Deep Learning in Computer Vision

Handwritten Digits Recognition Using LeNet

Instructor: Dr. Mehrandezh
Student: Marzieh Zamani

Traditional Feedforward NN vs. Convolutional NN

- Traditional NN =>All layers *fully connected* (FC)

- CNNs => FC layers only as the *very last layer(s)* .

Convolutional NN Key Advantageous

- ***Local Invariance***
  - Classify an image as containing a particular object *regardless* of where in the image the object appears.

- ***Compositionality***
  - Learn rich features deeper in the network. (e.g. build edges from pixels, shapes from edges, and then complex objects from shapes)

Convolutional NN Layer Types

- **Layer Types**
  - **Convolutional (CONV)**: the core building block of a CNN
  - **Activation (ACT or RELU)** : always following CONV (CONV => ACT)
  - **Pooling (POOL)**: to reduce the size of an input volume
  - **Fully-connected (FC)**: always placed at the end of the network
  - **Batch normalization (BN)**: to normalize the activations of a given input
  - **Dropout (DO)**: a form of *regularization* that aims to help prevent overfitting

- **CNN Network example**:
  - INPUT => CONV => RELU => FC => SOFTMAX

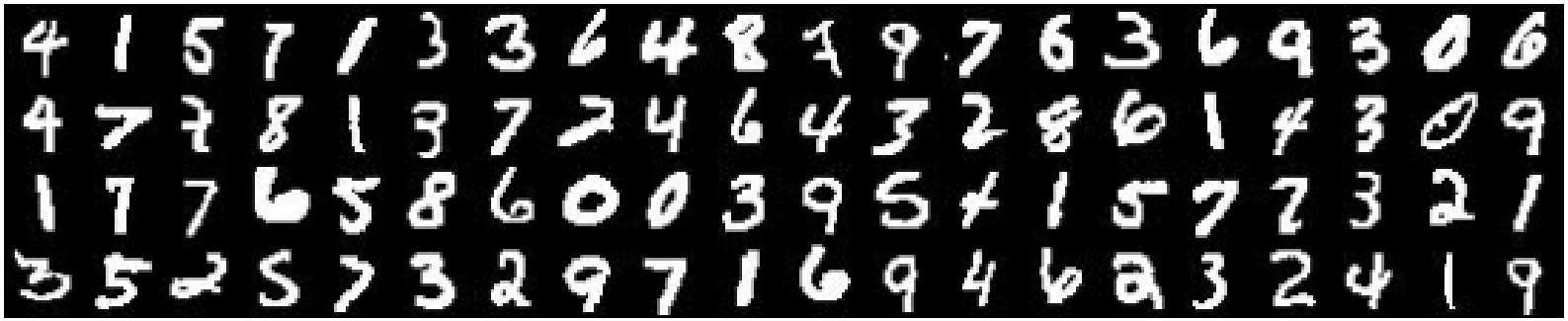
- **Learning layers**: CONV, FC, and, BN)

LeNet Network on MNIST

Layer Type	Output Size	Filter Size / Stride
INPUT IMAGE	$28 \times 28 \times 1$	
CONV	$28 \times 28 \times 20$	$5 \times 5, K = 20$
ACT	$28 \times 28 \times 20$	
POOL	$14 \times 14 \times 20$	2×2
CONV	$14 \times 14 \times 50$	$5 \times 5, K = 50$
ACT	$14 \times 14 \times 50$	
POOL	$7 \times 7 \times 50$	2×2
FC	500	
ACT	500	
FC	10	
SOFTMAX	10	

Implementation
Step #1 – Loading the MNIST dataset

- MNIST Dataset for Training and Validation
  - 60,000 training images
  - 10,000 testing images



- Python Code

```
((trainData, trainLabels), (testData, testLabels)) =
mnist.load_data()
```

Implementation
Step #2 – Instantiating the LeNet architecture

Layer Type	Output Size	Filter Size / Stride	Implementation in Python
INPUT IMAGE	28×28×1		model = Sequential() inputShape = (height, width, depth)
CONV	28×28×20	5×5;*K* = 20	model.add(Conv2D(20, (5, 5), padding="same", input_shape=inputShape))
ACT (ReLU)	28×28×20		model.add(Activation("relu"))
POOL	14×14×20	2×2	model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
CONV	14×14×50	5×5;*K* = 50	model.add(Conv2D(50, (5, 5), padding="same"))
ACT (ReLU)	14×14×50		model.add(Activation("relu"))
POOL	7×7×50	2×2	model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
FC	500		model.add(Flatten()) model.add(Dense(500))
ACT (ReLU)	500		model.add(Activation("relu"))
FC	10		model.add(Dense(classes))
SOFTMAX	10		model.add(Activation("softmax"))

Implementation
Step #3 – Training and Fitting LeNet

- Model is trained using the training batch from MNIST dataset.

- Python Code:

```
opt = SGD(lr=0.01)
model.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy"])
# train the network
print("[INFO] training network...")
H = model.fit(trainData, trainLabels,
validation_data=(testData, testLabels), batch_size=128,
epochs=20, verbose=1)
```

Implementation
Step #4 – Evaluating network performance

- Then, the trained model will be evaluated using the testing batch from MNIST dataset.

- Python Code:

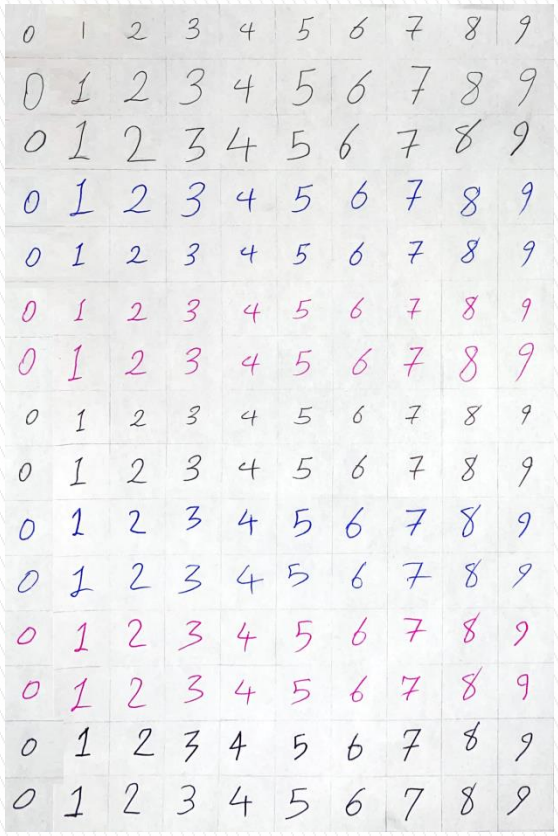
```
print("[INFO] evaluating network...")
predictions = model.predict(testData, batch_size=128)
print(classification_report(testLabels.argmax(axis=1),
predictions.argmax(axis=1), target_names=[str(x) for x in
le.classes_]))
```

Implementation
Step #5 – LeNet on Handwritten digits

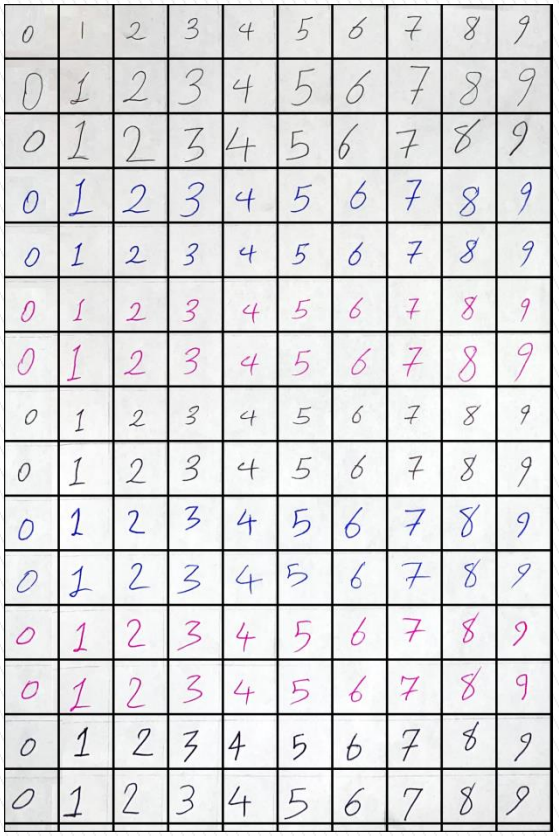
- 150 handwritten digits (written by different hands and pens)
- Pre-processed to following batches:

1. "**none**" batch => Plain scanned images
2. "**thresh**" batch => Smoothing and thresholding scanned images
3. "**crop_tight**" batch => Cropping empty background around digits
4. "**crop_sq**" batch => Padding cropped digits to make them squared
5. "**pad_[scale]**" batch => Padding squared images with scales = {1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0}

Implementation
Step #5 – LeNet on Handwritten digits



Scanned handwritten digits



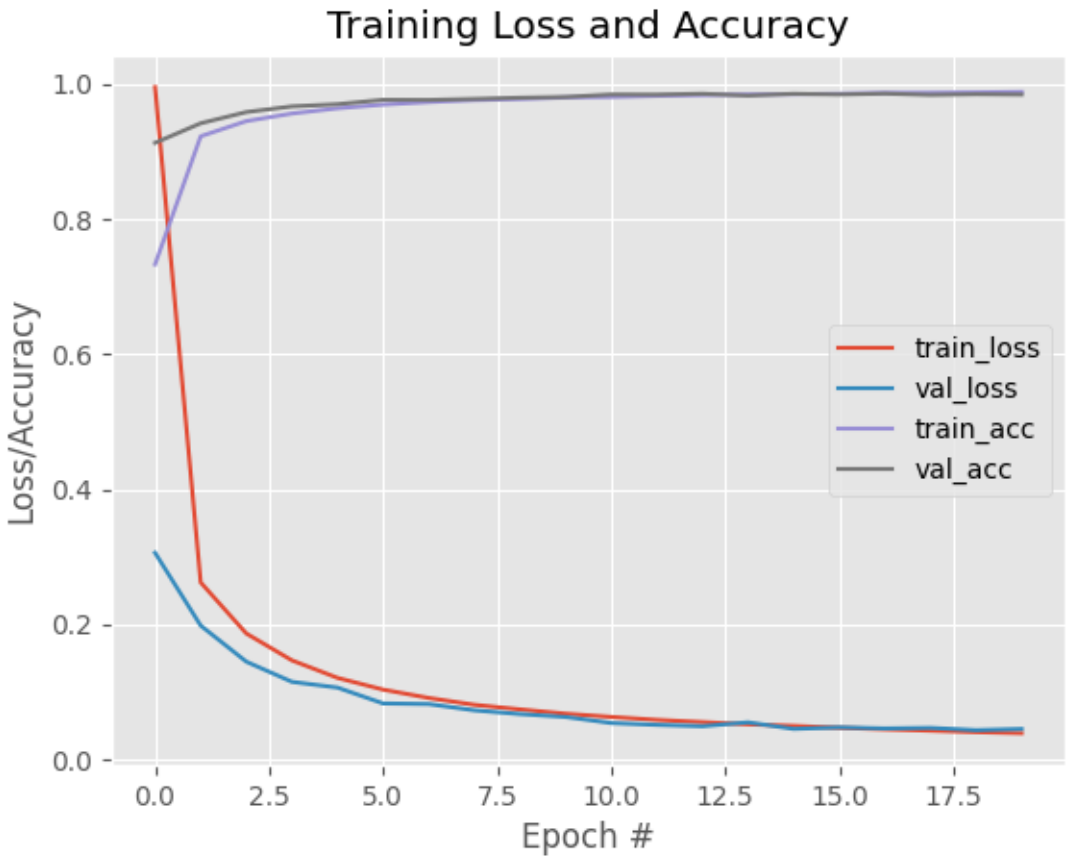
Partitioned handwritten digits

Implementation
Step #5 – LeNet on Handwritten digits

5 pre-processing styles:
{'none', 'thresh', 'crop_tight', 'crop_sq', 'pad_1.5'}

11 padding scales:
{1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0}

Results:
Training LeNet on MNIST



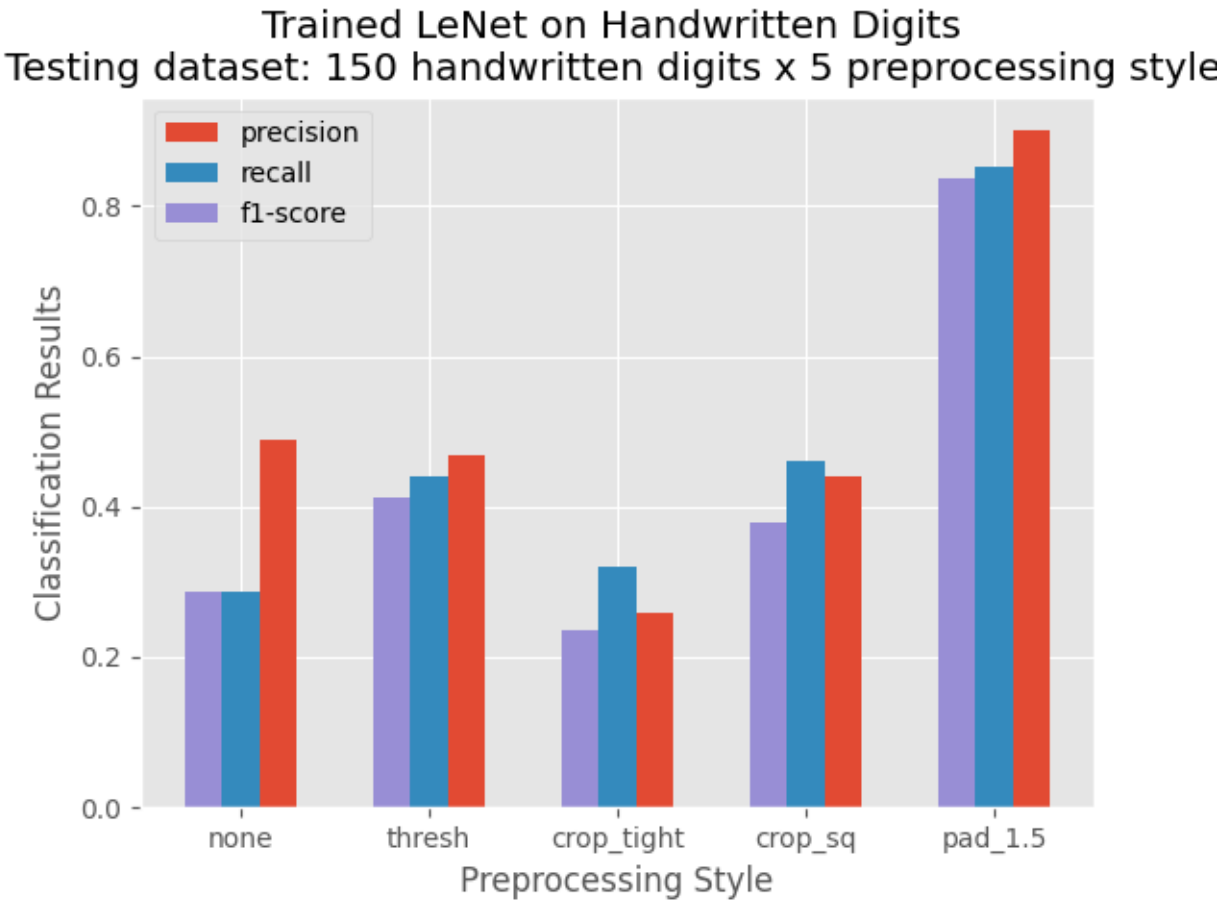
Results:
Training LeNet on MNIST

- **Loss**
  - Training and validation loss are have both decreased to 0.05. There are some fluctuations in validation loss until epoch 15 but the amplitude reduces to an acceptable amount. We can safely say that this network is not overfitted.

- **Accuracy**
  - With a similar pattern, training and validation accuracy have increased to 98% which is impressive. Again, there are some rise and fall in validation accuracy until epoch 15 but the amplitude reduces to a negligible amount.

- **Number of epochs**
  - It is worth mentioning that the required number of epochs is comparatively low. The network reaches 96% accuracy at epoch 5 and the final 98% validation accuracy is obtained with only 20 epochs.

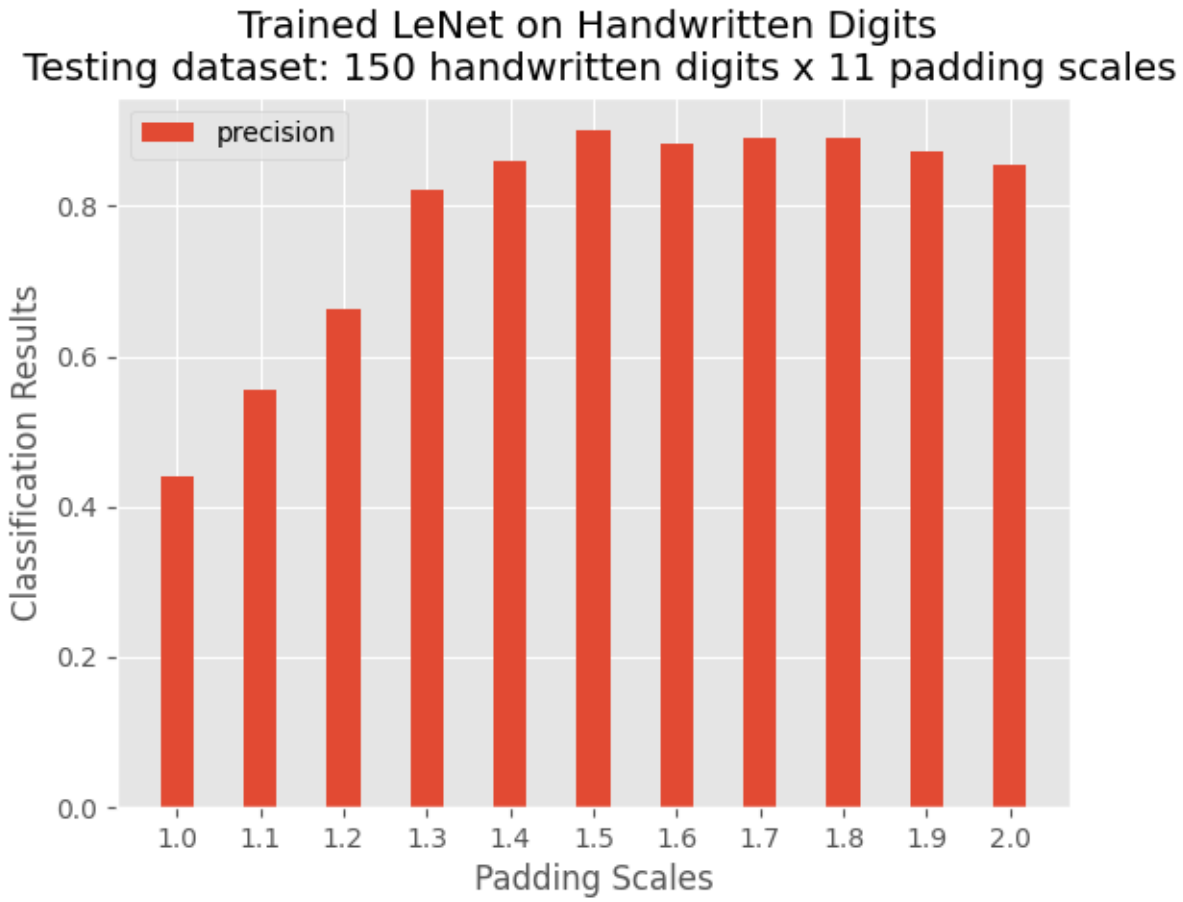
Trained LeNet on Handwritten Digits
[150 handwritten digits x 5 pre-processing styles]



Trained LeNet on Handwritten Digits [150 handwritten digits x 5 pre-processing styles]

- “none” => 49%
  - Plain scanned images are classified with 49% precision, significantly lower than 98% precision for MNIST testing dataset.
- “thresh” => 47%
  - Smoothing and thresholding alone do not have significant effect on classification precision.
- “crop_tight” => 26%
  - Tightly cropped digits do not have equal width and height and therefore will lose their original ration when being resized to 28x28. This change of width/height ratio has significantly reduced the precision to 26%.
- “crop_sq” => 44%
  - By padding cropped digits to squared dimension, their width/height ratio will remain constant during resizing. However, the classification accuracy is still lower than plain images.
- “pad_1.5” => 90%
  - Padding squared images with scales higher than 1 has a significant positive effect on the accuracy. Scale 1.5 results in highest precision which is 90

Trained LeNet on Handwritten Digits [150 handwritten digits x 11 padding scales]



Trained LeNet on Handwritten Digits [150 handwritten digits x 11 padding scales]

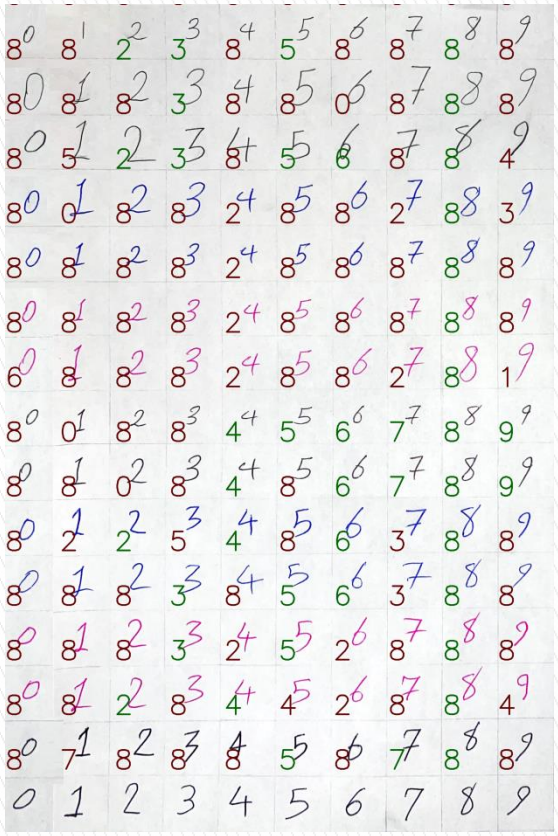
- Due to the significant effect of padding scale on classification precision, we tested 11 scales ranging from 1.0 to 2.0.
- Padding squared images with scales higher than 1 has a significant positive effect on the accuracy.
- The 44% precision for scale 1.0 has increased to 90% for scale 1.5 results.
- This is probably because this scale is closest to the scale of MNIST dataset.

Conclusion

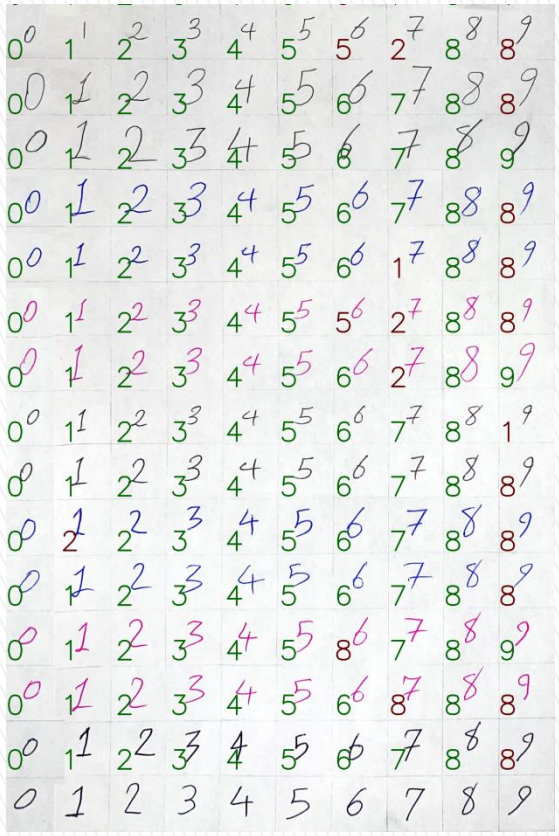
- LeNet network trained and tested using MNIST dataset obtains an excellent accuracy of 98% with only 20 epochs.
- When trying trained LeNet model on handwritten images:
  - Plain scanned images are classified with 49% precision, significantly lower than 98% precision for MNIST testing dataset.
  - Thresholding and smoothing scanned digits do not have any significant effect on classification precision.
  - Change of width/height ratio decreases the precision to 26%.
  - Padding squared images with scales higher than 1 has a significant positive effect on the accuracy. Scale 1.5 results in highest precision which is 90%.

Labeling digits with classification results

Note: Correct labels printed in green, incorrect labels printed in red.



‘none’ data batch
(49% precision)



‘pad_1.5’ data batch
(90% precision)

References

- Rosebrock, A. (2017). Deep learning for computer vision with python: starter bundle. PyImageSearch.
- MNIST Dataset: http://yann.lecun.com/exdb/mnist/

Thanks for your attention

Questions are welcome