STYLESCAN

Automated clothing recognition

Authored By: Faisal Saimeh & Ahmad Zalmout

Introduction

This report details the development of a machine learning model for wearable image classification. The model uses a Convolutional Neural Network (CNN) architecture to analyze images and identify different clothing items and wearables. This project explores the effectiveness of preprocessing operations have on the model to accurately classify various clothing Items. This brings us one step closer to building a model that finds matching clothes and produces accurate recommendations.

Objective

It is not an easy task to increase the accuracy of a model. Nor is it straight forward. And one of the important aspects is data preprocessing. Which is in fact the first challenge one encounters when developing a model.

For this matter, we chose the first format to tackle and explore the world of data preprocessing and observe their effectiveness on training a model.

Methodology

Data collection

CNN is a neural network algorithm, implying that it is a very complex model. And it requires special handling in preprocessing and tuning. But before it comes to that, a large amount of data is needed for the following reasons:

1. High Parameter Count:

   CNNs have a large number of parameters compared to simpler machine learning models. These parameters represent the connections and weights within the network that are adjusted during training. With a larger dataset, the network has more opportunities to learn the optimal values for these parameters, leading to better performance.

2. Learning Complex Features:

   CNNs are particularly skilled at recognizing patterns in images. These patterns can be low-level features like edges and lines, or higher-level features like shapes and objects. A large dataset exposes the CNN to a wider variety of these features, allowing it to learn robust representations that generalize well to unseen data.

Consequently, complicating the task of finding a sufficient size of a dataset. Especially when most of the image datasets stores each image in a separate folder. This makes it difficult for python to read the images all at once and cumbersome for us to collect them in one folder

especially when we are dealing with thousands of images and thus thousands of folders and subfolders. Other datasets had images with no labels, impeding data collection task even further as we were trying to build a supervised classification machine learning model.

For this reason, more than one source of data had to be collected and combined.

Data was collected from the following sources:

- https://www.kaggle.com/agrigorev/clothing-dataset-full
- https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-small
- https://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html

For a total of 71236 images labeled with 174 labels were combined to train our model.

Data preprocessing

The data was combined in one place to be read all at once. However, it was noted that it would be easier on the devices used to read each collection of images alone and then combine them in a dataframe. Afte that, it was linked to the CSV files that contains each image's label.

Data cleaning

When combined, there were 174 labels. Upon inspection, it was discovered that there are redundant and misleading classes in the data across different files. For further information refer to the chart below displaying the raw labels.

They were modified, renamed, and similar categories were combined, and the final result is displayed below:

And then, the labels were encoded using LabelEncoder.

Data Transformation

After settling down on which labels to keep and to combine, image preprocessing techniques came into play.

3 types of image preprocessing methods were examined: resizing, augmentation, and normalization.

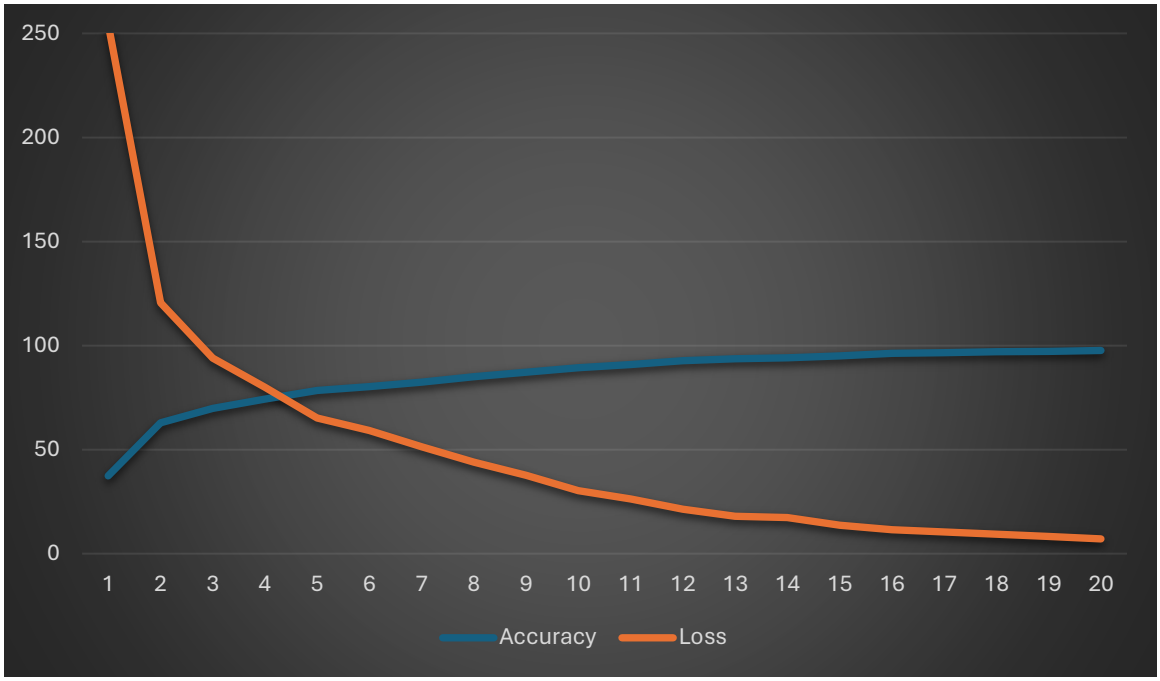
Here's a quick explanation:

1. Resizing: This ensures all images have a uniform size. Resizing forces all images into the same size grid, making them compatible with the CNN's input layer.
2. Augmentation: This artificially expands the dataset by creating variations of existing images. Similar to creating photocopies and then modifying them slightly (flipping, rotating, color adjustments). This helps the CNN learn from a wider range of variations and improve its ability to generalize to unseen data.
3. Normalization: This scales the pixel intensity values of your images to a common range (often 0 to 1 or -1 to 1). Imagine the brightness of each pixel goes from 0 (black) to 255 (white). Normalization puts all images on the same "brightness scale" so the CNN focuses on learning the patterns in the images, not just their relative brightness.

Resizing

At first, images were resized to a pixel size of (100, 100) and were sent to the model. It returned not bad results.

Below is a chart of the accuracy and loss across the epochs:



As shown, the accuracy increases with increasing epochs getting very close to a 100 and the loss started very high and then dropped near 0.

However for a better evaluation, classification specific metrics were measured as mentioned below:

Precision: 0.7416741992591599

Recall: 0.6503892427459307

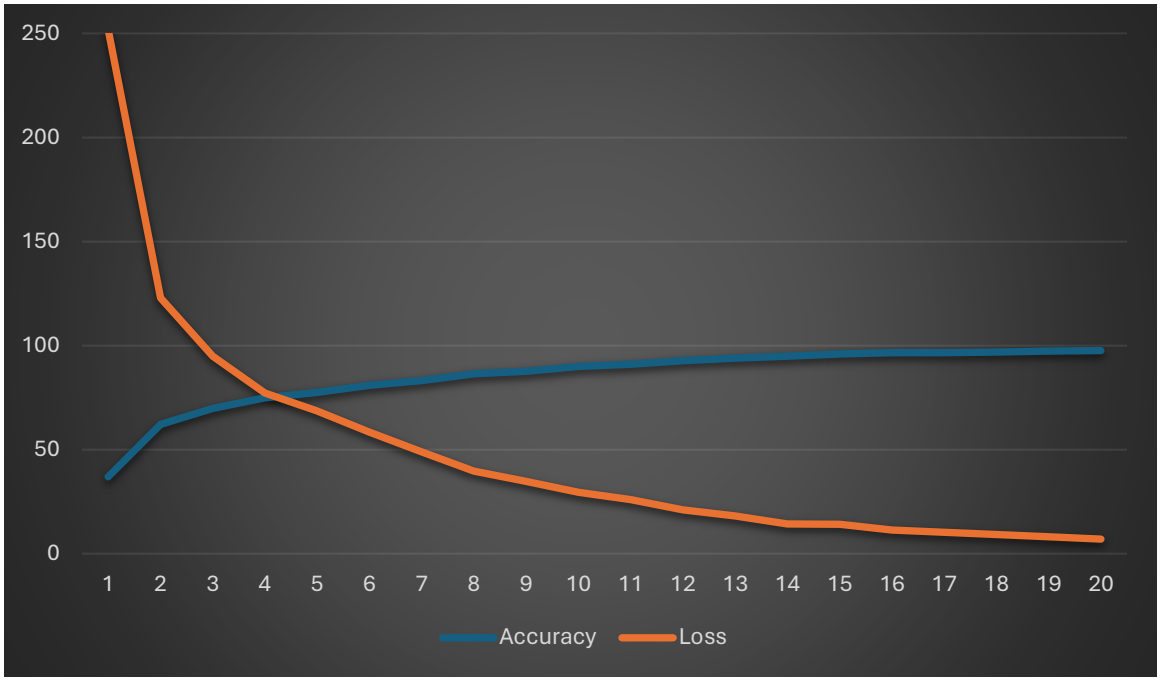
F1-Score: 0.6502664624433696

Which translates to a relatively weak model.

Resizing with normalization

Second, images were resized to (100, 100) **and** z-scored normalized (Standardization). This type of normalization subtracts the mean of all pixel values in the image from each pixel value and then divides the result by the standard deviation of all pixel values in the image.

Below is a chart of the accuracy and loss across the epochs:



As shown, the accuracy increases with increasing epochs getting very close to a 100 and the loss started very high and then dropped near 0.

However for a better evaluation, classification specific metrics were measured as mentioned below:

Precision: 0.8026751676657021

Recall: 0.7968860580325549

F1-Score: 0.7947280433794707

This model is a significant improvement from the one before.

Resizing, Augmentation, and Normalization

Finaly, all methods were used together.

The best arrangement for a CNN model involving Resizing, Augmentation, and Normalization is:

1. Resizing

2. Augmentation

3. Normalization

Here's why this order is preferred:

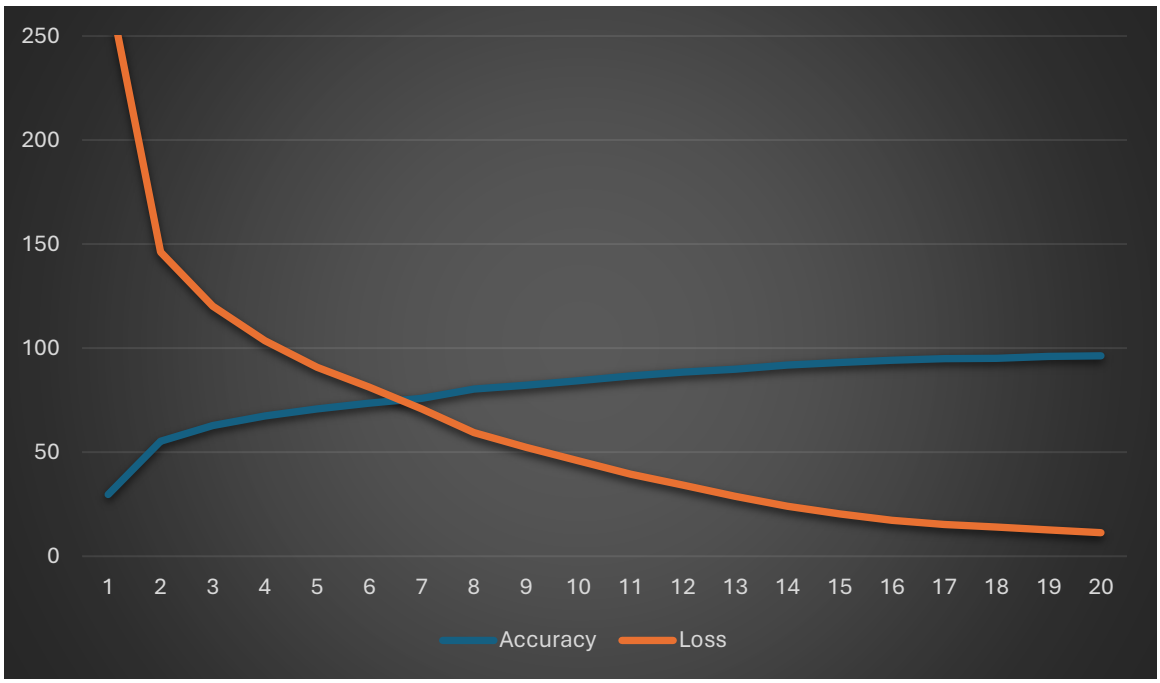
- Resizing: This ensures all images have a consistent size, making them compatible with the CNN's input layer. It's crucial to perform this first to ensure any subsequent operations work on images of the expected size.
- Augmentation: Augmentation creates variations of existing images. It is used after resizing to inject diversity into the dataset.

- Normalization: This scales pixel values to a common range. Performing normalization after any potential resizing and augmentation ensures that the final pixel values used by the CNN are within the expected range for optimal training and performance.

Benefits of this order:

- Efficiency: Resizing first avoids unnecessary computations on larger images before they are scaled down.
- Consistency: Normalization on the final image size ensures consistent scaling across all data points used for training.

Below is a chart of the accuracy and loss across the epochs:



As shown, the accuracy increases with increasing epochs getting very close to a 100 and the loss started very high and then dropped near 0.

However for a better evaluation, classification specific metrics were measured as mentioned below:

Precision: 0.7486868073202483

Recall: 0.7544232130219392

F1-Score: 0.7496339563449737

This model is better than the first model, however, inferior to the second. It could be interpreted that the model became worse, or the second model was a bit overfit, and this model gave a more accurate result.

Model Architecture

As the objective of the project is to test different preprocessing techniques of the model, the parameters of the model were not tested heavily. However, to produce reasonably good results more than one set of parameters were examined and then it was settled on one set.

The model had the following parameters across all tests:

- Convolutional neural network was used with Adam optimizer.
- 4 neural network layers were used.
  - First: 16 filters, kernel size of 5
  - Second: 32 filers, kernel size of 5
  - Third: 64 filters, kernel size of 5
  - Forth: 128 filters, kernel size of 5
- A learning rate of 0.0001 was used.

Conclusion

All models were good as the f1 score did not drop below 60%. On the other hand, it is advised to use different data preprocessing methods as they make the model more robust and reliable.

Below is a table comparing the models we trained:

Evaluation metric	Resizing	Resizing + Normalization	Resizing + Augmentation + Normalization
Precision	0.7416741992591599	0.8026751676657021	0.7486868073202483
Recall	0.6503892427459307	0.7968860580325549	0.7544232130219392
F1-score	0.6502664624433696	0.7947280433794707	0.7496339563449737

At first glance, the 2nd model appears to be the best. In the same time, the 3rd in our opinion is more robust as it eliminates overfitting by implementing Augmentation while maintaining a pretty high f1 score.