Assignment 2, Deep Learning Fundamentals, 2022
CNNs for image classification

Duc Nhan Do
The University of Adelaide
Adelaide, SA, 5000
ducnhan.do@student.adelaide.edu.au

Abstract

Convolutional Neural Network (CNN) is the most commonly used network for computer vision problems. This network effectively extracts patterns in the input image, such as lines, gradients, circles, or even faces and eyes, which makes it a powerful tool for image classification. In this experiment, as the goal is to gain deeper insights into the CNN, this network architecture will be self-implemented using Pytorch library and trained on the ***FashionMNIST*** *dataset. First, the self-implemented CNN architecture is compared to Multilayer Perceptron (MLP) to investigate the performance differences. Then, this network is also tested on different hyperparameters (batch size, learning rate, optimizer) to find the best configuration to achieve the highest accuracy on the test set.*

1. Introduction

Deep learning (DL) can be considered a subset of *machine learning*. While machine learning uses straightforward concepts, deep learning works with a broader algorithm called artificial neural networks. This algorithm is inspired and designed by the structure and function of the human brain to imitate how humans think and learn. The learning process of DL can be *supervised, semi-supervised* or *unsupervised*. DL has various architectures, such as *Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN)* and *Transformers*. These architectures have been applied to computer vision, speech recognition, natural language processing, medical image analysis and produced stunning results which can surpass human expert performance in some cases [1]. **Image Classification** is a fundamental task that attempts to comprehend an entire image. The goal is to classify the image by assigning it to a specific label. Typically, Image Classification refers to images in which only one object appears and is analyzed and one DL architecture called ***CNN*** is commonly used for this specific task.

Throughout the history of development, there are some remarkable and well-known models: *VGG* [15] uses architecture with small (3×3) convolution filters and shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to $16 - 19$ weight layers; *InceptionNet* [16] explores ways to scale up networks in ways that aim at utilizing the added computation as efficiently as possible; *ResNet* [10] explicitly reformulates the layers as learning residual functions regarding the layer inputs, instead of learning unreferenced functions; *EfficientNet* [17] is a new baseline network aim to carefully balance network depth, width, and resolution to achieve a better result. In addition, the current top 3 most state-of-the-art CNN models on the ***FashionMNIST*** dataset, based on *PaperWithCode*'s leaderboard are **Fine-Tuning DARTS** [18] with 96.91%, **Shake-Shake** [8] with 96.41% and **PreAct-ResNet18 + FMix** [9] with 96.36% accuracy. In this experiment, to investigate the fundamental of CNN, a simple baseline CNN is implemented using *Pytorch* library to classify different types of clothes in the ***FashionMNIST*** dataset. Furthermore, the CNN is tested and compared on different batch size, learning rates and optimizers to find the architecture that leads to the best performance. The implementation code is available at: https://github.com/DoDucNhan/DL-assignment.git

2. Convolutional Neural Network

A convolutional neural network is a class of DNN, most commonly applied to analyze visual imagery. CNNs are very good at picking up patterns in the input image, such as lines, gradients, circles, or even faces and eyes. This property makes CNNs so powerful for computer vision. Unlike earlier computer vision algorithms, CNNs can operate directly on a raw image and do not need any preprocessing. A CNN typically consists of three types of layers: *convolutional layer, pooling layer*, and *fully connected layer*.

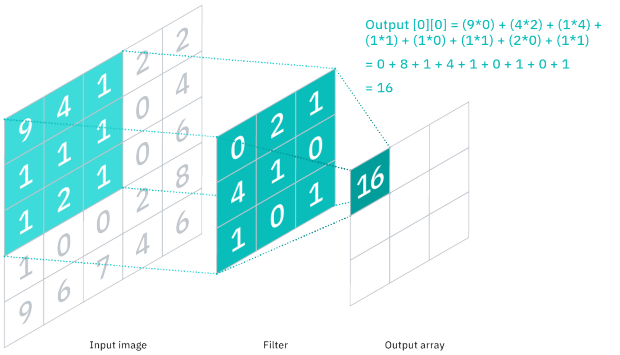


Figure 1. The example of convolutional layer.

2.1. Convolutional Layer

The convolutional layer (Fig. 1) is the core building block of CNN. It carries the primary portion of the network's computational load. This layer performs a dot product between two matrices. One matrix is the set of learnable parameters, otherwise known as a kernel, and the other is the input image. The kernel is spatially smaller than an image but is more in-depth. There are three hyperparameters that affect the volume size of the output that needs to be set before the training of the neural network begins. These include [7]:

1. The number of filters affects the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.
2. Stride is the number of pixels the kernel moves over the input matrix. Stride values of two or greater are rare. A larger stride value yields a smaller output.
3. Zero-padding is usually used when the filters do not fit the input image. This parameter sets all elements that fall outside the input matrix to zero, producing a larger or equally sized output. There are three types of padding:
    - **Valid padding**: This is also known as no padding. In this case, the last convolution is dropped if the dimensions do not align.
    - **Same padding**: This padding ensures that the output layer has the same size as the input layer.
    - **Full padding**: This type of padding increases the size of the output by adding zeros to the border of the input.

Suppose inputs are an image with a size of $(W_{in} \times H_{in})$, a kernel of dimensions $(K \times K)$, stride S and padding P. Then, the formula to compute the output size of an image after applying the convolutional layer is described as follows [3]:

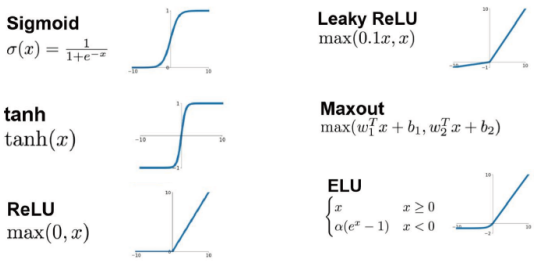


Figure 2. Common activation functions.

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1$$

$$H_{out} = \frac{H_{in} - K + 2P}{S} + 1$$

2.2. Activation Functions

Since the convolutional layer is a linear operation, a nonlinear activation is necessary to avoid turning the neural network into a *Linear Regression* algorithm. As shown in Fig. 2, there are several activation functions for hidden and output layers:

- Sigmoid
- Tanh
- ReLU [2]
- Leaky ReLU
- Maxout
- ELU [6]

For CNN, the **ReLU** activation function is applied to the output of every convolutional layer, and it is perhaps the most common function used for hidden layers.

2.3. Pooling Layer

Pooling layers, also known as downsampling, perform dimensionality reduction, reducing the number of parameters in the input. Like the convolutional layer, the pooling operation slides a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, and the calculated values form the output array. There are two main types of pooling (Fig. 3) [7]:

- **Max pooling**: As the filter moves across the input, it selects the pixel with the maximum value to send to the output array.
- **Average pooling**: As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

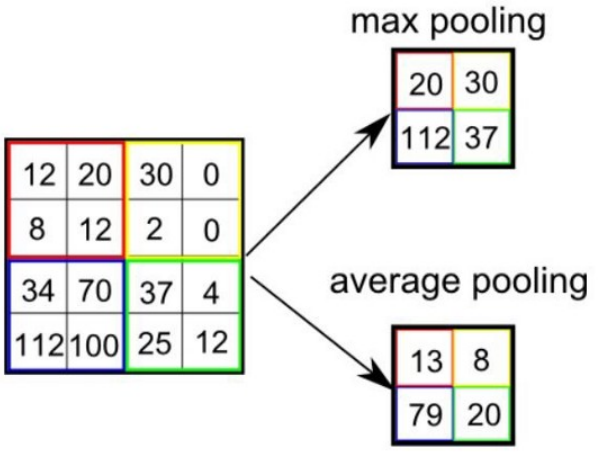


Figure 3. Example of 2×2 pooling layer [13]

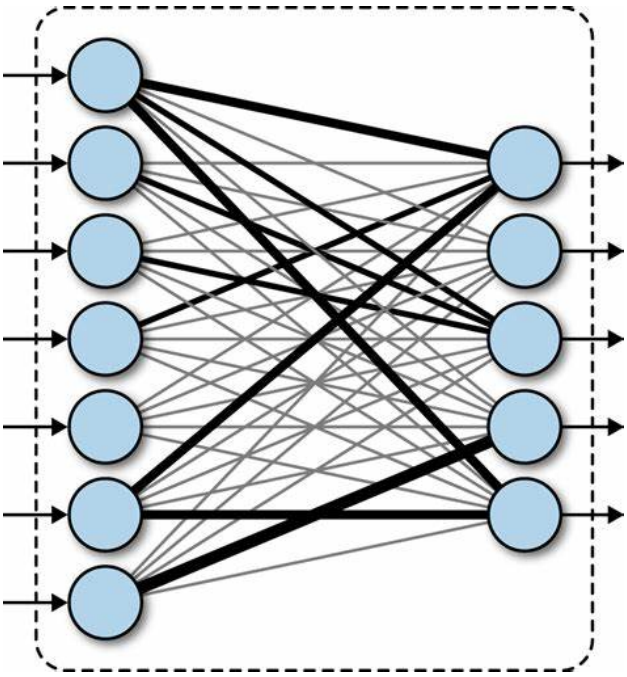


Figure 4. Example of fully connected layer.

While much information is lost in the pooling layer, it also has several benefits to CNN. They help to reduce complexity, improve efficiency, and limit the risk of overfitting.

2.4. Fully Connected Layer

Fully connected (FC) layers connect every neuron in one layer to every neuron in another layer (Fig. 4), which is similar to the traditional MLP. This layer performs the task of classification based on the features extracted through the previous layers and their different filters, which is the flattened output matrix. While convolutional and pooling layers use *ReLu* functions, FC layers help to map the representation between the input and the output and usually leverage a *Softmax* activation function to classify multi-classes appropriately, producing a probability from 0 to 1 [7].

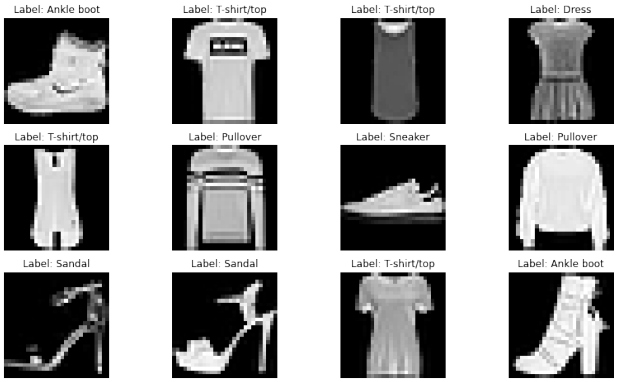


Figure 5. Samples of FashionMNIST dataset.

3. Experiment Analysis

In this experiment, MLP, CNN and CNN with batch normalization are compared to show the superior efficiency of CNN in image classification. Next, *hyperparameter tuning* is applied to find the best parameters to achieve the best performance on the **FashionMNIST** dataset. The evaluation metric to determine the better model is based on accuracy:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

3.1. Dataset

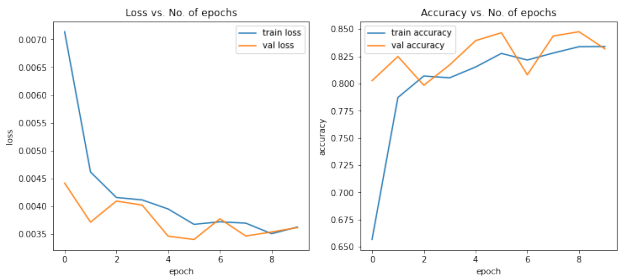
The dataset used in this experiment is **FashionMNIST** [19]. It is a dataset comprising of 28×28 grayscale images of $70,000$ fashion products from 10 categories, with $7,000$ images per category. The training set has $60,000$ images and the test set has $10,000$ images. Fashion-MNIST shares the same image size, data format and the structure of training and testing splits with the original MNIST. Some images in the dataset are shown in Fig. 5. In this experiment, 20% of the training set is used as validation set for evaluation while training.

3.2. Architecture Comparison

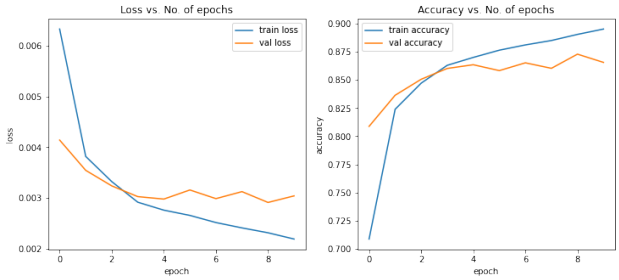
Model	Loss	Accuracy
MLP	0.004	0.82
CNN	0.003	0.857
CNN-BN	0.003	0.875

Table 1. The loss and accuracy on test set of three models.

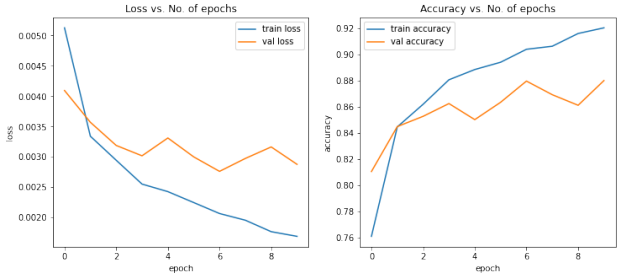
In this comparison, three different models are initialized as follows:



(a) The loss and accuracy of MLP.



(b) The loss and accuracy of CNN.



(c) The loss and accuracy of CNN-BN.

Figure 6. The loss and accuracy comparison of three models.

- **Multi-Layer Perceptron (MLP)**: Two hidden fully connected layers with *ReLU* activation function, *Dropout* to prevent overfitting; and one output layer with a *Softmax* activation function.

- **Convolutional Neural Network (CNN)**: Two convolutional layers with *ReLU* activation function and one output layer with a *Softmax* activation function.

- **Convolutional Neural Network with Batch Normalization [11] (CNN-BN)**: Same as CNN architecture except adding *batch normalization* after each convolutional layer.

Batch normalization is chosen in this comparison due to the following advantages [4, 5, 14]:

- Using batch normalization makes the network more stable during training. Therefore, it helps speed up the training process.

- It solves the problem of internal covariate shift. Through this, batch normalization ensures that the input for every layer is distributed around the same mean and standard deviation.

- Deep neural networks can be sensitive to the technique used to initialize the weights before training. However, the stability of training brought by batch normalization can make training deep networks less sensitive to the choice of weight initialization method.

- The *Dropout* can be removed. Batch Normalization provides similar regularization benefits as *Dropout* since the activations observed for a training example are affected by the random selection of examples in the same mini-batch.

The training process is performed with the same configuration:

- **Batch size**: 128

- **Epochs**: 10

- **Learning rate**: 0.01

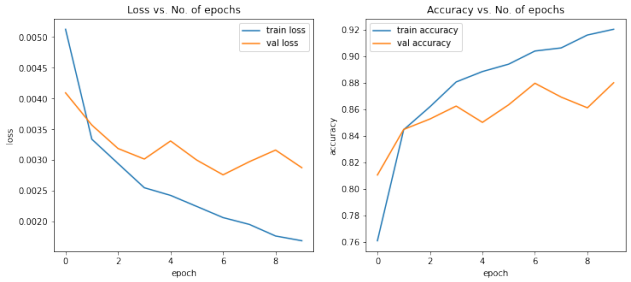
- **Optimizer**: Adaptive Moment Estimation (Adam)

- **Loss**: Negative Log-Likelihood

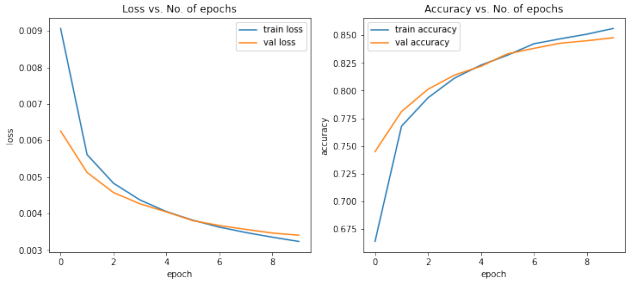
As shown in Table. 1, the CNN works better than MLP, which is as expected since the CNN is more effective at extracting useful information from images such as lines, curves and edges. Furthermore, thanks to the benefits of batch normalization, there is no doubt that the CNN-BN model gives a better result than CNN, with a test accuracy of 85.7%. .

3.3. Hyperparameters Tuning

Since CNN-BN is the best model in the comparison in section 3.2, further analysis will only be conducted on this model.



(a) The loss and accuracy with Adam optimizer.



(b) The loss and accuracy with SGD optimizer.

Figure 7. The loss and accuracy comparison of CNN-BN model between Adam and SGD optimizer.

3.3.1 Optimizer Comparison

There are some arguments about optimizers that Stochastic Gradient Descent (SGD) is better at generalizing solutions than Adam. The test performance worsens even when Adam achieves the same or lower training loss than SGD. Adam often displays faster initial progress on the training loss, but its performance quickly plateaus on the test error [12]. Therefore, an experiment is conducted to determine whether the argument above is correct. After the same training process as section 3.2, the results are shown in Table. 2 proves that Adam has a much faster convergence speed than SGD after 10 epochs, while the training curve in SGD (Fig. 7) fluctuates less.

Optimizer	Loss	Accuracy
Adam	0.003	0.875
SGD	0.003	0.857

Table 2. The loss and accuracy on test set of CNN-BN model using Adam and SGD optimizers.

3.3.2 Tuning With Ray Tune

Because each optimizer has its unique benefit, hyperparameter tuning will be applied to both optimizers to find the best option for this classification problem. In this process, one hidden layer is added to the CNN-BN model with an unknown *number of nodes (l1)*, whose most suitable value

batch size	l1	lr	loss	accuracy
64	64	0.01	0.0063063	0.866187
128	128	0.0001	0.00886682	0.691167
128	**64**	**0.001**	**0.00290222**	**0.8735**
128	256	0.01	0.00310922	0.86925
64	256	0.0001	0.0118035	0.747375
32	128	0.0001	0.022036	0.763437
128	128	0.01	0.00309437	0.863417
128	64	0.01	0.00431525	0.80635
32	64	0.0001	0.0235701	0.758104
32	256	0.001	0.013543	0.843333
64	64	0.0001	0.0152728	0.731479
32	64	0.01	0.0168359	0.803833
128	256	0.0001	0.00551326	0.755729
64	128	0.001	0.00611342	0.860458
32	64	0.001	0.0139218	0.83785

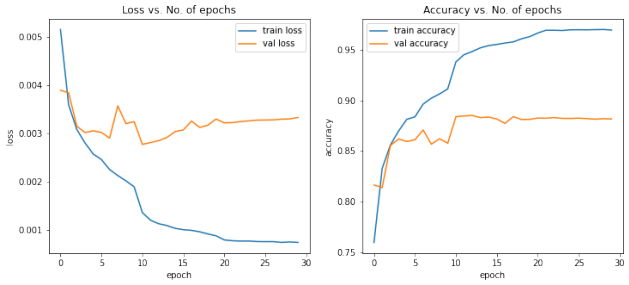
Table 3. Tuning results with Adam optimizer

batch size	l1	lr	loss	accuracy
32	128	0.0001	0.0467116	0.643312
32	128	0.01	0.011214	0.873021
64	128	0.0001	0.0292318	0.492229
64	64	0.0001	0.0334248	0.222208
128	256	0.001	0.00717465	0.727292
32	256	0.001	0.0423066	0.678625
32	64	0.001	0.0476746	0.644208
64	**64**	**0.01**	**0.00661299**	**0.848417**
128	64	0.0001	0.0162137	0.314979
64	128	0.01	0.00721858	0.827313
128	128	0.001	0.00778187	0.731875
32	256	0.0001	0.066671	0.25625
64	256	0.001	0.0210749	0.683292
128	64	0.001	0.00777006	0.724604
32	256	0.01	0.0131995	0.848208
32	128	0.001	0.0452574	0.640729
64	256	0.0001	0.035531	0.1095

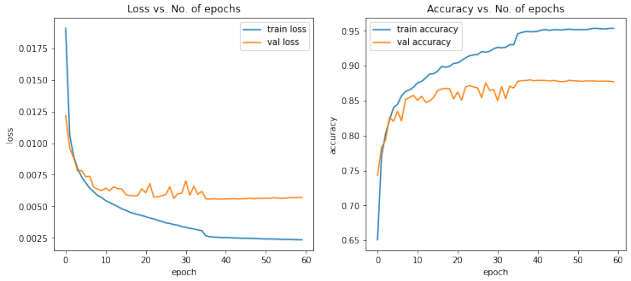
Table 4. Tuning results with SGD optimizer

will be found with the help of tuning along with parameters such as *batch size* and *learning rate*. This change aims to increase the complexity of the FC layers for class prediction with a desire to improve the prediction accuracy. The tuning process is performed with the help of the **Ray Tune** library, and the combination of parameters will be chosen randomly from these parameter sets:

- **Batch size**: $\{32, 64, 128\}$
- **Learning rate**: $\{1e-4, 1e-3, 1e-2\}$
- **l1**: $\{64, 128, 256\}$



(a) The loss and accuracy in final train with Adam optimizer.



(b) The loss and accuracy in final train with SGD optimizer.

Figure 8. The loss and accuracy comparison of CNN-BN model in final train between Adam and SGD optimizer.

After tuning with max epochs of 10, the best configurations for the CNN-BN model are **{l1: 64, learning rate: 0.001, batch size: 128}** and **{l1: 64, learning rate: 0.01, batch size: 64}** for Adam and SGD respectively. These results are obtained based on the validation losses which are shown in Table. 3 and Table. 4.

3.4. Final Training

Optimizer	Loss	Accuracy
Adam	0.003	0.88
SGD	0.006	0.872

Table 5. The final train evaluation on test set of CNN-BN model using Adam and SGD optimizers.

Based on the tuning results in section 3.3.2, the experiment moves to the final training process. The final training is divided into two parts for Adam and SGD optimizer, and the learning rate scheduler is applied differently in two parts in order to gain higher performance. In the first part with the Adam optimizer, instead of using the learning rate in the best configuration found earlier, the value of 0.01 is used to best fit with the learning rate scheduler. As a result, after 30 epochs and a learning rate step of 0.1 every 10 epochs, the test accuracy increased to 88% In the second part with the SGD optimizer, the learning rate step is 0.1 in every 35 epochs. Since the convergence speed of SGD is slower than Adam, the number of epochs in this training is extended to 60. As a result, the test accuracy obtained is 87.2%, which is better than the result in Table. 2, but still lower than the result achieved by Adam (Table. 5). The visualization of loss and accuracy in the final train of both optimizers are shown in Fig. 8. The results also show that adding an FC layer is effective, and the accuracy increases by 1.5% in SGD and 0.5% in Adam. However, the loss seems to be slightly raised, and the accuracy has a small drop in the last few epochs. This matter could be a sign that the model is on the edge of overfitting.

4. Conclusion

The model comparison in section 3.2 shows that the CNN model is more effective at extracting features from images than MLP. Furthermore, batch normalization can boost the model's performance thanks to its ability to ensure that the input for every layer is distributed around the same mean and standard deviation. The experiment also applied several training techniques, such as hyperparameter tuning and learning rate schedule, which also helped increase the test accuracy in the last train. In the final training, the test accuracy of CNN-BN model using Adam optimizer is 88% in just 30 epochs which is 0.8% higher than SGD optimizer with 60 epochs. This result also confirms that Adam converged faster. Although the plateau phenomenon happened (Fig. 8), it is still a better option than SGD in this **FashionMNIST** classification problem. For future direction, *data augmentation* and *early stopping* are the potential techniques to prevent overfitting. Another prospective approach to achieve better results is *transfer learning*. Due to the purpose of understanding the fundamental of CNN, transfer learning is not applied in this experiment. However, this method can enhance accuracy by leveraging pre-trained weights from advanced deep models such as *VGG16, InceptionNet*, and *ResNet*. Besides, leveraging high-performance model architecture will shorten the time for thinking about the network architecture design, such as adding an FC layer in the section 3.3.2.

References

[1] Deep learning - wikipedia. https://en.wikipedia.org/wiki/Deep_learning. (Accessed on 10/10/2022). 1

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018. 2

[3] Panagiotis Antoniadis. Calculate the output size of a convolutional layer. https://www.baeldung.com/cs/convolutional-layer-size, September 2022. (Accessed on 10/17/2022). 2

[4] Ramji Balasubramanian. Batch normalization and its advantages — medium. https://medium.com/mlearning-ai/batch-normalization-and-

its-advantages-a1dc52af83d0, March 2021. (Accessed on 10/17/2022). 4

[5] Jason Brownlee. A gentle introduction to batch normalization for deep neural networks. https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/?fbclid=IwAR3UFjHuMTocAoyCCUjY5OaJT61VhNOEjzrzsFk_USaIozhv_sDtHoaNDH8, January 2019. (Accessed on 10/17/2022). 4

[6] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015. 2

[7] IBM Cloud Education. What are convolutional neural networks? — ibm. https://www.ibm.com/cloud/learn/convolutional-neural-networks, October 2020. (Accessed on 10/10/2022). 2, 3

[8] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization, 2020. 1

[9] Ethan Harris, Antonia Marcu, Matthew Painter, Mahesan Niranjan, Adam Prügel-Bennett, and Jonathon Hare. Fmix: Enhancing mixed sample data augmentation, 2020. 1

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 1

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 4

[12] Sieun Park. A 2021 guide to improving cnns-optimizers: Adam vs sgd — medium. https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008, June 2021. (Accessed on 10/14/2022). 5

[13] Sumit Saha. A comprehensive guide to convolutional neural networks — towards data science. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, December 2018. (Accessed on 10/17/2022). 3

[14] Shipra Saxena. Batch normalization — what is batch normalization in deep learning. https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/, March 2021. (Accessed on 10/17/2022). 4

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. 1

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015. 1

[17] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. 1

[18] Muhammad Suhaib Tanveer, Muhammad Umar Karim Khan, and Chong-Min Kyung. Fine-tuning darts for image classification, 2020. 1

[19] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. 3