSTN.

For improved results, we could keep the tied weights between the localization and classification CNNs in the TSTN, while also add a recurrent layer on top of the localization network, like in Recurrent Spatial Transformer Networks (RSTNs) (Sønderby *et al.*, 2015). Since RSTNs obtain better results than STNs and TSTNs do the same, we could expect improved results by combining these two ideas (using TRSTNs).

TSTNs could also be applied to handwriting recognition in 'noisy environments' e.g. historical documents or textline recognition of textlines in which the result of the segmentation process produces noise (e.g. fragments of text from neighboring lines). They could also be used for more difficult tasks, an obvious choice being object recognition.

Acknowledgements

This research was supported by a DGA-MRIS scholarship. We wish to thank NVIDIA Corporation for the donation of the GPU which was used to perform this research.

6. References

Bastien F., Lamblin P., Pascanu R., Bergstra J., Goodfellow I. J., Bergeron A., Bouchard N., Bengio Y., "Theano: new features and speed improvements", , Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

Bergstra J., Breuleux O., Bastien F., Lamblin P., Pascanu R., Desjardins G., Turian J., Warde-Farley D., Bengio Y., "Theano: a CPU and GPU Math Expression Compiler", *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June, 2010. Oral Presentation.

Bluche T., Ney H., Kermorvant C., "The LIMSI handwriting recognition system for the HTRtS 2014 contest", *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, p. 86-90, 8, 2015.

Dieleman S., Schlüter J., Raffel C., Olson E., Sønderby S. K., Nouri D., Maturana D., Thoma M., Battenberg E., Kelly J., Fauw J. D., Heilman M., diogo149, McFee B., Weideman H., takacsg84, peterderivaz, Jon, instagibbs, Rasul D. K., CongLiu, Britefury, Degrave J., "Lasagne: First release.", August, 2015.

Glorot X., Bordes A., Bengio Y., "Deep Sparse Rectifier Neural Networks", *International Conference on Artificial Intelligence and Statistics*, p. 315-323, 2011.

Graves A., Schmidhuber J., "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks", *Advances in Neural Information Processing Systems*, p. 545-552, 2009.

He K., Zhang X., Ren S., Sun J., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", 2015.

Jaderberg M., Simonyan K., Zisserman A., Kavukcuoglu K., "Spatial Transformer Networks", 6, 2015.

Kingma D., Ba J., "Adam: A Method for Stochastic Optimization", 2014.

Krizhevsky A., Sutskever I., Hinton G. E., "ImageNet Classification with Deep Convolutional Neural Networks", *Advances in Neural Information Processing Systems*, p. 1097-1105, 2012.

Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z., Karpathy A., Khosla A., Bernstein M., Berg A. C., Fei-Fei L., "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision*, vol. 115, n° 3, p. 211-252, 2015.

Simonyan K., Zisserman A., "Very Deep Convolutional Networks for Large-Scale Image Recognition", 2014.

Sønderby S. K., Sønderby C. K., Maaløe L., Winther O., "Recurrent Spatial Transformer Networks", 9, 2015.

Srivastava N., Hinton G. E., Krizhevsky A., Sutskever I., Salakhutdinov R., "Dropout: a simple way to prevent neural networks from overfitting.", *Journal of Machine Learning Research*, vol. 15, n° 1, p. 1929-1958, 2014.

Sønderby S. K., "Cluttered MNIST dataset", , `https://s3.amazonaws.com/lasagne/recipes/datasets/mnist_cluttered_60x60_6distortions.npz`, 11, 2015a. [Online; accessed 30-November-2015].

Sønderby S. K., "Spatial Transformer Network code example ('recipe')", , `https://github.com/Lasagne/Recipes/blob/master/examples/spatial_transformer_network.ipynb`, 11, 2015b. [Online; accessed 30-November-2015].

Sønderby S. K., "Spatial Transformer Network code repository", , `https://github.com/skaae/transformer_network`, 11, 2015c. [Online; accessed 30-November-2015].

Yann LeCun Corinna Cortes C. J. B., "The MNIST database of handwritten digits", , `http://yann.lecun.com/exdb/mnist/`, 11, 2015. [Online; accessed 30-November-2015].