EGCI 463 Pattern Recognition

Final Project Report

Group Members

Pitchapa	Phisutpichet	6580065
Natnicha	Sukchuenanant	6580812
Chanon	Pluemhathaikij	6581103
Supakorn	Panyadee	6581117

Mahidol University International College

Asst. Prof. Dr. Mingmanas Sivaraksa

30 November 2025

Table of Content

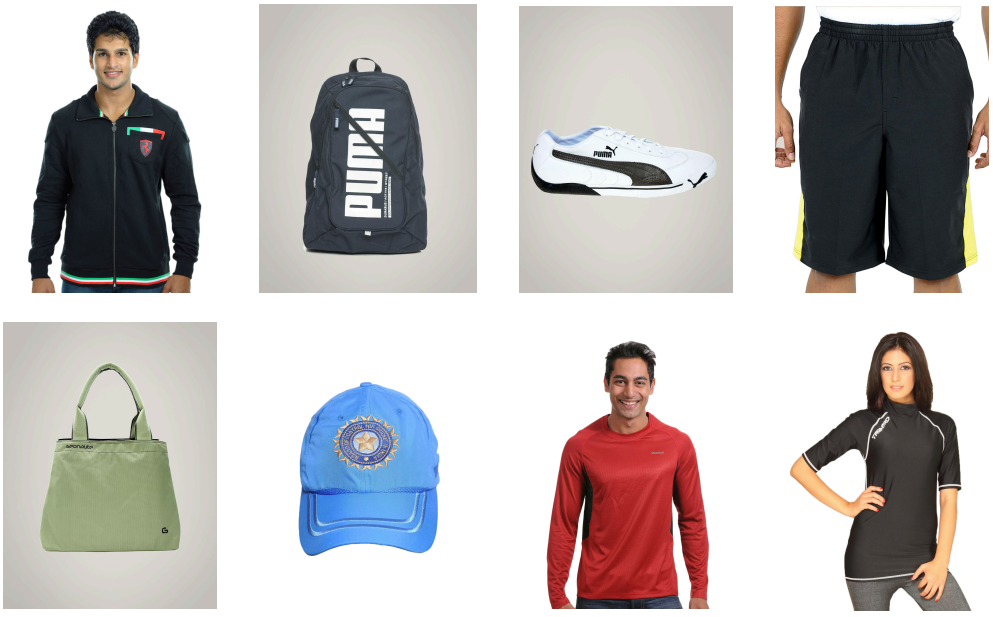
Project Background

Our data was collected from the Fashion Product Images Dataset. The dataset is about the clothings and accessories with more than 44,000 pictures and much information on each of the clothes and accessories.

Dataframe of the Fashion Product Images Dataset.

	id	gender	masterCategory	subCategory	articleType	baseColour	season	year	usage	productDisplayName
0	15970	Men	Apparel	Topwear	Shirts	Navy Blue	Fall	2011.0	Casual	Turtle Check Men Navy Blue Shirt
1	39386	Men	Apparel	Bottomwear	Jeans	Blue	Summer	2012.0	Casual	Peter England Men Party Blue Jeans
2	59263	Women	Accessories	Watches	Watches	Silver	Winter	2016.0	Casual	Titan Women Silver Watch
3	21379	Men	Apparel	Bottomwear	Track Pants	Black	Fall	2011.0	Casual	Manchester United Men Solid Black Track Pants
4	53759	Men	Apparel	Topwear	Tshirts	Grey	Summer	2012.0	Casual	Puma Men Grey T-shirt
5	1855	Men	Apparel	Topwear	Tshirts	Grey	Summer	2011.0	Casual	Inkfruit Mens Chain Reaction T-shirt
6	30805	Men	Apparel	Topwear	Shirts	Green	Summer	2012.0	Ethnic	Fabindia Men Striped Green Shirt
7	26960	Women	Apparel	Topwear	Shirts	Purple	Summer	2012.0	Casual	Jealous 21 Women Purple Shirt
8	29114	Men	Accessories	Socks	Socks	Navy Blue	Summer	2012.0	Casual	Puma Men Pack of 3 Socks
9	30039	Men	Accessories	Watches	Watches	Black	Winter	2016.0	Casual	Skagen Men Black Watch

We decided to choose a subCategory column as output for our model due to its number being in a good range and our input will be the picture of the clothes and accessories. The following pictures are examples of the pictures inside the dataset.



Examples of image from dataset

Data Preparation

```
print(f"Number of class in masterCategory: {df['masterCategory'].nunique()}")
print(f"Number of class in subCategory: {df['subCategory'].nunique()}")
print(f"Number of class in articleType: {df['articleType'].nunique()}")
print(f"Number of class in baseColour: {df['baseColour'].nunique()}")
print(f"Number of class in usage: {df['usage'].nunique()}")
```

0.0s

```
Number of class in masterCategory: 7
Number of class in subCategory: 45
Number of class in articleType: 143
Number of class in baseColour: 46
Number of class in usage: 8
```

From the extraction of the dataset's dataframe, we found masterCategory and subCategory to be suitable to be the model's output. However, the masterCategory classes are too general. We focus our model's goal to classify the specific type of clothes and accessories in certain specific categories such as topwear, bottomwear, and watches.

The comparison between masterCategory's classes and subCategory's classes

```
df['subCategory'].value_counts()
```

[2] 0.0s

subCategory	
Topwear	15402
Shoes	7343
Bags	3055
Bottomwear	2694
Watches	2542
Innerwear	1808
Jewellery	1079
Eyewear	1073
Fragrance	1011
Sandal	963
Wallets	933
Flip Flops	913
Belts	811
Socks	698
Lips	527
Dress	478
Loungewear and Nightwear	470
Saree	427
Nails	329
Makeup	307
Headwear	293
Ties	258
Accessories	129
Scarves	118

```
df['masterCategory'].value_counts()
```

[8] 0.0s

masterCategory	
Apparel	21397
Accessories	11274
Footwear	9219
Personal Care	2403
Free Items	105
Sporting Goods	25
Home	1

Name: count, dtype: int64

```
target_classes = ['Topwear','Shoes','Bottomwear', 'Bags','Watches']
index_map = {
    'Topwear' : 0,
    'Shoes' : 1,
    'Bottomwear': 2,
    'Bags' : 3,
    'Watches' : 4
}
df_filtered = df[df['subCategory'].isin(target_classes)]

df_filtered = df_filtered.groupby('subCategory').sample(n=1000, random_state=101)
df_filtered['subCategory'].value_counts()
```

0.0s

Afterwards we decided to use subCategory as the output. We will use the top 5 biggest classes in the subCategory column, which are topwear, shoes, bottomwear, bags, watches. Each

of the classes we sampled only 1,000 data equally to ensure that it will not create unbalanced classes problems.

```python
from pathlib import Path
from PIL import Image
from sklearn.preprocessing import FunctionTransformer

def flatten_images(X):
    X = np.asarray(X)
    n_samples = X.shape[0]
    return X.reshape(n_samples, -1)

flatten = FunctionTransformer(flatten_images, validate=False)

image_dir = Path('dataset/fashion-dataset/images')
image_size = (128,128)

def imgExist(img_id):
    full_path = image_dir / f"{img_id}.jpg"
    return os.path.exists(full_path)

def load_gray_img(img_id, size = image_size):
    full_path = image_dir / f"{img_id}.jpg"
    with Image.open(full_path) as img:
        img = img.convert('L') #greyScale
        if image_size is not None:
            img = img.resize(image_size)
        arr = np.array(img)
        arr = arr.reshape(-1)
        return arr
```

0.0s

Next, we created a function called load_grey_img() to load the picture from each data and change the size of image to the image_size that we declared and lastly change the picture into greyscale by using Image library.

We created a function imgExist() to check if the id image in the datasheet exists in the images dataset folder or not.

```python
mask = df_filtered['id'].apply(imgExist)
df_clean = df_filtered[mask].copy()

X = np.stack([load_gray_img(img_id, size=image_size) for img_id in df_clean['id'].values])
y = df_clean['subCategory'].map(index_map).to_numpy()
```

3m 18.4s

By using sklearn library, We split the train dataset and test dataset into proportions of 80:20 and keep the classes ratio during the split. Lastly, we use random_state = 101.

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2, stratify=y, random_state=101)
```

0.1s

Methods

1. Traditional

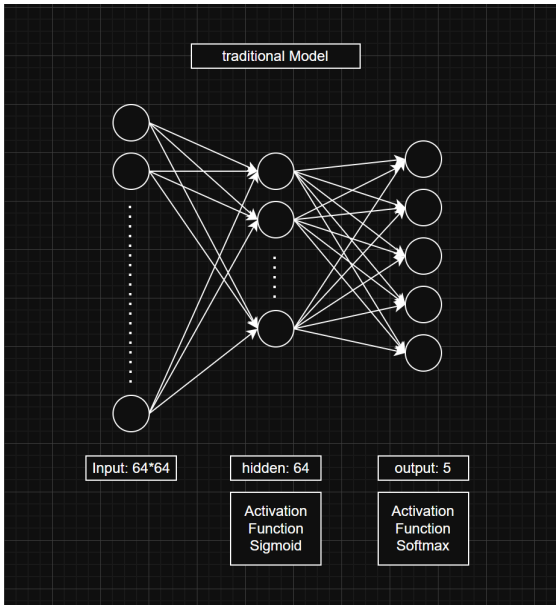
We create a class named nn to work as the neural network of 1 input layer, 1 hidden layer, 1 output layer. We wanted to create a neural network without using any library from sklearn or tensorflow.keras . We decided to use hyperparameters as below. For the sake of comparison, we try different Input layer sizes and Hidden layer sizes.

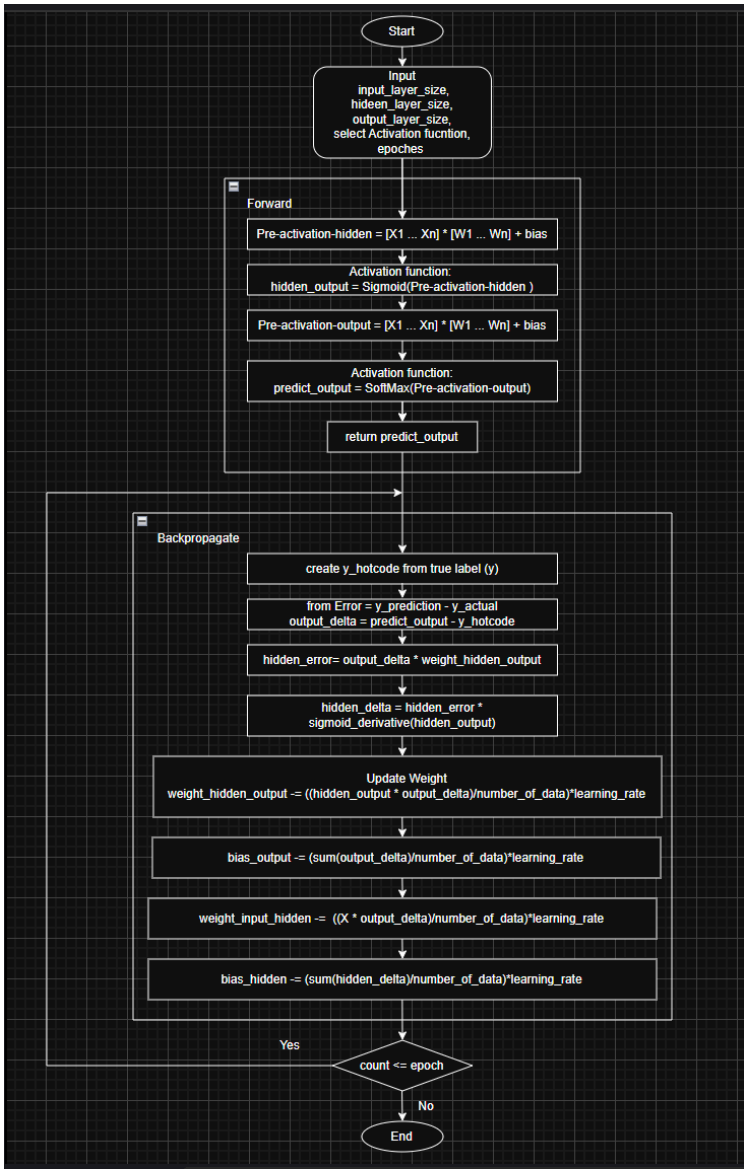
Input layer size: [28*28, 32*32, 64*64, 128*128]

Hidden layer size: [32, 64]

Output layer size : [5]

Activation function: Sigmoid function





2. PCA and Perceptron

We employed a traditional machine learning approach combining Principal Component Analysis (PCA) for feature extraction and Perceptron for classification.

- **Grid Search for Hyperparameter Tuning**

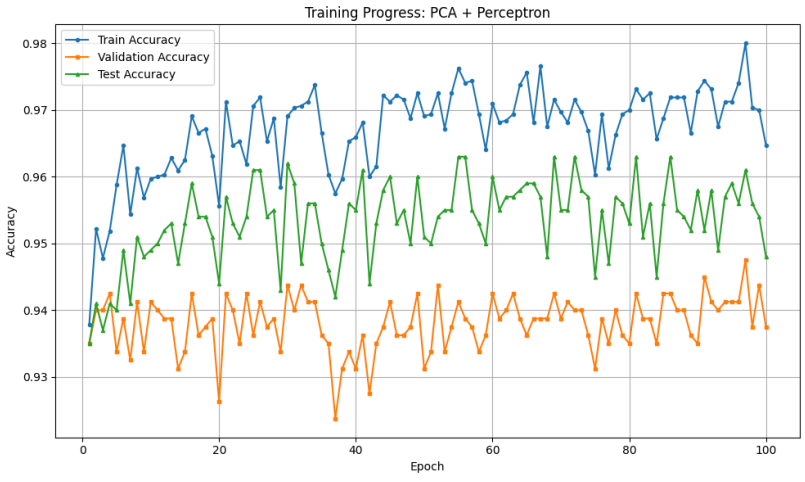
We implemented Grid Search to systematically optimize hyperparameters. The parameter space included: PCA components [50, 100, 150, 200], max iterations [500, 1000, 1500], learning rates [0.001, 0.01, 0.1], and tolerance [1e-3, 1e-4], totaling 72 combinations. Grid Search identified optimal configuration: **150 PCA components**, learning rate 0.001, 500 max iterations, and tolerance 0.001, achieving 95.13% validation accuracy.

- **PCA Feature Extraction**

Each 64×64 grayscale image contains 4096 features. PCA transforms the original feature space into principal components ordered by variance. Using **150 components**, we reduced dimensionality from 4096 to 150 features while retaining **90.75%** of variance. This addresses the curse of dimensionality, accelerates training, and reduces noise.

- **Perceptron Classifier**

Perceptron is a linear classifier that learns optimal weights through iterative updates. We chose it for simplicity, interpretability, and efficiency. Our implementation uses a one-vs-all strategy for multi-class classification across five fashion categories, trained on PCA-transformed features with Grid Search-optimized hyperparameters.



3. Multilayer Perceptron (MLP)

For the deep learning model, we implemented a Multilayer Perceptron (MLP) to classify images into the five selected subcategories: *Bags, Bottomwear, Shoes, Topwear,* and *Watches*. Since MLPs require one-dimensional inputs, the preprocessed images (28×28 grayscale) were flattened into 784-dimensional feature vectors prior to training.

- **Model Architecture**

The MLP architecture consists of two fully connected hidden layers designed to learn non-linear relationships in the input data:

- Hidden Layer 1: 256 neurons with ReLU activation
- Hidden Layer 2: 128 neurons with ReLU activation
- Dropout (0.3) applied after each hidden layer to mitigate overfitting
- Output Layer: 5 units with softmax activation for multi-class classification

This architecture is computationally efficient while still expressive enough to capture meaningful patterns from flattened images.

- **Hyperparameter Optimization**

To obtain the best-performing configuration, we applied GridSearchCV with Stratified 3-Fold Cross Validation.

This approach ensures:

- Balanced class distribution across folds
- Systematic exploration of hyperparameter combinations
- Reliable model selection using validation accuracy

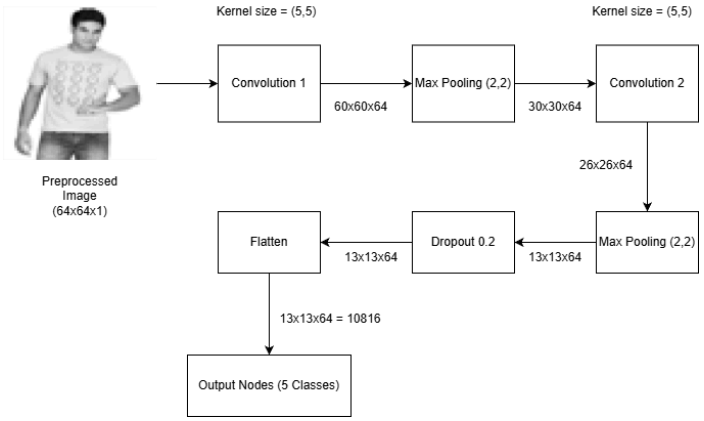
The search space included the number of neurons in both hidden layers, dropout rates, batch sizes, and the number of training epochs.

- **Training Procedure**

After selecting the optimal hyperparameters, the model was retrained on the combined training and validation datasets to obtain stable learning curves and prepare for final evaluation. The training process focused on monitoring validation behavior to ensure the model generalized well rather than memorizing training data.

4. Convolutional Neural Network

For the Convolutional Neural Network, we create a model to predict 5 different classes. Here is the model configuration that we are going to use as a baseline model for Convolutional Neural Network.



1st layer: Convolution with 64 filters of size (5,5), Activation function = ReLU

2nd layer: Max Pooling with pool size of (2,2)

3rd layer: Convolution with 64 filter of size (5,5), Activation function = ReLU

4th layer: Max Pooling with pool size of (2,2)

5th layer: Dropout 0.2

6th layer: Flatten

7th layer: Dense with 5 output (correspond to the number of class), Activation = softmax

Results

1. Traditional models result

Compare the difference between each model when changing the input image size

Model 1 : size of image = (28*28),epochs =3000, hidden layers = 64, learning rate = 0.1

Model 2 : size of image = (32*32), epochs =3000, hidden layers = 64, learning rate = 0.1

Model 3 : size of image = (64*64), epochs = 3000, hidden layers = 64, learning rate = 0.1

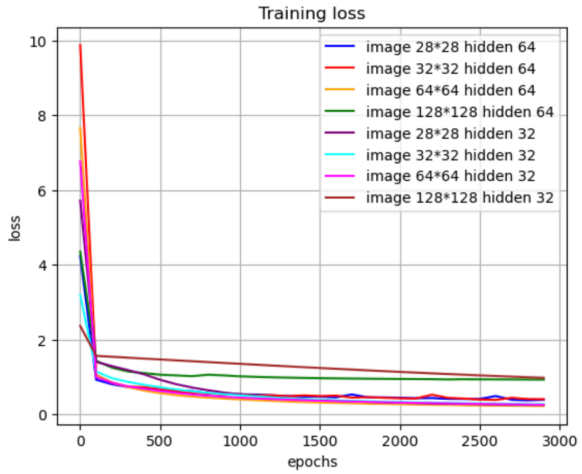
Model 4 :size of image = (128*128), epochs = 3000, hidden layers = 64, learning rate = 0.1

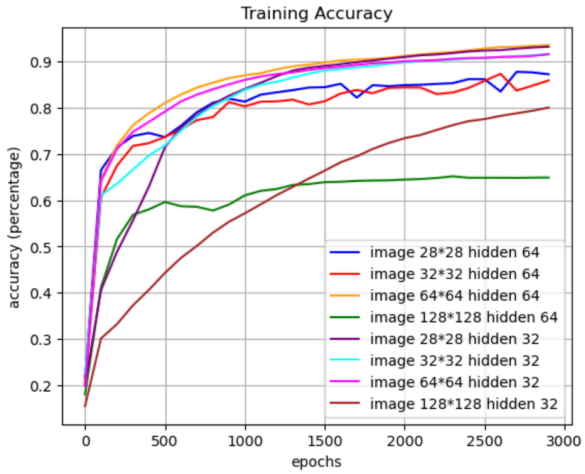
Model 5 : size of image (28*28),epochs =3000, hidden layers =32, learning rate = 0.1

Model 6 : size of image (32*32),epochs =3000, hidden layers = 32, learning rate = 0.1

Model 7 : size of image (64*64),epochs =3000, hidden layers = 32, learning rate = 0.1

Model 8 : size of image (128*128),epochs =3000, hidden layers = 32, learning rate = 0.1





From our training loss and training Accuracy, we found that 64*64 image size and 64 hidden layer size was the best model in terms of loss and accuracy. This might be the fact that if we increase input pixels from the image, the more noise we will get resulting in worse model's learning patterns. On the other hand, if we decrease input pixels, it will make the model generalize the data too much, resulting in losing the crucial for classification.

```
print("============== hidden_size : 32 ==============")
print(f"Test Accuracy input(28*28): {model_accuracy_28_32:.4f}")
print(f"Test Accuracy input(32*32): {model_accuracy_32_32:.4f}")
print(f"Test Accuracy input(64*64): {model_accuracy_64_32:.4f}")
print(f"Test Accuracy input(128*128): {model_accuracy_128_32:.4f}")
print("============== hidden_size : 64 ==============")
print(f"Test Accuracy input(28*28): {model_accuracy_28:.4f}")
print(f"Test Accuracy input(32*32): {model_accuracy_32:.4f}")
print(f"Test Accuracy input(64*64): {model_accuracy_64:.4f}")
print(f"Test Accuracy input(128*128): {model_accuracy_128:.4f}")
```

[175] ✓ 0.0s

```
============== hidden_size : 32 ==============
Test Accuracy input(28*28): 0.8340
Test Accuracy input(32*32): 0.9180
Test Accuracy input(64*64): 0.9200
Test Accuracy input(128*128): 0.8140
============== hidden_size : 64 ==============
Test Accuracy input(28*28): 0.8340
Test Accuracy input(32*32): 0.8600
Test Accuracy input(64*64): 0.9250
Test Accuracy input(128*128): 0.6470
```

Comparison on Test Accuracy

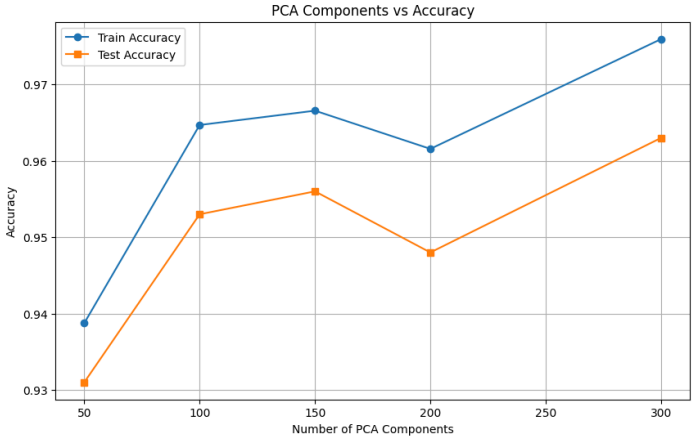
	precision	recall	f1-score	support
0	0.93	0.91	0.92	200
1	0.97	0.98	0.97	200
2	0.94	0.92	0.93	200
3	0.93	0.95	0.94	200
4	0.92	0.93	0.93	200
accuracy			0.94	1000
macro avg	0.94	0.94	0.94	1000
weighted avg	0.94	0.94	0.94	1000

Model 3 evaluate by using y_test

2. PCA and Perceptron model

From the PCA and Perceptron model using 1,000 images (200 per class), we evaluated 72 hyperparameter combinations across PCA components [50, 100, 150, 200], max iterations [500, 1000, 1500], learning rates [0.001, 0.01, 0.1], and tolerance [1e-3, 1e-4] using Grid Search. The best configuration was **150 PCA components**, 500 max iterations, learning rate 0.001, and tolerance 0.001, achieving 95.13% validation accuracy.

To analyze the impact of dimensionality reduction, we experimented with different numbers of PCA components while keeping other hyperparameters constant. Model 1 with 50 PCA components achieved 93.10% test accuracy, Model 2 with 100 components achieved 95.30%, **Model 3 with 150 components achieved 95.60%**, Model 4 with 200 components achieved 94.80%, and Model 5 with 300 components achieved 96.30%. Although Model 5 achieved the highest test accuracy, **Model 3 with 150 components was selected by Grid Search** as it provided the best validation performance, representing an optimal balance between information retention and generalization. Too few components lose crucial information for accurate classification, while too many may not generalize as well to validation data. **The 150 - component configuration retains 90.75% of original variance** while effectively filtering noise.



Using the optimal configuration with **150 PCA components** and 1,000 images, our final model achieved 96.66% training accuracy, 95.13% validation accuracy, and 95.60% test accuracy with a weighted F1-score of 0.96. The model converged after approximately 100 epochs with consistent performance across all sets, indicating good generalization. Class-wise performance showed Shoes as the best performing with F1-score of 0.99 (200 test samples), followed by Watches (0.97), Bags (0.96), Topwear (0.94), and Bottomwear (0.93).

3. **Multilayer Perceptron (MLP)**

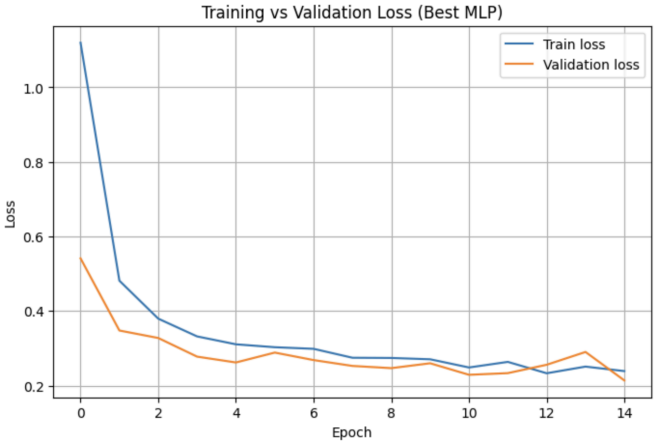
After applying GridSearchCV with 3-fold stratified validation across 24 hyperparameter combinations, the optimal configuration for the MLP was identified as 256 neurons in the first hidden layer, 128 neurons in the second hidden layer, 0.3 dropout, 15 epochs, and batch size 64. This configuration achieved the highest validation accuracy during cross-validation and was retrained on the combined training and validation set.

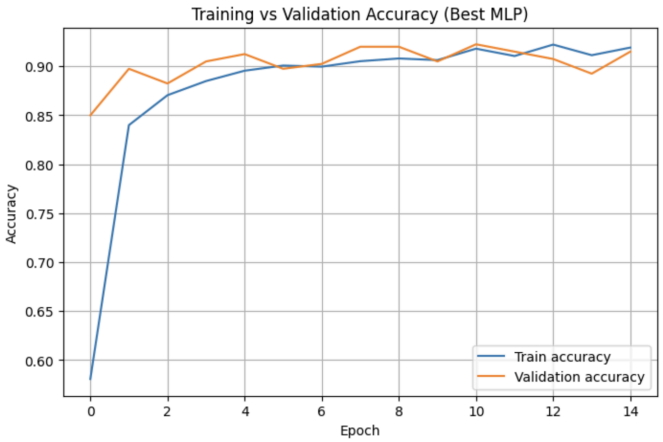
To assess learning behavior, we analyzed the training curves across 15 epochs. Training accuracy increased steadily and validation accuracy stabilized between 90–92%, while validation loss remained consistently close to the training loss. This indicates effective generalization without significant overfitting, a result of dropout and appropriate model capacity. The model converged early, around epoch 10–12, which aligns with the selected hyperparameters.

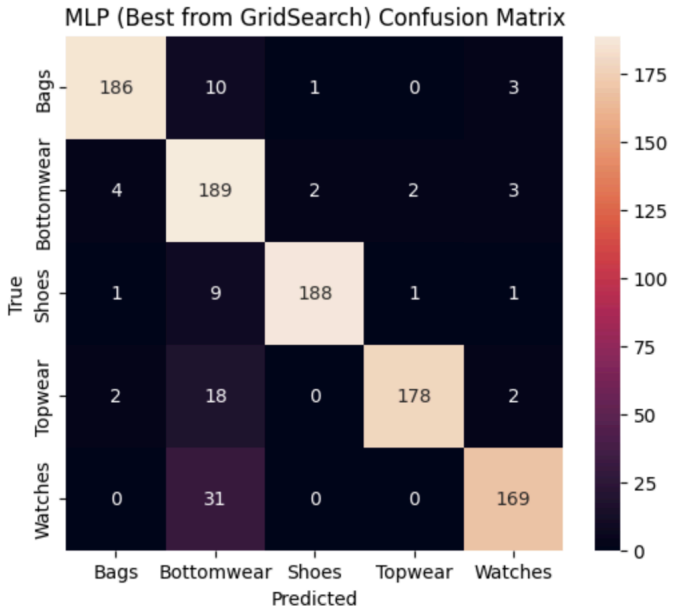
When evaluated on the held-out test set of 1,000 samples (200 per class), the final MLP achieved 91.00% test accuracy and a macro F1-score of 0.9128. These results are strong considering the simplicity of the 28×28 grayscale input and the fully connected architecture. The confusion matrix shows high class-wise performance for Bags (186/200 correct), Shoes (188/200 correct),

and Watches (169/200 correct). *Topwear* and *Bottomwear* exhibited the highest confusion: 31 instances of Bottomwear were predicted as Topwear, and 10 instances of Topwear were predicted as Bottomwear. This reflects the inherent limitation of flattened representations, which discard spatial layout information that would help differentiate clothing silhouettes.

Although the MLP does not leverage spatial features, it consistently learned meaningful global patterns across most categories and demonstrated stable performance across training, validation, and testing phases. Overall, the MLP model provides strong and reliable performance for this image classification task and effectively highlights the capabilities and constraints of fully connected neural networks on low-resolution visual data.







4. CNN

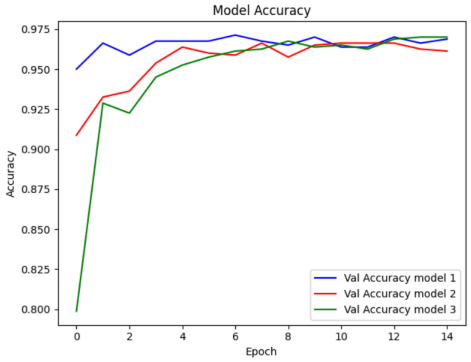
For the case of CNN, we have 3 models in total that we have tried. We use the image size of (64,64) with two convolutional layers and 0.2 dropout as a base model.

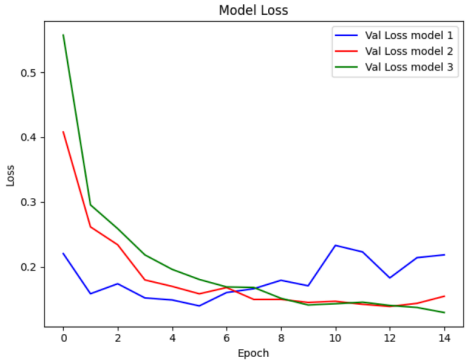
Hyperparameters:

Model 1: epochs = 15, Kernel size = (5,5), Max Pooling = (2,2), Dropout = 0.2

Model 2: epochs = 15, Kernel size = (5,5), Max Pooling = (4,4), Dropout = 0.2

Model 3: epochs = 15, Kernel size = (3,3), Max Pooling = (4,4), Dropout = 0.2





The result shows that the model with the best accuracy and loss value for the validation set is the model number 3. This could be the result of decreasing the number of kernel sizes from 5 to 3 which allows the model to capture finer details from the picture. If the kernel size is higher, the model will be more generalized due to the bigger size of the kernel which neglects some minor details of the image. But from my experience with using the model number 3, it shows signs of overfitting because if the picture used for prediction deviates from the training, it gives out the wrong result. In terms of practical usage, the first model performs the best.

	precision	recall	f1-score	support
Bags	0.97	0.99	0.98	200
Bottomwear	0.99	0.98	0.98	200
Shoes	0.99	0.98	0.99	200
Topwear	0.98	0.99	0.99	200
Watches	0.99	0.97	0.98	200
accuracy			0.98	1000
macro avg	0.99	0.98	0.99	1000
weighted avg	0.99	0.98	0.99	1000

Test Accuracy (Keras): 0.9850

Test Loss (Keras): 0.0548

Model 1 result (X_test)

	precision	recall	f1-score	support
Bags	0.98	0.97	0.98	200
Bottomwear	0.90	0.98	0.94	200
Shoes	1.00	0.99	0.99	200
Topwear	0.99	0.94	0.96	200
Watches	0.97	0.95	0.96	200
accuracy			0.97	1000
macro avg	0.97	0.97	0.97	1000
weighted avg	0.97	0.97	0.97	1000

Test Accuracy (Keras): 0.9670

Test Loss (Keras): 0.0807

Model 2 result (X_test)

	precision	recall	f1-score	support
Bags	0.98	0.98	0.98	200
Bottomwear	0.97	0.98	0.98	200
Shoes	1.00	0.99	1.00	200
Topwear	0.98	0.96	0.97	200
Watches	0.97	0.99	0.98	200
accuracy			0.98	1000
macro avg	0.98	0.98	0.98	1000
weighted avg	0.98	0.98	0.98	1000

Test Accuracy (Keras): 0.9820

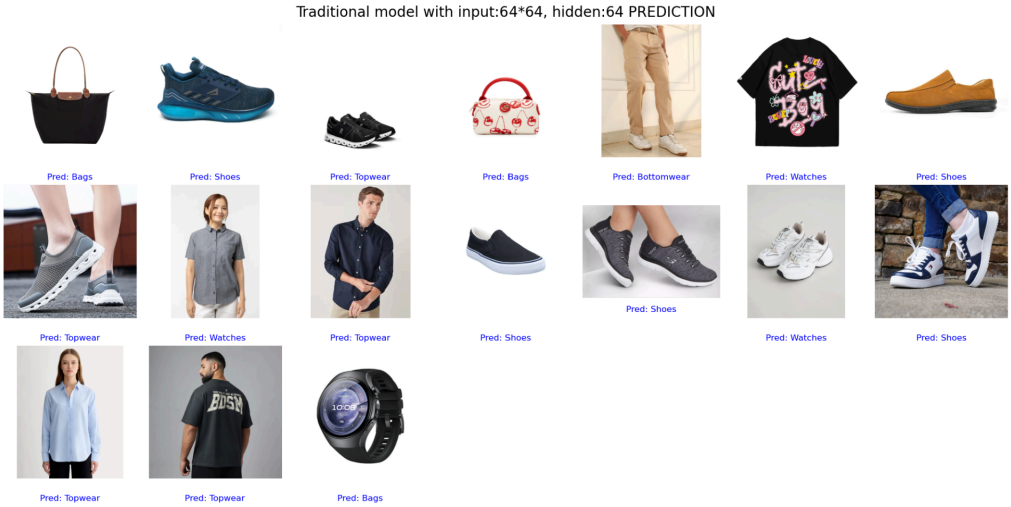
Test Loss (Keras): 0.0678

Model 3 result (X_test)

Discussion

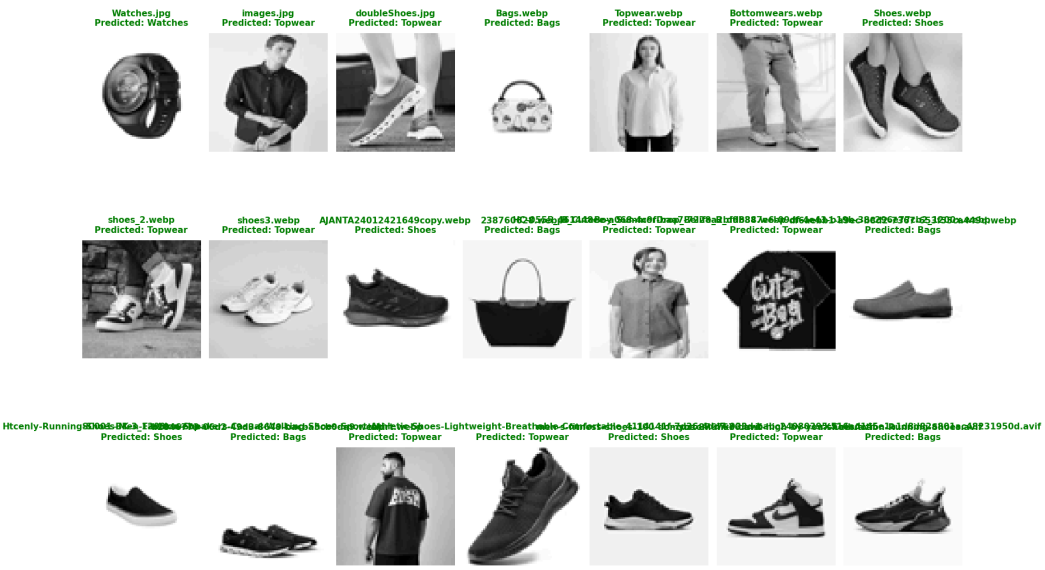
1. Traditional model results

From a test model with 21 real world images, We found that the traditional model has correctly predicted 15 images and incorrectly predicted 6 images.



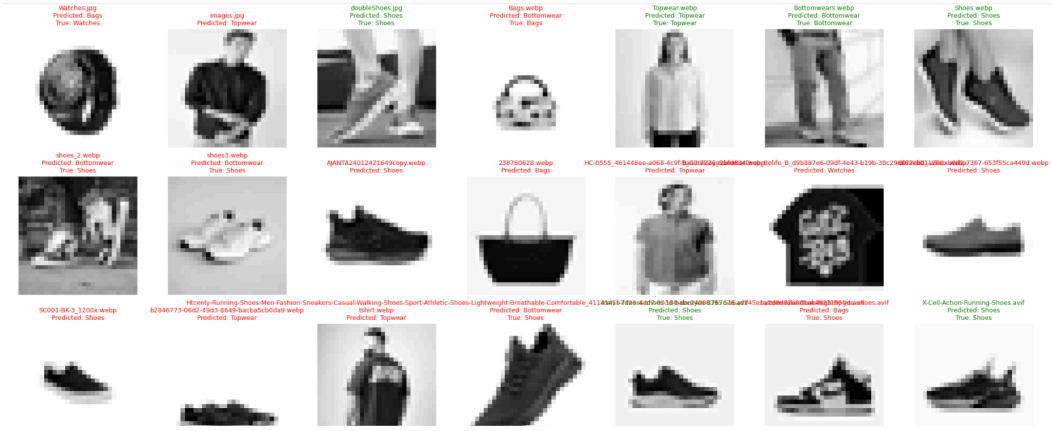
2. PCA and Perceptron results

The PCA and Perceptron also incorrectly predicted 6 images in which the most undetected objects are shoes.



3. **Multilayers Perceptron (MLP)**

After using the same set of images, the model achieved 50% accuracy on the identifiable cases. The lower performance is mainly due to background noise and the loss of detail from converting real photos into 28×28 grayscale inputs.



4. Convolutional Neural Network results

From the same 21 pictures, the Convolutional Neural Network model 1 and 3 predicted 8 images incorrectly. The model 2 gives the worst result with 10 images predicted incorrectly.

To summarize, from all the evaluated models, PCA and the Perceptron can be identified as the most efficient approaches. Although the traditional methods achieved similar prediction accuracy, PCA and the Perceptron required significantly fewer computational resources, making them more effective overall. The CNN model achieved the second-highest prediction performance, likely due to potential overfitting that limited its ability to generalize to unseen data. MLP performed the worst, reaching only 50% accuracy. Its low performance is mainly due to background noise and the loss of important details when real images were reduced to 28×28 grayscale.

Appendix

https://github.com/EGCI463-Fashion-model/classification_models

https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-dataset