BLG 506E Computer Vision 2022-2023 Spring Assignment V

Resul Dagdanov - 511211135

***Abstract*— This assignment focuses on implementing and using a convolutional neural network classifier for image classification. The development and use of the convolutional models along with the evaluation of the classifier's performance using numeric gradient checking are the main goals of this assignment. The convolutional neural network classifier is put into practice and used to categorize the CIFAR10 public dataset. Additionally, in this assignment, the forward and backward propagation in convolutional neural networks are implemented manually. Other methods, such as max-pooling, kaiming initialization, and batch normalization algorithms along with their gradient sanity checks are conducted in detail. Evaluating the results against each implementation, and utilizing numeric gradient testing and sanity checking to verify the code's accuracy is part of the assignment. The code is developed in the convolutional-networks.py file and the guidelines are in the convolutional-networks.ipynb which is given a notebook in order to complete the assignment. Additionally, the instructions for testing and evaluating the classifier's performance as well as guidance on how to incorporate them in the notebook are received. This assignment's overall goal is to increase your knowledge of and practical proficiency in creating and using convolutional neural networks for image classification tasks.**

***Index Terms*— Image Classifier, Deep Learning, Convolutional Neural Network, Max-Pooling, Batch Normalization, CIFAR10**

I. Introduction

Convolutional Neural Networks (CNNs) are a form of deep learning algorithm often used for image and video analysis. They are inspired by the structure of the human visual system, which processes complex images through hierarchical layers of cells that detect edges, forms, and textures. Similarly, CNNs extract key features from raw picture data using convolutional filters and then use these features to classify or identify objects in the image [1]. CNNs have demonstrated exceptional performance in a variety of computer vision tasks such as object identification [2], image segmentation, and image recognition. Their significance arises from their ability to learn and recognize complex patterns in big datasets, making them valuable in a variety of real-world applications like self-driving automobiles, medical imaging, and facial recognition systems.

The CIFAR10 dataset [3] is a popular benchmark dataset for image classification tasks in machine learning, which contains 60,000 color images divided into 10 classes, each having 6,000 images. The image size of the photos in the dataset is relatively modest at 32x32 pixels, compared to other similar datasets like ImageNet. The dataset comprises ten classes, including car, truck, boat, cat, deer, dog, frog, horse, ship, and airplane, with an equal number of images in each class in the training and test sets. The training set contains 50,000 images, while the test set contains 10,000 images. Due to the small image size and significant similarities between some classes, such as dogs and cats or ships and airplanes, the CIFAR10 dataset poses a significant challenge for classification tasks. The dataset has been widely used to evaluate the performance of various machine learning and deep learning models and has played a crucial role in the development of state-of-the-art computer vision methods. Figure 1 illustrates some of the images of the CIFAR10 dataset for corresponding labels.

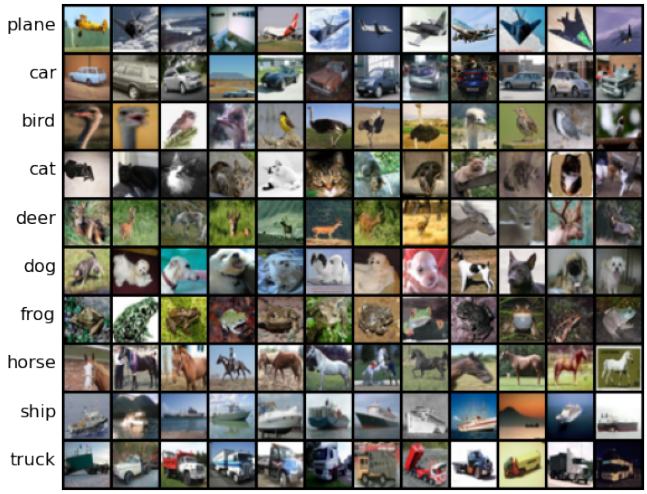

Fig. 1. CIFAR10 dataset example class label images [4].

In image processing and computer vision, grayscale and edge detection are essential feature extraction techniques. Grayscale reduces dimensionality and removes color information from picture data, making it easier for algorithms to find patterns and features in the image. Edge detection extracts important information from photos, such as the edges of objects, which can then be used to classify and segment images. These approaches are frequently employed in a variety of applications such as facial identification, object detection, and image analysis. As shown in Figure 2 illustrates the grayscale and edge detection of sample images of the CIFAR10 dataset.

Max pooling is a downsampling method that is extensively employed in CNNs for image analysis. It is used after convolutional layers to minimize the dimensionality of the feature maps and increase model efficiency. Max pooling works by dividing the feature map into non-overlapping rectangular sections and taking the maximum value from

R. Dagdanov is with Department of Aeronautical and Astonautical Engineering, Istanbul Technical University, Turkey dagdanov21 at itu.edu.tr

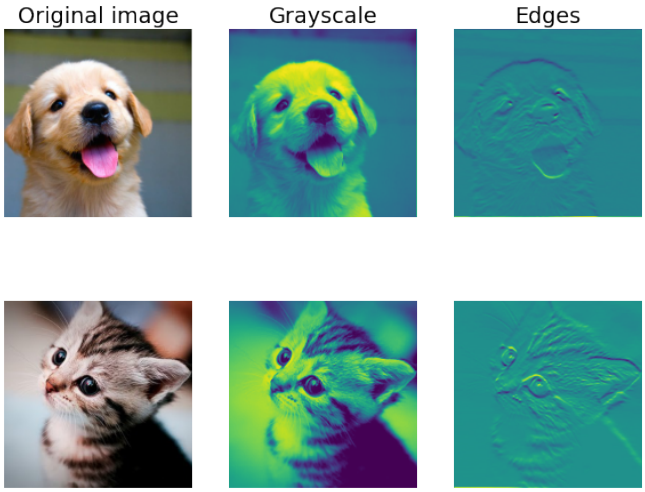

Fig. 2. Grayscale and edge detection features of the example CIFAR10 images.

each zone, resulting in a new down-sampled feature map with fewer spatial dimensions.

Max pooling has several advantages, including reducing overfitting by providing a form of regularization, making the model more computationally efficient by reducing the number of parameters, and providing a degree of translation invariance, which means the model can recognize features even if they appear in slightly different locations in the image. The size of the pooling window and stride length are hyperparameters that can be adjusted to improve the model's performance. Overall, max pooling is an effective strategy for assisting CNNs in extracting valuable features from images while boosting model efficiency.

Batch normalization is a technique frequently used in CNNs to increase the stability and speed of training. It operates by normalizing the previous layer activations for each batch during training, ensuring that the mean activation is near zero and the standard deviation is close to one. This normalization decreases internal covariate shift, which is the phenomenon in which the distribution of input to each layer varies during training, making it more difficult for the network to converge.

II. Methodology

In this section, a detailed inspection of the image classification with a convolutional neural network approach is carried out with related equations. It is possible to formulate the image classification with CNNs as follows:

- The input images are preprocessed to ensure that they are in a format suitable for the neural network. This may involve resizing, normalization, or other transformations.
- The convolutional neural network is trained on a set of labeled images, typically using a variant of supervised learning. The network learns to map input images to their corresponding labels by adjusting the weights of the connections between neurons in the network. The objective is to minimize a loss function that measures the difference between the predicted labels and the true labels.
- Once the CNN has been trained, it is evaluated on a set of unseen test images to measure its performance.
- The trained CNN can be used to predict the label of new, unseen images. The network takes an input image and produces a probability distribution over the possible labels. The label with the highest probability is typically chosen as the predicted label.

The convolution operation can be defined as:

$$y_{i,j} = \sum_{k,l} x_{i-k,j-l} \cdot h_{k,l} \tag{1}$$

where x is the input image, h is the filter or kernel, and y is the output feature map. The summation is taken over the kernel h with size $m \times n$, and centered at position (i, j) in the input image.

The output feature map can be computed by sliding the kernel over the input image with a certain stride, which determines the step size of the kernel. The stride is usually set to 1, which means the kernel moves one pixel at a time. The output feature map size is determined by the input size, kernel size, and stride.

After the convolution operation, a non-linear activation function is applied element-wise to the output feature map. The most common activation function used in CNNs is the Rectified Linear Unit (ReLU), which is defined as:

$$f(x) = \max(0, x) \tag{2}$$

where x is the input to the activation function. ReLU introduces non-linearity into the network and has been shown to improve the performance of CNNs.

In addition to convolutional layers, CNNs also use pooling layers to downsample the feature maps and reduce the computational cost. The most common pooling operation is max pooling, which selects the maximum value in each local region of the feature map. The max pooling operation can be defined as:

$$y_{i,j} = \max_{k,l} x_{i \times s + k, j \times s + l} \tag{3}$$

where x is the input feature map, y is the output feature map, and s is the stride of the pooling operation.

The batch normalization operation can be expressed mathematically as follows: Given a mini-batch of m training examples, with activations $x_1, x_2, ..., x_m$ for a given layer, the mean and standard deviation are computed as follows:

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x_i \quad \sigma = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)^2 + \epsilon} \tag{4}$$

where ϵ is a small constant added to the denominator for numerical stability.

The normalized activations y_i are then computed as:

$$y_i = \frac{x_i - \mu}{\sigma} \tag{5}$$

These normalized activations are then scaled and shifted by learnable parameters γ and β, respectively:

$$z_i = \gamma y_i + \beta \tag{6}$$

where γ and β are learnable parameters that are optimized during training. The resulting activations z_i are then passed on to the next layer in the network.

Overall, the convolutional, pooling, and batch normalization operations play a crucial role in the performance of CNNs, allowing them to learn meaningful features from input images and achieve state-of-the-art performance on various image recognition tasks. In this assignment, we have applied these methods to the image classification problem in the CIFAR10 dataset.

During backward propagation, the gradient of the loss with respect to the input feature map is computed. This gradient is then propagated back to the input layer using the chain rule of calculus. Since the max pooling operation is not differentiable, the gradient is assigned only to the location of the maximum value in each region, and all other locations are assigned a gradient of zero.

The backward propagation operation for max pooling could be expressed mathematically. Given an input feature map X of size $H \times W \times C$, and an output feature map Y of size $H' \times W' \times C$, the gradient of the loss with respect to the input feature map X is computed. For each element $y_{i,j,k}$ in the output feature map Y, find the corresponding position (p, q) in the input feature map X that was used to produce the maximum value.

$$(p, q) = \arg\max_{(i',j')} x_{(i'-1)s+p,(j'-1)s+q,k} \tag{7}$$

where s is the stride of the max pooling operation.

Then, the gradient of the loss with respect to the input feature map X is set to the gradient of the loss with respect to the output feature map Y at the corresponding position (p, q):

$$\frac{\partial L}{\partial x_{p,q,k}} = \frac{\partial L}{\partial y_{i,j,k}} \tag{8}$$

for all (i, j) such that $y_{i,j,k}$ is the maximum value in the corresponding region of the output feature map.

All other elements of the gradient of the loss with respect to the input feature map X are set to zero. This backward propagation operation ensures that the gradient is propagated only to the location of the maximum value in each local region of the input feature map, and allows the network to learn features that are robust to small translations and distortions in the input.

The backward propagation operation for batch normalization could be expressed mathematically. Given an input batch X of size $N \times D$, where N is the batch size and D is the dimensionality of each data point, and the normalized batch $\hat{X}$, the gradient of the loss with respect to the input batch X is computed. Compute the gradients of the loss with respect to the scale and shift parameters γ and β, respectively.

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{N} \frac{\partial L}{\partial \hat{x}i} \cdot x_i \quad \frac{\partial L}{\partial \beta} \qquad = \sum i = 1^N \frac{\partial L}{\partial \hat{x}_i} \tag{9}$$

where $\hat{x}_i$ is the i-th element of the normalized batch $\hat{X}$, and x_i is the i-th element of the input batch X.

Compute the gradients of the loss with respect to the normalized batch $\hat{X}$:

$$\frac{\partial L}{\partial \hat{x}_i} = \frac{\partial L}{\partial y_i} \cdot \gamma \tag{10}$$

where y_i is the i-th element of the output batch Y.

Compute the gradients of the loss with respect to the batch mean and variance:

$$\frac{\partial L}{\partial \mu} = \sum_{i=1}^{N} \frac{\partial L}{\partial \hat{x}i} \cdot (-\frac{1}{\sqrt{\sigma^2 + \epsilon}}) \tag{11}$$

$$\frac{\partial L}{\partial \sigma^2} = \sum i = 1^N \frac{\partial L}{\partial \hat{x}_i} \cdot (-\frac{1}{2} \cdot \frac{(x_i - \mu)}{(\sigma^2 + \epsilon)^{\frac{3}{2}}}) \tag{12}$$

where μ and σ^2 are the batch mean and variance, respectively, and ϵ is a small constant added for numerical stability.

Finally, compute the gradients of the loss with respect to the input batch X:

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma^2 + \epsilon}} \quad + \frac{\partial L}{\partial \sigma^2} \cdot \frac{2(x_i - \mu)}{N} \; + \frac{\partial L}{\partial \mu} \cdot \frac{1}{N} \tag{13}$$

This backward propagation operation ensures that the gradients are propagated through the batch normalization layer in a stable and efficient manner.

III. Experiments

Initially, the backward operation is applied to the convolutional layer is applied and the gradient errors for the input, weight, and bias parameters are low as expected. Consecutively, the backward operation for the max pooling operation and the gradient errors for the input, weight, and bias parameters are as expected. The results could be observed from the jupyter notebook implementation document.

After implementing, Fast layers, the differences between the implementation and FastConv are less than 1e-10. When moving from the implementation to FastConv CPU, we see speedups of at least 100x. When comparing the implementation to FastConv CUDA, we see speedups of more than 500x. Additionally, when comparing the implementation against FastMaxPool on CPU, we see speedups of more than 100x. When comparing the implementation against FastMaxPool on GPU, we see speedups of more than 500x.

Test the implementations of the sandwich layers by running the corresponding section of code blocks, we see errors less than $1e-7$.

After using numeric gradient checking to make sure that the backward pass of the convolutional neural network is correct, we use a small amount of artificial data and a small number of neurons at each layer for sanity checks. We see errors less than $1e-5$.

To overfit the small dataset, there is a nice trick to train the model with just a few training samples. We are able to overfit small datasets, which will result in very high training accuracy and comparatively low validation accuracy as shown in Figure 3 and Figure 4.

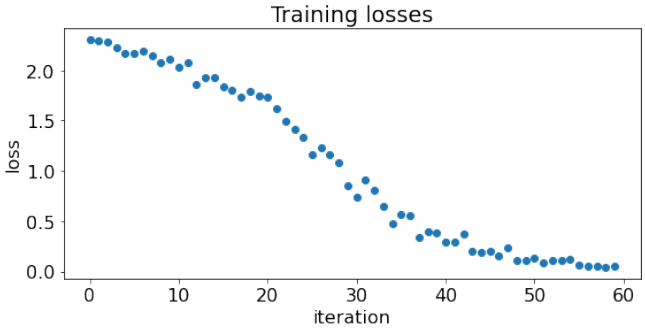

Fig. 3. Overfitting to the small dataset for sanity checks (Training Loss).

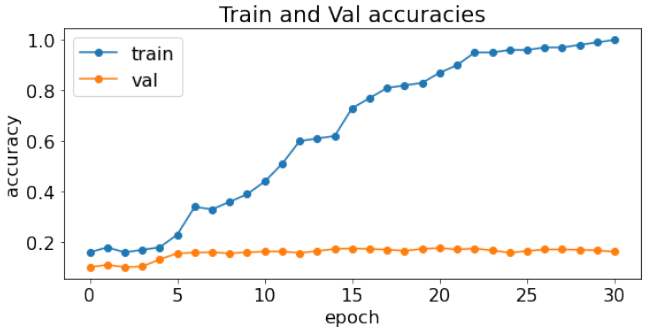

Fig. 4. Overfitting to the small dataset for sanity checks (Accuracies).

Additionally, we visualized the first-layer convolutional filters from the trained network as illustrated in Figure 5.

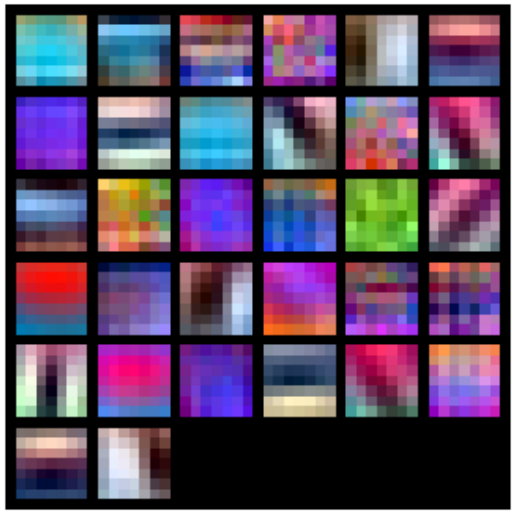

Fig. 5. Visualizing the firsts-layer convolutional filters.

As another sanity check, we can overfit a small dataset of the random 50 images of the CIFAR10 dataset. After tweaking the learning rate and weight initialization scale to overfit, we achieved 100 percent training accuracy within 30 epochs. The training loss of this overfitting experiment is illustrated in Figure 6. The training dataset accuracy reached 100 percent as illustrated in the corresponding notebooks. This overfitted model is saved as "overfit-deepconvnet.pth".

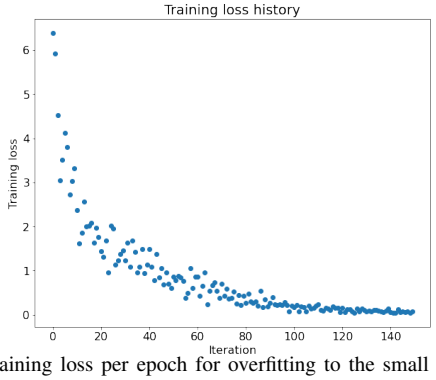

Fig. 6. Training loss per epoch for overfitting to the small dataset of 50 images.

In another experiment, we trained a 31-layer network with four different weight initialization schemes. Among them, only the Kaiming initialization method achieved a non-random accuracy after one epoch of training. The performance results could be observed in the following Figure 7.

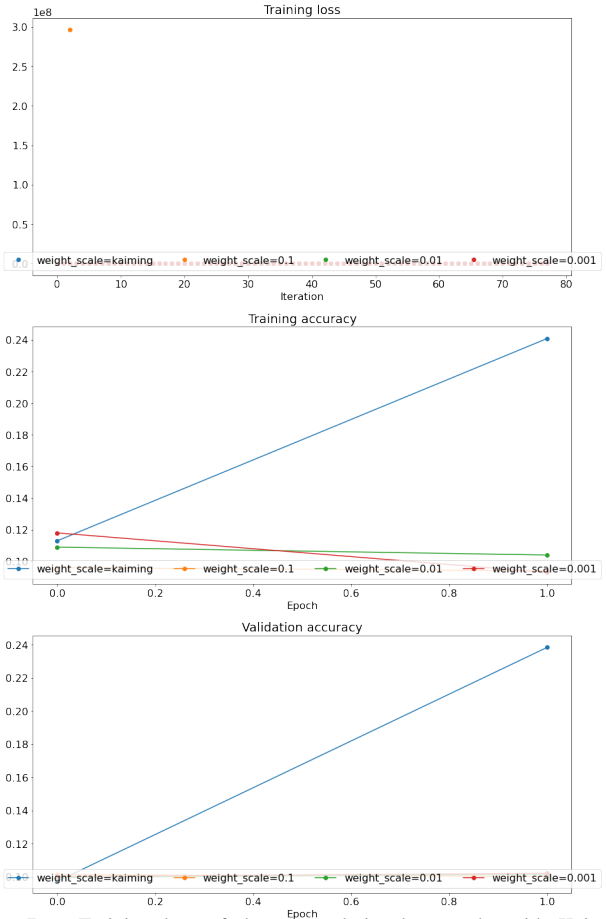

Fig. 7. Training loss of deep convolutional networks with Kaiming initialization.

After training the best convolutional model that we could on the CIFAR10 dataset, we stored the best model in the "best-model" variable. We achieved at least **%71 accuracy** on the validation set using a convolutional net, within 60 seconds of training. The validation set accuracy reached as high as **%76.39**.

Next, we tested the best model on the test set of the CIFAR10 dataset. It was expected to achieve at least %71 accuracy on the validation set and %70 accuracy on the test set. Our best model obtained a test set accuracy of **%74.48**. The best model is saved as "one-minute-deepconvnet".

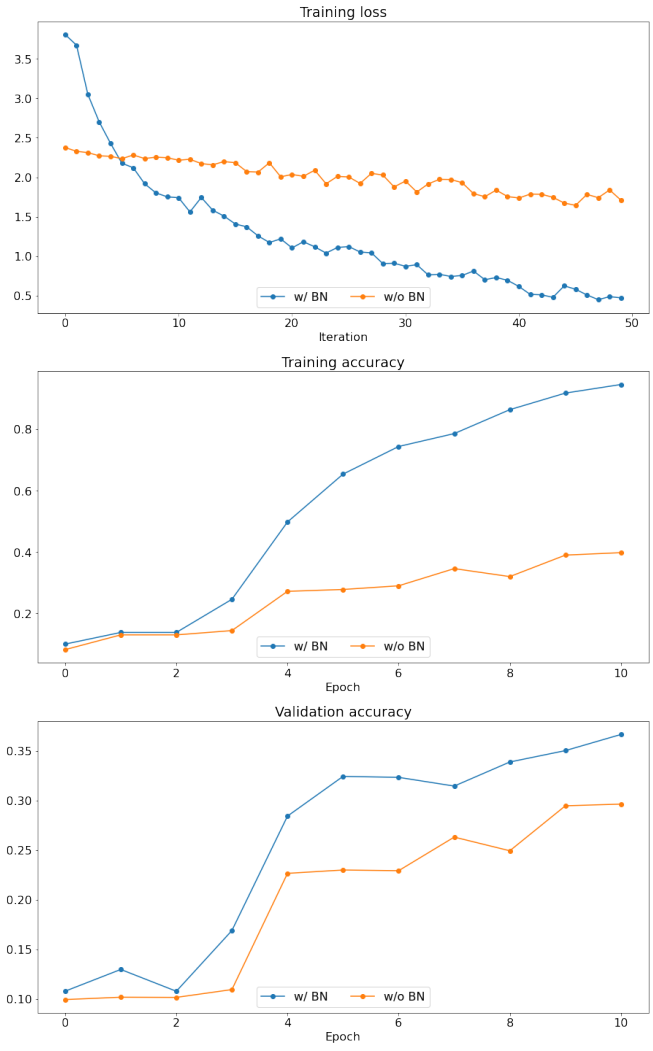

Fig. 8. Performance of Deep CNN with batch normalization addition.

After implementing the forward pass for batch normalization, we run the sanity check of the implementation. After running batch normalization with beta=0 and gamma=1, the data has zero mean and unit variance. After running batch normalization with nontrivial beta and gamma, the output data has a mean approximately equal to beta and a standard deviation approximately equal to gamma. To derive the backward pass we write out the computation graph for batch normalization and backprop through each of the intermediate nodes. By comparing the performance of the batch normalization implementation, the speed-up reached **6.07x** increase.

After implementing the backward pass for spatial batch normalization, we performed numeric gradient checking on the implementation and observed errors less than $1e-6$.

We run a small experiment to study the interaction of batch normalization and learning rate. We trained convolutional networks with different learning rates and plotted training accuracy and validation set accuracy over time. We found that using batch normalization helps the network to be less dependent on the learning rate as illustrated in Figure 9.

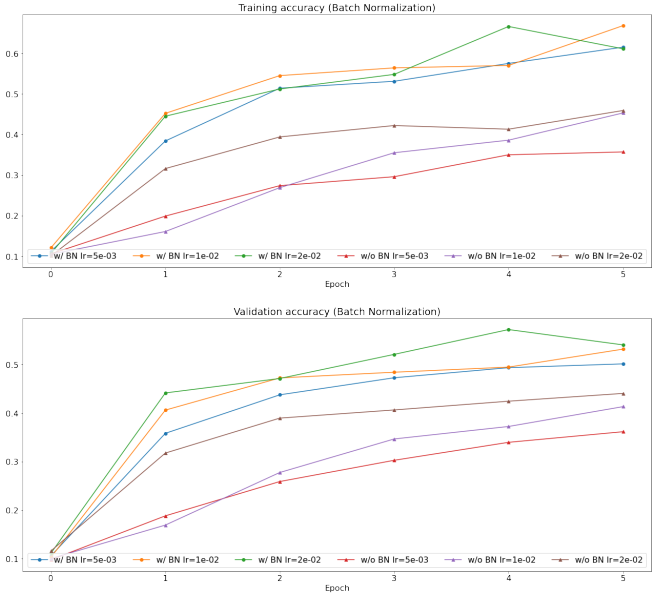

Fig. 9. Accuracies of CNN with batch normalization for different learning rates.

ACKNOWLEDGEMENT

This work is conducted as an assignment of the "BLG 506E Computer Vision" graduate course lecture given by Prof. Hazım Kemal Ekenel.

REFERENCES

[1] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[2] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification," in *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*. Springer, 2010, pp. 143–156.

[3] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[4] Y. Abouelnaga, O. S. Ali, H. Rady, and M. Moustafa, "Cifar-10: Knn-based ensemble of classifiers," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2016, pp. 1192–1195.