Image Classification

Jingxu (Jack) Hu
jingxu.hu@mail.mcgill.ca

Alex Goulet
alex.goulet@mail.mcgill.ca

Karanvir Sidhu
karanvir.sidhu@mail.mcgill.ca

Abstract

This mini-project assesses the performance of different architectures of Convolutional Neural Network (CNN) on the MINIST dataset. The training and test dataset contained $60,000$, and $10,000$ gray-scaled images, respectively, belonging to 9 distinct classes. In this paper, the efficiency and the accuracy of different CNN architectures were investigated. Various, architectural schemes ranging from simple 2-4 layered CNNs to very deep architectures such as VGG-11,VGG-13, VGG-16, and VGG-19 were implemented and examined in depth. The choice of different optimization algorithms such as AdaDelta, Stochastic Gradient Descent, and the Adam optimization algorithm was examined, and the paper uses the Adam Optimization method as it offers computational efficiency and has a higher performance [1]. Moreover, the influence of hidden layer activation functions, such as ReLU and leaky ReLU, was examined, and it was found that leaky ReLU resulted in higher accuracy. The effect of dropout rate and batch size was tested, and it was found out that the batch size of 64 and dropout rate of 25% yielded higher accuracy. The final model accuracy on the test set for VGG-11, VGG-13, VGG-16 with image augmentation, and VGG-19 were 90.55%, 91.33%, 95.56%, and 89.77%, respectively.

1 Introduction

1.1 Overview

Machine learning has countless applications in the modern world as it is a powerful tool that can solve complex problems for which the solution is often too difficult to obtain. One of the significant fields of study in machine learning is computer vision. In this field, a popular vision algorithm, namely Convolution Neural Network (CNNs), attempts to gain an in-depth understanding of real-world scenes by performing various types of recognition tasks on visual data. The focus of this paper shall be to examine the classification performance (in terms of accuracy) of various CNN architectures given images containing multiple clothing items with their respective price labels. Thus, distinct CNN models were constructed in order to compare different performance benchmarks and help determine the best architecture. Furthermore, the hyper-parameters for each model were tuned in order to optimize the efficiency of the algorithms as well as to maximize their accuracies on the test set. Finally, the CNNs were trained and evaluated on a modified version of the Fashion-MNIST dataset [4].

1.2 Findings

Throughout the conducted experiments, multiple CNN architectures were used in the classification process. Out of all the CNN architectures presented, VGG-16 with random image rotations provided the highest test set accuracy. In order to obtain the optimal performance from these CNN architectures, their respective hyper-parameters were tuned until satisfactory. The final model accuracy on the test set for VGG-11, VGG-13, VGG-16 with random image rotation, and VGG-19 were found to be 90.55%, 91.33%, 95.56%, and 89.77%, respectively.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

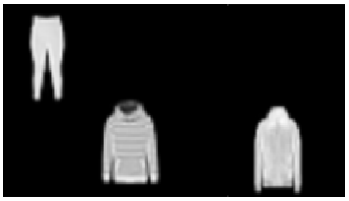


Figure 1: Example image from the Fashion-MNIST dataset [4].

2 Data Sets

2.1 Overview

The CNNs are used to solve a multi-class image classification problem, namely price identification. The dataset used in this paper represents a modified version of the original Fashion MNIST dataset [4]. Each image within the set is a grey-scale (single channel) image of size 64x128 pixels. Each pixel intensity value ranges from 0 (black) to 255 (white). All the images comprise exactly three articles of clothing, each with their respective price. The articles and respective prices are as follows: t-shirt = \$1, trouser = \$2, pullover = \$3, dress = \$4, and coat = \$5. The label of any given image corresponds to the total price of all of the three articles present in said image. Figure 1 shows an example image from the Fashion-MNIST dataset. It should be noted that the total price of the articles ranges from \$5 to \$13 for this given dataset, and thus the problem presents a 9-way multi-classification output. The training dataset is composed of 60000 images, along with the corresponding class labels. The test-set, on the other hand, contains 10000 images for which the class labels are unknown.

2.2 Preprocessing

The training set undergoes multiple preprocessing steps before being fed into the various CNN architectures. Firstly, the images are converted from Numpy intensity arrays into Pytorch intensity tensors to allow for high dimensional arithmetic operations in the GPU [2]. The second step is to normalize the images such that pixels range between -1 and 1. Raw grey-scale pixel values can also be directly fed into the CNN models, but this can result in longer training time and reduced accuracy. A further optional step would be to apply image augmentation to the training set in order to increase the number of training examples without acquiring additional new images [5]. The images are augmented in such a way as to preserve the key pixel features while re-arranging the pixels enough that it adds some noise to differentiate them from the original images and turn them into suitable, non-superfluous training examples [5]. In this paper, one image augmentation technique, **randomRotation()** was applied to both the training set and the test set. This augmentation technique was able to increase the validation and test set accuracies by a substantial amount. The reason being, all the proposed CNN models, were exposed to this wide variety of re-arranged key pixel features captured by these rotated images.

3 Proposed Approach

Image recognition tasks have been made possible due to recent advancements in processing resources (GPUs), and the availability of large public image repositories with varying complexities [1]. Each image classification task on a specific image repository requires a different pipeline/CNN configuration to achieve a desirable accuracy. The most important factors include convolution + pooling kernel sizes, strides, padding, input sizes, output sizes, weight decays, number of layers, activation functions, batch normalization, batch sizes, loss functions, and optimizers. It is important to note that as the number of convolution layers increases, the number of training parameters/filters increases as well. This specific increase may potentially cause the model to only filter certain patterns of objects and render it incapable of generalizing to unseen data. Thus, each CNN configuration requires meticulous tuning of its hyper-parameters so the model neither over-fits nor under-fits for a given image dataset.

3.1 Convolutional Neural Network Architecture

After several experiments with various CNN configurations, the VGG architecture was determined to generalize the best to unknown data given the proposed depth of the network [1]. The architecture was implemented in the Pytorch framework [2] by referencing its past literature [1]. The validation accuracy obtained for each VGG architecture was greater than each of its other simple CNN counterparts. In ascending order of validation accuracies, they are: VGG-11, VGG-19, VGG-13, and VGG-16. In addition, this architecture adapts well to a wide range of classification tasks and image datasets [1]. In most cases, VGG nets were able to match or outperform complex recognition pipelines built around shallow CNN image representations [1].

The VGG architectures used within this paper were slightly modified from the original specifications. The input channel size was reduced from 3 to 1 as the images used were single channel greyscale images. In addition, batch normalization and dropout layers were added at the convolution and FC layers to combat over-fitting. Lastly, the output layer size of the last fully connected layer was modified to support a 9-way classification output. These architectures consisted of VGG-11, VGG-13, VGG-16, and VGG-19. The convolution layer configurations fixed the convolution kernel size to 3 pixels, pooling kernel to 2 pixels, convolution stride of 1 pixel, pooling stride of 2 pixels, and padding of 1 pixel in both height and width dimensions [1]. In the FC layer configurations, the use of 3 layers is proposed where the size of the first two layers were fixed to 4096-channels, and the last to 1000-channels [1]. In-between the convolution layers, the ReLU activation function was used to model the non-linearity of the data [1]. The last layer consisted of a softmax layer to return class probabilities [1]. In all 4 VGG configurations, 5 Max-Pooling and 3 Fully-Connected layers were used. They all shared the same convolution layer specifications but differed in terms of the convolution layer depth [1].

3.2 CNN Hyper-parameter Tuning

This subsection shows the process of hyperparameter tuning used for the proposed VGG-16 technique. A similar hyperparameter tuning process was used for tuning the VGG-11, VGG-13, and VGG-19 CNN architectures.

3.2.1 Optimizers and Loss Functions

For the proposed VGG-16 architecture, we used Adam optimizer. The other choices for the optimizer included AdaDelta and the Stochastic Gradient Descent. Adam method is an expansion of the AdaDelta method since it uses the second-order gradients and momentum (momentum is used to smoothen the information obtained from the first-order gradients). Also, the Adam method is a faster optimizer method compared to stochastic Gradient Descent and the AdaDelta method [1; 3]. Through trial and error, it was determined that the default values provided by PyTorch worked the best for the hyperparameters such as momentum, β_1, and β_2. For the error metrics, the Cross-Entropy Loss was found to be the best error metric as suggested in [1].

3.2.2 Learning Rate

The Adam Optimization Algorithm adaptively computes the learning rate. The input learning rate sets an upper bound for the adaptive learning rate computed by the Adam optimizer. The higher input learning rates lead to a lower validation accuracy; the best input learning rate was determined using trial and error and was found to be equal to 1×10^{-5}. However, using the input learning rate of the 1×10^{-5} led to the plateauing of the accuracy around 72%. To overcome the plateauing, an error learning rate scheduler was used to adaptively change the learning rate to a lower value after every 3 epochs [2].

3.2.3 Dropout Rate

Dropout is a regularization technique used to prevent overfitting [6]. In contrast to regularization methods, which modifies the loss function, the dropout method randomly drops neurons from the neural networks during training [6]. This is equivalent to training each iteration on a different neural network. In this project, the dropout rate was systematically chosen between 0% to 40%. Figure 2 shows three experiments where the dropout rate was set to 0%, 25%, and 40%. As can be seen in Figure 1, the dropout rate of 0% leads to overfitting, while the dropout rate of 40% resulted in an

overall decrease in accuracy. The best dropout rate was found to be equal to 25% since the validation and training accuracies are of the same order and provide a higher overall yield.

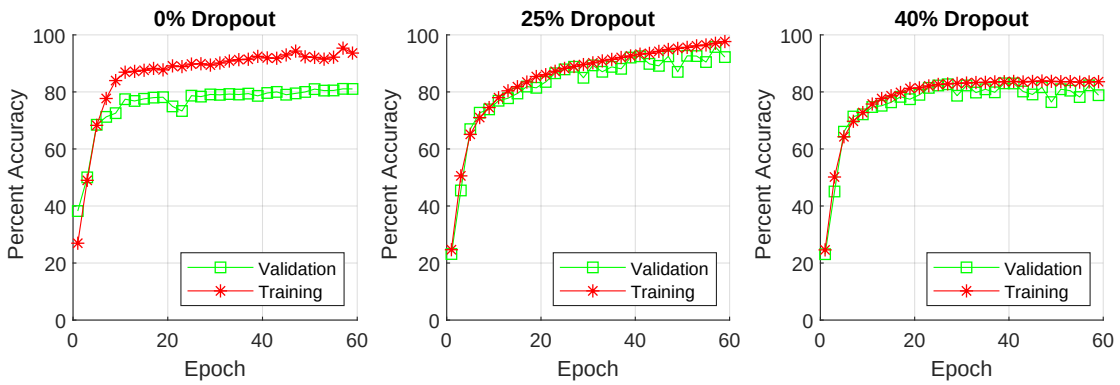


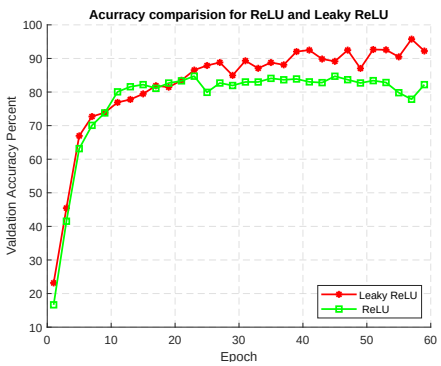
Figure 2: The validation accuracy and the training accuracy for the different Dropout rates.

3.2.4 Activation Functions

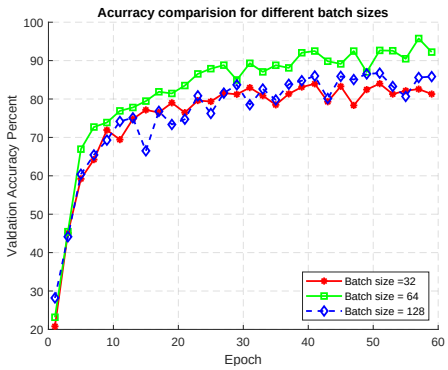
This subsection provides a brief overview of the two activation functions and the results obtained in the experiments. ReLU stands for the rectified linear unit; it is the most commonly used activation function used in CNNs; it can be defined as $y = max(0, x)$. ReLU results in sparser and more concise models, compared to the Sigmoid function, which leads to higher accuracy and lower computational effort. However, for negative values of x, the derivative of the ReLU function is equal to zero, which implies that no learning is performed on the layers below the dead ReLU. Due to this reason, the Leaky ReLU was used instead. This variant of ReLU performs better compared to the original as the derivative is never equal to zero. As such, the Leaky ReLU reduces the training time and increases accuracy. For choosing the activation function, we experimented with ReLU and leaky ReLU activation functions. The results of the experiments are shown in Figure 3a. As can be seen in the figure, the leaky ReLU learns faster than the standard ReLU.

3.2.5 Batch Size

The batch size controls the accuracy of the error gradient and thus affects the accuracy of the CNN. Therefore, three different batch sizes of 32, 64, and 128 were experimented with. The results of the batch size are shown in Figure 3b. As can be seen in Figure 3b, the batch size of 64 provides an optimal value of the accuracy for the given data set.



(a) Accuracy comparison for the ReLU and leaky ReLU activation functions.



(b) Accuracy comparison for different batch sizes.

Figure 3: The above figure (a) shows the accuracy comparison for different activation functions and the figure (b) shows accuracies for different batch sizes.

4 Results

This section describes the results of the various experiments performed using all the CNN configurations described in Table 1. The entire dataset was divided into two sets, 80% of the data were used for training, and the remaining data were used for validation. The validation set was used to select the hyper-parameters to create the best model. After choosing well-performing hyperparameters, the CNN architecture was re-trained on the full data set. The epoch numbers listed in 1 indicate a point in time at which the validation accuracies reached a plateau. As can be seen in Table 1, the simple CNN and the VGG-11 architecture did not perform well. On the other hand, the VGG-13 and VGG-16 architectures yielded higher validation and training accuracies. Consequently, VGG-13 and VGG-16 architectures also provided a higher test set accuracy of 90.55% and 91.33%, respectively. The highest test set accuracy of 95.56% was achieved using the VGG-16 architecture with image augmentation.

Table 1: This table summarises the results for all the CNN architectures used.

	CNN Architecture	Epochs	Training	Validation Accuracy	Testset Accuracy	Run Time
1.	Simple CNN 2-4 layers	20	51.92%	50.523%	—	13.33 min
2.	VGG-11	110	78.64%	76.86%	—	91.6 min
3.	VGG-13	263	93.43%	92.33%	90.55%	219.2 min
4.	VGG-16	261	98.19%	97.62%	91.33%	394.5 min
5.	VGG-16 + Image Aug.	190	98.73%	98.56%	**95.56%**	380.1 min
6.	VGG-19	110	95.27%	88.61%	89.7%	275 min

5 Discussion and Conclusion

As can be seen in Table 1, the simpler CNN architectures performed poorly in classifying the images. This suggests that a deeper and more complex model is required for achieving predictions. According to Table 1, both training and validation accuracies significantly increased as the number of convolution layers increased, and the CNNs deepened. However, for the case of VGG-19, the increased complexity in layer configuration led to over-fitting. This is because, in the case of VGG-19, the validation and test set accuracies were lower compared to training accuracy. Therefore, it can be concluded that there exists a fine balance between the number of convolution layers and prediction accuracy. Another noteworthy observation drawn from the results is that VGG-16 with image augmentation provided a higher test set accuracy compared to the VGG-16 architecture without image augmentation. This is despite the fact that the number of training epochs used for the model without image augmentation is higher. This suggests a substantial significance of image augmentation in data preprocessing. Thus, other image augmentation techniques such as image translation, cropping, noise injection, etc. can be applied to achieve a higher classification accuracy [5].

5.1 Future Work

Several directions for further work based on the results of this project are described below:

- **Different CNN Architectures** In this report, the VGG based CNN architectures were explored in depth. The other CNN architectures such as ResNet, AlexNet, and LeNet can be further implemented to investigate whether higher accuracies can be achieved [5]
- **PCA Based Image Compression** A PCA based image compression is very popular in the literature, and it can be implemented to reduce the image feature size. Moreover, apart from reducing the size of the image, PCA helps in constructing a more significant yet smaller set of features that span the image.
- **Distributed Training using Multiple GPU's**: As can be seen in Table 1, the run time is very long for deeper CNN architectures. The DDP architecture should be considered, as it is a widely adopted single-program multiple-data training paradigm. In DDP, the model is replicated on every GPU process, and every model replica will be fed with a different set of input data samples. During execution, DDP takes care of gradient communications to

keep model replicas synchronized and overlaps it with the gradient computations to speed up training [2].

6 Statement of Contributions

The project tasks were split equally amongst the three team members. Alex Goulet and Karanvir Sidhu were in charge of running different experiments to optimize the CNN architectures. Jack Hu was in charge of implementing the data preprocessing pipeline and the various CNN architectures used in the experiments.

7 Appendix

7.1 Appendix A: Pytorch Library

The entire data preprocessing pipeline, convolution neural network architectures, training logic, and prediction logic were implemented in Facebook's deep learning framework, namely Pytorch. The following classes and libraies were used from the aforementioned framework [2].

- **torch.nn**: For packing various CNN configurations.
- **torch.Tensor**: For transforming image pixels from numpy intensities to Pytorch GPU supported tensors.
- **torch.cuda**: For checking the availability of GPUs and creating a GPU instance.
- **torch.nn.functional**: For CNN convolution/linear layer functions, loss functions, and activation functions.
- **Torchvision**: For implementing the image preprocessing pipeline (various image transforms)
- **torch.optim**: For the types of gradient descent algorithms used and various learning-rate schedulers.
- **torch.Storage**: For storing and reloading the trained CNN models from checkpoints.

References

[1] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), Kuala Lumpur, 2015, pp. 730-734, doi: 10.1109/ACPR.2015.7486599.

[2] "PyTorch documentation," PyTorch documentation: PyTorch 1.7.0 documentation. [Online]. Available: https://pytorch.org/docs/stable/index.html. [Accessed: 06-Dec-2020].

[3] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[4] K. Rasul, H. Xiao, "Fashion-MNIST". [Online]. Available: https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/. [Accessed: 01-Dec-2020].

[5] Shorten C, Khoshgoftaar TM. A survey on image data augmentation for deep learning. Journal of Big Data. 2019 Dec 1;6(1):60.

[6] Garbin, Christian, Xingquan Zhu, and Oge Marques. "Dropout vs. batch normalization: an empirical study of their impact to deep learning." Multimedia Tools and Applications (2020): 1-39.