Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference

정몽주, 박태준

공간통계연구실

2022년 8월

Contents

Section 1

Introduction

1. Introduction

- **Convolutional neural networks (CNNs)** are popular deep learning tool for image processing.
- CNNs require huge amounts of data for regularisation and quickly over-fit on small data.
- In contrast **Bayesian neural networks (NNs)** are robust to over-fitting, offer uncertainty estimates, and can easily learn from small datasets.

- Because of the vast number of parameters and extremely large models commonly used in practical applications.
- However, modelling a distribution over the kernels (also known as filters) of a CNN has never been attempted successfully before.

1. Introduction

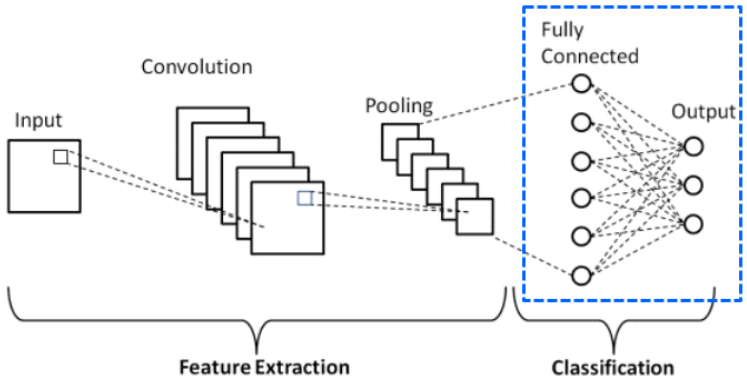


Figure: Convolutional Neural Networks (CNNs) Architecture Diagram

- **Dropout** is often not used after convolution layers.
- In existing literature, dropout is used in CNNs only after inner-product layers
- Because test error suffers, which renders small dataset modelling a difficult task.

1. Introduction

- We propose new practical dropout CNN architectures, mathematically identical to Bayesian CNN
- Our model is implemented by performing dropout after convolution layers.

- We show these models obtain better test accuracy compared to existing approaches in the field.
- We show that the proposed model reduces over-fitting on small datasets compared to standard techniques.
- We demonstrate improved results with MC dropout on existing CNN models in the literature.

Topic

Bayesian CNN with Variational Inference

- Casting dropout as variational inference in Bayesian neural networks.
- This Bayesian interpretation of dropout allows us to propose the use of MC dropout for convolutions.

Section 2

Bayesian Convolutional Neural Networks

Convolutional Neural Networks (CNNs)

- This section introduces **convolutional neural networks**, also known as **convnets**
- CNNs is a type of deep-learning model almost universally used in computer vision applications.

Convolutional Neural Networks (CNNs)

- Convolution layers learn local patterns.
- It gives two properties:
  1. The patterns they learn are translation invariant.
  2. They can learn spatial hierarchies of patterns.

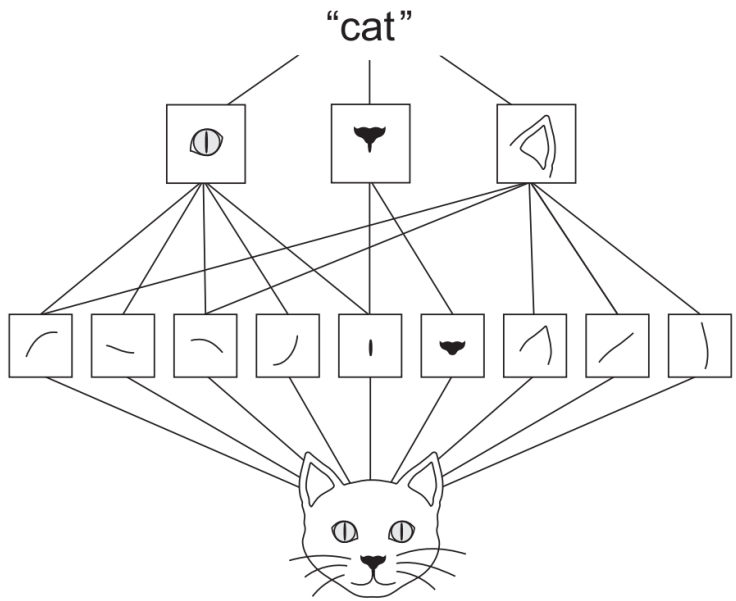


Figure: The visual world forms a spatial hierarchyof visual modules.

Convolutional Neural Networks (CNNs)

- We will look at a simple MNIST digits classification example to understand the structure of CNNs.

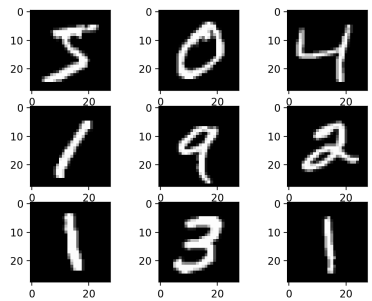


Figure: The **MNIST database** is a large database of handwritten digits containing 60,000 training images and 10,000 testing images.

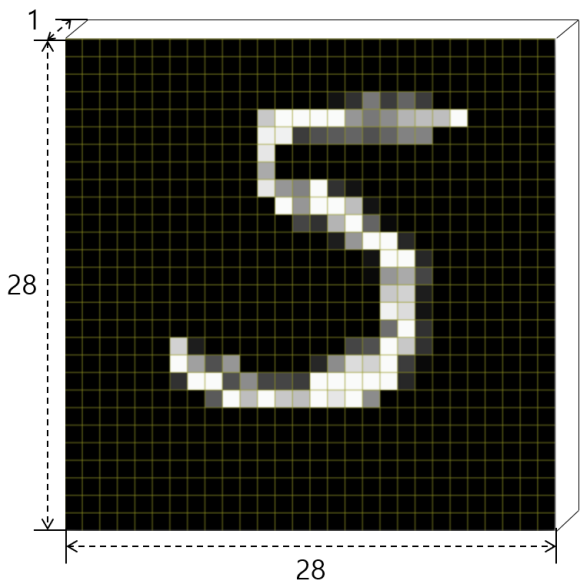
Convolutional Neural Networks (CNNs)

- Convolutions operate over 3D tensors, called **feature maps:**

$$(\text{height}, \text{width}, \text{channels})$$

  $\rightarrow$ Spatial axes and depth axis.

- In the MNIST example, the first convolution layer takes a feature map of size (28, 28, 1).



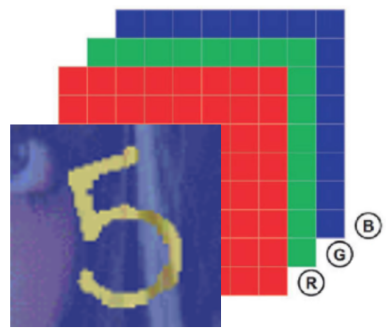


Figure: Input feature map of size (28, 28, 1). For an RGB image, (28, 28, 3)

Convolutional Neural Networks (CNNs)

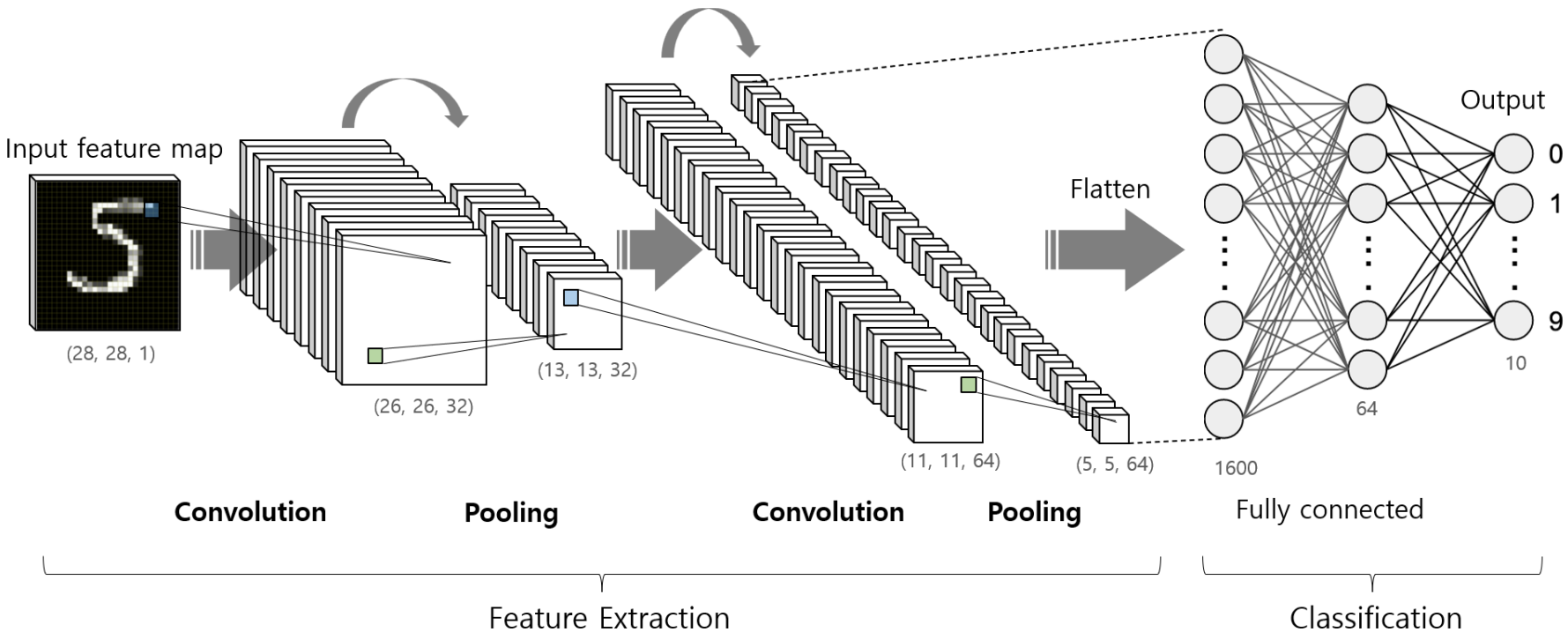


Figure: Structure of CNNs

Convolutional Neural Networks (CNNs)

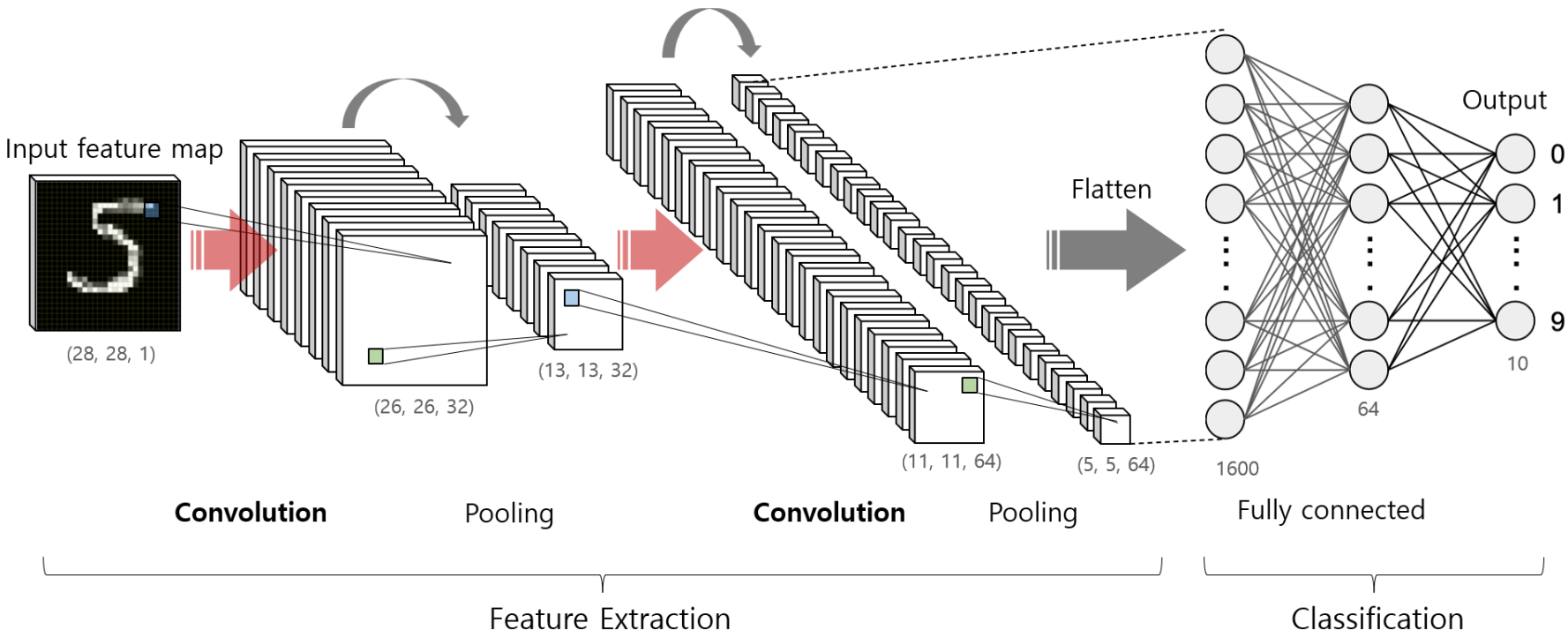
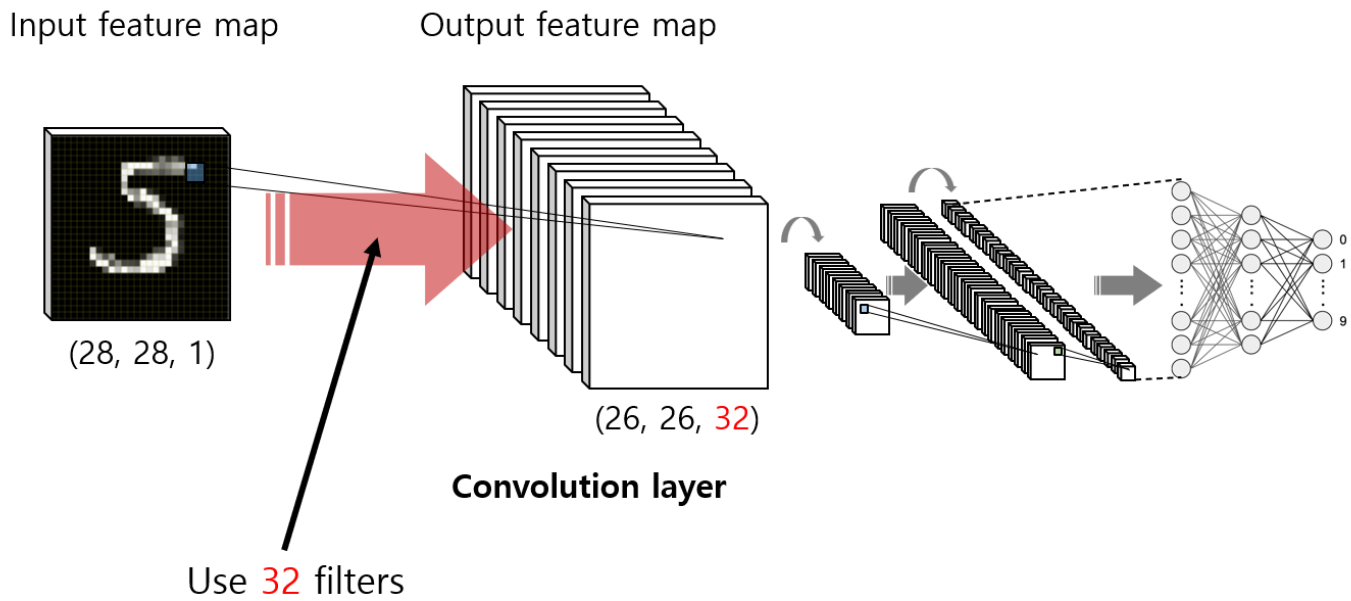


Figure: Two convolution layers

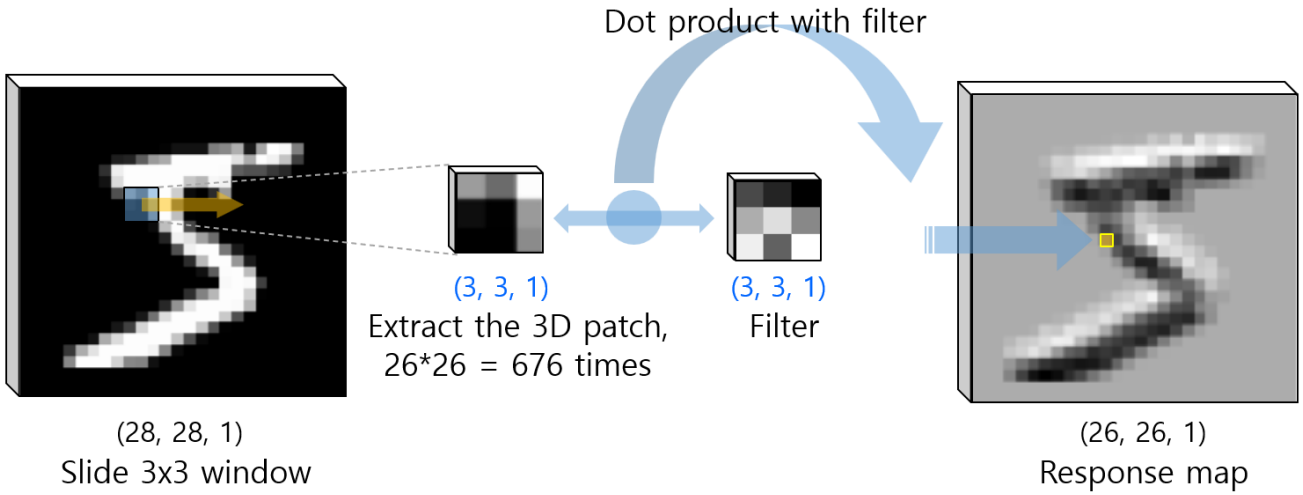
Convolutional Neural Networks (CNNs)



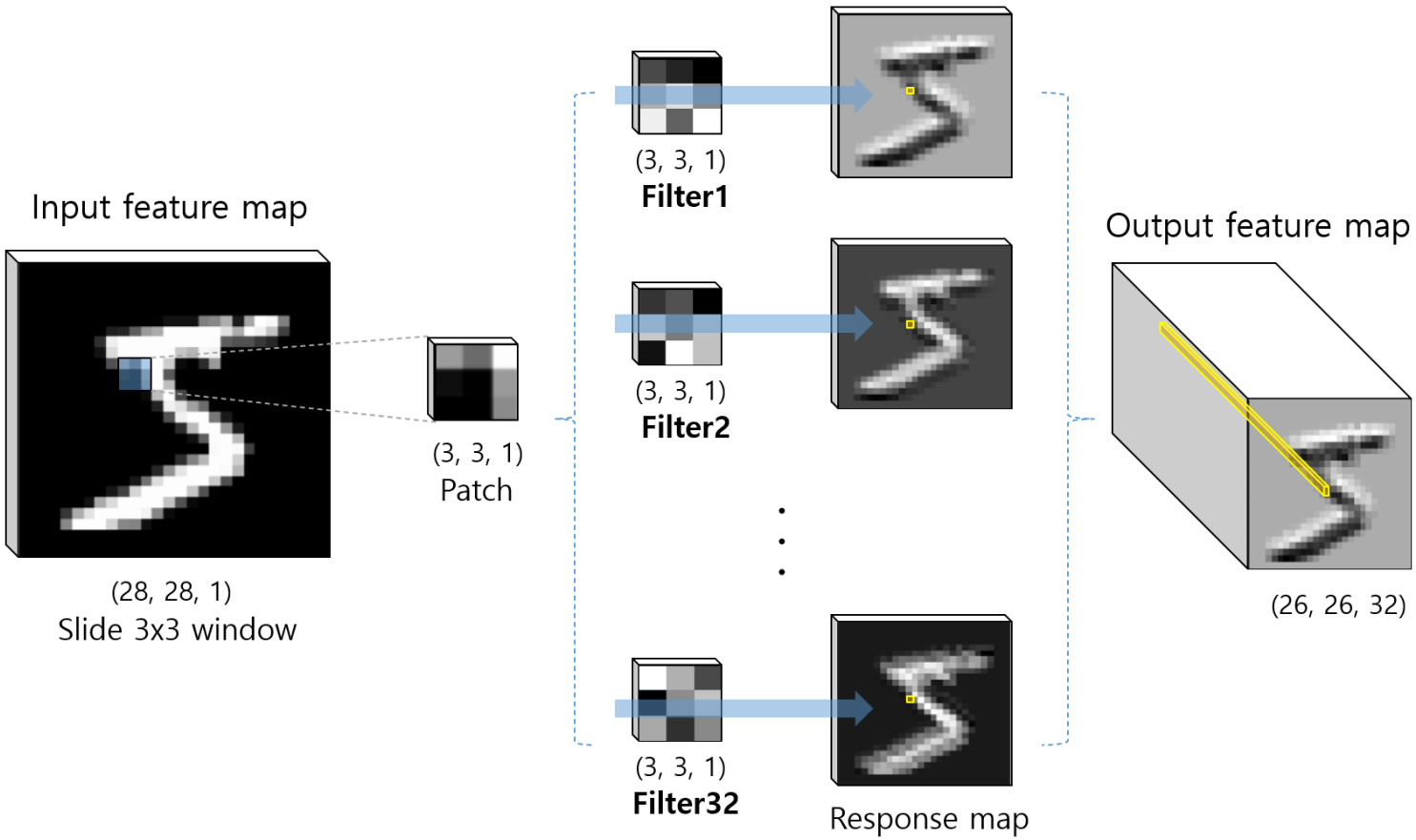
- The convolution operation extracts **patches** from its input feature map

$\Rightarrow$ and applies the same transformation to all of these patches, using **filters**, producing an output feature map.

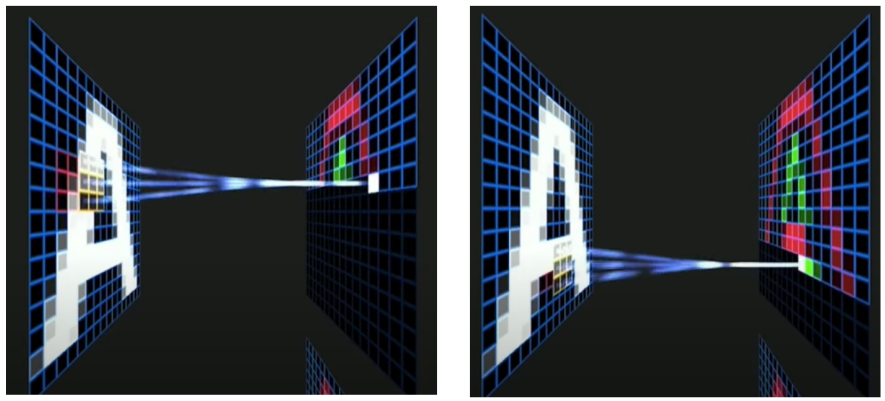
Convolutional Neural Networks (CNNs)



Convolutional Neural Networks (CNNs)

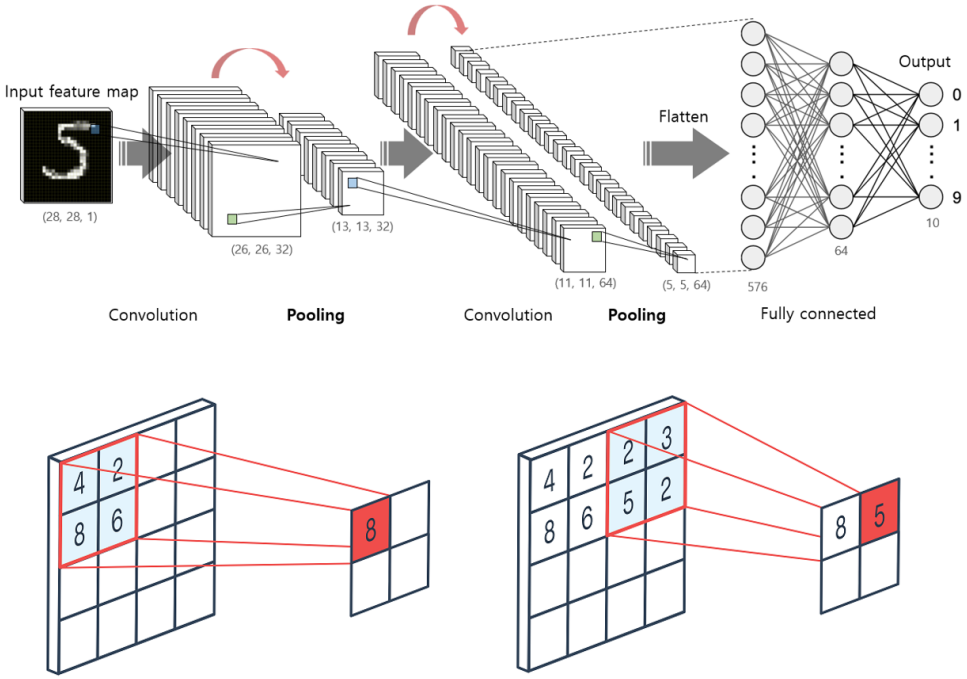


Convolutional Neural Networks (CNNs)



- A convolution works by sliding these windows of size 3×3 or 5×5 over the 3D input feature map, and extracting the 3D patch of shape *(window height, window width, input depth)*.
- Each such 3D patch is then transformed into a 1D vector of shape *(output depth,)*,
  - via a tensor product with the same learned **weight matrix**, called the convolution kernel.

Convolutional Neural Networks (CNNs)



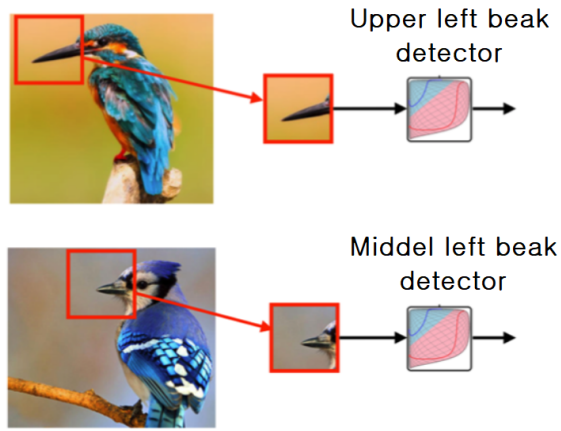


Figure: Max-Pooling

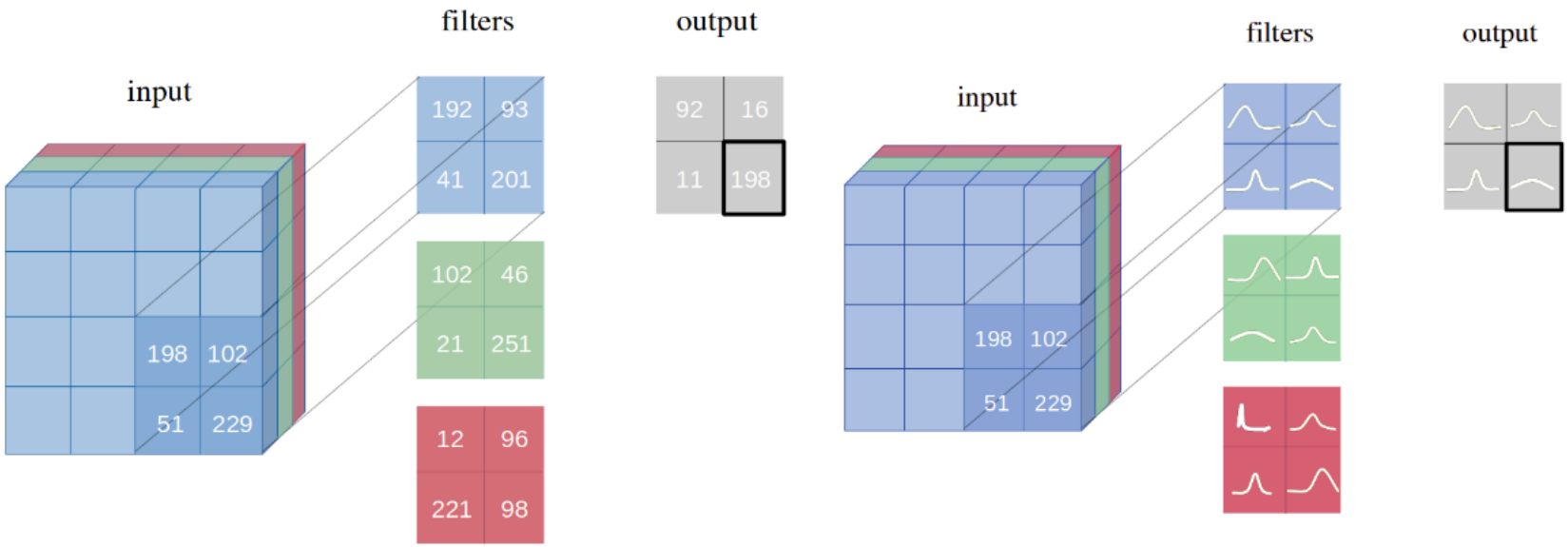
1. Over-fitting
2. Translation invariant.

Convolutional Neural Networks (CNNs)

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D  (None, 13, 13, 32)        0
)
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)          0
2D)
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 64)                102464
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 121,930
Trainable params: 121,930
Non-trainable params: 0
_________________________________________________________________
```

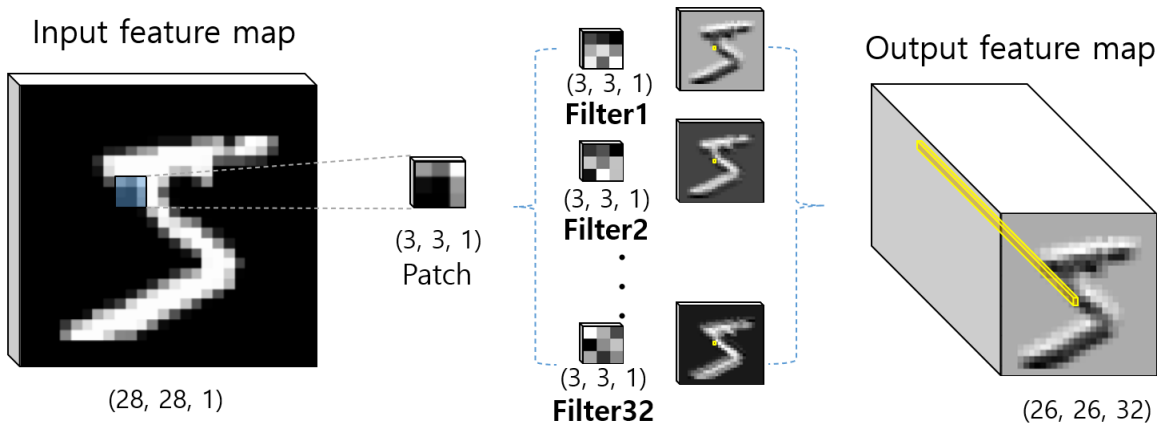
Convolutional Neural Networks (CNNs)



To implementing a Bayesian CNN,
how to apply dropout after all convolution layers?

Bayesian Convolutional Neural Networks

- To integrate over the kernels, we reformulate the convolution as a linear operation.



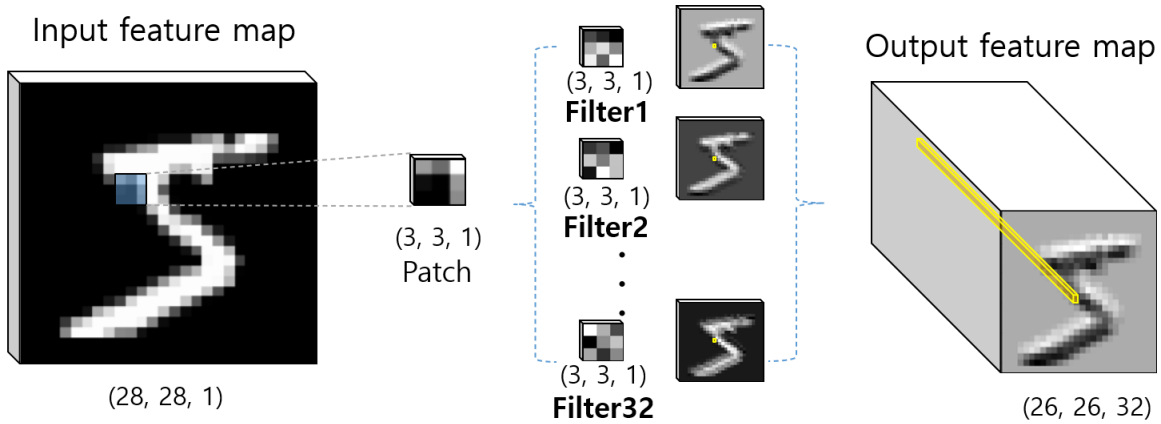
- Input to the layer:

$$\mathbf{x} \in \mathbb{R}^{28\times 28\times 1}.$$

- Kernels:

$$\mathbf{k}_1, \cdots, \mathbf{k}_{32} \in \mathbb{R}^{3\times 3\times 1}.$$

Bayesian Convolutional Neural Networks

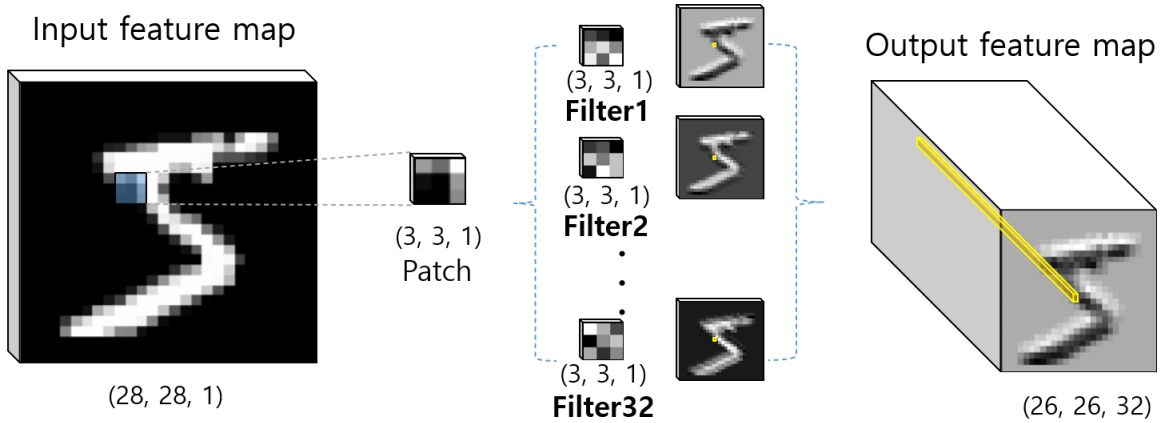


- We extract $3 \times 3 \times 1$ dimensional patches from the input and vectorize these.
- Collecting the vectors in the rows of a matrix, we obtain a new representation for our input

$$\bar{\mathbf{x}} \in \mathbb{R}^{676 \times 9},$$

with $n = 26 \cdot 26 = 676$ patches.

Bayesian Convolutional Neural Networks



- The vectorized kernels form the columns of the **weight matrix**:

$$\mathbf{W}_1 \in \mathbb{R}^{9\times 32}.$$

- The convolution operation is then equivalent to the matrix product:

$$\bar{\mathbf{x}}\mathbf{W}_1 \in \mathbb{R}^{676\times 32}.$$

- The columns of the output can be re-arranged to a 3 dimensional tensor:

$$\mathbf{y} \in \mathbb{R}^{26\times 26\times 32}.$$

Bayesian Convolutional Neural Networks

- Pooling can then be seen as a non-linear operation on the matrix $\mathbf{y}$.

Bayesian Convolutional Neural Networks

- We place a prior distribution over each kernel.
- We sample Bernoulli random variables $\mathbf{z}_{i,j,n}$ and multiply by the weight matrix

$$\mathbf{W}_i \cdot \mathrm{diag}\left(\left[\mathbf{z}_{i,j,n}\right]_{j=1}^{K_i}\right)$$

with K_i channels in the i'th layer.

- For our case,

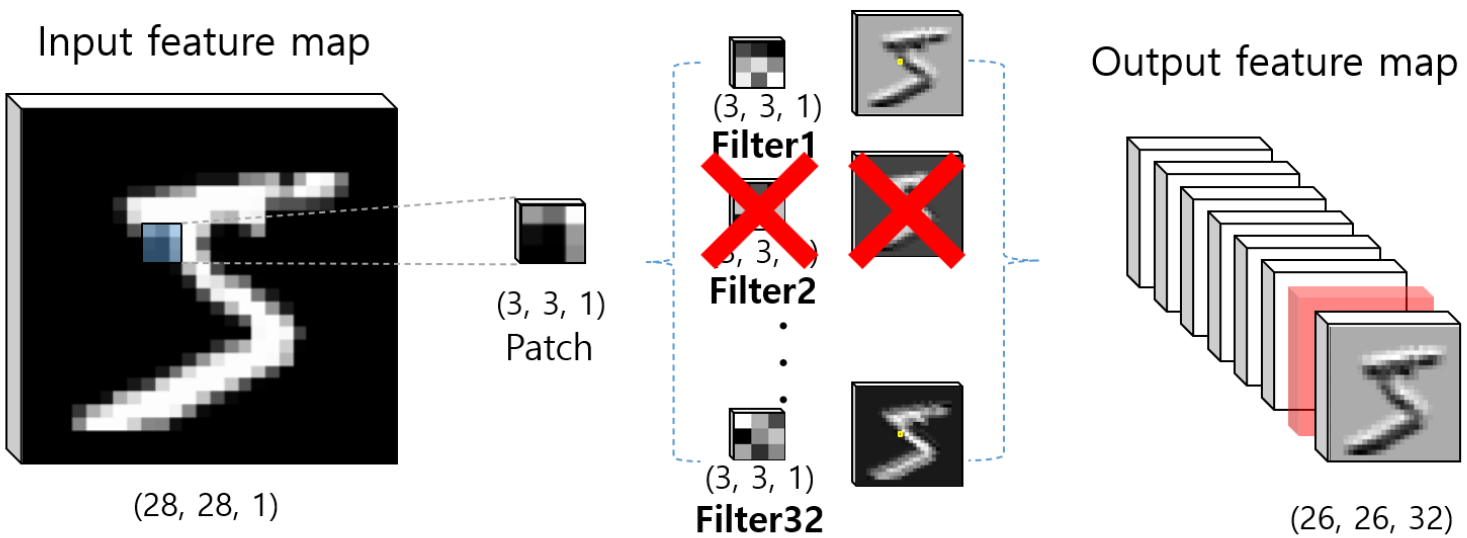
$$\mathbf{W}_1 \cdot \mathrm{diag}\left(\left[\mathbf{z}_{1,j,676}\right]_{j=1}^{32}\right)$$

$$= \begin{pmatrix} | & | & & | \\ \text{kernel 1} & \text{kernel 2} & \cdots & \text{kernel 32} \\ | & | & & | \end{pmatrix} \begin{pmatrix} \mathbf{z}_1 & 0 & & 0 \\ 0 & \mathbf{z}_2 & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & \mathbf{z}_{32} \end{pmatrix}$$

with $\mathbf{W}_1 \in \mathbb{R}^{9\times 32}$.

Bayesian Convolutional Neural Networks

- This distribution randomly sets kernels to zero for different patches.
- This is also equivalent to applying dropout for each element in the tensor $\mathbf{y}$ before pooling.



Bayesian Convolutional Neural Networks

- Therefore, implementing our Bayesian CNN is as simple as using dropout after every convolution layer before pooling.

Recall that:

- To relate the approximate inference in our Bayesian NN to dropout training, we define our approximating variational distribution $q(\mathbf{W}_i)$ for every layer i as

$$\mathbf{W}_i = \mathbf{M}_i \cdot \text{diag}([\mathbf{z}_{i,j}]_{j=1}^{K_i})$$
$$\mathbf{z}_{i,j} \sim \text{Bernoulli}(p_i), \quad i = 1, \cdots, L, \; j = 1, \cdots, K_{i-1}$$

  - Here $\mathbf{z}_{i,j}$ are Bernoulli distributed random variables with some probability p_i.
  - $\mathbf{M}_i$ are variational parameters to be optimized.

Bayesian Convolutional Neural Networks

- The standard dropout test time approximation does not perform well when dropout is applied after convolutions.
- We solve this by approximating the predictive distribution

Recall that:

$$p(y^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) \approx \int p(y^*|\mathbf{x}^*, \boldsymbol{\omega}) q(\boldsymbol{\omega}) d\boldsymbol{\omega}$$
$$\approx \frac{1}{T} \sum_{t=1}^{T} p(y^*|\mathbf{x}^*, \hat{\boldsymbol{\omega}}_t)$$

Section 3

Experiments

Experiments

Two sets of data

- MNIST
- CIFAR10
  - 32×32 pixel, 60,000 color images
  - Labeled with 10 classes

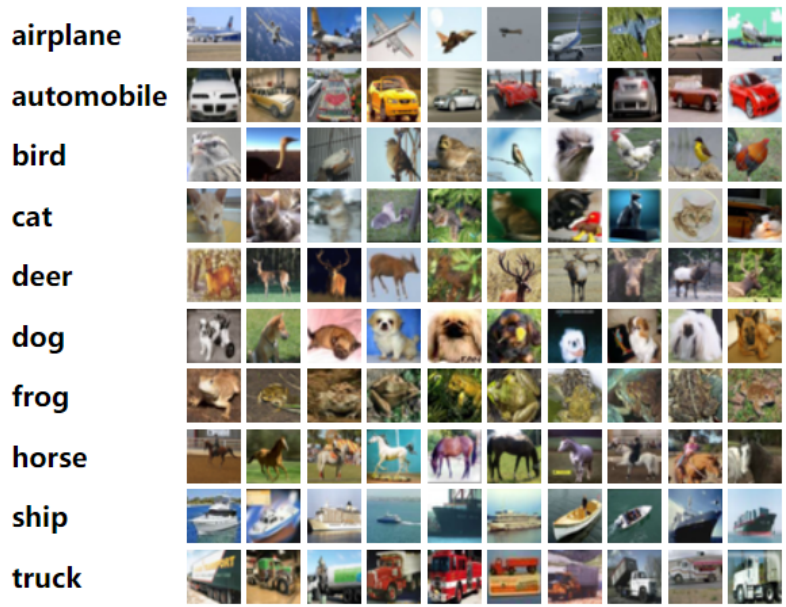


Figure: Labels of CIFAR-10 dataset

Experiments

3.1 Bayesian Convolutional Neural Networks

Testing techniques with dropout

- Standard dropout

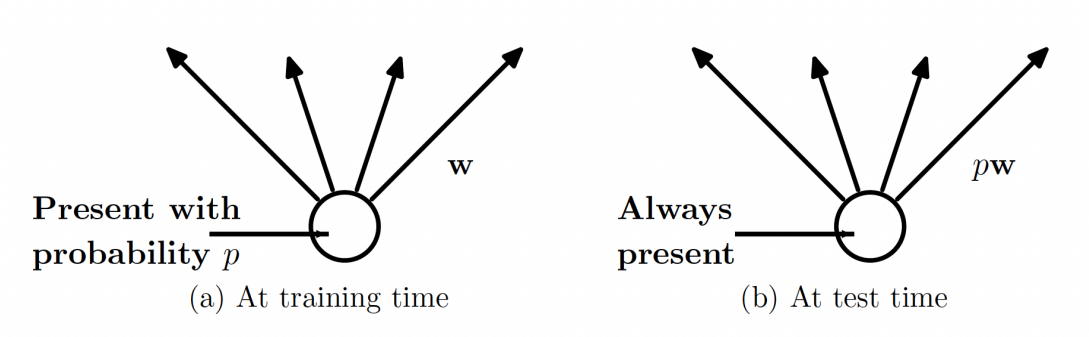


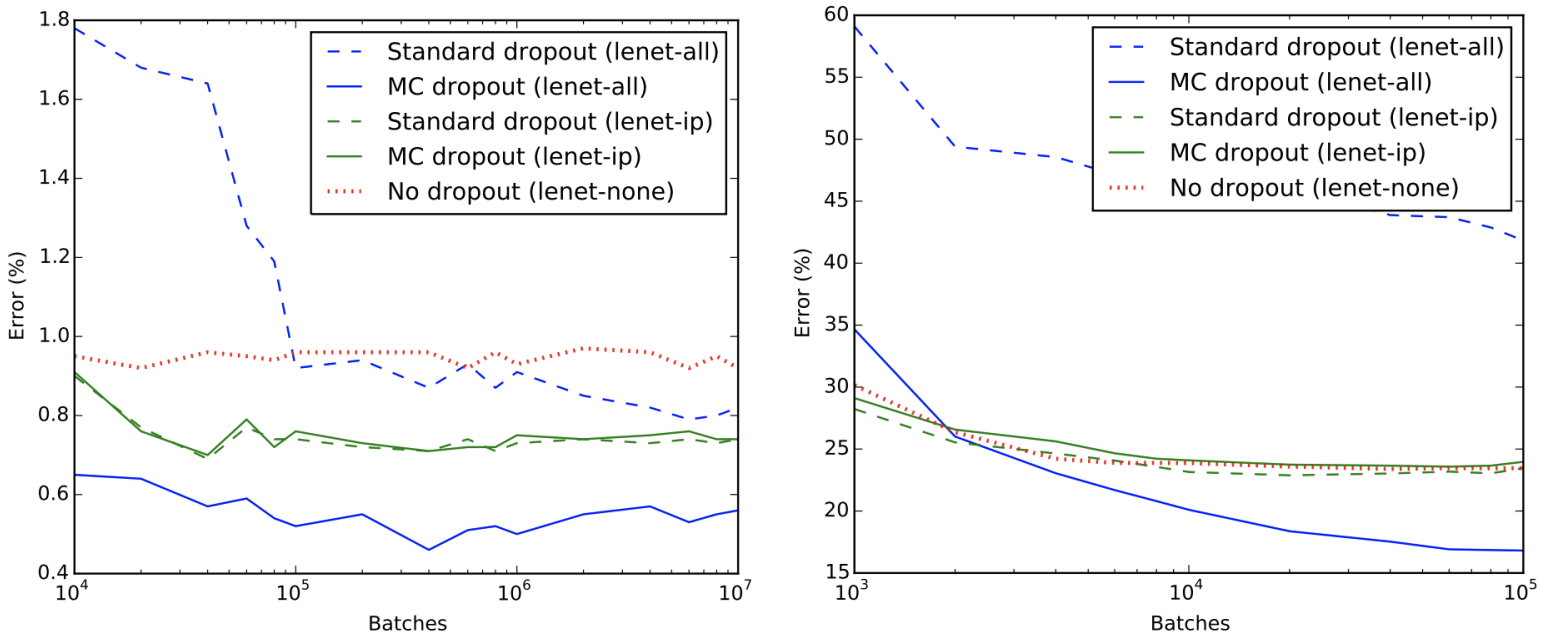
Figure: Fig2 in Srivastava et al. (2014)

- MC dropout

$$\mathbb{E}_{q(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*] = \int \mathbf{y}^* q(\mathbf{y}^*|\mathbf{x}^*) d\mathbf{y}^*$$
$$\approx \frac{1}{T}\sum_{i=1}^{T} \hat{\mathbf{y}}^*(\mathbf{x}^*, \mathbf{W}_1^t, \cdots, \mathbf{W}_L^t),$$

Experiments

3.1 Bayesian Convolutional Neural Networks



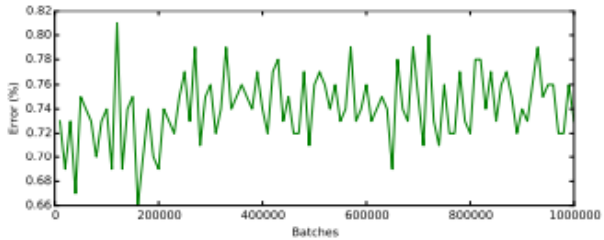
(a) MNIST

(b) CIFAR-10

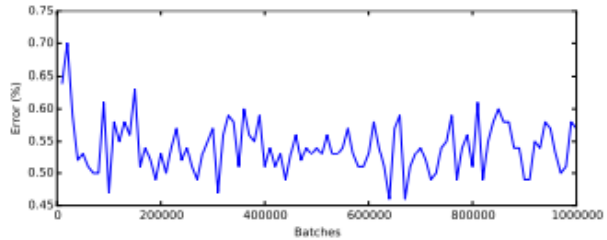
Figure: Text error for LeNet with dropout applied after every weight layer (lanet-all our Bayesian CNN implementation), dropout applied after the fully connected layer alone (lanet-ip), and without dropout (lanet-none).

Experiments

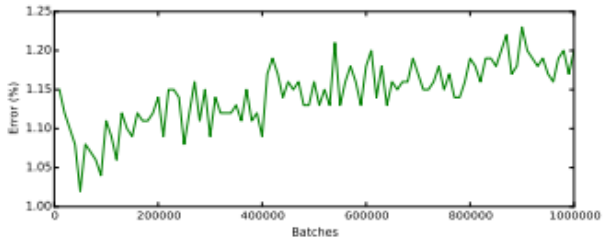
3.2 Model Over-fitting



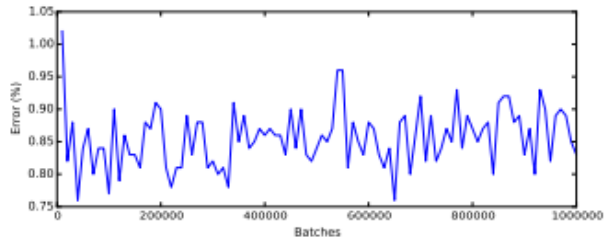
(a) Entire MNIST, Standard dropout + lenet-ip



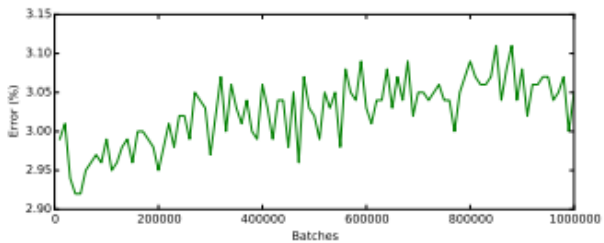
(b) Entire MNIST, MC dropout + lenet-all



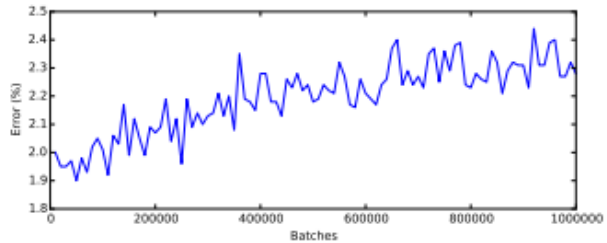
(c) 1/4 of MNIST, Standard dropout + lenet-ip



(d) 1/4 of MNIST, MC dropout + lenet-all



(e) 1/32 of MNIST, Standard dropout + lenet-ip



(f) 1/32 of MNIST, MC dropout + lenet-all

Figure: Test error of LeNet trained on random subsets of MNIST decreasing in size. MC dropout was used with 10 samples.

Experiments

3.3 MC Dropout in Standard Convolutional Neural Networks

	CIFAR Test Error (and Std.)	
Model	**Standard Dropout**	**MC Dropout**
NIN	10.43	$\mathbf{10.27 \pm 0.05}$
DSN	9.37	$\mathbf{9.32 \pm 0.02}$
Augmented-DSN	7.95	$\mathbf{7.71 \pm 0.09}$

Figure: Test error of CIFAR10 with the same networks evaluated using Standard dropout versus MC dropout ($T = 100$, averaged with 5 repetitions and given with standard deviation.)

Experiments

3.4 MC Estimate Convergence

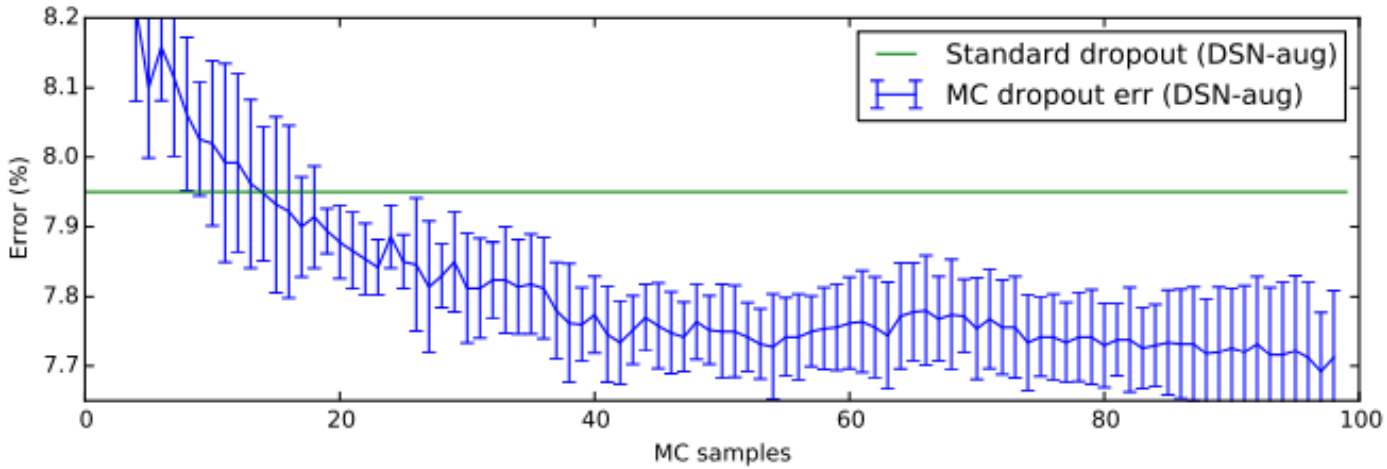


Figure: Augmented-DSN test error for different number of averaged forward passes in MC dropout (blue) averaged with 5 repetitions, shown with 1 standard deviation. In green is test error with Standard dropout.

References

- FRANÇOIS CHOLLET, Deep Learning with Python. 2017
- Yarin Gal, Dropout as a Bayesian Approximation: Appendix. 2016

감사합니다.