Kaggle Task: Digit Recognizer Presentation

Xia Rui

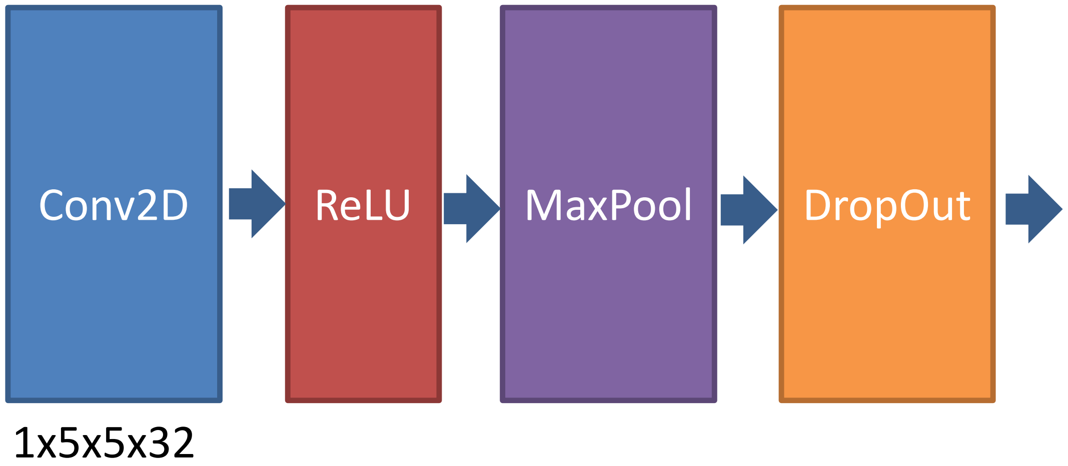
Resources

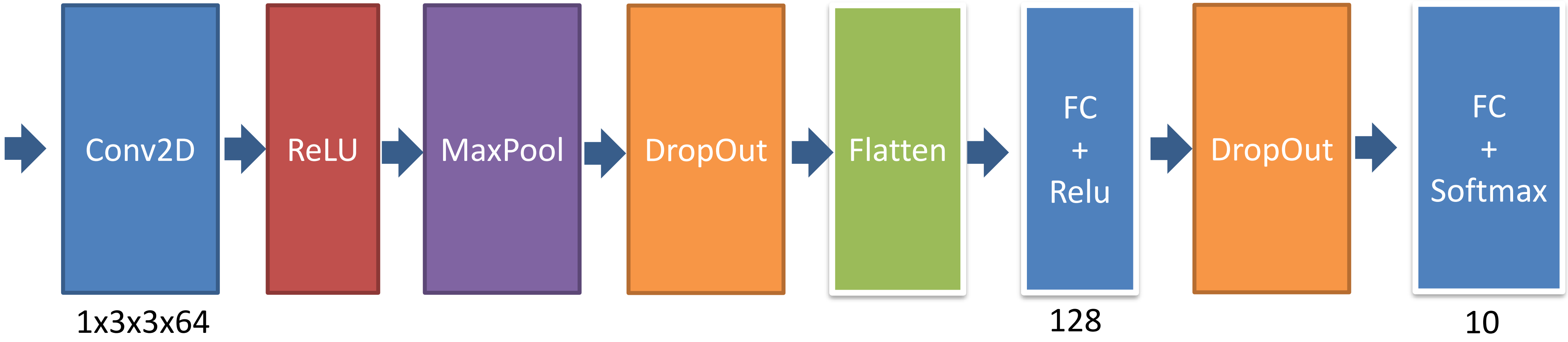
- Stanford University cs231n Convolutional Neural Networks for Visual Recognition
- Python / Jupyter Notebook / Keras
- Research Articles on Batch Normalization / softmax + crossentropy
- Other’s kernel

Basic Models

Idea:

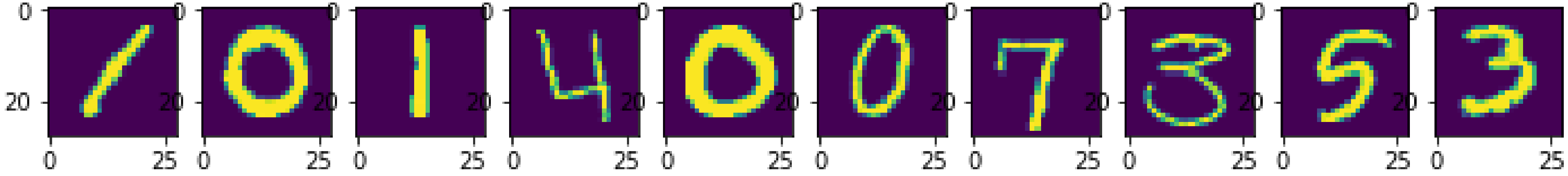
1. Leaky ReLU
2. Add more Conv2D layers
3. Batch Normalization
4. Increase epochs
5. Data argumentation





Data Preparation

label	pixel 0	pixe l1	pixel 2	pixel 3	pixel 4	pixel 5	pixel 6	pixel 7	pixel 8	...	pixel 773	pixel 774	pixel 775	pixel 776	pixel 777	pixel 778	pixel 779	pixel 780	pixel 781	pixel 782	pixel 783
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0



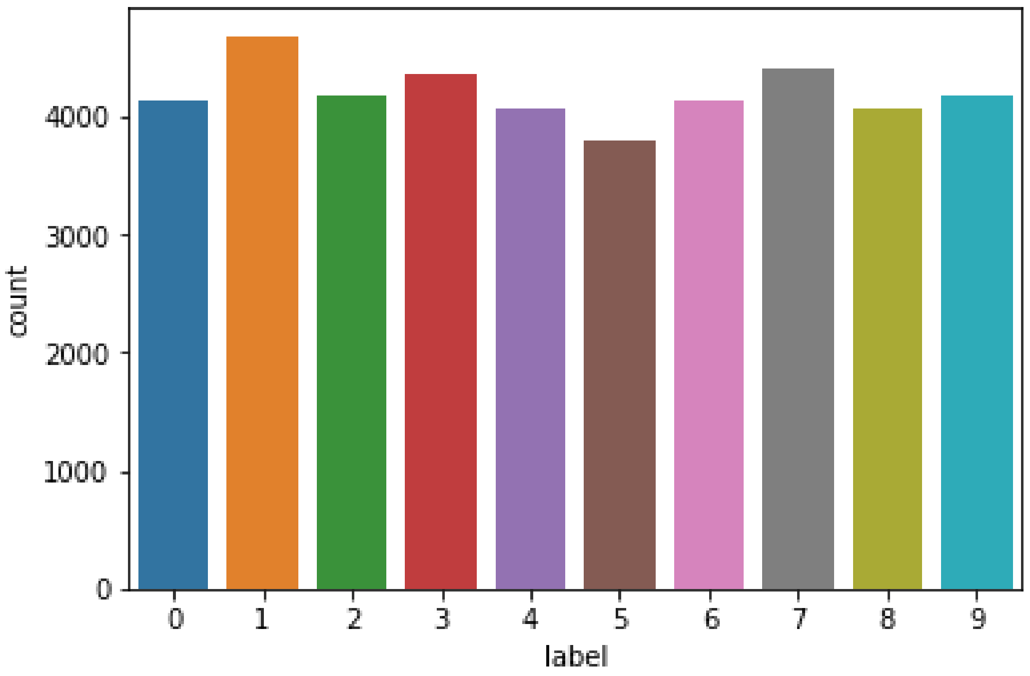
784 Pixels = 28x28 Pixels   integer 0-255   10 digits: from 0-9

Data Preparation

- Data Distribution

- Checking for Missing Values

- Normalization /255
- To reduce the illumination's influence
- Gradient Descent will converge faster on input range (0,1) than (0,255)



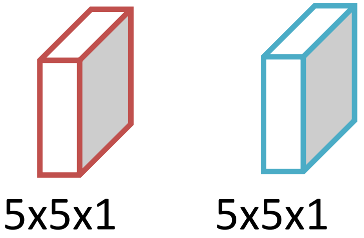
$$\frac{x - x_{min}}{x_{max} - x_{min}}$$

- Data reshape 1x784 to 28x28

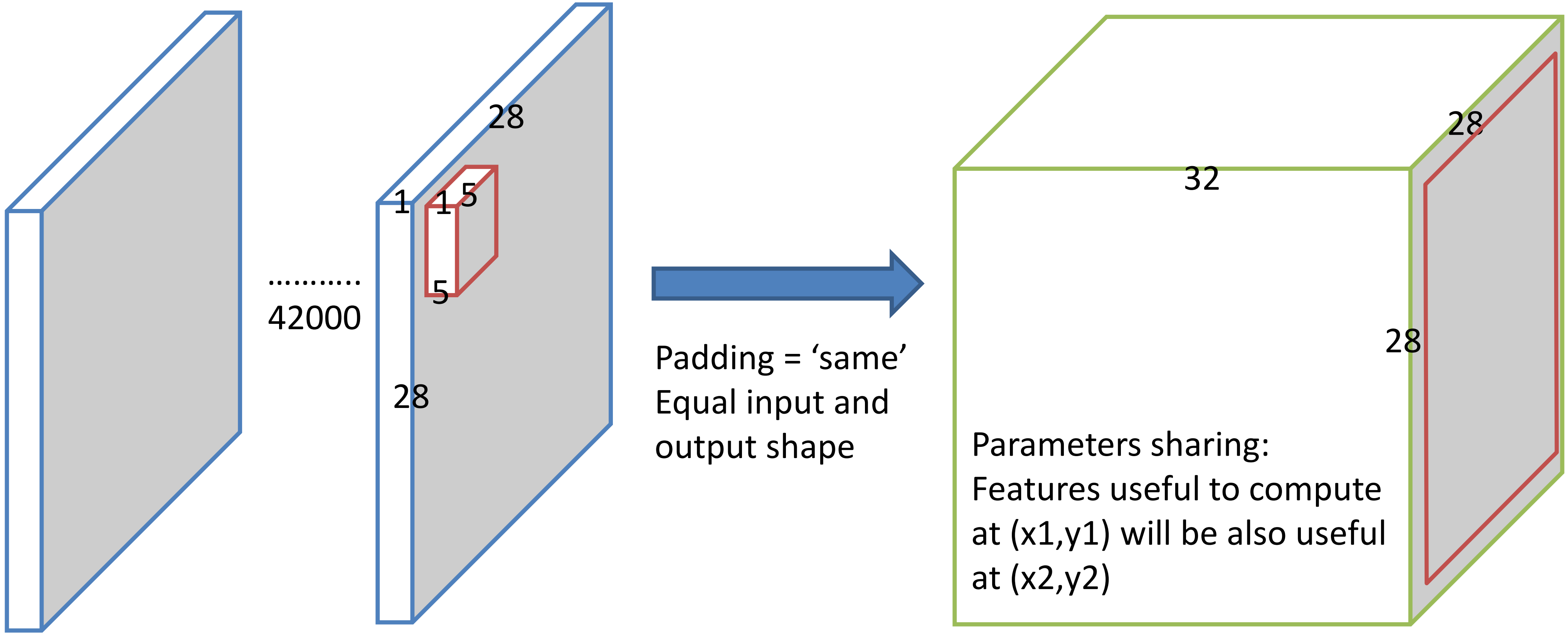
- Encode labels to one hot vectors 9 -> (0,0,0,0,0,0,0,0,0,1)
- Convert a class vector (integers) to binary class matrix

- Split the training set to training set and validation set for fitting

Convolutional Layers



FC: 28x28x1 too large
Conv: connected to a small region of the layer



28x28 each neurons calculate gradients

Σ

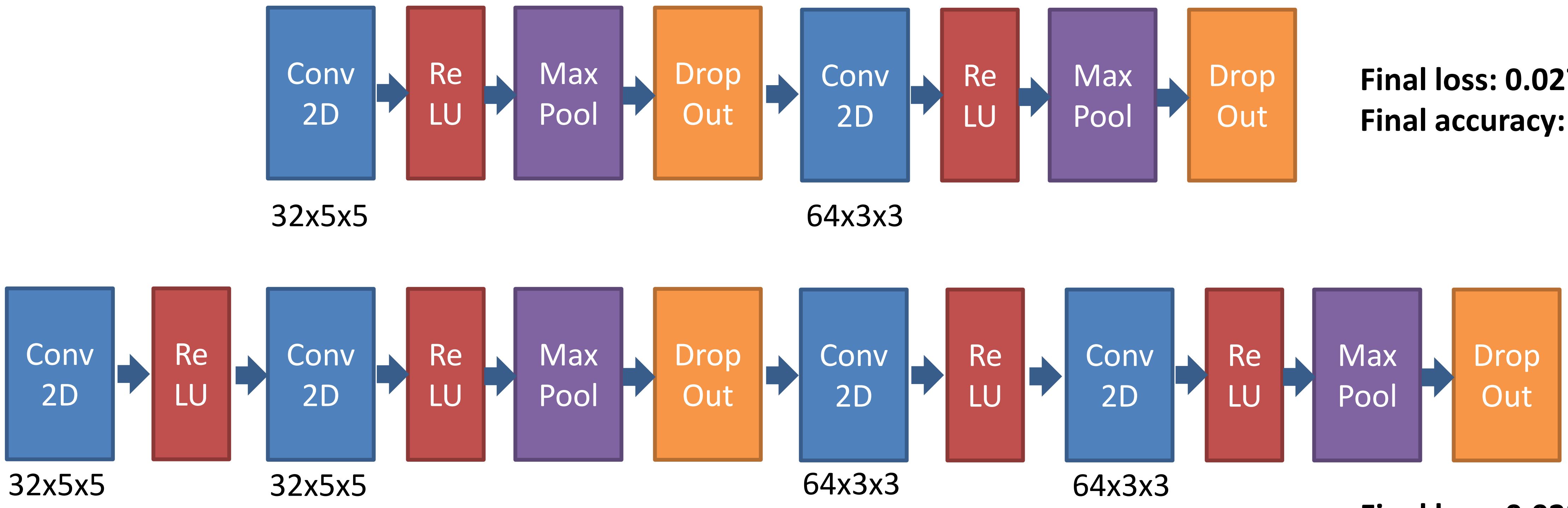
Update a single 5x5x1 weights

28x28x32 neurons
5x5x1 parameters

32x5x5x1 parameters

Convolutional Layers

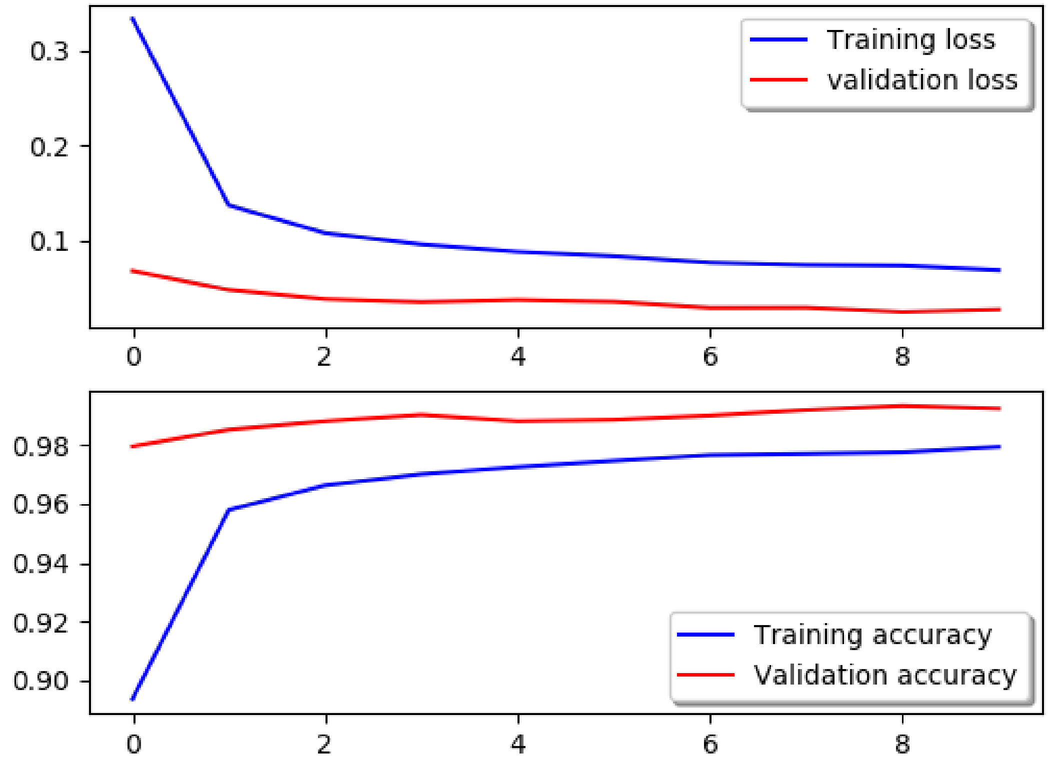
32x5x5 overall features, more large-scale features
64x3x3 local features, more detailed features, more filters

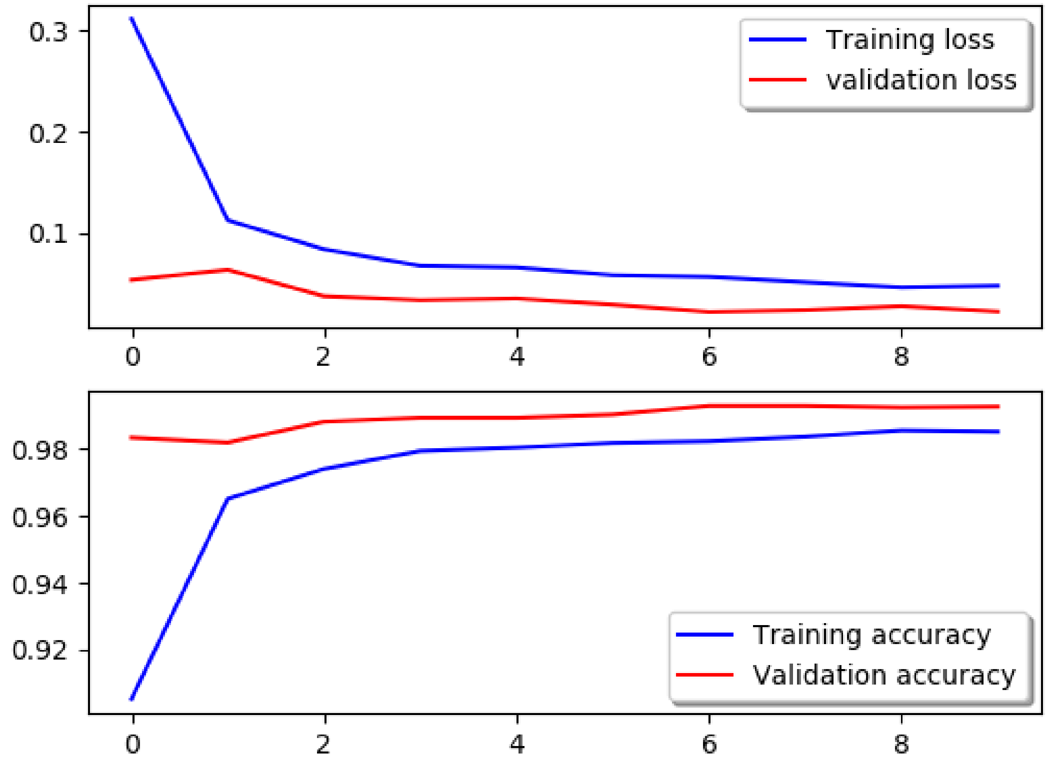


Final loss: 0.0277
Final accuracy: 0.9923

Final loss: 0.0226
Final accuracy: 0.9926

Convolutional Layers





Training curves are closer to the validation curves --- more Convolutional layers tend to overfit more

Data Argumentation

Reason for why validation loss curve is lower
Training harder to identify

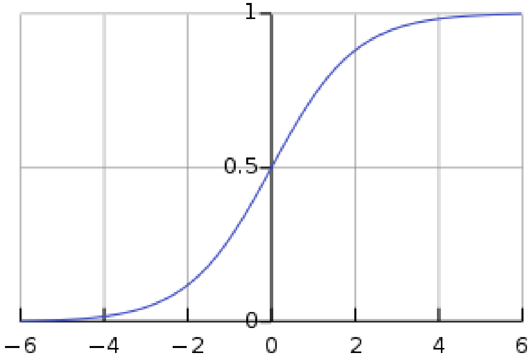
Not to overfit, artificially expand the dataset

Someone will write bigger/smaller numbers, scale
Rotate
Shift horizontally/vertically

```
datagen = ImageDataGenerator(
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1)
```

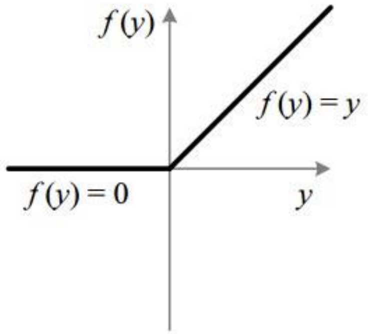
Activation Layers

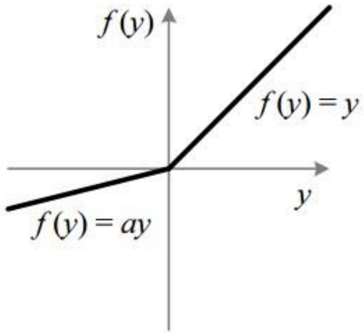
Add non-linearity to the models



Sigmoid & Tanh

- Gradient vanishing (derivative = g(x)(1-g(x)))
- Not zero-centred (always greater than 0, zig-zagging dynamics)
- Time-consuming to calculate





ReLU max(0,x)

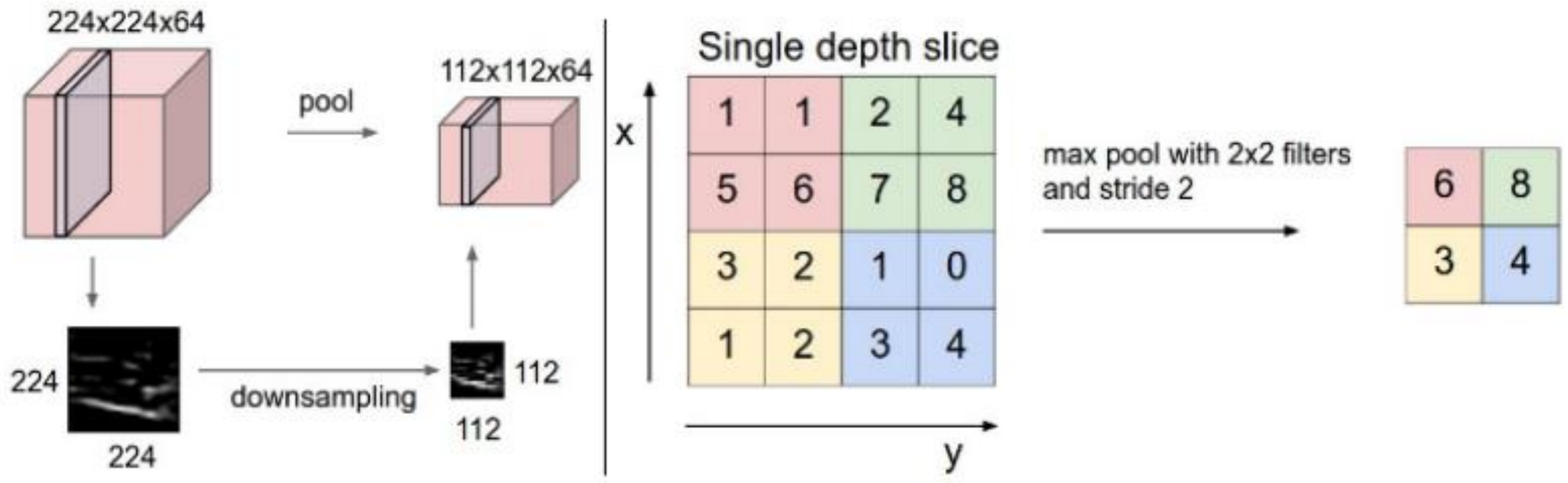
- Not zero-centred
- Simply thresholding
- Can ‘die’: Every input x put ReLU zero → die
  At least one x activate ReLU (fixed by small lr)

Leaky ReLU max(0.01x, x)

- Should be better since solved not zero-centred problem
- **Accuracy decrease from 0.9926 to 0.9864**
- **Error raise from 0.0226 to 0.0431**

MaxPooling Layers

Down-sampling, pick maximum value in a 2x2 square
Reduce computation cost, reduce overfitting (reduce dimension)
28x28 → 14x14



Dropout Layers

Randomly ignore some nodes in layer (Making new nns)
Forces network to learn in a distributed way

≈ training different neural network and then take average

Batch Normalization Layers

- Internal covariance shift
- Change in the distribution of network activations due to the change in network parameters during training

- $F_2(F_1(x, \theta_1), \theta_2)$, θ_2 does not have to readjust to compensate for the change in the distribution of x

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

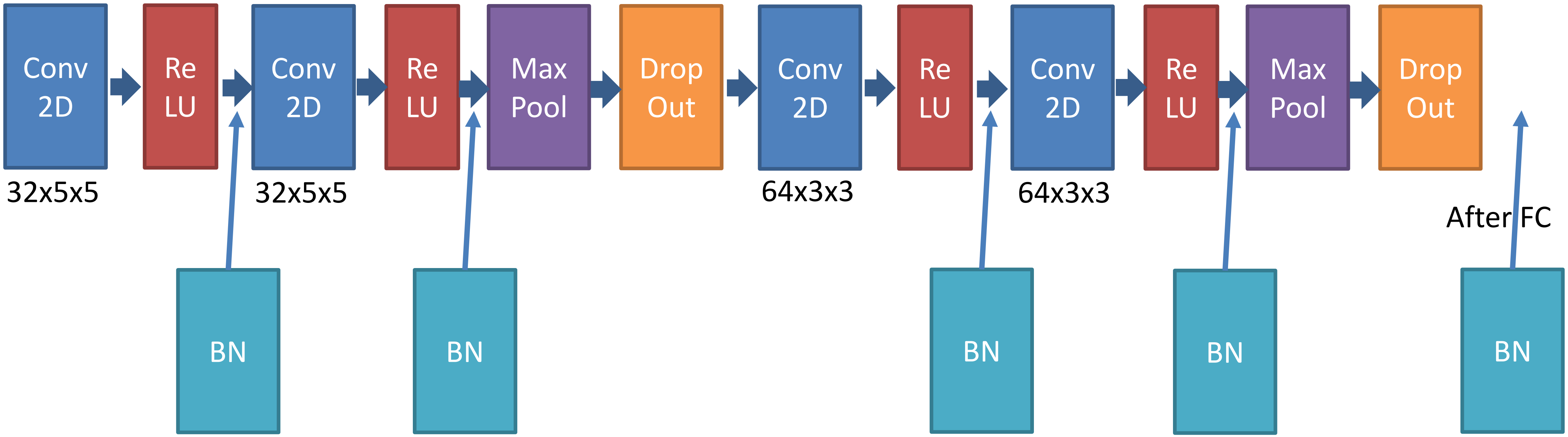
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Normalize each scalar feature independently
- Mini-batch estimate mean & variance
- New parameters are learned, in order to restore the representation power of the network

Batch Normalization Layers



4Conv2D + BN + ReLU
Accuracy: 0.9926 Error: 0.0226

4Con2D + ReLU
Accuracy: 0.9905 Error: 0.0323

Flatten + FC + Softmax Layers

- After Flatten: 1D vector
- After Last FC layer: 1D vector: 10x1x1 (10 categories)

- $\sigma(x) =$ Softmax(x) = normalized(exp(x)) = $\frac{\exp(x_i)}{\sum_j \exp(x_j)}$
- Endow score with meaning(probability distribution), Sum up to be 1

- Softmax + categorical_crossentropy (loss function, categorical classification >2)

- When calculating gradient descent, derivative of softmax(x) =$\sigma(x)$(1-$\sigma(x)$)
- Get eliminated by the derivative of cross-entropy loss

Optimizer

Functions to iteratively improve parameters

- SGD (slow), mini-batch Gradient Descent (partial)
  - Every time different batch of inputs (not stable)

- Momentum
  - Consider previous gradients

$$v_t = \gamma \cdot v_{t-1} + \alpha \cdot \nabla_\Theta J(\Theta)$$

$$\Theta = \Theta - v_t$$

- Adagrad

$$n_t = n_{t-1} + g_t^2$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} * g_t$$

- RMSprop

4Conv2D + ReLU + RMSprop
Accuracy: 0.9905 Error: 0.0323

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\Theta_{t+1} = \Theta_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t$$

- Adam

4Conv2D + ReLU + Adam
Accuracy: 0.9933 Error: 0.0232

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) g_t^2$$

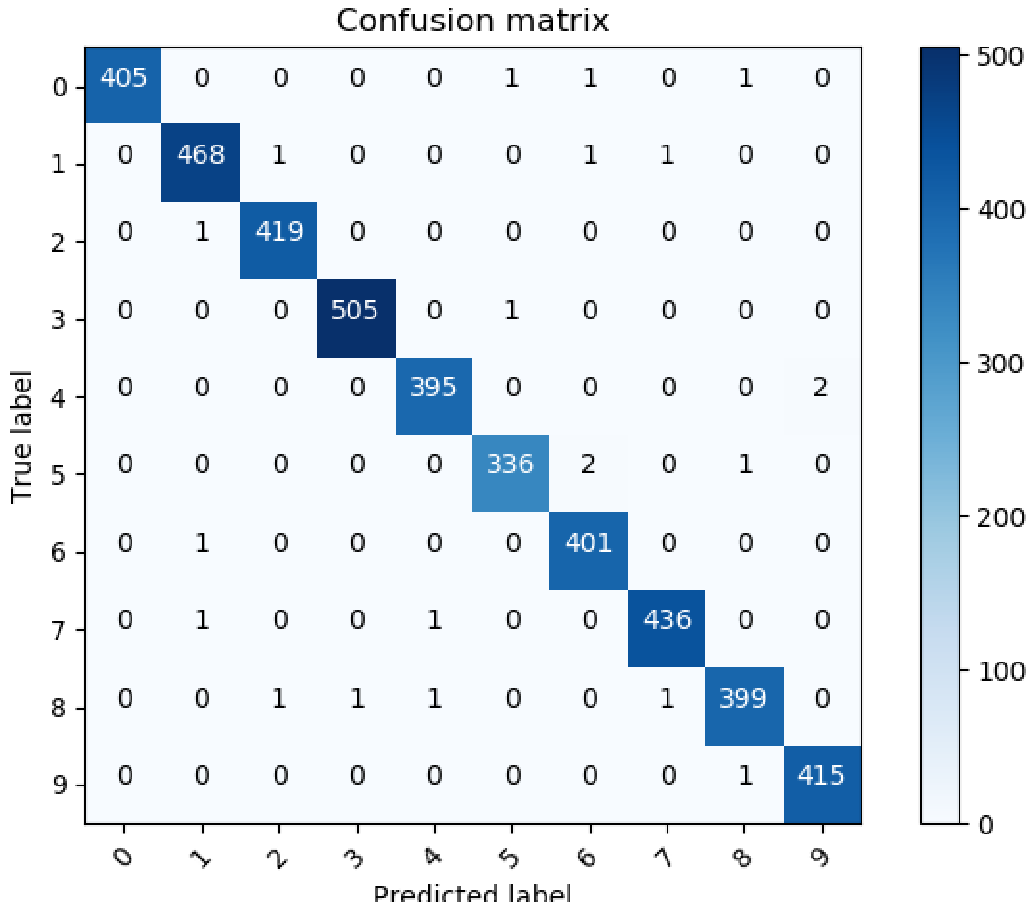
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\Theta_{t+1} = \Theta_t - \frac{\alpha}{\sqrt{\hat{v}_t}+\epsilon} \hat{m}_t$$

Result Evaluation : Confusion Matrix & Classification Report

[Conv2D + ReLU + BN] x2 + MaxPooling + DropOut + [Conv2D + ReLU + BN] x2 + MaxPooling + DropOut + Flatten + FC + DropOut + FC + Softmax
Epochs → 30

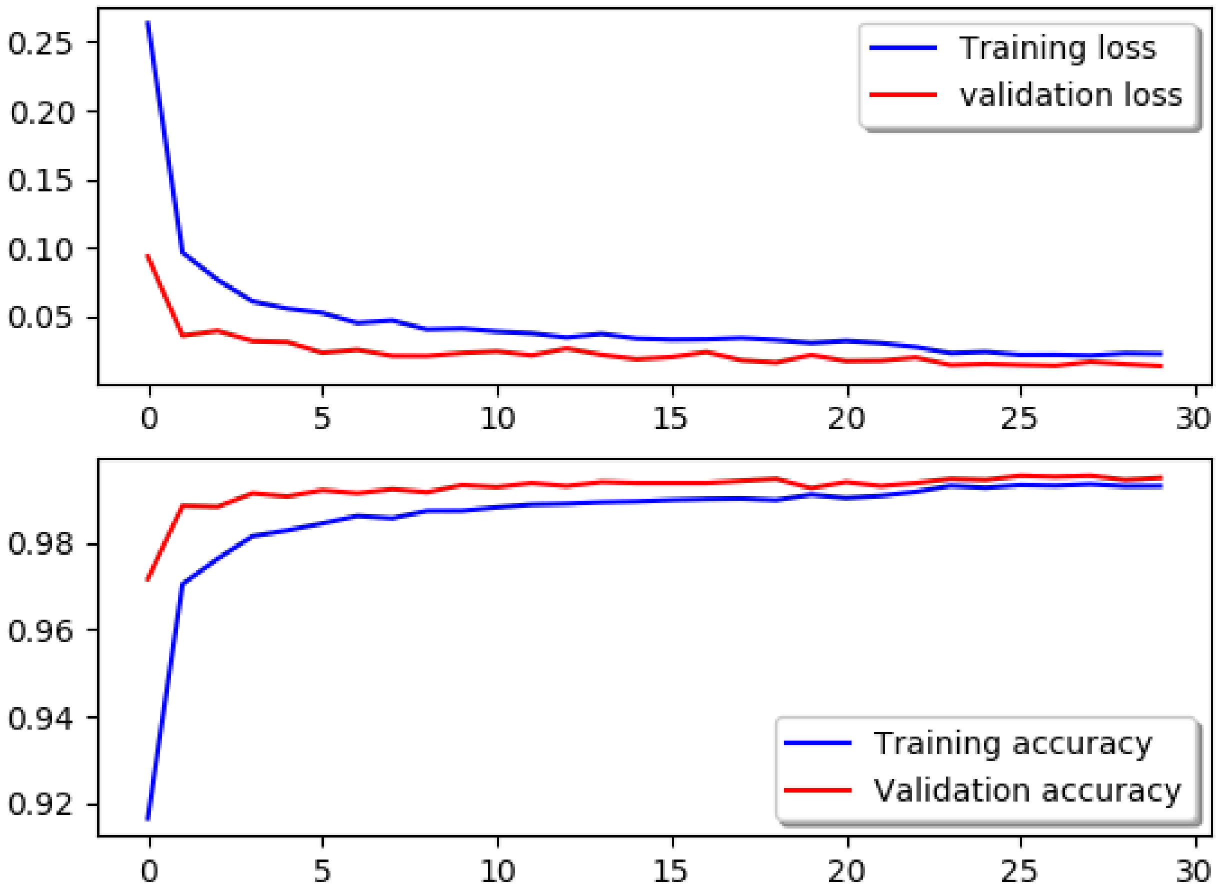


Final loss: 0.013653, final accuracy: 0.995000

	precision	recall	f1-score	support
0	1.00	0.99	1.00	408
1	0.99	0.99	0.99	471
2	1.00	1.00	1.00	420
3	1.00	1.00	1.00	506
4	0.99	0.99	0.99	397
5	0.99	0.99	0.99	339
6	0.99	1.00	0.99	402
7	1.00	1.00	1.00	438
8	0.99	0.99	0.99	403
9	1.00	1.00	1.00	416
avg / total	1.00	0.99	0.99	4200

Result Evaluation : Training & Validation Curves

[Conv2D + ReLU + BN] x2 + MaxPooling + DropOut + [Conv2D + ReLU + BN] x2 + MaxPooling + DropOut + Flatten + FC + DropOut + FC + Softmax
Epochs → 30



Result Evaluation : Top 6 Errors

[Conv2D + ReLU + BN] x2 + MaxPooling + DropOut + [Conv2D + ReLU + BN] x2 + MaxPooling + DropOut + Flatten + FC + DropOut + FC + Softmax
Epochs → 30