***IBIMA Publishing***
*Communications of the IBIMA*
*https://ibimapublishing.com/articles/CIBIMA/2026/267504/*
*Vol. 2026 (2026), Article ID 267504, 15 pages, ISSN: 1943-7765*
*https://doi.org/10.5171/2026.267504*

*Research Article*

# Impact of Batch Normalization and Spatial Dropout on VGG-11, NiN, and GoogLeNet

**Rafal BOGIEL**

Military University of Technology, Warsaw, Poland
rafal.bogiel@student.wat.edu.pl

Received date:23 march 2026; Accepted date:19 June 2026; Published date: 23 July 2026



**Abstract**

We study the effects of batch normalization and spatial dropout on the performance of VGG-11, Network in Network, and GoogLeNet architectures. We incorporate these regularization techniques into each model and evaluate their impact on classification accuracy and average loss. Experimental results show that spatial dropout has a limited influence on performance metrics, whereas batch normalization consistently improves model performance. Moreover, combining both methods does not yield significant gains beyond those achieved by batch normalization alone.

**Keywords**: VGG-11, Network in Network, GoogLeNet, regularization, batch normalization, spatial dropout

## Introduction

Deep convolutional neural networks (CNNs) have achieved state-of-the-art performance in a wide range of computer vision tasks, including image classification and object recognition (Huang et al., 2017) (He et al., 2016). Architectures such as VGG (Simonyan and Zisserman, 2015), Network in Network (Lin et al., 2014), and GoogLeNet (Szegedy et al., 2015) have played a significant role in the development of deep learning by demonstrating the effectiveness of increased depth and architectural innovation. However, training deep CNNs remains challenging due to optimization instability, slow convergence, and overfitting (Darwish, 2024) (Gupta and Jindal, 2025) (Korkmaz, 2025).

Batch normalization (Ioffe and Szegedy, 2015) was introduced to address these issues by normalizing intermediate activations, thereby stabilizing gradient propagation and accelerating training. In addition to improving convergence, batch normalization often provides implicit regularization. In parallel, dropout-based methods (Srivastava et al., 2014) (Tompson et al., 2015) have been widely used to reduce overfitting. While standard dropout is effective in fully connected layers, it may be less suitable for convolutional layers due to strong spatial correlations. Spatial dropout addresses this limitation by randomly suppressing entire feature maps, encouraging more robust channel-level representations (Tompson et al., 2015).

**Cite this Article as:** Rafal BOGIEL (2026)," Impact of Batch Normalization and Spatial Dropout on VGG-11, NiN, and GoogLe Net", Communications of the IBIMA, Vol. 2026 (2026), Article ID 267504, https://doi.org/10.5171/2026.267504

The effectiveness of normalization and regularization techniques strongly depends on network architecture. VGG employs deep homogeneous convolutional stacks (Simonyan and Zisserman, 2015), Network in Network introduces multilayer perceptrons within convolutional layers (Lin et al., 2014), and GoogLeNet utilizes Inception modules for multi-scale feature extraction (Szegedy et al., 2015). These structural differences suggest that batch normalization and spatial dropout may affect each architecture in distinct ways.

Despite their widespread use, relatively few studies have systematically analyzed the impact of batch normalization and spatial dropout across different CNN families (Cai et al., 2020) (Kim et al., 2023) (Lee and Lee, 2020). Consequently, clear guidelines for selecting appropriate training strategies remain limited.

In this paper, we present an empirical evaluation of batch normalization and spatial dropout applied to VGG-11, Network in Network, and GoogLeNet. For each architecture, we compare four configurations: a baseline model, a model with batch normalization, a model with spatial dropout, and a model combining both techniques. Spatial dropout layer is used in place of standard dropout, similarly to (Lee and Lee, 2020). We analyze their effects on classification accuracy and convergence behavior, providing practical insights for CNN design and training.

## Related Work

### *Fully connected layer*

A fully connected (FC) layer performs a linear transformation followed by a nonlinear activation (Rumelhart et al., 1986). For a given input vector $\boldsymbol{x}$, the output of the layer is computed as:

$$\boldsymbol{h} = \phi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

where $\boldsymbol{W}$ and $\boldsymbol{b}$ denote the weight matrix and bias vector, respectively, and $\phi(\cdot)$ is a nonlinear activation function. Fully connected layers enable global interactions among input features and are commonly used for feature integration and final prediction stages in neural networks. However, due to their dense connectivity and large number of parameters, FC layers may suffer from increased computational cost and overfitting, particularly when applied to high-dimensional inputs.

### *Convolutional layer*

A convolutional layer applies learnable filters to local regions of the input in order to extract spatially structured features (LeCun et al., 1998). Given an input tensor $X$, the output tensor $H_k$ is computed as:

$$H_k = X * W_k + B_k$$

where $*$ denotes the convolution operation, $W_k$ and $B_k$ are the kernel and bias corresponding to the $k$-th output channel.

Equivalently, the convolution can be expressed in index-wise form (Zhang et al., 2023a) as:

$$H[i,j,k] = B[i,j,k] + \sum_{a=0}^{p-1}\sum_{b=0}^{q-1}\sum_{c=0}^{c_{in}-1} W[a,b,c,k]\cdot X[i+a, j+b, c]$$

where $c_{in}$ denotes the number of input channels and $p \times q$ is the spatial size of the convolutional kernel.

By leveraging local connectivity and weight sharing, convolutional layers significantly reduce the number of parameters compared to fully connected layers and introduce inductive biases such as translation equivariance, making them particularly effective for processing grid-structured data such as images and signals (Zhang et al., 2023a).

Convolutional layers are commonly followed by pooling layers, which perform a fixed downsampling operation over local neighborhoods. For example, max pooling computes:

$$H[i,j,k] = \max_{0\le a\le p-1} \max_{0\le b\le q-1} X[i+a, j+b, k]$$

Pooling layers reduce spatial resolution, improve computational efficiency, and introduce robustness to small spatial variations (LeCun et al., 1998).

***Batch normalization***

Batch normalization is a technique used to stabilize and accelerate the training of deep neural networks by normalizing the input of each layer (Ioffe and Szegedy, 2015). For a mini-batch $\mathcal{B} = \{x_1, x_2, \dots, x_n\}$, the normalized output is computed as:

$$\mathrm{BN}(\boldsymbol{x}) = \gamma \odot \frac{\boldsymbol{x} - \widehat{\mu_{\mathcal{B}}}}{\sqrt{\widehat{\sigma_{\mathcal{B}}^2} + \epsilon}} + \beta.$$

where $\widehat{\mu_{\mathcal{B}}}$ and $\widehat{\sigma_{\mathcal{B}}^2}$ are mean and variance of batch, $\epsilon$ is a small constant for numerical stability, and $\boldsymbol{\gamma}, \boldsymbol{\beta}$ are learnable scale and shift parameters. BN allows for higher learning rates, faster convergence, and improved generalization. It is commonly applied after fully connected or convolutional layers, before the activation function.

During inference, batch normalization operates differently from training, as the batch statistics cannot be computed on-the-fly. Instead, fixed estimates of the mean and variance, typically obtained by aggregating statistics over the entire training dataset, are used to normalize the inputs.

***Dropout***

Dropout is a regularization technique designed to reduce overfitting in neural networks by randomly deactivating a subset of neurons during training (Srivastava et al., 2014). For a given output of fully connected layer $\boldsymbol{h}$, dropout is applied as:

$$\tilde{h} = \boldsymbol{r} \odot \boldsymbol{h}, r_i \sim Bernoulli(p)$$

where $\boldsymbol{r}$ is binary mask, $p$ denotes the probability of retaining a neuron, and $\odot$ represents element-wise multiplication. During inference, dropout is disabled and the activations are scaled accordingly to account for the expected value of the mask. By preventing co-adaptation of neurons, dropout improves generalization and is commonly applied to fully connected layers.

***Spatial Dropout***

Spatial dropout is a structured regularization technique that extends standard dropout by randomly dropping entire feature maps rather than individual activations (Tompson et al., 2015). Given a convolutional layer output $H_k$ for channel $k$, spatial dropout is defined as:

$$\widetilde{H_k} = \boldsymbol{r}_k H_k, r_{ki} \sim Bernoulli(p)$$

where $\boldsymbol{r}_k$ is channel-wise binary mask and $p$ is the probability of retaining a feature map. By enforcing independence across channels, spatial dropout mitigates feature co-adaptation and improves robustness in convolutional neural networks, while preserving spatial correlations within each feature map. During inference, spatial dropout is disabled and output activations are scaled by probability $p$.

***VGG Network***

VGG-11, denoted as ConvNet with 11 layers in the original paper (Simonyan and Zisserman, 2015), is a deep convolutional neural network architecture

characterized by a simple and uniform design based on stacked $3 \times 3$ convolutional layers followed by max-pooling operations. The network consists of 11 learnable layers, including eight convolutional layers and three fully connected layers, with ReLU activations applied after each convolution. By progressively increasing the number of feature channels while reducing spatial resolution, VGG-11 captures hierarchical image representations.

***Network in Network***

Network in Network (NiN) (Lin et al., 2014) is a convolutional neural network architecture that improves feature representation by introducing multilayer perceptron convolution (mlpconv) layers. An mlpconv layer begins with a conventional spatial convolution to capture local receptive field information, followed by one or more $1 \times 1$ convolutional layers, all with nonlinear activation functions. This sequence effectively implements a micro multilayer perceptron at each spatial location, enabling richer nonlinear feature transformations while preserving spatial structure.

In addition, NiN replaces traditional fully connected layers with global average pooling. Global average pooling computes the spatial average of each feature map, producing a single scalar per channel that is directly used for classification. This operation significantly reduces the number of parameters, improves robustness to spatial translations, and mitigates overfitting by enforcing a stronger correspondence between feature maps and output categories.

The original NiN consists of four mlpconv layers that are separated by three $3 \times 3$ max polling layers. The network terminates with global average pooling layer.

***GoogLeNet***

GoogLeNet (Szegedy et al., 2015) is a deep convolutional neural network architecture designed to improve computational efficiency while increasing representational capacity. Its core component is the Inception module, which performs parallel convolutions with multiple kernel sizes along with pooling operations, and concatenates their outputs along the channel dimension. The use of $1 \times 1$ convolutions for dimensionality reduction significantly decreases computational cost and the number of parameters. By stacking multiple Inception modules, GoogLeNet achieves substantial depth while maintaining efficiency, and replaces fully connected layers with global average pooling to further reduce model complexity.

The original paper describes Inception module consisting of four parallel convolutions stacks. First of them is a single $1 \times 1$ convolution. Second and third consist of $1 \times 1$ convolution, followed by $3 \times 3$ and $5 \times 5$ convolutions, respectively. Last path is built from $1 \times 1$ convolution preceded by $3 \times 3$ max pooling layer.

GoogLeNet architecture comprises approximately one hundred layers, of which around sixty layers are organized into Inception layers. Network concludes with a global average pooling layer, similarly as in (Lin et al., 2014), followed by an additional fully connected layer.

**Methodology**

***Implementation Details***

All networks were implemented using the PyTorch framework (Paszke et al., 2019), with layer parameters set to their default values unless stated otherwise.

We detail the architectural specifications of the VggBlock layer and the VGG-11 network in Fig. 1 and Fig. 2 respectively. In our implementation, the original architecture (Simonyan and Zisserman, 2015) was extended through the integration of batch normalization and spatial dropout layers.

We outline the internal structure of the Mlpconv layer in Fig. 3, while the configuration of the Network in Network model is presented in Fig. 4. As kernel sizes were not specified in the primary reference (Lin et al., 2014), we adopted the parameter settings of AlexNet (Krizhevsky et al., 2017), following the implementation suggested in (Zhang et al., 2023b). For the purposes of this study, we further modified the architecture to incorporate batch normalization and spatial dropout.

We present the detailed configuration of the Inception layer in Fig. 5, with the complete GoogLeNet architecture described in Fig. 6. Since

the original architecture (Szegedy et al., 2015) does not incorporate regularization layers, we extended the implementation to include batch normalization, dropout, and spatial dropout.

We integrated batch normalization layers after each convolutional layer and prior to the activation function, following the architectural convention established in ResNet (He et al., 2016).

Given that the optimal placement of dropout and spatial dropout remains a non-trivial design choice (Kim et al., 2023), we positioned these layers in close proximity to maintain architectural consistency. In the VGG-11 configuration, we integrated the spatial dropout layer after the final convolutional layer. In the Network in Network implementation, it was inserted immediately after the existing dropout layer. In the absence of pre-existing regularization in the original GoogLeNet architecture, we integrated dropout and spatial dropout layers prior to the second-to-last Inception module to mirror the structural design of the Network in Network model.

For the purposes of this study, all implemented regularization layers could be selectively disabled.

**Require:** $X$ - input tensor, $size$, $repeat$
$Y \leftarrow X$
**for** $i \leftarrow 1$ to $repeat$ **do**
    $Y \leftarrow \text{Conv2d}(Y, out\_channels = size, kernel\_size = 3, padding = 1, stride = 1)$
    $Y \leftarrow \text{BatchNorm2d}(Y, num\_features = size)$
    $Y \leftarrow \text{ReLU}(Y)$
$Y \leftarrow \text{MaxPool2d}(kernel\_size = 2, padding = 0, stride = 2)$
**return** $Y$

**Fig. 1 VggBlock**

**Require:** $X$ - input tensor
$Y \leftarrow \text{VggBlock}(X, size = 64, repeat = 1)$
$Y \leftarrow \text{VggBlock}(Y, size = 128, repeat = 1)$
$Y \leftarrow \text{VggBlock}(Y, size = 256, repeat = 2)$
$Y \leftarrow \text{VggBlock}(Y, size = 512, repeat = 2)$
$Y \leftarrow \text{VggBlock}(Y, size = 512, repeat = 2)$
$Y \leftarrow \text{Dropout2d}(Y)$
$Y \leftarrow \text{Linear}(Y, out\_features = 4096)$
$Y \leftarrow \text{ReLU}(Y)$
$Y \leftarrow \text{Dropout}(Y)$
$Y \leftarrow \text{Linear}(Y, out\_features = 4096)$
$Y \leftarrow \text{ReLU}(Y)$
$Y \leftarrow \text{Dropout}(Y)$
$Y \leftarrow \text{Linear}(Y, out\_features = 10)$
**return** $Y$

**Fig. 2 VGG-11**

**Require:** $X$ - input tensor, $size$, $kernel\_size$, $padding$, $stride$
1: $Y \leftarrow \text{Conv2d}(X, out\_channels = size,\ kernel\_size = kernel\_size,\ padding = padding,\ stride = stride)$
2: $Y \leftarrow \text{BatchNorm2d}(Y, num\_features = size)$
3: $Y \leftarrow \text{ReLU}(Y)$
4: $Y \leftarrow \text{Conv2d}(Y, out\_channels = size,\ kernel\_size = 1,\ padding = 0,\ stride = 1)$
5: $Y \leftarrow \text{BatchNorm2d}(Y, num\_features = size)$
6: $Y \leftarrow \text{ReLU}(Y)$
7: $Y \leftarrow \text{Conv2d}(Y, out\_channels = size,\ kernel\_size = 1,\ padding = 0,\ stride = 1)$
8: $Y \leftarrow \text{BatchNorm2d}(Y, num\_features = size)$
9: $Y \leftarrow \text{ReLU}(Y)$
10: **return** $Y$

**Fig. 3 Mlpconv**

**Require:** $X$ - input tensor
1: $Y \leftarrow \text{Mlpconv}(X, size = 96,\ kernel\_size = 11,\ padding = 0,\ stride = 4)$
2: $Y \leftarrow \text{MaxPool2d}(Y, kernel\_size = 3,\ padding = 0,\ stride = 2)$
3: $Y \leftarrow \text{Mlpconv}(Y, size = 256,\ kernel\_size = 5,\ padding = 2,\ stride = 1)$
4: $Y \leftarrow \text{MaxPool2d}(Y, kernel\_size = 3,\ padding = 0,\ stride = 2)$
5: $Y \leftarrow \text{Mlpconv}(Y, size = 384,\ kernel\_size = 3,\ padding = 1,\ stride = 1)$
6: $Y \leftarrow \text{MaxPool2d}(Y, kernel\_size = 3,\ padding = 0,\ stride = 2)$
7: $Y \leftarrow \text{Dropout}(Y)$
8: $Y \leftarrow \text{Dropout2d}(Y)$
9: $Y \leftarrow \text{Mlpconv}(Y, size = 384,\ kernel\_size = 3,\ padding = 1,\ stride = 1)$
10: $Y \leftarrow \text{AdaptiveAvgPool2d}(Y, output\_size = 1)$
11: **return** $Y$

**Fig. 4 Network in Network**

**Require:** $X$ - input tensor, $size[]$

$B_1 \leftarrow \text{Conv2d}(X, \text{out\_channels} = size[0], \text{kernel\_size} = 1, \text{padding} = 0, \text{stride} = 1)$
$B_1 \leftarrow \text{BatchNorm2d}(B_1, num\_features = size[0])$
$B_1 \leftarrow \text{ReLU}(B_1)$
$B_2 \leftarrow \text{Conv2d}(X, \text{out\_channels} = size[1][0], \text{kernel\_size} = 1, \text{padding} = 0, \text{stride} = 1)$
$B_2 \leftarrow \text{BatchNorm2d}(B_2, num\_features = size[1][0])$
$B_2 \leftarrow \text{ReLU}(B_2)$
$B_2 \leftarrow \text{Conv2d}(B_2, \text{out\_channels} = size[1][1], \text{kernel\_size} = 3, \text{padding} = 1, \text{stride} = 1)$
$B_2 \leftarrow \text{BatchNorm2d}(B_2, num\_features = size[1][1])$
$B_2 \leftarrow \text{ReLU}(B_2)$
$B_3 \leftarrow \text{Conv2d}(X, \text{out\_channels} = size[2][0], \text{kernel\_size} = 1, \text{padding} = 0, \text{stride} = 1)$
$B_3 \leftarrow \text{BatchNorm2d}(B_3, num\_features = size[2][0])$
$B_3 \leftarrow \text{ReLU}(B_3)$
$B_3 \leftarrow \text{Conv2d}(B_3, \text{out\_channels} = size[2][1], \text{kernel\_size} = 5, \text{padding} = 2, \text{stride} = 1)$
$B_3 \leftarrow \text{BatchNorm2d}(B_3, num\_features = size[2][1])$
$B_3 \leftarrow \text{ReLU}(B_3)$
$B_4 \leftarrow \text{MaxPool2d}(X, \text{kernel\_size} = 3, \text{padding} = 1, \text{stride} = 1)$
$B_4 \leftarrow \text{Conv2d}(B_4, \text{out\_channels} = size[3], \text{kernel\_size} = 1, \text{padding} = 0, \text{stride} = 1)$
$B_4 \leftarrow \text{BatchNorm2d}(B_4, num\_features = size[3])$
$B_4 \leftarrow \text{ReLU}(B_4)$
$Y \leftarrow \text{Concatenate}(B_1, B_2, B_3, B_4)$
**return** $Y$

**Fig. 5 Inception Layer**

**Require:** $X$ - input tensor
1: $Y \leftarrow \text{Conv2d}(X, out\_channels = 64,\ kernel\_size = 7,\ padding = 3,\ stride = 2)$
2: $Y \leftarrow \text{BatchNorm2d}(Y, num\_features = 64)$
3: $Y \leftarrow \text{ReLU}(Y)$
4: $Y \leftarrow \text{MaxPool2d}(Y, kernel\_size = 3, padding = 1, stride = 2)$
5: $Y \leftarrow \text{Conv2d}(X, out\_channels = 64,\ kernel\_size = 1,\ padding = 0,\ stride = 1)$
6: $Y \leftarrow \text{BatchNorm2d}(Y, num\_features = 64)$
7: $Y \leftarrow \text{ReLU}(Y)$
8: $Y \leftarrow \text{Conv2d}(Y, out\_channels = 192,\ kernel\_size = 3,\ padding = 1,\ stride = 1)$
9: $Y \leftarrow \text{BatchNorm2d}(Y, num\_features = 192)$
10: $Y \leftarrow \text{ReLU}(Y)$
11: $Y \leftarrow \text{MaxPool2d}(Y, kernel\_size = 3, padding = 1, stride = 2)$
12: $Y \leftarrow \text{Inception}(Y, size = [64, (96, 128), (16, 32), 32])$
13: $Y \leftarrow \text{Inception}(Y, size = [128, (128, 192), (32, 96), 64])$
14: $Y \leftarrow \text{MaxPool2d}(Y, kernel\_size = 3, padding = 1, stride = 2)$
15: $Y \leftarrow \text{Inception}(Y, size = [192, (96, 208), (16, 48), 64])$
16: $Y \leftarrow \text{Inception}(Y, size = [160, (112, 224), (24, 64), 64])$
17: $Y \leftarrow \text{Inception}(Y, size = [128, (128, 256), (24, 64), 64])$
18: $Y \leftarrow \text{Inception}(Y, size = [112, (144, 288), (32, 64), 64])$
19: $Y \leftarrow \text{Inception}(Y, size = [256, (160, 320), (32, 128), 128])$
20: $Y \leftarrow \text{MaxPool2d}(Y, kernel\_size = 3, padding = 1, stride = 2)$
21: $Y \leftarrow \text{Dropout}(Y)$
22: $Y \leftarrow \text{Dropout2d}(Y)$
23: $Y \leftarrow \text{Inception}(Y, size = [256, (160, 320), (32, 128), 128])$
24: $Y \leftarrow \text{Inception}(Y, size = [384, (192, 384), (48, 128), 128])$
25: $Y \leftarrow \text{AdaptiveAvgPool2d}(Y, output\_size = 1)$
26: $Y \leftarrow \text{Linear}(Y, out\_features = 10)$
27: **return** $Y$

**Fig. 6 GoogLeNet**

***Dataset Description***

For the purposes of this study, we constructed a ten-class dataset based on the ImageNet-1k dataset (Deng et al., 2009). We selected the image classes pseudo-randomly using the *random.sample()* function (Python Software Foundation, 2026a) from the Python 3 *random* module. To ensure reproducibility, we fixed the random seed to 42 using the *random.seed()* function (Python Software Foundation, 2026b). The selected classes are listed in Tab. 1.

All images in the dataset were converted to the RGB color space. We rescaled the shorter side of each image to 256 pixels and extracted a centered 224 × 224 crop. Finally, we normalized the images using a mean vector of [0.485,0.456,0.406] and a standard deviation vector of [0.229,0.224,0.225].

**Tab. 1 Imagenet-10**

| ID | Label |
|---|---|
| 0 | Minibus |
| 1 | Slug |
| 2 | European fire salamander, Salamandra salamandra |
| 3 | Reflex camera |
| 4 | Tabby, tabby cat |
| 5 | Siberian husky |
| 6 | Komondor |
| 7 | Dowitcher |
| 8 | Radio, wireless |
| 9 | Wallaby, brush kangaroo |

## Experiments

### *Details*

We trained all models for 40 epochs and evaluated them on the test dataset after each epoch to obtain the average loss and classification accuracy.

When we incorporated spatial dropout layers into our models, the standard dropout layers were always disabled. Since VGG-11 and Network in Network originally included dropout layers, these layers were retained in models with only batch normalization but omitted in models with spatial dropout or with both regularization methods. The original implementation of GoogLeNet does not include any dropout layers, therefore they were omitted in all variations.

### *VGG-11*

As shown in Fig. 7, we observed that regularization with spatial dropout delayed the onset of overfitting and reduced the minimum average loss but slowed down the convergence rate compared to the original network. We found that batch normalization eliminated overfitting over the 40-epoch training period and further reduced the minimum average loss relative to spatial dropout, however, it also resulted in a slower convergence rate. The combination of spatial dropout and batch normalization allowed us to maintain a convergence rate comparable to the spatial dropout variant while achieving a minimum average loss similar to that obtained with batch normalization.

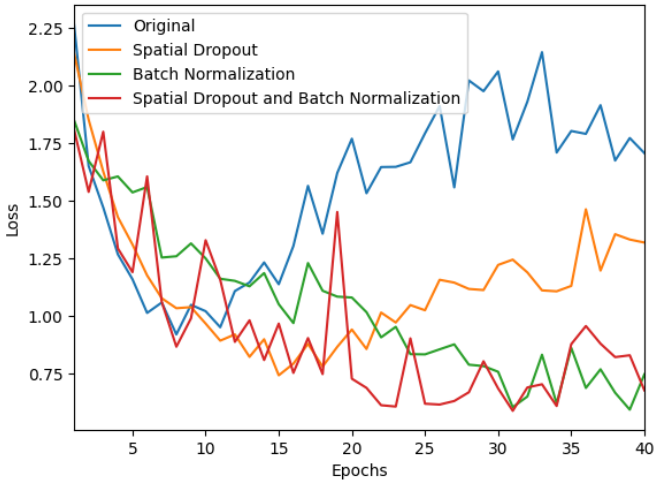


**Fig. 7 VGG-11 cumulative loss**

As presented in Fig. 8, we observed that incorporating spatial dropout into the model slowed the rate of accuracy improvement during training but led to higher accuracy near the 40th epoch. Batch normalization further reduced the rate of accuracy increase, while producing a higher final accuracy compared to spatial dropout alone. In models incorporating both regularization methods, we observed a combination of effects. The accuracy increase rate resembled that of the spatial dropout variant, whereas the final accuracy approached that of the model with batch normalization.

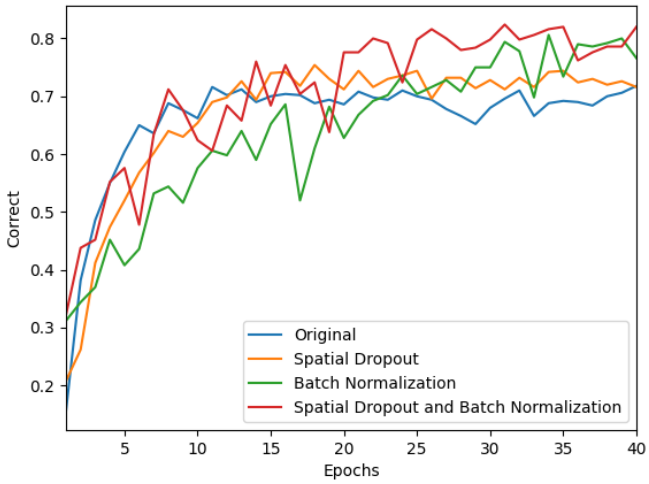


**Fig. 8 VGG-11 correct guesses**

***Network in Network***

As illustrated in Fig. 9, we observed that spatial dropout marginally reduced the network's minimum average loss, without noticeable improvements in the convergence rate or the emergence of overfitting. In contrast, batch normalization eliminated overfitting and visibly enhanced both the convergence rate and the minimum average loss. When applied in combination, we did not observe any improvement compared to the variant with batch normalization alone.

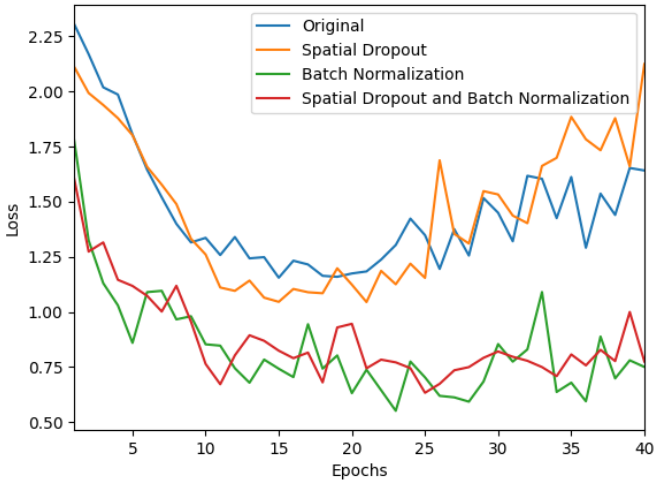


**Fig. 9 Network in Network cumulative loss**

As shown in Fig. 10, we observed that spatial dropout, when applied independently, slowed down the rate of accuracy improvement during training, but improved the final network accuracy. Batch normalization further enhanced both metrics. When both methods were combined, we did not observe any improvement over the variant with batch normalization alone.

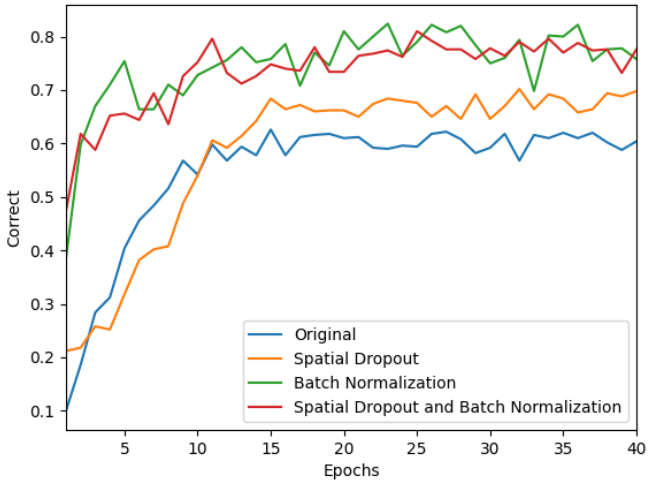


**Fig. 10 Network in Network correct guesses**

***GoogLeNet***

As illustrated in Fig. 11, we observed that spatial dropout improved the convergence rate relative to the original architecture, without yielding a noticeable reduction in the minimum average loss. Batch normalization improved the convergence rate and reduced the minimum average loss compared to spatial dropout. We did not observe any visible improvement in models incorporating both regularization methods compared to the variant with batch normalization alone. Furthermore, we observed the onset of overfitting in all network variants near the 40th epoch of training.

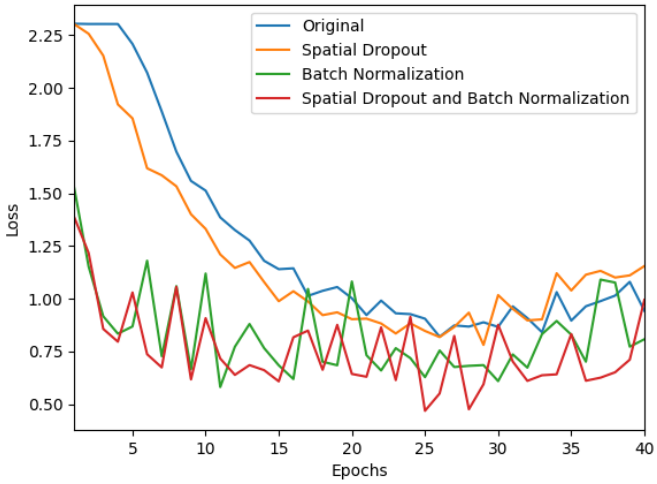


**Fig. 11 GoogLeNet cumulative loss**

As illustrated in Fig. 12, we observed that spatial dropout increased the rate of accuracy improvement during training, without enhancing the final accuracy relative to the original architecture. Batch normalization improved both the final accuracy and the accuracy improvement rate. When both methods were incorporated, we did not observe any improvement compared to the variant with batch normalization applied independently.

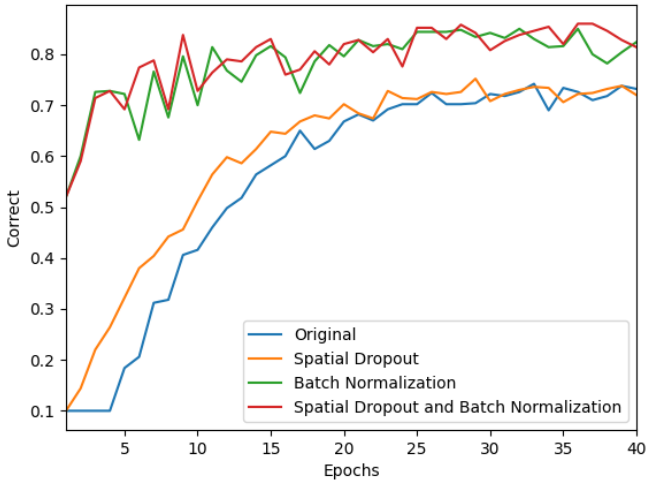


**Fig. 12 GoogLeNet correct guesses**

**Summary**

We evaluated the influence of batch normalization and spatial dropout on the performance of VGG-11, Network in Network, and GoogLeNet architectures.

We observed that spatial dropout reduced the average loss substantially in VGG-11 and marginally in the Network in Network architecture. While it resulted in a slower convergence rate in VGG-11, an improvement in convergence was observed in GoogLeNet. Spatial dropout also delayed the emergence of overfitting in VGG-11. Furthermore, it reduced the rate of accuracy improvement during training in VGG-11 and Network in Network but increased it in GoogLeNet. The final accuracy was improved in both VGG-11 and Network in Network models.

In comparison to spatial dropout, batch normalization resulted in a slower convergence rate in VGG-11, but improved convergence in the remaining architectures. The average loss was reduced in all evaluated models relative to spatial dropout. We found that batch normalization eliminated overfitting within the 40-epoch training period in both VGG-11 and Network in Network architectures. Although the rate of accuracy improvement during training was lower compared to spatial dropout in VGG-11, the final accuracy was higher. In Network in Network and GoogLeNet, batch normalization consistently led to faster accuracy improvement and higher final accuracy.

The combination of both methods yielded performance metrics similar to those obtained with batch normalization alone in Network in Network and GoogLeNet. In VGG-11, the combined approach resulted in an average loss and final accuracy comparable to batch normalization, while the convergence rate and accuracy improvement rate resembled those of the spatial dropout variant.

We presented the best achieved accuracy of each model in Tab. 2.

**Tab. 2 Accuracy of evaluated models**

| Network | Original | Spatial Dropout | Batch Normalization | Both |
|---|---|---|---|---|
| VGG-11 | 0.72 | 0.75 | 0.81 | 0.82 |
| Network in Network | 0.63 | 0.70 | 0.82 | 0.81 |
| GoogLeNet | 0.74 | 0.75 | 0.85 | 0.86 |

## Conclusions

Our results indicate that although spatial dropout and batch normalization may reduce the convergence rate and slow accuracy improvement during training, they do not adversely affect the final average loss or classification accuracy. Additionally, batch normalization is more likely to result in improved performance metrics than spatial dropout. Finally, combining both methods does not appear to provide additional benefits in terms of accuracy or average loss compared to batch normalization alone, particularly in more recent network architectures.

## References


- Cai, S., Shu, Y., Chen, G., Ooi, B.C., Wang, W. and Zhang, M. (2020) "Effective and Efficient Dropout for Deep Convolutional Neural Networks." Available at: http://arxiv.org/abs/1904.03392.
- Darwish, D. (2024) "Improving Techniques for Convolutional Neural Networks Performance," European Journal of Electrical Engineering and Computer Science, 8(1).
- Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. and Fei-Fei, L. (2009) "ImageNet: A Large-Scale Hierarchical Image Database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009.
- Gupta, R. and Jindal, R. (2025) "Impact of Too Many Neural Network Layers on Overfitting," International Journal of Computer Science and Mobile Computing, 14(5).
- He, K., Zhang, X., Ren, S. and Sun, J. (2016) "Deep residual learning for image recognition," Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.
- Huang, G., Liu, Z., Van Der Maaten, L. and Weinberger, K.Q. (2017) "Densely connected convolutional networks," Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017.
- Ioffe, S. and Szegedy, C. (2015) "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 32nd International Conference on Machine Learning, ICML 2015.
- Kim, B.J., Choi, H., Jang, H., Lee, D. and Kim, S.W. (2023) "How to Use Dropout Correctly on Residual Networks with Batch Normalization," Proceedings of Machine Learning Research.
- Korkmaz, O.E. (2025) "Revisiting convolutional design for efficient CNN architectures in edge-aware applications," Scientific Reports, 15(1).
- Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2017) "ImageNet classification with deep convolutional neural networks," Communications of the ACM, 60(6).
- LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) "Gradient-based learning applied to document recognition," Proceedings of the IEEE, 86(11).
- Lee, S. and Lee, C. (2020) "Revisiting spatial dropout for regularizing convolutional neural networks," Multimedia Tools and Applications, 79(45–46).
- Lin, M., Chen, Q. and Yan, S. (2014) "Network in network," 2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A.,

Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. and Chintala, S. (2019) "PyTorch: An imperative style, high-performance deep learning library," Advances in Neural Information Processing Systems.

- Python Software Foundation (2026a) "random.sample()". Available at: https://docs.python.org/3/library/random.html#random.sample (Accessed: 18 February 2026).
- Python Software Foundation (2026b) "random.seed()". Available at: https://docs.python.org/3/library/random.html#random.seed (Accessed: 18 February 2026).
- Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986) "Learning representations by back-propagating errors," Nature, 323(6088).
- Simonyan, K. and Zisserman, A. (2015) "Very deep convolutional networks for large-scale image recognition," 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014) "Dropout: A simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, 15.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015) "Going deeper with convolutions," Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.
- Tompson, J., Goroshin, R., Jain, A., LeCun, Y. and Bregler, C. (2015) "Efficient object localization using Convolutional Networks," Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.
- Zhang, A., Lipton, Z.C., Li, M. and Smola, A.J. (2023a) "Dive into deep learning. Chapter 7. Convolutional Neural Networks." Cambridge University Press. Available at: https://d2l.ai/chapter_convolutional-neural-networks/index.html (Accessed: 18 February 2026).
- Zhang, A., Lipton, Z.C., Li, M. and Smola, A.J. (2023b) "Dive into deep learning. Chapter 8. Modern Convolutional Neural Networks." Cambridge University Press. Available at: https://d2l.ai/chapter_convolutional-modern/index.html (Accessed: 18 February 2026).