Using Spatial Transformers for Digit Identification

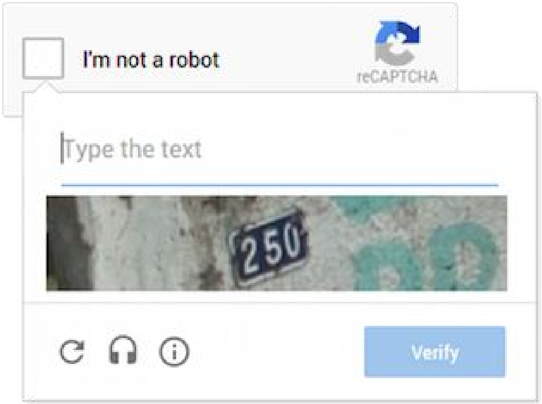
Scott Chow and Robert Cyprus

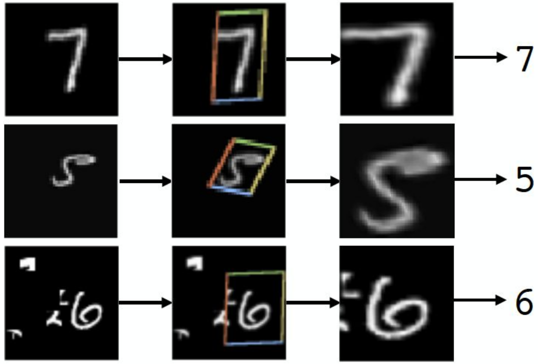
Motivation

Task: Identify digits and characters from images

Issue:

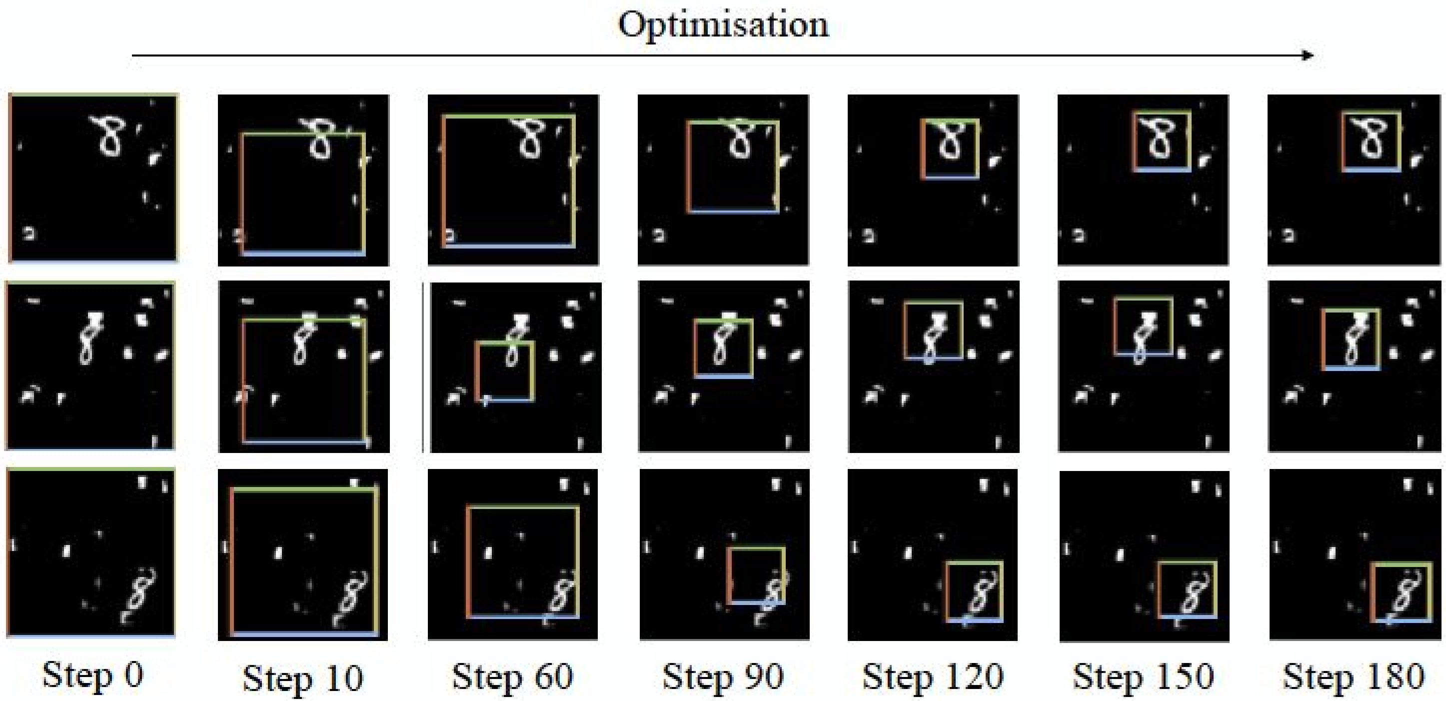
- Words can be rotated or distorted in images.
- Standard Convolutional Neural Nets (CNN) do not deal with these distorted characters.





Primary Text of Interest

"Spatial Transformer Networks" (2016) by Jaderberg et al. at Google DeepMind,

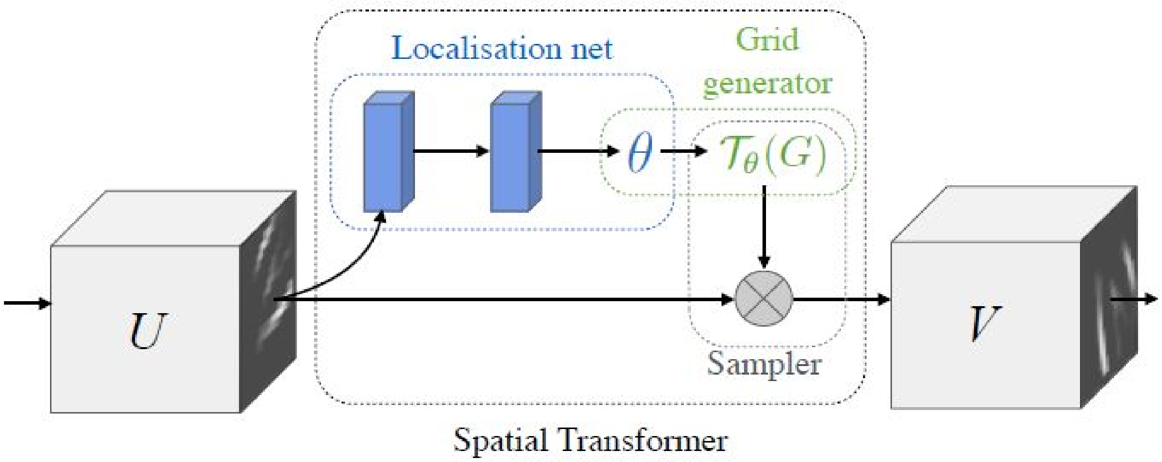


Introduction to Spatial Transformers

Spatial Transformers “explicitly allows the spatial manipulation within the network.”

Consists of three parts:

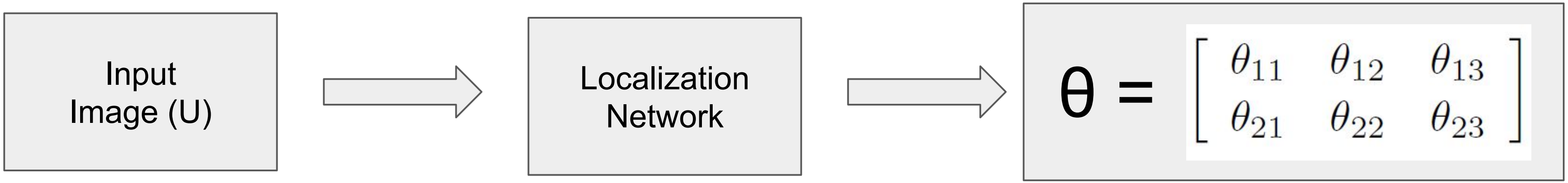
a) Localization Network
b) Grid Generator
c) Sampler



Spatial Transformers: Localization Network

A function that takes in the input image and outputs the parameters of the transformation to be applied to the feature map.

Usually a fully-connected network or convolutional network.

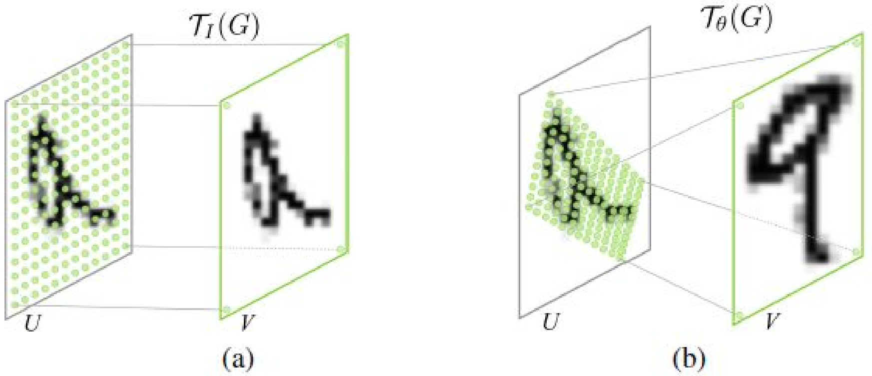


Spatial Transformers: Grid Generator

We want to create a grid (G) that maps points from input image □ points on output

Using θ from Localization Network, compute modified grid, $T_\theta(G)$

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \mathtt{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$



Spatial Transformers: Sampler

Given the input image (U) and the grid of sampling points $T_\theta(G)$, compute the output image (V) by applying a sampling kernel *k*

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m; \Phi_x) k(y_i^s - n; \Phi_y) \;\; \forall i \in [1 \dots H'W'] \;\; \forall c \in [1 \dots C]$$

Let’s break this equation down...

STN: Sampler

Given the input image (U) and the grid of sampling points $T_\theta(G)$, compute the output image (V) by applying a sampling kernel *k*

$$V_i^c$$

For the i-th pixel in the output image

Spatial Transformers: Sampler

Given the input image (U) and the grid of sampling points $T_\theta(G)$, compute the output image (V) by applying a sampling kernel *k*

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c$$

For the i-th pixel in the output image,
Sum across each point of the input image

Spatial Transformers: Sampler

Given the input image (U) and the grid of sampling points $T_\theta(G)$, compute the output image (V) by applying a sampling kernel *k*

$$V_i^c = \sum_{n}^{H} \sum_{m}^{W} U_{nm}^c k(\qquad) k(\qquad)$$

For the i-th pixel in the output image,
Sum across each point of the input image
After applying the sampling kernel *k*

Spatial Transformers: Sampler

Given the input image (U) and the grid of sampling points $T_\theta(G)$, compute the output image (V) by applying a sampling kernel *k*

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m; \Phi_x) k(y_i^s - n; \Phi_y)$$

For the i-th pixel in the output image,
Sum across each point of the input image
After applying the sampling kernel *k*
With kernel parameters:

(x_i, y_i) = *i*-th point in $T_\theta(G)$

(Φ_x, Φ_y) = parameters for sampling kernel

Spatial Transformers: Sampler

Given the input image (U) and the grid of sampling points T_θ(G), compute the output image (V) by applying a sampling kernel *k*

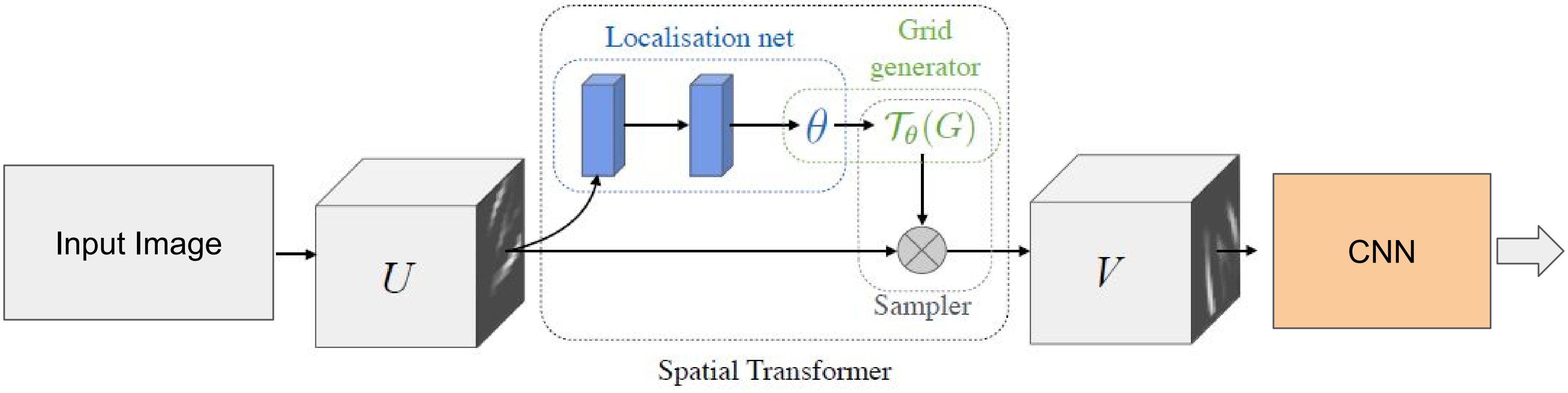
$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m; \Phi_x) k(y_i^s - n; \Phi_y) \;\; \forall i \in [1 \ldots H'W'] \;\; \forall c \in [1 \ldots C]$$

For the i-th pixel in the output image,
Sum across each point of the input image
Applying the sampling kernel *k*
With kernel parameters:
  (x_i, y_i) = *i*-th point in T_θ(G)
  (Φ_x,Φ_y) = parameters for sampling kernel
For all points in the output image (*H'W'*) and all image channels (*C*)

Incorporating Spatial Transformers into the Pipeline

“Spatial Transformers can be added into Convolutional Neural Network architecture at any point,” creating a Spatial Transformer Network (STN)

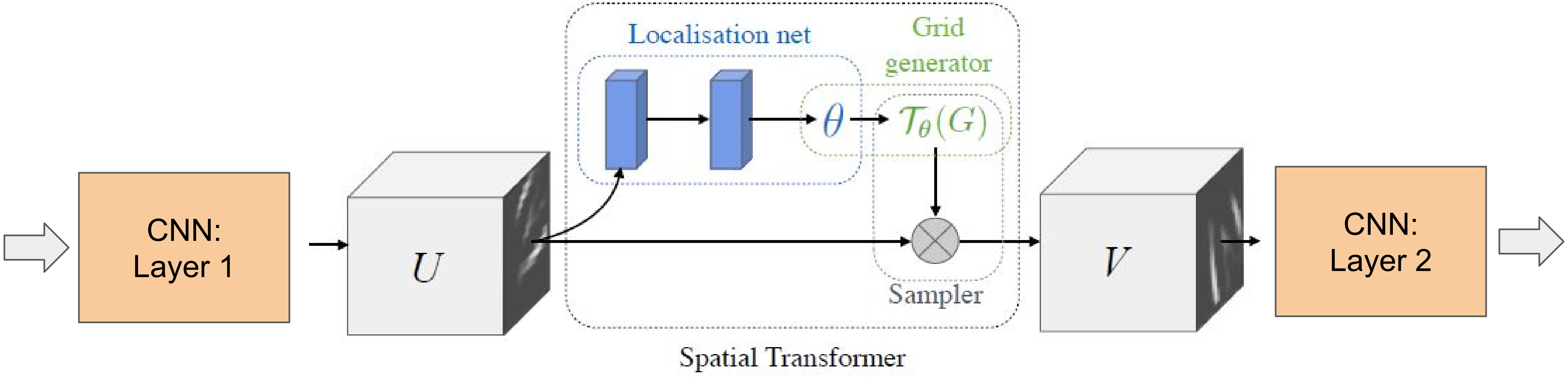
Spatial Transformers learn via Back-propagation, thus very flexible in placement.



Extension: STNs between CNN Layers

"Spatial Transformers can be added into a Convolutional Neural Network architecture **at any point**," creating a Spatial Transformer Network (STN)

Authors briefly mention that we could add spatial transformers between CNN layers. To be explored...

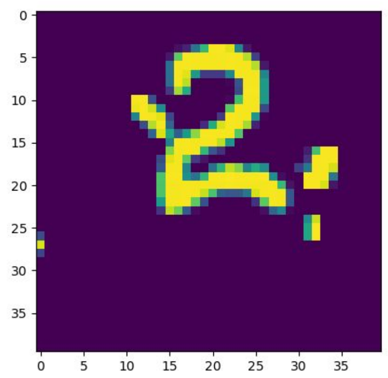


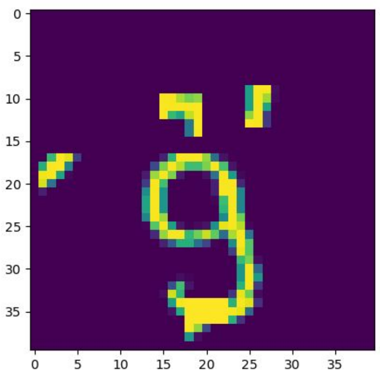
STN Experiments

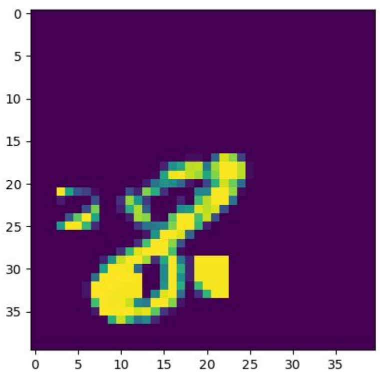
Data Set

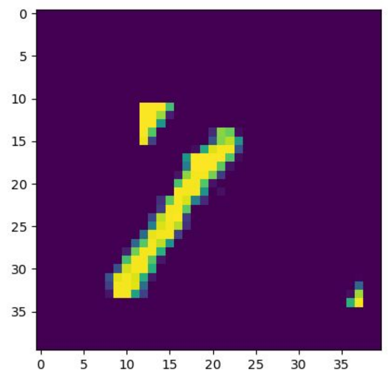
Cluttered MNIST Data Set

Used subset of data:
10000 Training Set /1000 Test Set /1000 Validation Set









Hardware / Software Used

- Hardware
  - NVidia GTX 970 Graphics Card (3.5GB)
  - Intel i5 6600K @ 3.5GHz
- Software
  - Python 3.5
  - Tensorflow (GPU version)

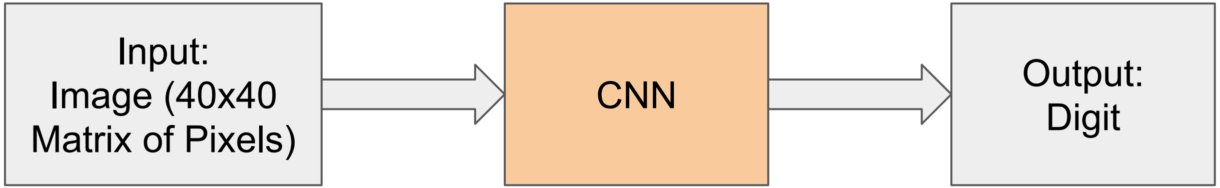
Three Neural Net Designs

- Image □ CNN □ Digit
  - Classic digit identification methodology
  - Our control test and benchmark for digit identification

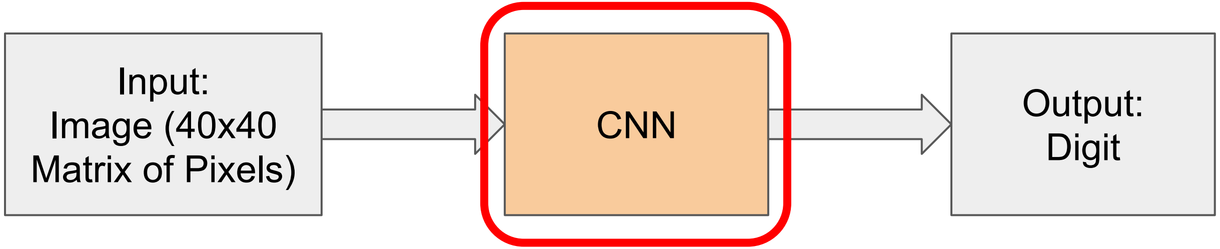
- Image □ STN □ CNN □ Digit
  - Used by Google Deepmind

- Image □ STN □ CNN □ STN □ CNN □ Digit
  - One type of our modified Neural Nets
  - We call this MSTN, for Multi-STN

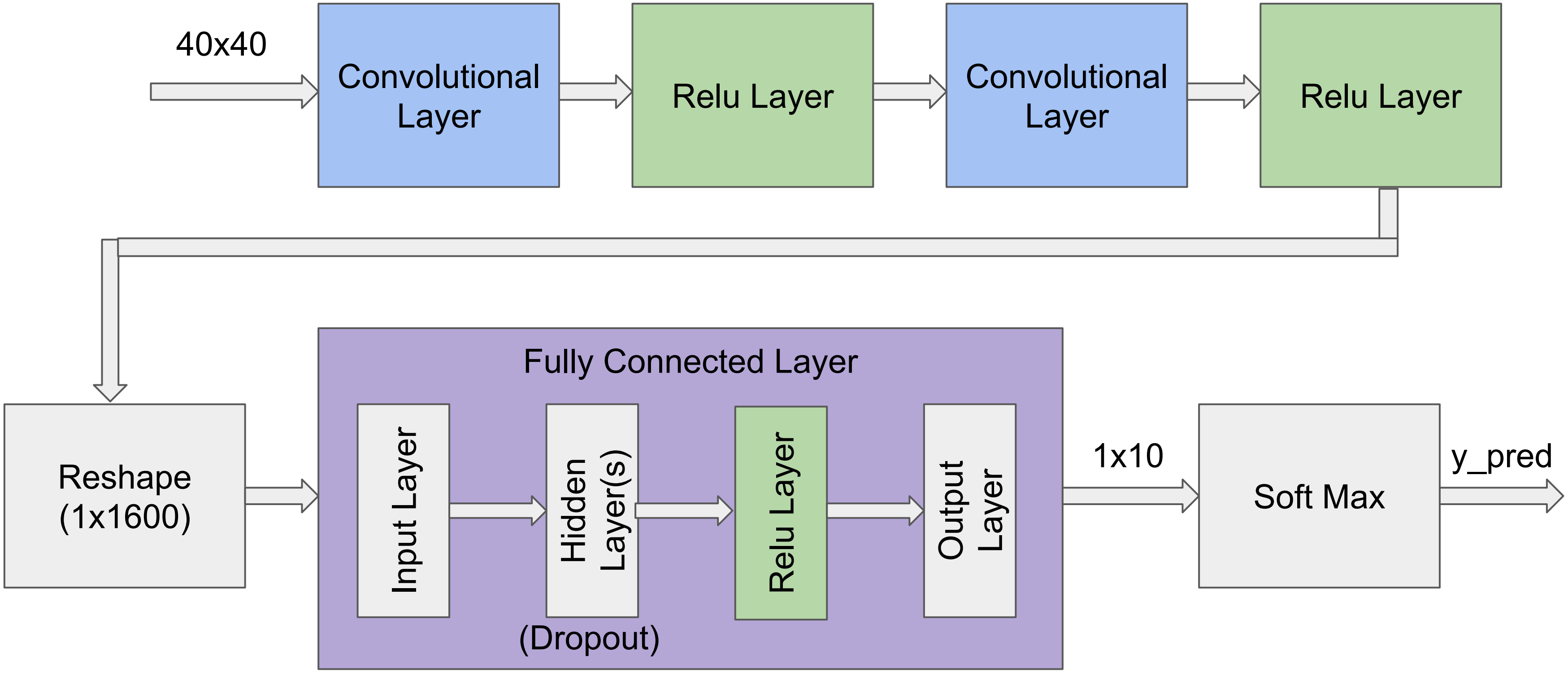
Standard CNN For Digit Identification



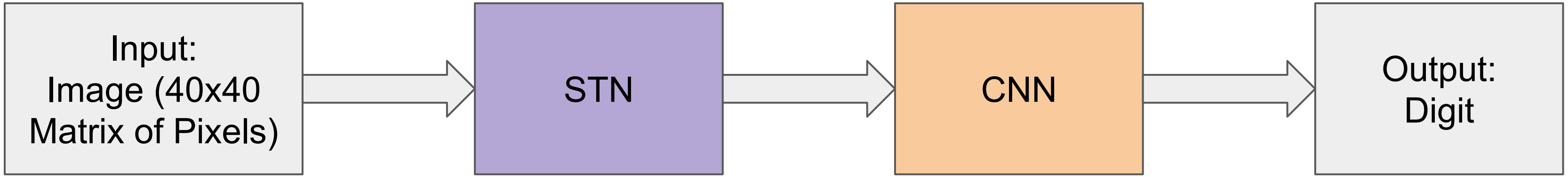
Standard CNN For Digit Identification



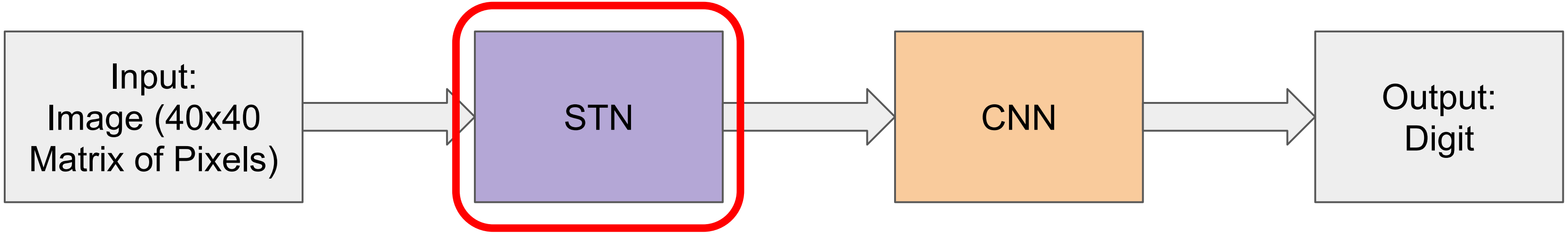
CNN Design


40x40
Convolutional Layer
Relu Layer
Convolutional Layer
Relu Layer
Reshape (1x1600)
Fully Connected Layer
Input Layer
Hidden Layer(s)
(Dropout)
Relu Layer
Output Layer
1x10
Soft Max
y_pred

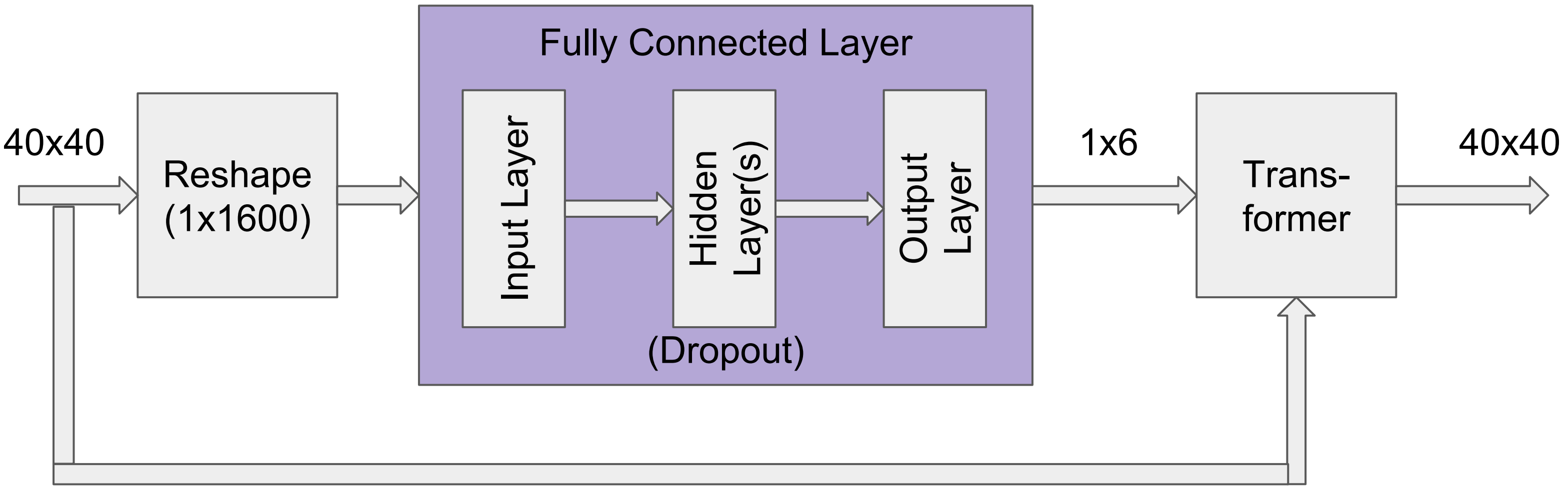
STN Digit Identification Network



STN Digit Identification Network

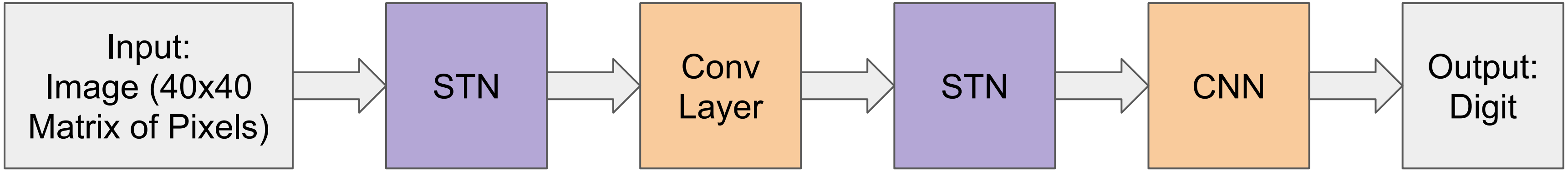


STN Design



40x40
Reshape (1x1600)
Fully Connected Layer
Input Layer
Hidden Layer(s)
Output Layer
(Dropout)
1x6
Trans- former
40x40

MSTN Digit Identification Network

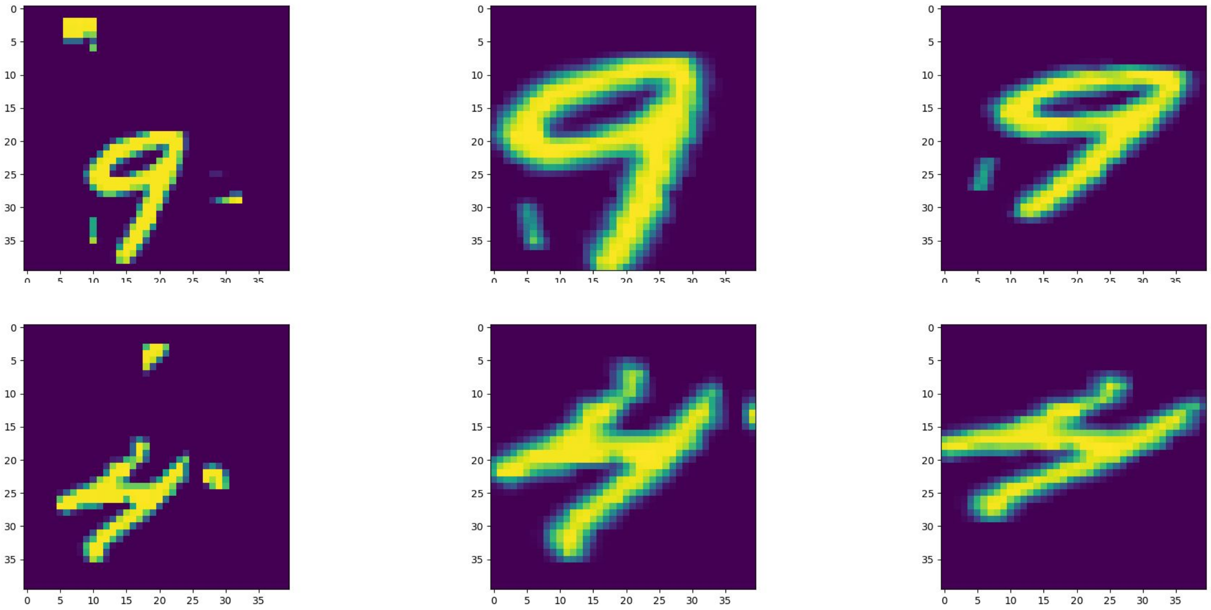


Test Process of the Networks

- All 10,000 train images and 1,000 test images from Cluttered MNIST data set were used in each trial
- Each network training had 3 runs of 100, 200, 300, 400, and 500 epochs
- 50 iterations / epoch

Sampled STN Results

Original | STN | MSTN

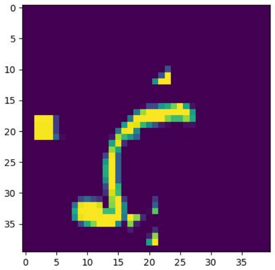


Harder Digits to Identify

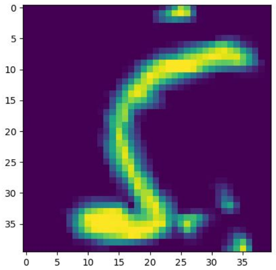
Original

STN

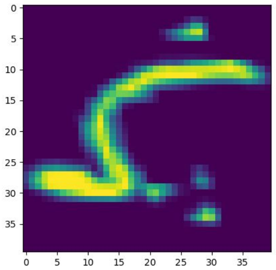
MSTN



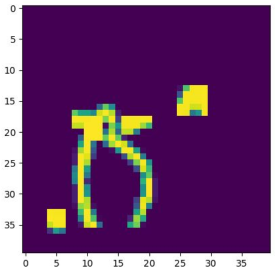
Digit: 5



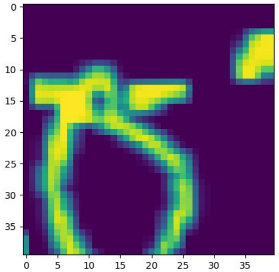
(Correct)



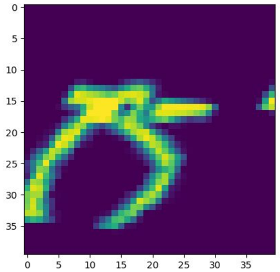
(Correct)



Digit: 0

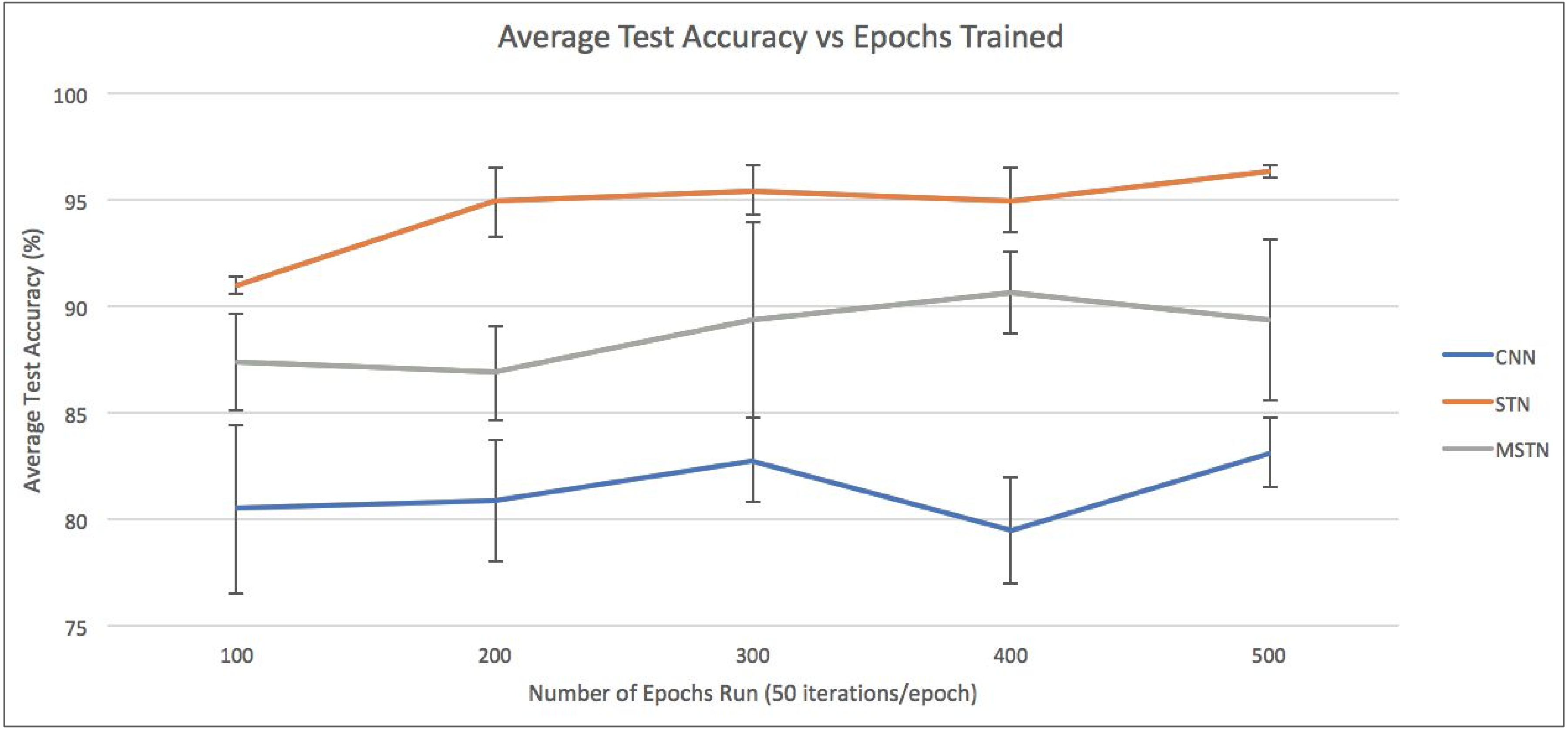


(Incorrect)



(Incorrect)

Comparison of Network Results



Future Work

- Train/test our models with a larger data set
- More complex ConvNet Models for STN and main CNN

Questions?