Image Classification with Deep Learning Methods

Prabhsimran Singh
McGill University
Montreal, Canada
prabhsimran.singh@mail.mcgill.ca

Maninder Bir Singh Gulshan
McGill University
Montreal, Canada
maninder.gulshan@mail.mcgill.ca

Amal Koodoruth
McGill University
Montreal, Canada
amal.koodoruth@mail.mcgill.ca

Abstract

In this project, we have investigated the performance of supervised machine learning classification models to predict the total price of an image containing three clothing articles in the Modified Fashion-MNIST dataset. The goal of this research is to find a model which can accurately predict the total price of the articles in an image. For this study, we implemented Neural Network(NN)[12], Convolution Neural Network (CNN)[8] and deep CNN. We developed a basic model which included multiple CNN layers and fully connected NN layers with ReLu[3] non-linearity function. We also explored the performance of the classifiers trained with Residual Neural Network (ResNet), VGG-16 and AlexNet. Preprocessing techniques like normalization and changing the size of the image are used to improve the performance of the models. The VGG-16 model performed best with an accuracy of 95.55% on the validation set. It had an accuracy of 97.73% on Kaggle.

1 INTRODUCTION

Image Classification is a fundamental task that attempts to comprehend an entire image as a whole. Moreover, it refers to images in which one or more objects appear and are analyzed. In this report, we classified images from a modified version of the Fashion-MNIST dataset and predicted the total price as output. For the classification, we used ResNet18[4], ResNet152[5], AlexNet[6], and VGG16[7](OxfordNet) architecture models, modified to suit the requirements of the study. In order to find the best model, we tuned the hyper-parameters of our model. We evaluated the performance of our models using different learning rates, epoch values and different depths of NN and CNN layers. We also varied the size of image filters to find the best filter for our classification model.

To find the best optimizer for our models we used SGD(Stochastic Gradient Descent)[9][13] and Adam(Adaptive Moment Estimation)[14]. For SGD we used different values of momentum and learning rate and for Adam optimizer we used different learning rate to explore the effects on our predictions. Moreover, we used scheduler to dynamically update the learning rate. To calculate the loss of our model, cross entropy loss was used. We observed that the models with more number of parameters took more training time. We trained the models until the validation set accuracy stabilized and training loss became almost zero. The model trained using a architecture to similar VGG16 with Adam optimizer performed best and had an accuracy of 95.55% on the validation set and 97.73% on Kaggle.

2 DATASETS

The modified Fashion MNIST[2] dataset has a total of 60,000 grey scale images. Each image has a height of 64 and width of 128 and contained 3 clothing articles, belonging to a set of 5 articles that are T-shirt/top, Trouser, Pullover, Dress and Coat. Each article has a cost price, with coat being the most expensive article costing 5$ and T-shirt/top being

Figure 1: Image with class label 5

Figure 2: Image with class label 13

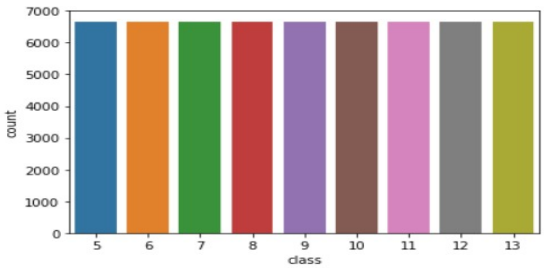

Figure 3: Class Labels

Article Name	Price($)
T-shirt/top	1
Trouser	2
Pullover	3
Dress	4
Coat	5

Figure 4: Articles with price list

least expensive costing 1$. The class label of each image is the total price of articles contained in an image. For an instance, Figure 1, has a class label of 5, because the total cost of articles in the Figure 1 is 5$, whereas Figure 2, has a class label of 13. Figure 4 shows the price of every article in ascending order. The range of class labels is from 5 to 13, making it a 9 class classification problem. To find the distribution of class labels in the dataset we plotted a bar graph, Figure 3. From the Figure 3, it can observed that the data is uniformly distributed with 6667 images belong to class label 5 to 10 and 6666 images belong to 10 to 13 class label.

For preprocessing, we used $transforms.ToTensor()$[1], $transforms.Normalize((mean),(std))$[1] , $transforms.Resize(H,W)$[1]. $transforms.ToTensor()$ is used to convert PIL image to tensor image which has shape of (C, H, W) where C is a number of channels, H and W are image height and width. Every image of modified Fashion MNIST dataset has shape of (1, 64, 128). Because the images are grey scale, therefore it has channel(C) equal to 1. $transforms.Resize()$ is used to change the height and width of images. For this study we have used image size of 64 x 128, 226 x 226 and 64 x 64. $transforms.Normalize()$ is used to normalize a tensor image with mean and standard deviation (std).After normalization $output[channel] = (input[channel] - mean[channel])/std[channel]$. For this study we used mean equal to 0.5 or 0.1307 and std equal to 0.5 or 0.3081.

Because the class labels were in range 5 to 13, we subtracted the minimum value i.e 5 from all the class labels changing the range to 0 to 8. This made it easier for us to train our models. For test data predictions we added 5 to get the real values for class labels.

3 PROPOSED APPROACH

In this section, we will discuss different methods we implemented or used for image classification, training and testing and for tuning the hyper-parameters.

3.1 Base Model

The base model we implemented was the Convolutional Neural Network (ConvNet or CNN)[13] which is one of the special type of Neural Network used effectively for image recognition as well as classification. We had used 80% i.e. 48000 images for the training data and 20% i.e. 12000 images for the validation data from the training dataset. The architeture of basic model is as follows:

→ 1 x convolution layer of 10 channel of 3x3 kernel and 1 padding
→ 1 x maxpool[10] layer of 2x2 pool size and stride 2x2
→ 1 x convolution layer of 20 channel of 3x3 kernel and same padding
→ 1 x maxpool layer of 2x2 pool size and stride 2x2
→ 1 x fully connected linear layer with output 50
→ 1 x fully connected linear layer with output 9

For non-linearity we used ReLu function and for output we used softmax. We trained and tested four base models, with different normalization values and image size.

3.2 AlexNet

AlexNet was another model that we tried to implement. AlexNet comprises of eight layers with weights; the first five are convolutional and the remaining three are fully- connected. The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels. We edited the model in 3 ways. Instead of feeding in 224x224 RGB images, our input was 128x64 grayscale images. Also, the number of classes in this project is 9, instead of 1000 used in the original paper. [9]

3.3 Residual Neural Network

Another approach we used was the implementation of residual neural networks, more commonly known as ResNet. This method aims at reducing the problem of vanishing gradient by skipping one or more layers in a deep neural network, as illustrated in Figure 5 below[4]. For this study we explored the performance of ResNet18 and ResNet152. The difference between the two is that ResNet18 contains 18 layers (11,174,857 trainable parameters) while ResNet152 contains 152 layers (58,227,785 trainable parameters).

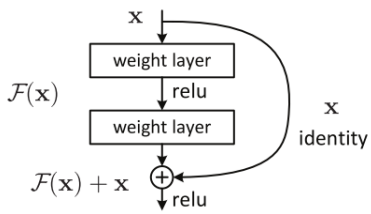

Figure 5: ResNet skip path[15]

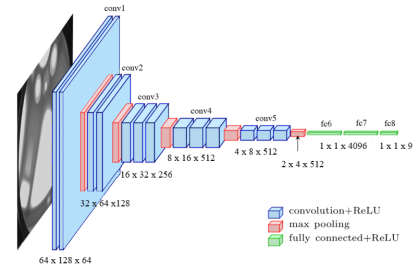

Figure 6: VGG-16 Architecture

3.4 VGG-16

Finally, we implemented an edited VGG-16 model. Originally, the model is designed to take in 224 x 224 RGB images and classify them. The authors used convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. The detailed architecture is as follows:
→ 2 x convolution layer of 64 channel of 3x3 kernel and same padding
→ 1 x maxpool layer of 2x2 pool size and stride 2x2
→ 2 x convolution layer of 128 channel of 3x3 kernel and same padding
→ 1 x maxpool layer of 2x2 pool size and stride 2x2
→ 3 x convolution layer of 256 channel of 3x3 kernel and same padding
→ 1 x maxpool layer of 2x2 pool size and stride 2x2
→ 3 x convolution layer of 512 channel of 3x3 kernel and same padding
→ 1 x maxpool layer of 2x2 pool size and stride 2x2

→ 3 x convolution layer of 512 channel of 3x3 kernel and same padding
→ 1 x maxpool layer of 2x2 pool size and stride 2x2

We adapted this model to our dataset: we changed the input channel size to 1 because our dataset is in grayscale. Also, the input size of image was 64 × 128 instead of 224 × 224 as in the original model. As a consequence we are left with a model with 48,337,865 trainable parameters.Refer to Figure 6 to view our modified VGG-16 architecture. For model summary refer Figure 10.

4 RESULTS

As mentioned in section 3, we implemented different models. Their performance based on training with 70% of the shuffled dataset is presented in Table 1 below.

Table 1: Accuracy of different models

Architecture	**Optimizer**	**Learning Rate**	**Momentum**	**Batch Size**	**Accuracy %**
VGG 16	Adam	0.01	-	128	94.94
VGG 16	**Adam**	**0.01**	-	**32**	**95.55**
AlexNet	Adam	0.0001	-	100	56.6
ResNet18	Adam	0.01	-	100	93.4
ResNet18	SGD	0.01	0.5	100	94.6
ResNet18	SGD	0.01	0.7	100	95.1
ResNet152	Adam	0.01	-	100	94.5
ResNet152	SGD	0.01	0.5	100	94.7
ResNet152	SGD	0.01	0.9	100	93.1
Base Model	Adam	0.0001	-	100	72
Base Model	SGD	0.0001	0.5	100	11
Base Model	SGD	0.0001	0.7	100	52

The hyperparameters being varied were the optimizer, learning rate, momentum (for Stochastic Gradient Descent) and batch size. We also used a scheduler[16] for our optimizer with starting learning rate 0.01 as shown in the table above. This means that the learning rate was being adjusted during the training phase depending on the gradient of the loss.

Figure 7 shows how the training accuracy of the base model varied with the number of iterations (1 epoch = 500 iterations). The hyperparameters being changed in that experiment were the normalization constants and the image size. The batch size and the learning rate was set to 100 and 0.0001 respectively. Basic model used Adam optimizer. The model descriptions are as follows:

- Base Model 1 : image size: 64×128, normalization: 0.5, 0.5
- Base Model 2 : image size: 64×128, normalization: 0.1307, 0.3081
- Base Model 3 : image size: 64×64,normalization: 0.1307, 0.3081
- Base Model 4 : image size: 64×64 normalization : 0.5, 0.5

Since we observed that training worked best with normalization constants 0.1307 and 0.3081, we used this for the rest of the experiments.

We observe in Figure 8 how the training loss decreases for VGG-16 with the number of epochs.It can be observed that after 30 epochs, the training loss converged and became very small and relatively stable.

In the Figure 9, we compare the validation accuracies for the different models. We can see that the best models include ResNet152 (Adam optimizer), ResNet18 (SGD as optimizer with 0.7 momentum) and VGG-16(batch size of 32). Each model attempted to classify the test set on kaggle and the respective accuracies were 78%, 87% and 97%. Based on this, we can conclude that ResNet152 was overfitting the training and the validation set.

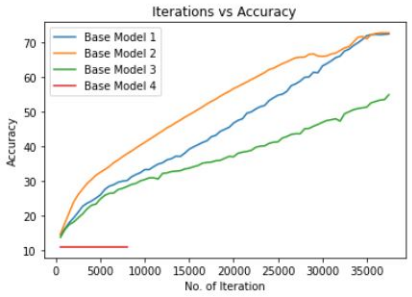

Figure 7: Base model accuracies

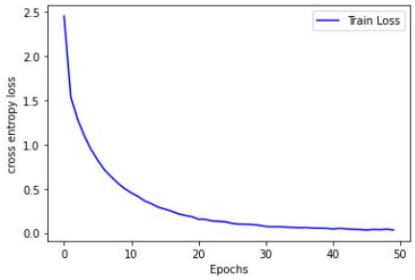

Figure 8: VGG-16 Training loss

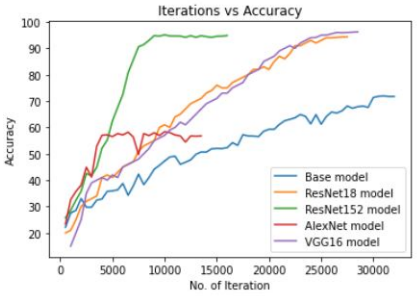

Figure 9: Models Accuracies

5 DISCUSSION AND CONCLUSION

In the research, the model based on our modified VGG 16 architecure with Adam optimizer performed best with an accuraccy of 95.55% on validation set and 97.73% on Kaggle. This study shows how accurately convolutional neural network can predict the class label. We explored the performance of the state-of-art CNN architectures on our dataset. We also used different optimizers, batch sizes and learning rate to find the best model.

To study the effect of normalization and image size on the accuracy we trained a base model, On evaluating the base model with different hyperparmeters as shown in Figure 7, base model with image size: 64×128 and normalization values of 0.1307 and 0.3081 performed better, thus we evaluated the rest of models with image size of 64 x 128 and normalization values of 0.1307 and 0.3081.

In case of ResNet18, the models trained with Adam and SGD had similar validation accuracies, but the SGD optimizer with momentum of 0.7 performed better, whereas in case of ResNet152 model with Adam optimizer performed better. Even though the ResNet18 and ResNet152 had similar validation accuracies of 95.1% and 94.7%, but on the test set Resnet18 had 87% whereas, ResNet152 had 79% accuracy. Based on these results we concluded ResNet152 may have overfitted the data. This shows that having more layers in the network not always result in higher accuracy.

In case of AlexNet we observed that loss became almost zero at 25 epochs but validation acccuracy was remained low around 56% therefore, for further investigation we implemented VGG-16 which performed much better than AlexNet. VGG-16 with Adam optimizer had validation accuracy of 95.55% whereas, ResNet18 had an accuracy of 95.1%. Because VGG-16 performed best we tried to study the effect of batch size, we re-trained our model with 128 batch size, but there was not much change in accuracy of the model but memory usage increased therefore, we kept the batch size of 32 for VGG-16.

In this project, we analyzed the images from modified Fashion-MNIST dataset, and tried to correctly classify the class labels by using 5 different CNN architectures and 2 different optimizers. To enhance the accuracy, we modified state-of-art CNN architectures to suit the needs of this study. The performances of these classifiers are investigated in terms of accuracy by changing hyper-parameters such as learning rate, optimizer, momentum and batch size. We also used preprocessing techniques like image resizing and normalization to further improve the accuracy. The models trained using VGG-16, ResNet18, ResNet152 had accuracies above 90%.

For future investigation, effect of other optimizer like Adadelta, RMSprop can be explored. A callback function can be implemented to early stop the training when loss becomes relatively small and stable. Also, architectures such as GoogLeNet can be used.

6 STATEMENT OF CONTRIBUTION

The work we present is the collective outcome of the effort made by every individual member of the team. Every member trained models with different optimizers, learning rate and batch sizes. For the coding part, every member learned to create CNN based classifier.

7 REFERENCES

[1] “torchvision.transforms,” torchvision.transforms - PyTorch 1.7.0 documentation. [Online]. Available: https://pytorch.org/docs/stable/torchvision/transforms.html

[2] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, Nov. 2012, doi: 10.1109/MSP.2012.2211477.

[3] H. Ide and T. Kurita, "Improvement of learning for CNN with ReLU activation by sparse regularization," 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, 2017, pp. 2684-2691, doi: 10.1109/IJCNN.2017.7966185.

[4] H. Song, Y. Zhou, Z. Jiang, X. Guo and Z. Yang, "ResNet with Global and Local Image Features, Stacked Pooling Block, for Semantic Segmentation," 2018 IEEE/CIC International Conference on Communications in China (ICCC), Beijing, China, 2018, pp. 79-83, doi: 10.1109/ICCChina.2018.8641146.

[5] WEI LIU, MIAOHUI ZHANG, ZHIMING LUO, and YUANZHENG CAI, “An Ensemble Deep Learning Method for Vehicle Type Classification on Visual Traffic Surveillance Sensors,” IEEE Xplore Full-Text PDF: [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8082506. [Published: 25-Oct-2017].

[6] Jing Sun, Xibiao Cai, Fuming Sun and J. Zhang, "Scene image classification method based on Alex-Net model," 2016 3rd International Conference on Informative and Cybernetics for Computational Social Systems (ICCSS), Jinzhou, 2016, pp. 363-367, doi: 10.1109/ICCSS.2016.7586482.

[7] H. Qassim, A. Verma and D. Feinzimer, "Compressed residual-VGG16 CNN model for big data places image recognition," 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, 2018, pp. 169-175, doi: 10.1109/CCWC.2018.8301729.

[8] N. Jmour, S. Zayen and A. Abdelkrim, "Convolutional neural networks for image classification," 2018 International Conference on Advanced Systems and Electric Technologies (ICASET), Hammamet, 2018, pp. 397-402, doi: 10.1109/ASET.2018.8379889.

[9] E. M. Dogo, O. J. Afolabi, N. I. Nwulu, B. Twala and C. O. Aigbavboa, "A Comparative Analysis of Gradient Descent-Based Optimization Algorithms on Convolutional Neural Networks," 2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS), Belgaum, India, 2018, pp. 92-99, doi: 10.1109/CTEMS.2018.8769211.

[10] Sushma L , Dr. K. P. Lakshmi, 2020, An Analysis of Convolution Neural Network for Image Classification using Different Models, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH TECHNOLOGY (IJERT) Volume 09, Issue 10 (October 2020).

[11] S. Shalev-Shwartz & S. Ben-David, *Understanding machine learning: from theory to algorithms*. Cambridge: Cambridge University Press, 2019, ch. 20, pp. 28-284.

[12] S. Shalev-Shwartz & S. Ben-David, *Understanding machine learning: from theory to algorithms*. Cambridge: Cambridge University Press, 2019, ch. 14, pp. 184-201.

[13] C. M. Bishop & C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY: Springer Science + Business Media, LLC, 2006, ch. 5, sec. 5.5.6, pp. 267-289

[14] Z. Zhang, "Improved Adam Optimizer for Deep Neural Networks," 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), Banff, AB, Canada, 2018, pp. 1-2, doi: 10.1109/IWQoS.2018.8624183.

[15] V. Fung, “An Overview of ResNet and its Variants,” Medium, 17-Jul-2017. [Online]. Available: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035. [Accessed: 07-Dec-2020].

[16] “torch.optim,” torch.optim - PyTorch 1.7.0 documentation. [Online]. Available: https://pytorch.org/docs/stable/optim.html. [Accessed: 07-Dec-2020].

8 APPENDIX A

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 64, 128]             640
       BatchNorm2d-2          [-1, 64, 64, 128]             128
              ReLU-3          [-1, 64, 64, 128]               0
            Conv2d-4          [-1, 64, 64, 128]          36,928
       BatchNorm2d-5          [-1, 64, 64, 128]             128
              ReLU-6          [-1, 64, 64, 128]               0
         MaxPool2d-7           [-1, 64, 32, 64]               0
            Conv2d-8          [-1, 128, 32, 64]          73,856
       BatchNorm2d-9          [-1, 128, 32, 64]             256
             ReLU-10          [-1, 128, 32, 64]               0
           Conv2d-11          [-1, 128, 32, 64]         147,584
      BatchNorm2d-12          [-1, 128, 32, 64]             256
             ReLU-13          [-1, 128, 32, 64]               0
        MaxPool2d-14          [-1, 128, 16, 32]               0
           Conv2d-15          [-1, 256, 16, 32]         295,168
      BatchNorm2d-16          [-1, 256, 16, 32]             512
             ReLU-17          [-1, 256, 16, 32]               0
           Conv2d-18          [-1, 256, 16, 32]         590,080
      BatchNorm2d-19          [-1, 256, 16, 32]             512
             ReLU-20          [-1, 256, 16, 32]               0
           Conv2d-21          [-1, 256, 16, 32]         590,080
      BatchNorm2d-22          [-1, 256, 16, 32]             512
             ReLU-23          [-1, 256, 16, 32]               0
        MaxPool2d-24           [-1, 256, 8, 16]               0
           Conv2d-25           [-1, 512, 8, 16]       1,180,160
      BatchNorm2d-26           [-1, 512, 8, 16]           1,024
             ReLU-27           [-1, 512, 8, 16]               0
           Conv2d-28           [-1, 512, 8, 16]       2,359,808
      BatchNorm2d-29           [-1, 512, 8, 16]           1,024
             ReLU-30           [-1, 512, 8, 16]               0
           Conv2d-31           [-1, 512, 8, 16]       2,359,808
      BatchNorm2d-32           [-1, 512, 8, 16]           1,024
             ReLU-33           [-1, 512, 8, 16]               0
        MaxPool2d-34            [-1, 512, 4, 8]               0
           Conv2d-35            [-1, 512, 4, 8]       2,359,808
      BatchNorm2d-36            [-1, 512, 4, 8]           1,024
             ReLU-37            [-1, 512, 4, 8]               0
           Conv2d-38            [-1, 512, 4, 8]       2,359,808
      BatchNorm2d-39            [-1, 512, 4, 8]           1,024
             ReLU-40            [-1, 512, 4, 8]               0
           Conv2d-41            [-1, 512, 4, 8]       2,359,808
      BatchNorm2d-42            [-1, 512, 4, 8]           1,024
             ReLU-43            [-1, 512, 4, 8]               0
        MaxPool2d-44            [-1, 512, 2, 4]               0
           Linear-45                 [-1, 4096]      16,781,312
      BatchNorm1d-46                 [-1, 4096]           8,192
             ReLU-47                 [-1, 4096]               0
           Linear-48                 [-1, 4096]      16,781,312
      BatchNorm1d-49                 [-1, 4096]           8,192
             ReLU-50                 [-1, 4096]               0
           Linear-51                    [-1, 9]          36,873
================================================================
Total params: 48,337,865
Trainable params: 48,337,865
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.03
Forward/backward pass size (MB): 52.72
Params size (MB): 184.39
Estimated Total Size (MB): 237.14
```

Figure 10: Parameters for VGG-16