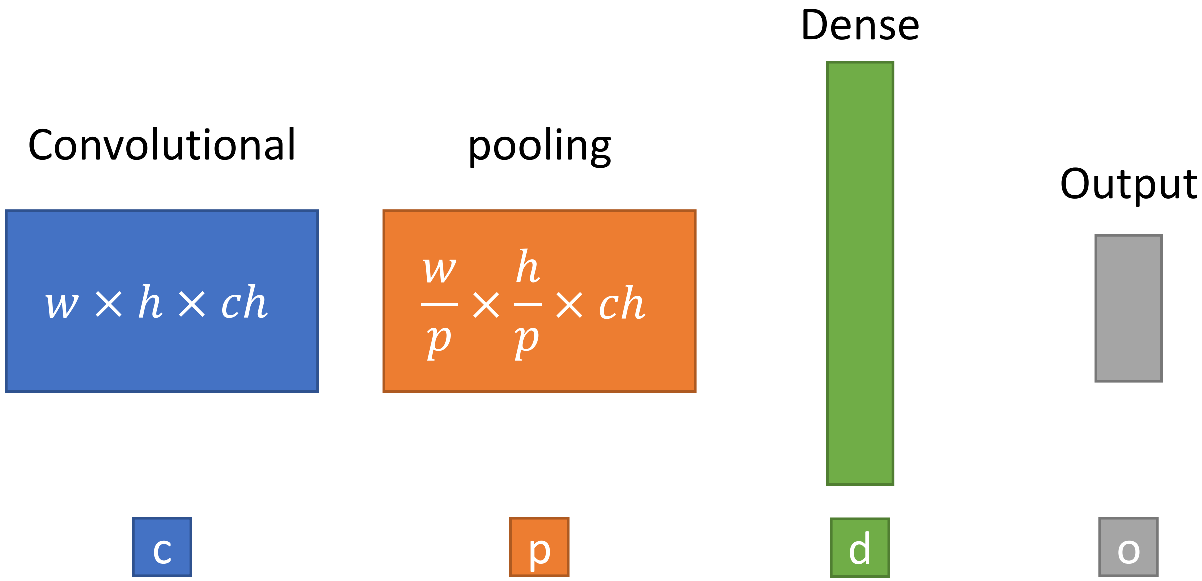
Classification Networks

CNN4N Journal Club

Amr Elsawy

Layers are Building Blocks

- We took different type of layers (convolutional, pooling, dense, and output).
- We can think of these layers as Lego blocks.



Data

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC).
- The ImageNet project is a large visual database designed for use in visual object recognition software research.
- More than 14 million labelled images (classification).
- 1 million of the images have bounding boxes.
- It contains more than 20,000 categories.

J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

https://www.image-net.org/
https://en.wikipedia.org/wiki/ImageNet

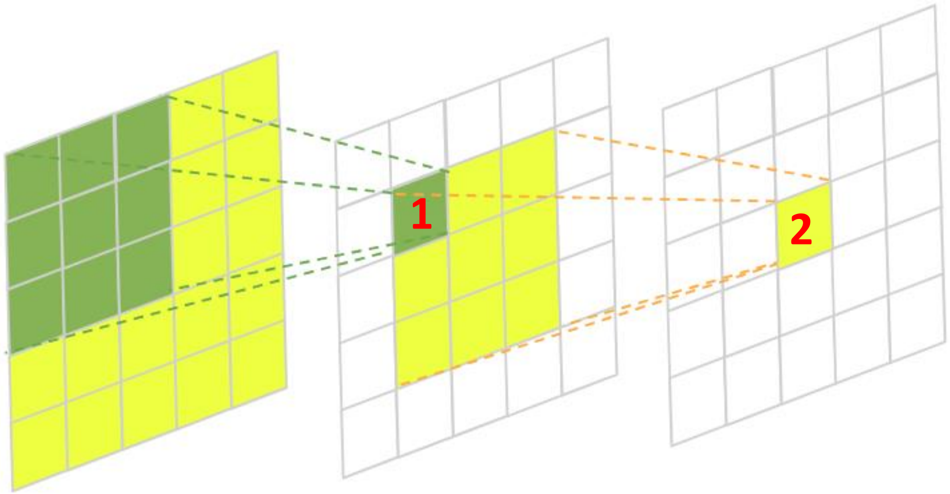
Cont.

- ILSVRC-2012 dataset.
- It has 1000 classes.
- Training (1.3M), validation (50K), and test (100K).

Receptive Field

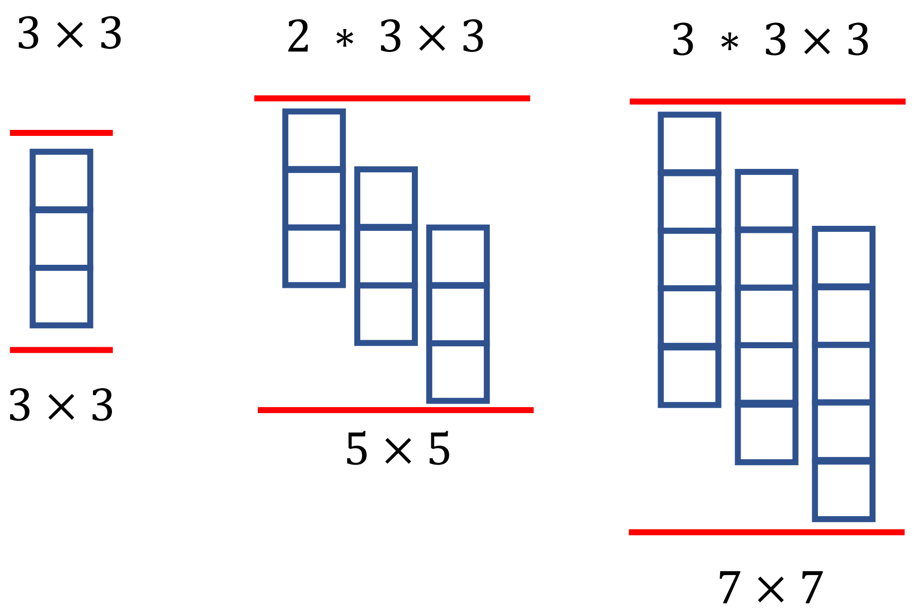
- We can think of it as the *net window* involving calculations with respect to the input image.

What are the sizes of regions 1 and 2?

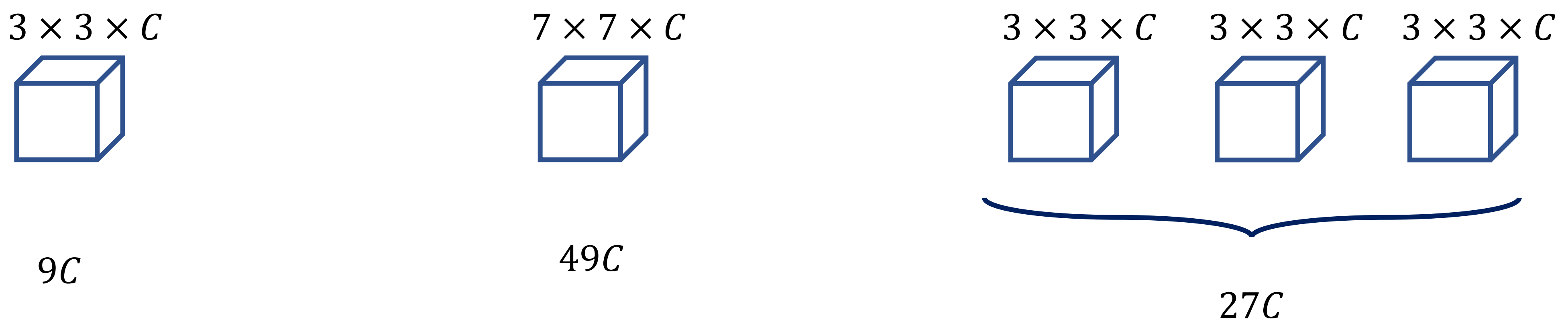


https://www.baeldung.com/cs/cnn-receptive-field-size

Cont.



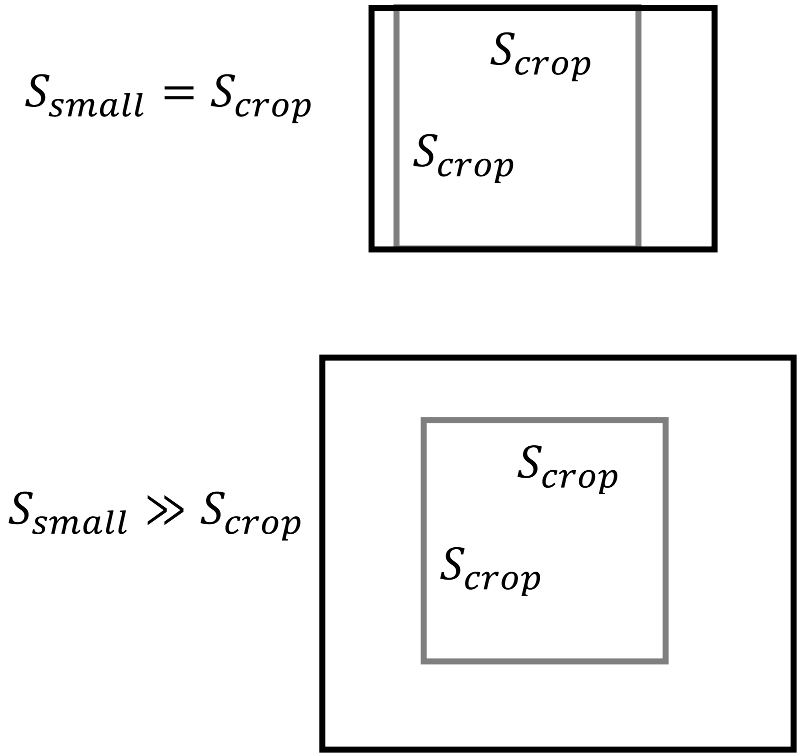
Cont.



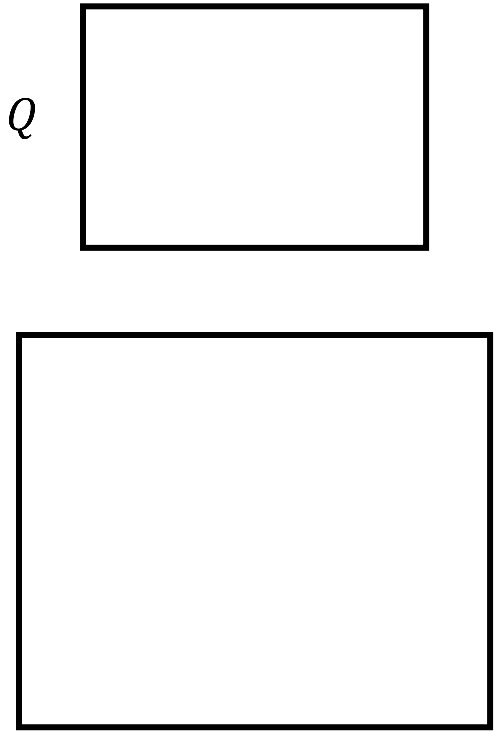
"This can be seen as imposing a regularization on the 7×7 conv. filters, forcing them to have a decomposition through the 3×3 filters."

Inputs

Training



Testing



S_{small} $is\ fixed$ 256 or 386 $or\ random\ in\ range[S_{min} = 256;\ S_{max} = 512]$

Evaluation

- Top-1 and top-5 errors.
- Single scale $Q = S$.
- Multiple scale $Q = 0.5(S_{min} + S_{max})$.

VGG Configurations

VGG

Weight layers (i.e., learnable layers) includes convolutional and dense layers.

Conv, size = 3x3, filters = 64

Channels = x, 2x, 4x, 8x, 8x

local response normalization (LRN)

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

ConvNet Configuration VGG16

A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64
	LRN	**conv3-64**	conv3-64	conv3-64 2	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
		conv3-128	conv3-128	conv3-128 2	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256 3	conv3-256
			conv1-256	**conv3-256**	conv3-256
					conv3-256
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512 3	conv3-512
			conv1-512	**conv3-512**	conv3-512
					conv3-512
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512 3	conv3-512
			conv1-512	**conv3-512**	conv3-512
					conv3-512
maxpool					
FC-4096					
ReLU FC-4096				3	
FC-1000					
soft-max					

Performance at Single Test Scale

Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E Fixed	256	256	27.3	9.0
	384	384	26.9	8.7
Jittering	[256;512]	384	**25.5**	**8.0**

Performance at Multiple Test Scale

Table 4: **ConvNet performance at multiple test scales.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

$Fixed\ \{S-32, S, S+32\}$

$Jitter\ \{S_{min}, 0.5(S_{min+}S_{max}), S_{max}\}$

Performance at Multicrop

Table 5: **ConvNet evaluation techniques comparison.** In all experiments the training scale S was sampled from $[256; 512]$, and three test scales Q were considered: $\{256, 384, 512\}$.

ConvNet config. (Table 1)	Evaluation method	top-1 val. error (%)	top-5 val. error (%)
D	dense	24.8	7.5
	multi-crop	24.6	7.5
	multi-crop & dense	**24.4**	**7.2**
E	dense	24.8	7.5
	multi-crop	24.6	7.4
	multi-crop & dense	**24.4**	**7.1**

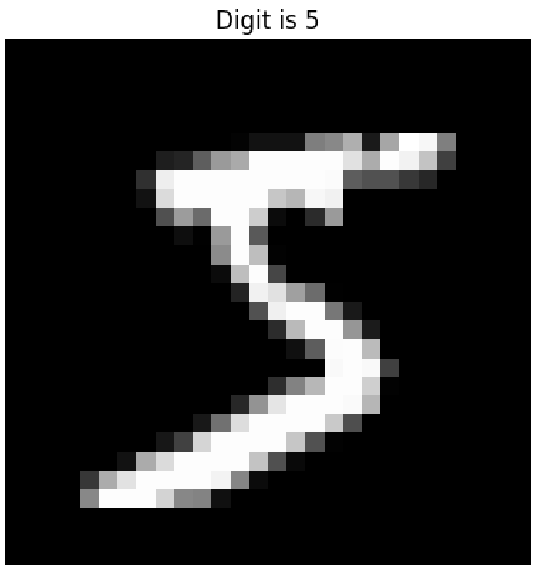
Demo

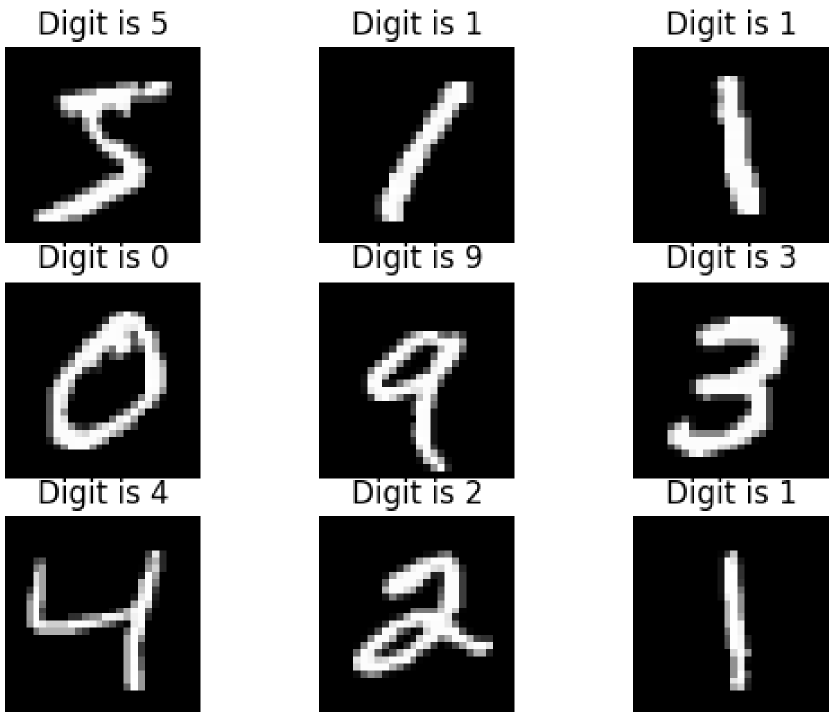
Classification of MNIST Dataset

Data Loading

```
keras.datasets.mnist.load_data(path='mnist.npz')

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```





Data Adjusting

```
x_train = np.expand_dims(x_train, axis=-1)

x_test = np.expand_dims(x_test, axis=-1)
```

```
y_train = keras.utils.to_categorical(y_train)

y_test = keras.utils.to_categorical(y_test)
```

Model Development Steps

- Define Layers
- Connecting Layers
- Defining Model
- Training Settings
- Fitting

Defining Layers

Think of layers as functions

```
# creating layers
# input
inputs = keras.layers.Input((28, 28, 1))

# 2 conv
conv1 = keras.layers.Conv2D(32, 3, 1, activation='relu', padding='same')
conv2 = keras.layers.Conv2D(64, 3, 1, activation='relu', padding='same')

# 2 pool
pool1 = keras.layers.MaxPool2D((2, 2))
pool2 = keras.layers.MaxPool2D((2, 2))

# flat
gap = keras.layers.GlobalAvergePooling2D()

# 2 dense
dense1 = keras.layers.Dense(64, 'relu')
dense2 = keras.layers.Dense(10)

# 2 dropouts
dropout1 = keras.layers.Dropout(0.5)
dropout2 = keras.layers.Dropout(0.5)

# output (softmax)
act = keras.layers.Activation('softmax')
```

Connecting Layers

Connections is a series of function calls

```
# connecting layers
x = conv1(inputs)
x = pool1(x)
x = conv2(x)
x = pool2(x)
x = gap(x)
x = dense1(x)
x = dropout1(x)
x = dense2(x)
x = dropout2(x)
x = act(x)
```

First input

Last output

Defining Model

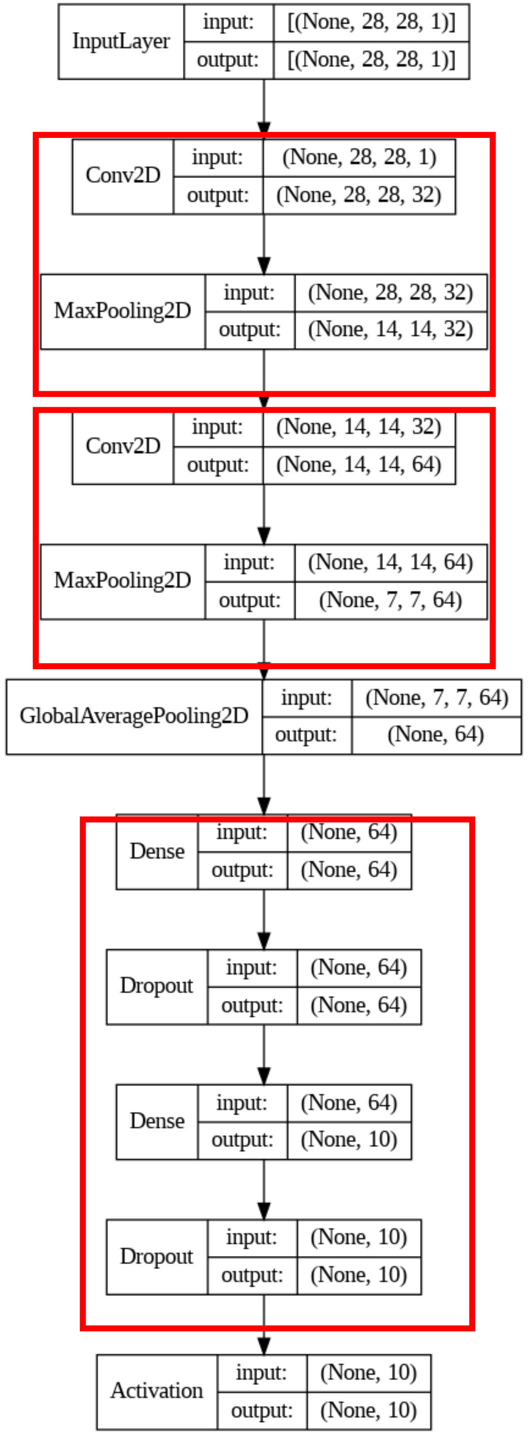
```
model = keras.models.Model(inputs=inputs, outputs=x)
```

https://nih.zoomgov.com/j/1602963310?pwd=dVlNUjQ1WGd3R3htMzJGc2pkWnNSdz09

First input

Last output

Network Visualization



Training Settings

```
model.compile(optimizer=keras.optimizers.Adam(0.001),
              loss=keras.losses.CategoricalCrossentropy(),
              metrics='acc')
```

Training (Fitting Model to Data)

```
model.fit(x_train, y_train, batch_size=25, epochs=10,
validation_data=[x_test, y_test], validation_batch_size=25)
```

Cont.

```
Epoch 1/10 2400/2400 [==============================] - 103s 41ms/step - loss: 2.3162 - acc: 0.2253 - val_loss: 1.7086 - val_acc: 0.6046
Epoch 2/10 2400/2400 [==============================] - 96s 40ms/step - loss: 1.8376 - acc: 0.3400 - val_loss: 1.2862 - val_acc: 0.7948
Epoch 3/10 2400/2400 [==============================] - 99s 41ms/step - loss: 1.6386 - acc: 0.4043 - val_loss: 0.9568 - val_acc: 0.8580
Epoch 4/10 2400/2400 [==============================] - 95s 40ms/step - loss: 1.5295 - acc: 0.4371 - val_loss: 0.8067 - val_acc: 0.8978
Epoch 5/10 2400/2400 [==============================] - 96s 40ms/step - loss: 1.4442 - acc: 0.4599 - val_loss: 0.6712 - val_acc: 0.9195
Epoch 6/10 2400/2400 [==============================] - 94s 39ms/step - loss: 1.3978 - acc: 0.4743 - val_loss: 0.5179 - val_acc: 0.9309
Epoch 7/10 2400/2400 [==============================] - 98s 41ms/step - loss: 1.3462 - acc: 0.4881 - val_loss: 0.4836 - val_acc: 0.9284
Epoch 8/10 2400/2400 [==============================] - 92s 38ms/step - loss: 1.3058 - acc: 0.4993 - val_loss: 0.4246 - val_acc: 0.9332
Epoch 9/10 2400/2400 [==============================] - 92s 38ms/step - loss: 1.2788 - acc: 0.5073 - val_loss: 0.3776 - val_acc: 0.9422
Epoch 10/10 2400/2400 [==============================] - 93s 39ms/step - loss: 1.2523 - acc: 0.5139 - val_loss: 0.3036 - val_acc: 0.9465
```

Further Reading

[1] https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/

[2] https://keras.io/guides/transfer_learning/

[3] https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/

Thanks ☺