Md. Mahfuzur Rahman Mahim

ML intern task-2

Contents

Introduction

I undertook a challenging task that required me to leverage my knowledge of Convolutional Neural Networks (CNNs). To achieve this, I delved deep into understanding the basics of various CNN models and their architectures. I applied this knowledge to the Apparel Dataset[1] from Kaggle[8], which has 8 different apparel categories and 9 unique colors. I also learned to develop my own deep neural network and applied it to the dataset. In my research, I studied VGG net[16], Residual Network (ResNet)[6], Inception Network (inceptionNet)[17], Mobile Net[7], and Efficient Net[18]. I used Keras[19] library to apply VGG16 and Pytorch[14] library to apply ResNet and InceptionNet. Additionally, I built my own model using both Keras and Pytorch. My multilabel classification models achieved accuracy of 66%, 80.11%, 80.519% respectively. Where the best accuracy being achieved using InceptionNet, followed by ResNet and my own models. To further my knowledge, I relied heavily on the courses of Andrew Ng and referred to Pytorch and Keras documentation for coding implementations. I also completed Andrew Ng's Convolutional Neural Network course from his Deep Learning Specialization[11]. Throughout this task, I made the most of the free computation power offered by Colab[4] and Kaggle. However, I encountered some challenges with Colab's limited computational points and therefore shifted to using Kaggle, which offers 30 hours/week of more powerful GPU usage.

Background Studies

0.1 Convolutional Neural Network

Convolutional Neural Networks (CNNs)[12][11] are powerful artificial neural networks designed to process data with a grid-like topology, such as an image. Their unique architecture makes CNNs exceptional at image and video processing tasks. They use small local receptive fields, or filters, that convolve across the image, allowing the network to focus on small regions at a time, detecting local features like edges or textures. Each filter is replicated across the entire visual field, enabling the network to detect the same feature anywhere in the image. This principle of shared weights leads to a dramatic reduction in the number of parameters, making training more efficient. CNNs are not limited to just classification and detection tasks. They can also be used for semantic segmentation, instance segmentation, image synthesis and generation, image super-resolution, style transfer, video analysis, and 3D object recognition. Thus, CNNs are a key tool in many areas of research and application involving spatial data. Below is the equation for the convolution operation:

$$\textbf{Conv} = \left(\frac{n + 2p - f}{s}\right) + 1$$

0.2 VGG

The VGG-16[16][11] network is a convolutional neural network known for its simplicity and effectiveness in image classification. It uses a straightforward architecture with repeated blocks of convolutional layers followed by max-pooling layers, all using three-by-three filters and same padding. The network architecture is uniform, with the number of filters roughly doubling in each block of convolutional layers. Despite being large, with 138 million parameters, its

simplicity and systematic design make it attractive for researchers. It achieves impressive results in image classification tasks, almost comparable to larger networks like VGG-19, making it a popular choice in practice.

0.3 Residual Networks (ResNet)

Residual Networks (ResNets)[6][11], help solve the problem of training very deep neural networks. The main issue with deep networks is the vanishing and exploding gradient problem. ResNets use skip connections, also known as shortcuts, to pass information from one layer directly to a much deeper layer. This allows the network to avoid the vanishing gradient problem and train effectively even with over a hundred layers. Without ResNets, as networks get deeper, training error tends to increase, but with ResNets, training error can continue to decrease even with very deep networks. Overall, ResNets have revolutionized deep learning by enabling the training of much deeper networks without significant loss in performance.

0.4 Inception Network

The Inception network[17][11] introduces the idea of using multiple filter sizes and pooling layers within a single layer of a Convolutional Network (ConvNet), allowing the network to capture different features simultaneously. Instead of choosing one filter size or pooling strategy, it uses 1x1, 3x3, and 5x5 convolutions, as well as pooling, and concatenates the outputs. This approach increases complexity but improves performance. However, it comes with high computational costs. To address this, a bottleneck layer with 1x1 convolutions is added to reduce computation while maintaining performance. This bottleneck layer reduces the dimensionality before applying larger convolutions, significantly reducing computational costs. This approach is used in the Inception network, which has been successful in various tasks like image recognition.

0.5 MobileNet

MobileNets[7][11] are a type of convolutional neural network that uses depthwise separable convolution, a special type of convolution that consists of two steps: depthwise convolution and pointwise convolution. This approach reduces computational costs significantly and makes MobileNets suitable for low-power devices like mobile phones. The computational cost is about

one-ninth of a normal convolution in typical scenarios. The document provides examples to explain the computational steps involved in both normal convolution and depthwise separable convolution. Additionally, it clarifies how depthwise separable convolution can handle different numbers of input channels and simplify the visualization of convolutional operations for ease of understanding.

0.6 Data Augmentation

In cases where the available data is insufficient to train a machine learning model, data augmentation[11] techniques can be employed to artificially expand the dataset. A widely used method involves manipulating existing data through rotation, shearing, and resizing to produce fresh variations of the original images. These variations may include slightly tilted or distorted versions of the original images, as well as those that have been magnified or reduced in size. Incorporating these augmented images into a convolutional neural network (CNN) model can furnish the model with additional features to learn from, thereby enhancing its precision.

Methodology

0.7 Overview

For this project, I utilized CNN models such as VGG16[16], InceptionNet[17], and ResNet[6] to train the Apparel dataset[1] from Kaggle[8]. Additionally, I created two models with different libraries, Keras[19] and Pytorch[14], using the same architecture. While working with pre-trained models like InceptionNet, ResNet, and VGG16, I discovered that only InceptionNet showed a significant difference, performing best on images sized (299,299). However, the other two models worked well with any image size. This was corroborated by Christian Szegedy et al.'s[17] InceptionNet paper, which noted that this model works optimally on (299,299) pixel images. Overall, this project allowed me to gain insight into how CNN models are constructed and how they function in computer vision.

0.8 Dataset

Kais[9] created this dataset with the intention of practicing multi-label classification, inspired by Jeremy Howard's FastAi lecture 3[10]. The dataset comprises 8 different clothing categories in 9 distinct colors. The primary aim of multi-label classification is to label items present in pictures based on these categories. The dataset consists of 16,170 images that were scraped from Google, Bing, and DuckDuckGo, encompassing a total of 37 categories. The creator compiled this dataset by gathering images from various sources on the internet, including trolukovich's dataset[2] and Adrian's simple and small dataset containing 3 clothing categories, which he used in his article on pyimagesearch's[15] multi-label classification with keras. To assemble the dataset, Kais utilized cwerner's fastclass package[3].

0.9 Model Description

I created a model called incomplete-CNN-Model, but due to limited GPU time on Colab[4], I was unable to fully converge the model. However, before that, I utilized VGG16 to train the apparel dataset to gain insight into how models function on datasets and to learn how to utilize pretrained models on datasets. VGG16 yielded a validation accuracy of 75.659% and training accuracy of 75.66% after 10 epochs. In contrast, my incomplete-CNN-Model only achieved an accuracy of 11.44% after 12 epochs. Due to the GPU constraints on Colab, I switched to using Kaggle. I utilized ResNet and inceptionNet as pretrained models to train my dataset. After implementing ResNet for 5 epochs, it yielded a validation accuracy of 82.3%.

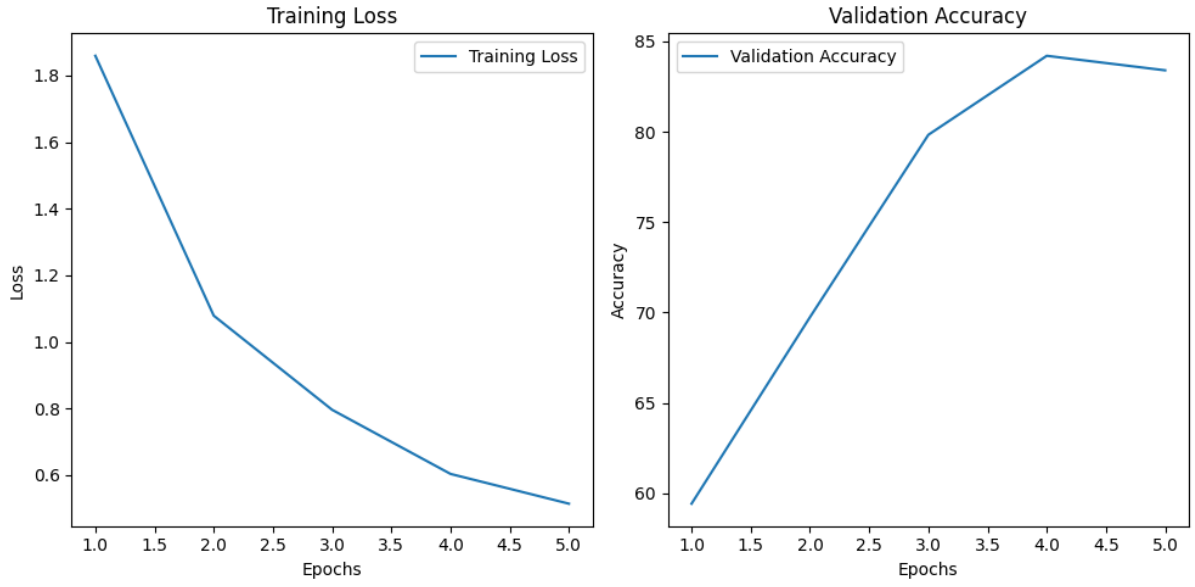

Figure 1: ResNet performance graph.

I utilized Pytorch library to implement both ResNet and inceptionNet. Upon implementing inceptionNet, I achieved an accuracy of 92.8% after a total of 10 epochs. I had previously run 5 epochs and obtained an accuracy of 86.26%, saved the model, and then re-ran the saved model to obtain the final accuracy. However, while applying inceptionNet, the loss was decreasing and the curve was not fluctuating, but when it came to validation, the results were fluctuating, ranging from 80% to 90%, then suddenly to 82%, before experiencing a sudden rise to 94%.

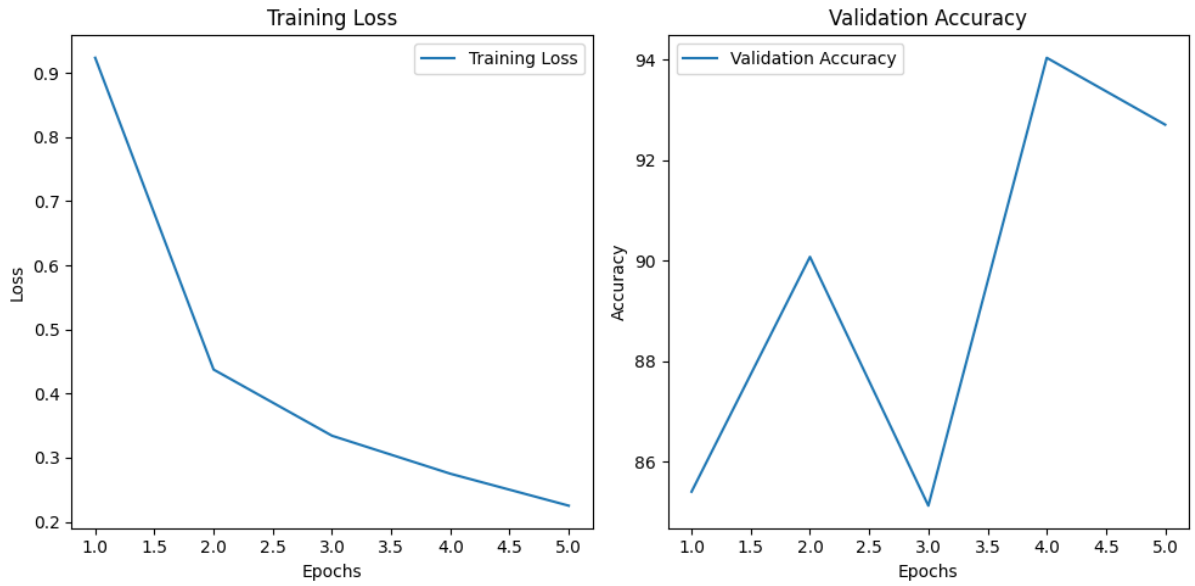

Figure 2: InceptionNet performance graph.

Consequently, my model was overfitting while using inceptionNet. To address this, I developed 3 complete CNN models - two using tensorflow keras library and one using Pytorch. The first completed CNN model, my-cnn-model-kaggle, presented some unnecessary warnings while attempting to use Kaggle GPU service using Keras library. I reinstalled tensorflow with cuda support and also installed HDF5(h5py)[5], PyPI typing extensions[20], and wheel[13] from pip due to my belief that some problems with dependencies compatibility on Kaggle and for storage issues, kaggle wasn't able to work with large data, and wasn't up to date with new python libraries system feature. However, I later discovered that the issue was with Kaggle because none of these import attempts worked. After running my CNN model, my-cnn-model-kaggle, for a total of 45 epochs, I achieved a training accuracy of 81.69% and validation and test accuracies of 66.82%. Though it performed well in training, the model struggled with validation and testing. However, my second complete CNN model, my-cnn-model-from-colab, fared much better in both training and testing with 80.26% and 80.11% accuracies respectively. This was due to the incorporation of BatchNormalization after the convolution layers. In my third CNN model, my-cnn-model-pytorch, which was built using the pytorch library, I achieved a validation accuracy of 80.519%. By implementing batch normalization and increasing the layers of my CNN model, I was able to significantly improve its accuracy.

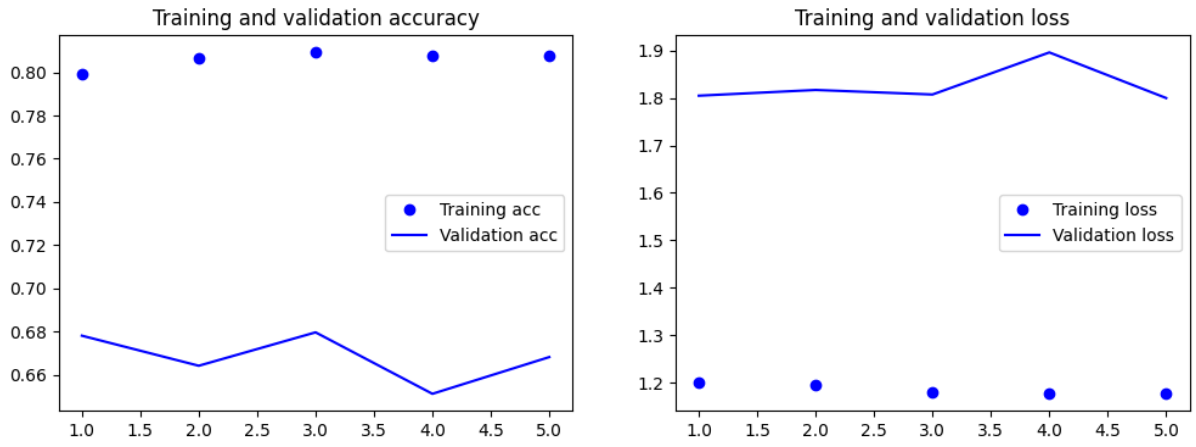

Figure 3: performance graph of my-cnn-model-kaggle.

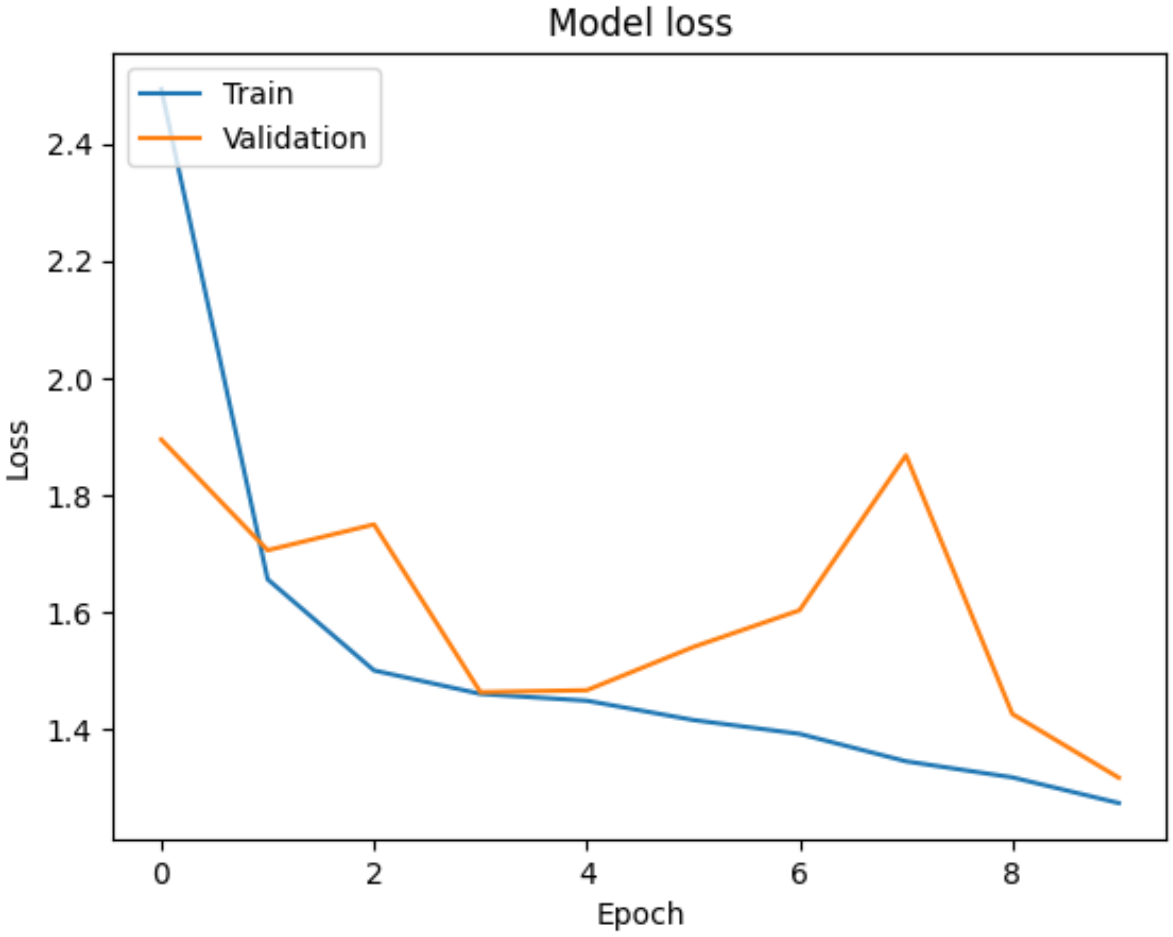

Figure 4: my-cnn-model-from-colab performance graph

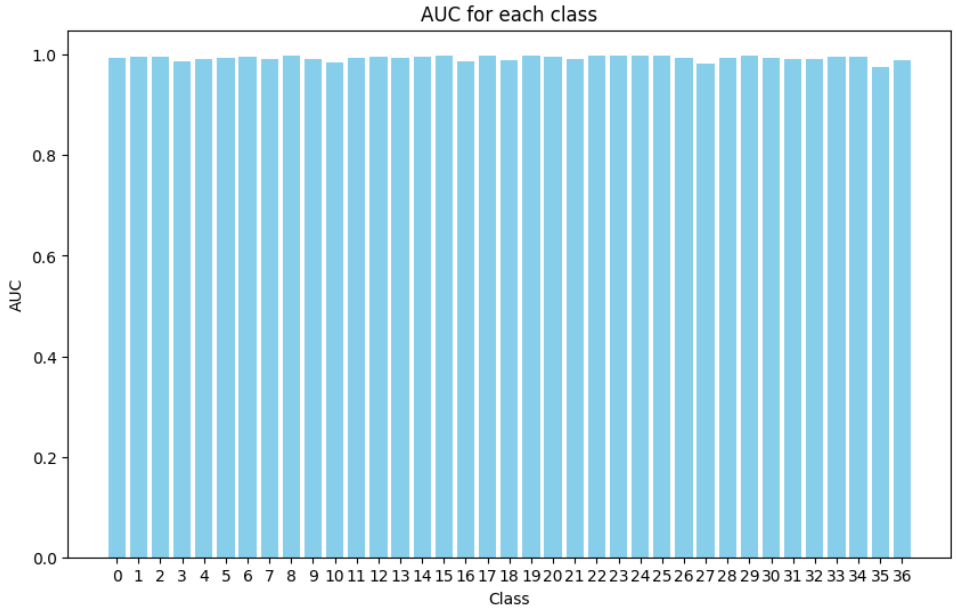

Figure 5: my-cnn-model-from-colab AUC-ROC of each class

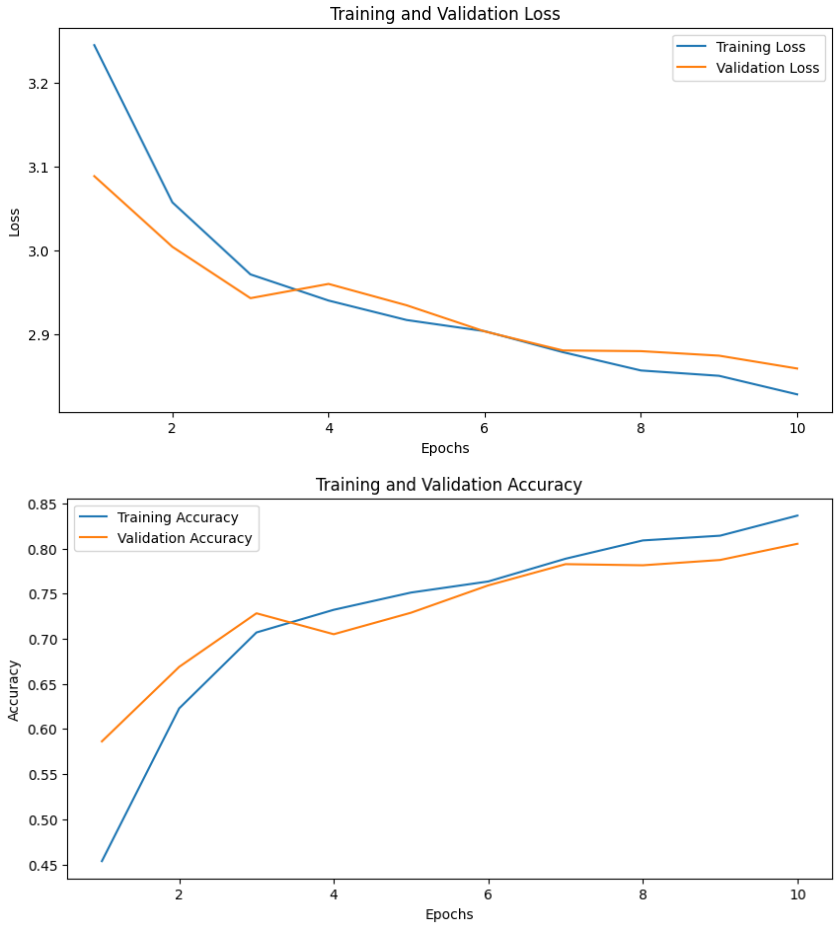

Figure 6: Performance graph of my-cnn-model-pytorch

Experimental Setup

I performed My models on Google Colab[4] and Kaggle[8]. Although I mostly used Kaggle notebooks as it has more GPU computation time than colab which helped me a lot as I was training with image dataset which requires more GPU computational power and more time. In Kaggle I had set up the environment as I was getting less computational power which affected my model performance. Still, I was facing the performance issue and filed a report to Kaggle. Here is the git repository link for the task.,gitlink

Result Analysis

In my experiments, I found that InceptionNet had the highest accuracy at 94%, but it was overfitted. ResNet was the next most accurate model, with a result of 82.3%. Among my own three models, my-cnn-model-pytorch achieved 80.519%, my-cnn-model-from-colab achieved 80.11%, and my-cnn-model-kaggle achieved 66.82%. Upon comparing these results, my-cnn-model-pytorch and my-cnn-model-from-colab were very close. Interestingly, the pretrained VGG16 model achieved better results than my models, with a score of 75.66% after only 10 epochs. The lowest performing model was my-cnn-model-kaggle, with a score of 66.82%. Although this was not a comparative task, I believe that with more fine tuning and experimentation, I could have achieved even better results with VGG16 and gained a deeper understanding of how the models behave. I plan to explore this further in the future.

Models	InceptionNet	ResNet	my-cnn-model-pytorch	my-cnn-model-from-colab	my-cnn-model-kaggle
0	94%	80.519%	80.11%	66.82%	75.66%

Figure 7: Results on different Models

Conclusion

This task was undertaken with the goal of learning about CNN models, which are crucial for computer vision tasks. During the process, I gained knowledge about several CNN model architectures, including VGG (VGG16)[16], ResNet[6], InceptionNet[17], and MobileNet[7]. Each of these models has a unique architecture and varying levels of work efficiency. For instance, MobileNets are preferred for mobile devices due to their lower computational cost compared to ResNet or InceptionNets. Although I also learned about EfficientNet, I need to invest more time to implement it effectively. Overall, these learnings helped me to grow, and I look forward to developing a deeper understanding of these models in the future.

References

[1] *Apparel Dataset — kaggle.com*. https://www.kaggle.com/datasets/kaiska/apparel-dataset. [Accessed 08-04-2024].

[2] *Apparel images dataset — kaggle.com*. https://www.kaggle.com/datasets/trolukovich/apparel-images-dataset. [Accessed 08-04-2024].

[3] *GitHub - cwerner/fastclass: Little tools to download and then weed through images, delete and classify them into groups for building deep learning image datasets (based on crawler and tkinter) — github.com*. https://github.com/cwerner/fastclass. [Accessed 08-04-2024].

[4] Google LLC. *Google Colaboratory*. Accessed on Access date. Google. Year accessed. URL: https://colab.research.google.com/.

[5] *HDF5 for Python 2014; h5py 3.10.0 documentation — docs.h5py.org*. https://docs.h5py.org/en/stable/. [Accessed 08-04-2024].

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[7] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV].

[8] Kaggle Inc. *Kaggle*. Accessed on Access date. Kaggle. Year accessed. URL: https://www.kaggle.com/.

[9] *Kais — Contributor — kaggle.com*. https://www.kaggle.com/kaiska. [Accessed 08-04-2024].

[10] *Lesson 3: Practical Deep Learning for Coders 2022 — youtube.com*. https://www.youtube.com/watch?v=hBBOjCiFcuo. [Accessed 08-04-2024].

[11] Andrew Ng. *Deep Learning Specialization*. Taught by Andrew Ng, Stanford University. Coursera. URL: https://www.coursera.org/specializations/deep-learning.

[12] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE].

[13] *pip wheel - pip documentation v24.0 — pip.pypa.io*. https://pip.pypa.io/en/stable/cli/pip_wheel/. [Accessed 08-04-2024].

[14] *PyTorch — pytorch.org*. https://pytorch.org/. [Accessed 08-04-2024].

[15] Adrian Rosebrock. *Multi-label classification with Keras - PyImageSearch — pyimagesearch.com*. https://pyimagesearch.com/2018/05/07/multi-label-classification-with-keras/. [Accessed 08-04-2024].

[16] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[17] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].

[18] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[19] Keras Team. *Keras: Deep Learning for humans — keras.io*. https://keras.io/. [Accessed 08-04-2024].

[20] *typing-extensions — pypi.org*. https://pypi.org/project/typing-extensions/. [Accessed 08-04-2024].