# A Machine Learning Approach to Personalized Fashion

**Christian I. Dzul Canulă** 2009047@upy.edu.mx
*Robotics Engineering Student*
*Universidad Politécnica de Yucatán*

**Adriana D. Rivera Martínez** 2009115@upy.ed.mx
*Robotics Engineering Student*
*Universidad Politécnica de Yucatán*

**Fernando Rodríguez Zapata** 2009118@upy.ed.mx
*Robotics Engineering Student*
*Universidad Politécnica de Yucatán*

**Mauricio D. Zaldivar Medina** 2009147@upy.ed.mx
*Robotics Engineering Student*
*Universidad Politécnica de Yucatán*

## Abstract

This project investigates the intersection of technology and fashion, tackling the intricate challenge of staying stylish amidst the ever changing fashion landscape. Employing advanced machine learning techniques: supervised, unsupervised, and reinforcement learning. The project aims to revolutionize our approach to fashion complexities. In the supervised learning phase, models are trained to discern specific apparel, textures, or colors within a vast array of fashion images, showcasing adaptability to diverse imagery. Unsupervised learning is instrumental in revealing implicit patterns in styles from datasets without explicit labels, employing clustering techniques and dimensionality reduction. The reinforcement learning component focuses on crafting outfits aligned with desired styles through an iterative process, incorporating user feedback for continual improvement. This project report provides a comprehensive exploration of methods employed, challenges faced, and innovative solutions devised, aspiring to not only address current fashion challenges but also contribute to a more personalized and accessible fashion landscape, where technology and style converge to shape the future of fashion.

**Keywords:** Fashion, Machine Learning, Clustering, CNN, Supervised, Unsupervised, Reinforcement.

## 1. Introduction

In the dynamic and ever-evolving realm of fashion, the pursuit of staying stylish and on-trend presents a complex challenge. The continuous evolution of fashion trends, coupled with diverse individual preferences, underscores the need for innovative solutions that bridge the gap between the dynamic fashion landscape and personal style choices. Recognizing the potential of advanced technologies, particularly within the domain of machine learning, the project endeavors to tackle these challenges head-on.

This project report explores the intricate intersection of technology and style, delving into the utilization of supervised, unsupervised, and reinforcement learning to revolutionize the way fashion is approached. As trends in fashion perpetually shift, the challenge lies not only in identifying in-vogue apparel and colors but also in crafting personalized and aesthetically pleasing outfit combinations. By leveraging the capabilities of machine learning, the project seeks to offer a comprehensive solution to this multifaceted problem.

Through the lens of supervised learning, the challenge of pinpointing specific apparel is address, textures, or colors within a vast array of fashion images. Unsupervised learning, on the other hand, becomes instrumental in deciphering prevalent patterns in styles from datasets lacking explicit labels. Furthermore, the project explores reinforcement learning as a means to craft outfits that align seamlessly with a desired style or user preference, fostering an iterative process of trial, feedback, and adjustment.

This report takes you on a journey through the project's development, highlighting the methods employed, challenges encountered, and the innovative solutions devised. When delving into the realms of machine learning and fashion, the aim is not only to overcome current challenges but also to contribute to the creation of a more personalized and accessible fashion landscape.

## 2. Main Problem

Fashion is an ever-evolving sector. Trends change consistently, and what's in vogue today might not be tomorrow.Moreover, style preferences vary from one individual to the next. For those looking to dress fashionably and stylishly,there's the challenge of identifying in-trend apparel and colors,figuring out how to combine them, and tailoring general-fashion advice to individual tastes.

- *Supervised:*
  Identifying specific apparel, textures, or colorsin a vast array of fashion images.

- *Unsupervised:*
  Determining prevalent patterns in styles froma dataset without explicit labels.

- *Reinforcement:*
  Crafting outfits that match a desired style oruser preference.

## 3. Description of the data used.

To begin, two datasets were employed, each chosen for a specific task. The first, intended for color classification, was a dataset containing images of colored garments, accessories, and footwear. The second dataset, specifically chosen for garment detection purposes, was a well-known clothing classification dataset with 10 labels in grayscale images was used. These images featured a distinct contrast between the background and the garment, limiting our model to photos with such well-defined backgrounds and well-captured subjects. In the Software of Visual Studio Code, to avoid using the RAM of collab and being able to upload 40k images at once, many attempts were made using different ideas.

### 3.1 Idea 1: Not using images

As stated, the first idea was not using the images directly, but instead using the color histograms of the three channels (BGR) as showed in Figure 1 and apply a K-nearest neighbors (KNN) algorithm with that data. The idea being that if the colors are similar, they should be close together. However, this approach did not work as expected, achieving only an accuracy of 54% (As seen in Figure 2). After this attempt it was noted that this data could have been used with a decision tree or a regular neural network for a better outcome.

```
url_csv = 'data1.csv'
dataframe = pd.read_csv(url_csv)

dataframe
```

| | id | Canal 0 | Canal 1 | Canal 2 | Canal 3 | Canal 4 | Canal 5 | Canal 6 | Canal 7 | Canal 8 | ... | Canal 7990 | Canal 7991 | Canal 7992 | Canal 7993 | Canal 7994 | Canal 7995 | Canal 7996 | Canal 7997 | Canal 7998 | Canal 7999 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 10000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 90.0 | 1331.0 |
| 1 | 10001 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 5.0 | 591.0 |
| 2 | 10002 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 21.0 | 906.0 |
| 3 | 10003 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 58.0 |
| 4 | 10004 | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 334.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 10015 | 9995 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 10.0 | 842.0 |
| 10016 | 9996 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 55.0 | 192.0 | 13.0 | 0.0 | 0.0 | 4.0 | 14.0 | 782.0 |
| 10017 | 9997 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 3.0 | 5.0 | 6.0 | 23.0 | 987.0 |
| 10018 | 9998 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 6.0 | 1127.0 |
| 10019 | 9999 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 3.0 | 2.0 | 3.0 | 6.0 | 7.0 | 13.0 | 15.0 | 26.0 | 41.0 | 980.0 |

Figure 1: Color Histogram of the images

### 3.2 Idea 2: Using Images

For the second idea, the images were split and separated into folders for training and testing. In this part, irrelevant data for the classifier was filtered out. Different classes such as accessories, makeup, underwear, and others were excluded and instead, the focus was on clothing similar to the classes in the "MNIST Fashion" dataset, this with the objective of reducing the number of observations, which include:

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.neighbors import KNeighborsClassifier

    X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.2, random_state=42)

    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train, y_train)
    accuracy = knn.score(X_test, y_test)

    print("Precisión del modelo KNN en los datos de prueba:", accuracy)

    Precisión del modelo KNN en los datos de prueba: 0.5523952095808383
```

Figure 2: KNN Test

- T-shirt/top
- Trouser
- Pullover
- Dress
- Coat
- Sandal
- Shirt
- Sneaker

It's important to clarify that the code splits the dataset into an 80-20 ratio not only for the entire dataset but also for each color individually. Colors with fewer than 100 observations were disregarded since there were other classes with thousands of observations. Nevertheless, this was changed due to a lack of improvement in the classifier, which then ended with the following used classes:

- Navy Blue
- Blue
- Black
- Grey
- Green
- Purple
- White
- Brown
- Pink
- Red

Visual Studio Code was employed to transform images into rows within a .csv file. This was achieved with the combination of OpenCV and the .read function. Given the structure of the images as (height, width, 3 channels in this instance), applying flatten consolidates this composition into a singular, refined row. Managing the images in this way enhances efficiency. Additionally, knowing the dimensions provides the advantage of being able to "reassemble" the image using a reshape operation.

The code reads each image, identified by its "id".jpg nomenclature in the image .csv. This id is linked to its features. Thus, by extracting the image's name from the .csv, this could be associated with its corresponding color. Subsequently, a row is generated with a length of 25x35x3 (the image is read as a tensor of three dimensions with these specific dimensions), and the label denoting its color is appended at the end.

# 4. Supervised Solution

## 4.1 Color Classification

### 4.1.1 Preprocessing

After dividing the dataset into two different files to manage the data of the picture. Another dropout was made for the color classes with less than 800 observations, and so the 10 most common were left, and since there were already 0-n label for each class, the 0-9 labels had to be updated (Table 1).

| Number Assigned | Class |
|---|---|
| 0 | Black |
| 1 | White |
| 2 | Blue |
| 3 | Grey |
| 4 | Brown |
| 5 | Red |
| 6 | Green |
| 7 | Navy Blue |
| 8 | Pink |
| 9 | Purple |

Table 1: Number labeled to each color

```
[ ] class_counts = train_data['baseColour'].value_counts()

[ ] color_800 = class_counts[class_counts >= 800].index
    train_data= train_data[train_data['baseColour'].isin(color_800)]

[ ] class_counts = train_data['baseColour'].value_counts()

[ ] class_counts

    2     4383
    6     3110
    1     2911
    3     1521
    7     1445
    10    1296
    4     1246
    0     1056
    8      808
    5      808
    Name: baseColour, dtype: int64

[ ] classes = [2, 6, 1, 3, 7, 10, 4, 0, 8, 5]

    test_data = test_data[test_data['baseColour'].isin(classes)]
```

```
[ ] classes = [2, 6, 1, 3, 7, 10, 4, 0, 8, 5]
    new_classes = {original: new_no for new_no, original in enumerate(classes)}

    train_data['baseColour'] = train_data['baseColour'].map(new_classes)
    test_data['baseColour'] = test_data['baseColour'].map(new_classes)

[ ] class_counts = train_data['baseColour'].value_counts()
    class_counts

    0    4383
    1    3110
    2    2911
    3    1521
    4    1445
    5    1296
    6    1246
    7    1056
    8     808
    9     808
    Name: baseColour, dtype: int64
```

Figure 3: Dataset Observations and Classes Updated

In this phase of the process, the independent (Xs) and dependent (Ys) variables were obtained to address the color classification problem. Initially, the identification column was excluded since it does not provide significant information for our purpose. Next, the variables X and Y were selected. It is crucial to note that the "reshape" operation was applied with the dimensions (-1, 35, 25, 3). Using "-1" allows the number of observations to be determined automatically, thus generating a tensor. In this context, a tensor is defined as a multidimensional array. The resulting tensor stores three-dimensional arrays (35, 25, 3) for each observation in our data set. This dimensional structure is essential to adequately represent the visual information of our images in the color classification process.

```
[ ]  train = train_data.drop('id', axis=1)
     test = test_data.drop('id', axis=1)

[ ]  X_train = train.iloc[:, :-1].values
     Y_train = train.iloc[:, -1].values

[ ]  Y_train = np.array(Y_train)

[ ]  X_test = test.iloc[:, :-1].values
     Y_test = test.iloc[:, -1].values

[ ]  X_train = X_train.reshape(-1, 35, 25, 3)
     X_test = X_test.reshape(-1, 35, 25, 3)
```

Figure 4: Features and Labels

At this stage, data augmentation techniques were implemented, it is a fundamental process to artificially enrich our set of observations. Data augmentation involves manipulation of images, including operations such as rotation, inversion, and zoom, for the purpose of generating additional variations. Specifically, for each existing image in the set, three new images have been created by applying various transformations. These modifications not only diversify the dataset, but also strengthen the model's ability to generalize patterns, thereby improving its performance in color classification. This strategic focus on data augmentation contributes significantly to the robustness and effectiveness of the classification model.

Subsequently, the data set joining, or concatenation phase was carried out. This process involves the combination of the original dataset with the new images generated from the data augmentation. The merging was executed so that the original dataset and the additional images became a single, coherent entity. This integration is essential to ensure that the classification model has access to the totality of observations, both the original and those generated by the data augmentation, thus providing a more complete and diversified basis for model training. The conjunction of both sets lays the foundation for a more robust and effective training process.

```
[ ] import numpy as np
    from tensorflow.keras.preprocessing import image
    from tensorflow.keras.preprocessing.image import ImageDataGenerator


    datagen = ImageDataGenerator(
        horizontal_flip=True,
        rotation_range=45,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
    )

    augmented_images = []
    augmented_labels = []

    for img, label in zip(X_train, Y_train):
        img = img.reshape((1,) + img.shape)
        label = np.array([label])

        for x_batch, y_batch in datagen.flow(img, label, batch_size=3):
            augmented_images.append(x_batch[0])
            augmented_labels.append(y_batch[0])
            break

    augmented_images = np.array(augmented_images)
    augmented_labels = np.array(augmented_labels)
```

Figure 5: Data Augmentation

```
[ ] X_train = np.concatenate((X_train, augmented_images), axis=0)
    Y_train = np.concatenate((Y_train, augmented_labels), axis=0)

[ ] X_train = X_train / 255.0
    X_test = X_test / 255.0
```

Figure 6: Dataset concatenation and Re-sacaling of pixel values

The division by 255 operation is performed for the purpose of normalizing the pixel values on a scale of 0 to 1. This transformation is essential to facilitate the model training process. By scaling the pixel values to a smaller range, from 0 to 1, numerical stability during training is improved. In addition, this practice is beneficial for activation and optimization functions that are commonly used in deep learning models. Pixel normalization contributes to more efficient convergence during the training process, allowing the model to more effectively and accurately learn the patterns present in the input data.

Then, the original labels, which were in a range from 0 to 9 were transformed using the "one-hot" coding technique or also known as dummies. This conversion involves representing each label as a binary vector, where only one element corresponding to the class of the label has the value of 1, while the others are 0.

This strategy was implemented with the objective of evaluating whether one-hot encoding has any significant impact on model performance during training. A comparative analysis between the use of labels in the range of 0 to 9 and one-hot coding was performed in order to determine whether this particular transformation positively or negatively affects the learning process of the color classification model.

```
[ ]  Y_train_one_hot = to_categorical(Y_train, 10)
     Y_test_one_hot = to_categorical(Y_test, 10)
```

Figure 7: New Variables for One-hot Encoding

In the context of unbalanced data sets, the class penalty technique was implemented. It consists of assigning a differential weight to classes, prioritizing minority classes and thus providing additional stimulus during the training process.

Despite the expectation that this penalty would improve the classification performance of underrepresented classes, practical results were inconclusive. In one specific attempt, it was observed that the application of penalization did not generate significant improvements and, in some cases, even worsened model performance. This finding suggests that, in certain scenarios, class penalization may not be an effective solution to address the imbalance in the dataset, and alternative approaches are required to improve the predictive capability of the model in minority classes.

### 4.1.2 Convolutional Neural Network (CNN)

During the design and tuning phase of the CNN model, several structures were explored, focusing especially on the implementation of the pooling technique. Pooling consists of reducing the size of the image while applying filters, allowing different features of the image to be highlighted.

Through multiple attempts with different configurations, it was possible to identify a structure that proved to be the most effective so far, reaching an accuracy of 73%. It is important to note that in this iteration it was chosen to work with labels in the range 0 to 9, without applying one-hot encoding.

This strategic pooling approach contributes to the model's ability to capture distinctive features of the images, resulting in improved classification performance. The decision not to use one-hot encoding for labels was made considering the specific evaluation of this scenario and its implications on model performance. For example, one of the many attempts was without pooling, the result obtained showed an accuracy of 70%. In this scenario, size reduction by pooling was dispensed with, implying that specific image features were processed without applying this type of operation.

There was also one attempt with One-hot encoding to see the model's performance. Each test of the model training process involved systematic modifications to the convolutional layers, with the objective of evaluating and improving model performance. Various configurations were explored, varying the presence or absence of pooling, the inclusion of dropout layers, among other adjustments, in a constant search to optimize the model design. These iterative adjustments were essential to understand how different configurations affect the performance of the model in the color classification task.

## 4.2 Apparel Classification

### 4.2.1 Preprocessing

For this algorithm, the "Fashion MNIST" dataset was used; this algorithm has 28x28 images in grayscale and is separated into the following classes:

| Number Assigned | Class |
|---|---|
| 0 | T-shirt/top |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle Boot |

Table 2: Number labeled to each apparel

As the selected dataset was well done with grayscales and backgrounds well defined in contrast with the cloth; previous preprocessing was not implemented.

For the splitting into training and testing, the dependent (Ys) or target was the first column of the dataset, this being the class of the image, and the independent (Xs) or features were the rest of the columns, these being the pixels of the image.

Then, the same process as the Color Classification was done: as the images were from 0 to 255 and they were needed from 0 to 1, a division by 255was made; right after, as done for the Color Classification, a reshape was done to form the image once again the X datasets were reshaped for it to be 28x28.

### 4.2.2 Convolutional Neural Network (CNN)

As well as the Color Classification was implemented for the apparel classification images. Various architectures were experimented with, and the models consistently achieved around 89-90% accuracy. Two models were saved during this process, providing options for final selection.

```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_accuracy}')
```

```
313/313 [==============================] - 2s 7ms/step - loss: 0.2493 - accuracy: 0.9072
Test accuracy: 0.9071999788284302
```

Figure 8: Accuracy of Apparel Classification.

### 4.3 Implementation

The different models were tested with new images, distinct from the test and train data. The model exhibiting the best performance was ultimately chosen. During these tests, some models struggled with accurately classifying colors, and the final models were selected based on their overall performance. Additionally, a function was developed where an image is entered along with the chosen models. The function then generates the detection/classification of the cloth, including its color.

```
model_color.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 33, 23, 32)        896

 max_pooling2d (MaxPooling2  (None, 33, 23, 32)        0
 D)

 dropout (Dropout)           (None, 33, 23, 32)        0

 conv2d_1 (Conv2D)           (None, 31, 21, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 31, 21, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 29, 19, 32)        9248

 max_pooling2d_2 (MaxPoolin  (None, 14, 9, 32)         0
 g2D)

 dropout_1 (Dropout)         (None, 14, 9, 32)         0

 conv2d_3 (Conv2D)           (None, 12, 7, 64)         18496

 max_pooling2d_3 (MaxPoolin  (None, 6, 3, 64)          0
 g2D)

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 128)               147584

 dropout_2 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 128)               16512

 dropout_3 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 210890 (823.79 KB)
Trainable params: 210890 (823.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model_apparel.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_1 (Conv2D)           (None, 11, 11, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 32)          0
 g2D)

 flatten (Flatten)           (None, 800)               0

 dense (Dense)               (None, 128)               102528

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 64)                4160

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 125162 (488.91 KB)
Trainable params: 125162 (488.91 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 9: CNN Models Layers

A function called "Detection" was created, which takes the image and the models for color and apparel classification as inputs. First, the image is read, and a copy is saved for later display of the original image. The image undergoes a blur operation. Subsequently, pixels are cropped from the top, bottom, and sides to focus more on the subject and eliminate unnecessary background. The image is then converted to grayscale for apparel classification and to color for color detection, each with their respective reshapes, as the models require images with the same dimensions they were trained on. As said in the color preprocessing, the pixel values are normalized by dividing them by 255 to scale them between 0 and 1.

To handle the dummy outputs ([0,0,1...,0]) from the models, dictionaries were introduced. These dictionaries help map the model output index to the corresponding label. Predictions are then made using the .predict method, inputting the images. Finally, the original image is displayed using Matplotlib, with the title indicating the color and apparel classifications based on the predictions.

Testing with images downloaded from the internet, as well as a photo taken by one team member, was conducted. These tests were specifically designed to evaluate the model's performance on images that it had not encountered during training or testing. This helps assess the model's generalization ability and how well it can handle real-world scenarios with previously unseen data.

The following image shows two of the predictions made by the models, the first one (Figure 11, left) portrays a purple dress predicted correctly, whereas the second one (Figure 11, right) shows that even it did detect the color correctly, the apparel was wrong, as it said it was a T-shirt/Top when it should be an Ankle boot.

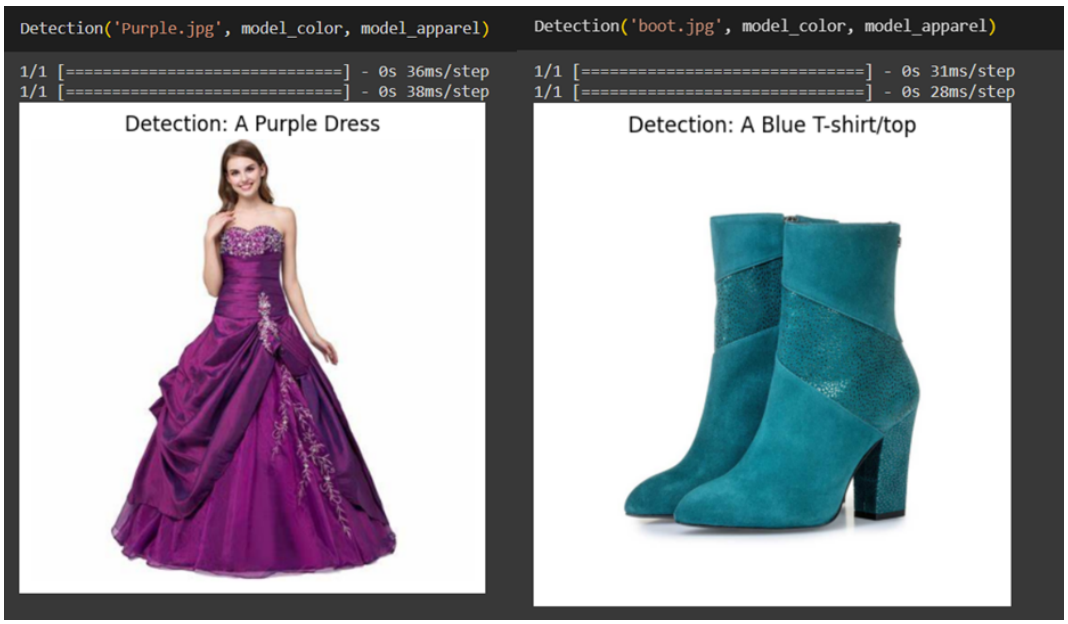


Figure 10: Tests

## 5. Unsupervised Solution

### 5.1 Development

Principal Component Analysis (PCA) was employed, it is a dimensionality reduction technique that transforms the original features into a new set of orthogonal components, capturing the variance in the data. These components were then used in conjunction with K-means clustering, that is an unsupervised machine learning algorithm that partitions data points into k clusters based on their similarity.

In this machine learning project, several Python libraries were used to facilitate various tasks related to image processing, data manipulation, clustering, and visualization.

- Scikit-learn (sklearn):
  - KMeans: The KMeans class is imported for clustering, a technique used to group data points based on similarity.
  - make_blobs: This function from sklearn.datasets is used to generate synthetic datasets with clusters.
  - PCA: Principal Component Analysis is utilized for dimensionality reduction.
- Plotly Express (px): Plotly Express, imported as px, is a high-level interface for creating interactive visualizations. It simplifies the process of generating complex plots and is built on top of the Plotly library.

In the initial preprocessing phase of the project, the training data was reshaped. The reshape operation transforms this data, creating a new array with 37168 rows. The primary objective of this reshaping is to convert the data into a 2D array format where each row corresponds to an individual sample, and the columns represent different features.

Continuing, for the training process, two fundamental techniques are strategically applied to extract meaningful patterns from the fashion dataset. Firstly, dimensionality reduction is performed using Principal Component Analysis (PCA), considered an essential step when dealing with high-dimensional data. In this context, an attempt is made to distill the data's complexity into three principal components, capturing the most significant sources of variation. The specified number of components initializes the scikit-learn PCA class, and the fit_transform method is then employed on the reshaped training data (X_train_reshaped).

The resultant transformed dataset, referred to as X_train_pca, encapsulates the information in a lower-dimensional space, providing a more concise representation. Subsequently, K-Means clustering is utilized to uncover inherent structures and groupings within the data. In this instance, it was necessary to identify 10 distinct clusters (n_clusters = 10). The scikit-learn KMeans class is instantiated accordingly, and the fit_predict method is applied to the PCA-transformed data (X_train_pca). The resulting labels array encapsulates the assigned cluster for each data point.

This approach enables the algorithm to autonomously discern patterns and similarities among the fashion items in the dataset. These unsupervised learning techniques are pivotal

in unraveling the latent structures inherent in the fashion data, thereby paving the way for more informed and intelligent decision-making in the subsequent stages of the project.

In the exploratory phase of the project, data visualization techniques using Plotly Express are employed to gain insights into the structure of the dataset after Principal Component Analysis (PCA) transformation. The construction of a DataFrame (‘df‘) from the PCA-transformed data (‘X_train_pca‘) marks the beginning of this code segment. The columns of this DataFrame are labeled ’PC1’, ’PC2’, and ’PC3’, representing the three principal components derived from the unsupervised learning process. Furthermore, a new column, ’color’, is introduced, with values assigned based on the first principal component.

The subsequent step encompasses the creation of an interactive 3D scatter plot. In this plot, the three principal components (’PC1’, ’PC2’, ’PC3’) are mapped to the x, y, and z axes, respectively. The color of each data point is determined by the ’color’ column, introducing a visual representation of the first principal component’s influence. The resulting figure, titled ’PCA Visualization,’ is rendered with an opacity setting of 0.7, enhancing the clarity of the plotted points.

This interactive visualization provides an intuitive platform for exploring the distribution and clustering patterns of fashion items in the reduced-dimensional space. It serves as a valuable tool for interpreting the outcomes of the unsupervised learning techniques, offering a visual representation of the relationships and groupings identified by the PCA and K-Means clustering processes. The insights gained from this visualization lay the foundation for subsequent analyses and recommendations in the pursuit of intelligent outfit generation.

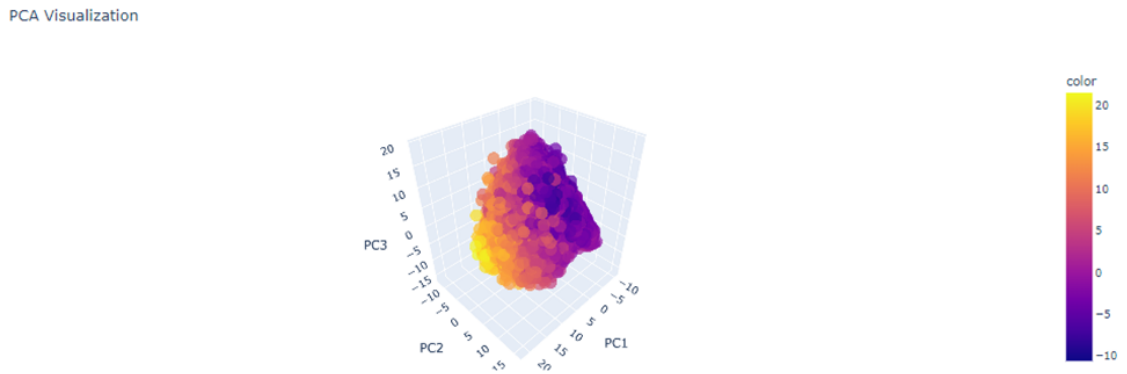


Figure 11: Visualizing PCA results in 3D with Plotly Express

In the pursuit of gaining qualitative insights into the clusters identified by the K-Means algorithm, a visualization strategy has been implemented to showcase a selection of images from each cluster. Lists (‘cluster_indices‘) are initialized to store the indices of images associated with each cluster. Subsequently, the dataset is iterated through by the loop, and indices are assigned to their respective clusters based on the labels obtained from the K-Means clustering process.

The visualization is orchestrated to display a curated selection of images from each cluster. For each cluster, a subplot is generated, and a predetermined number of images (‘num_images_per_cluster‘) are randomly selected from the cluster’s indices. To ensure suitable visualization, pixel values of the original images are normalized, and a conversion from BGR to RGB format is applied. The resulting visualizations, presented in a side-by-side layout, offer a qualitative assessment of the diverse fashion items encapsulated within each identified cluster.

This approach provides a valuable means of qualitatively interpreting the contents of the clusters, shedding light on the common characteristics and stylistic elements that have been discerned within the fashion dataset by the K-Means algorithm. The random sampling ensures a representative display of images within each cluster, facilitating a nuanced understanding of the distinctive features captured by the unsupervised learning process.


Figure 12: Visual sampling from each cluster

## 6. Reinforcement Solution

### 6.1 Development

The model's focus encompasses the upper part, lower part, and footwear, considering color groups and their respective clusters during the learning process. In the current implementation, color groups are directly chosen as part of the model's training. However, it is acknowledged that an optimal approach involves employing unsupervised learning techniques for the selection of color groups. This adjustment aims to enhance the model's ability to discern and categorize colors more organically. Similar to color groups, clusters, representing like colors, are labeled from 1 to 10 in the existing model. A recommended improvement involves utilizing unsupervised learning methods for determining cluster assignments. This adjustment ensures a more data-driven and nuanced understanding of the relationships between different color clusters.

Despite the considerations, the current model undergoes training where rewards are assigned based on specific criteria. Rewards are granted when the chosen garments are correctly positioned, when the colors of each garment belong to the same color group, and when the clusters align closely (e.g., proximity between cluster 1 and 2, but distance from cluster 10). This reinforcement learning approach aims to instill a sense of spatial and thematic coherence in the generated outfits.

This section of the report outlines the essential components and libraries utilized in the development of a reinforcement learning environment for training and evaluating agents. The objective is to provide a succinct overview of the key Python libraries and tools incorporated into the project.

To begin with, the "FashionEnv" class is created and derived from the OpenAI Gym's base environment class (gym.Env). It encapsulates the essential components required for training reinforcement learning agents. It is defined the action and observation spaces which consist of:

**Action Space:** The action space is defined as a MultiDiscrete space, denoting a multi-dimensional discrete space. In this case, the action space consists of nine dimensions, each ranging from 0 to 9, representing choices for clothing, color, and cluster categories.

**Observation Space:** Similarly, the observation space is a MultiDiscrete space with nine dimensions, corresponding to the current state of the environment. These dimensions represent the chosen clothing, color, and cluster for the outfit. The environment implements an epsilon-greedy exploration strategy. With a probability of epsilon (0.3 in this case), a random action is selected from the action space. This strategy introduces randomness to the agent's actions, promoting exploration.

Then, the reinforcement learning environment assigns rewards based on the chosen outfit configuration. The reward logic is structured as follows:

- Positive rewards are given if the selected clothing items are well-positioned (e.g., top, bottom, footwear).
- Rewards are provided for choosing colors that belong to the same predefined color groups.

- Additional rewards are granted for selecting clusters that are the same or at least two clusters that match.

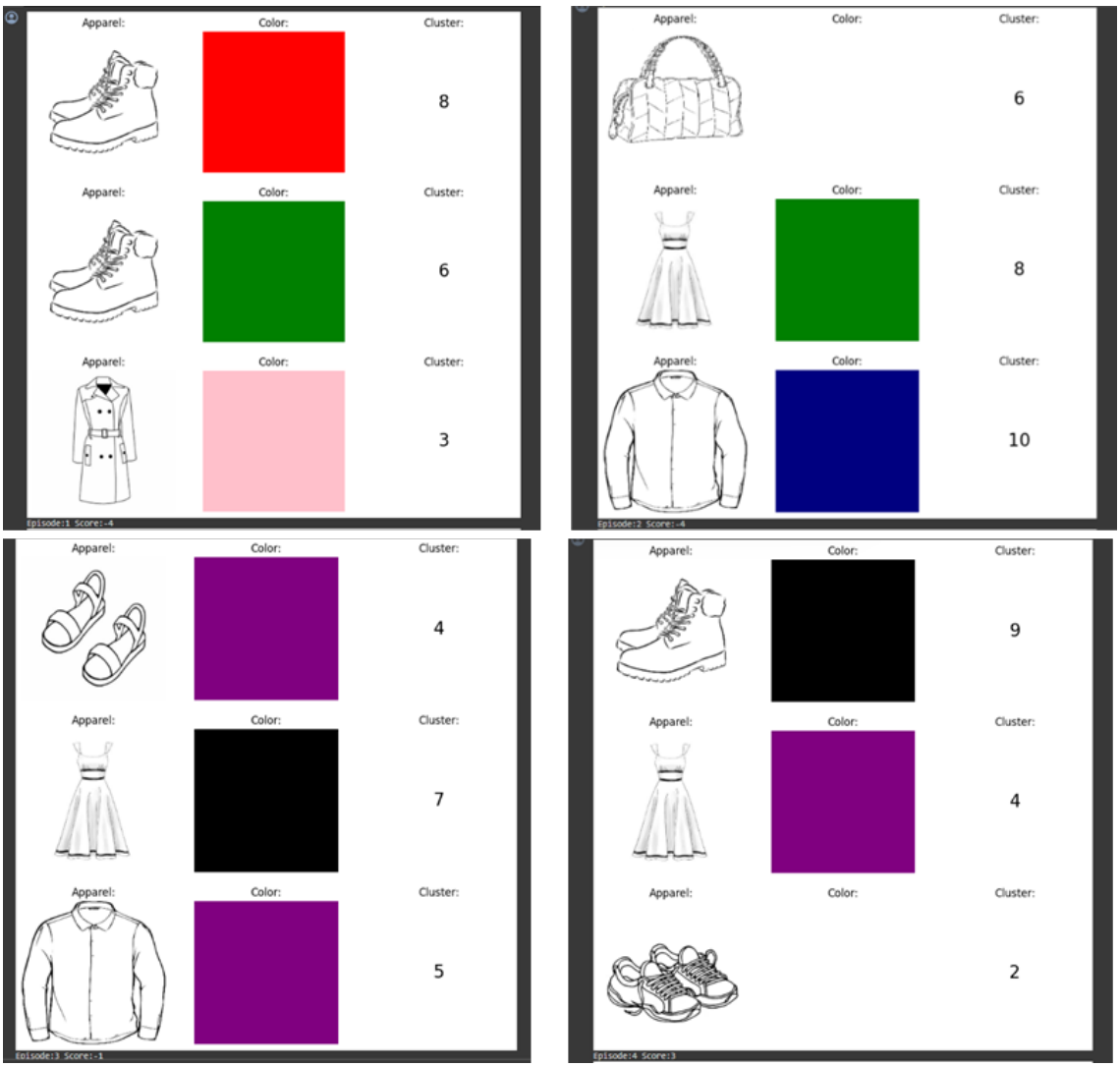


Figure 13: Outputs of the environment

Continuing, the render method generates a visual representation of the chosen outfit based on the current state. It utilizes images of clothing items, color representations, and cluster labels to create a visual display of the outfit. Finally, the reset method initializes the environment's state, setting all components to an initial state of 10. This method is typically called at the beginning of each episode.

Then, using the environment created tests were made to see the behavior. First, it initializes an instance of the "FashionEnv" class, using the environment for training reinforcement learning agents. The training loop runs for a specified number of episodes,

where each episode begins by resetting the environment to its initial state. Within each episode, the agent takes random actions from the defined action space, representing choices related to clothing, color, and cluster categories. The environment responds with the new state, the obtained reward, and a flag indicating whether the episode is completed. The 'env.render(n_state)' function visually displays the chosen outfit at each step. The agent accumulates rewards throughout the episode, and the final score is printed after the episode concludes . This iterative process allows for testing the agent's performance in generating well-coordinated outfits based on the reinforcement learning model implemented in the "FashionEnv" environment. The environment is closed after completing the specified number of episodes, finalizing the test.

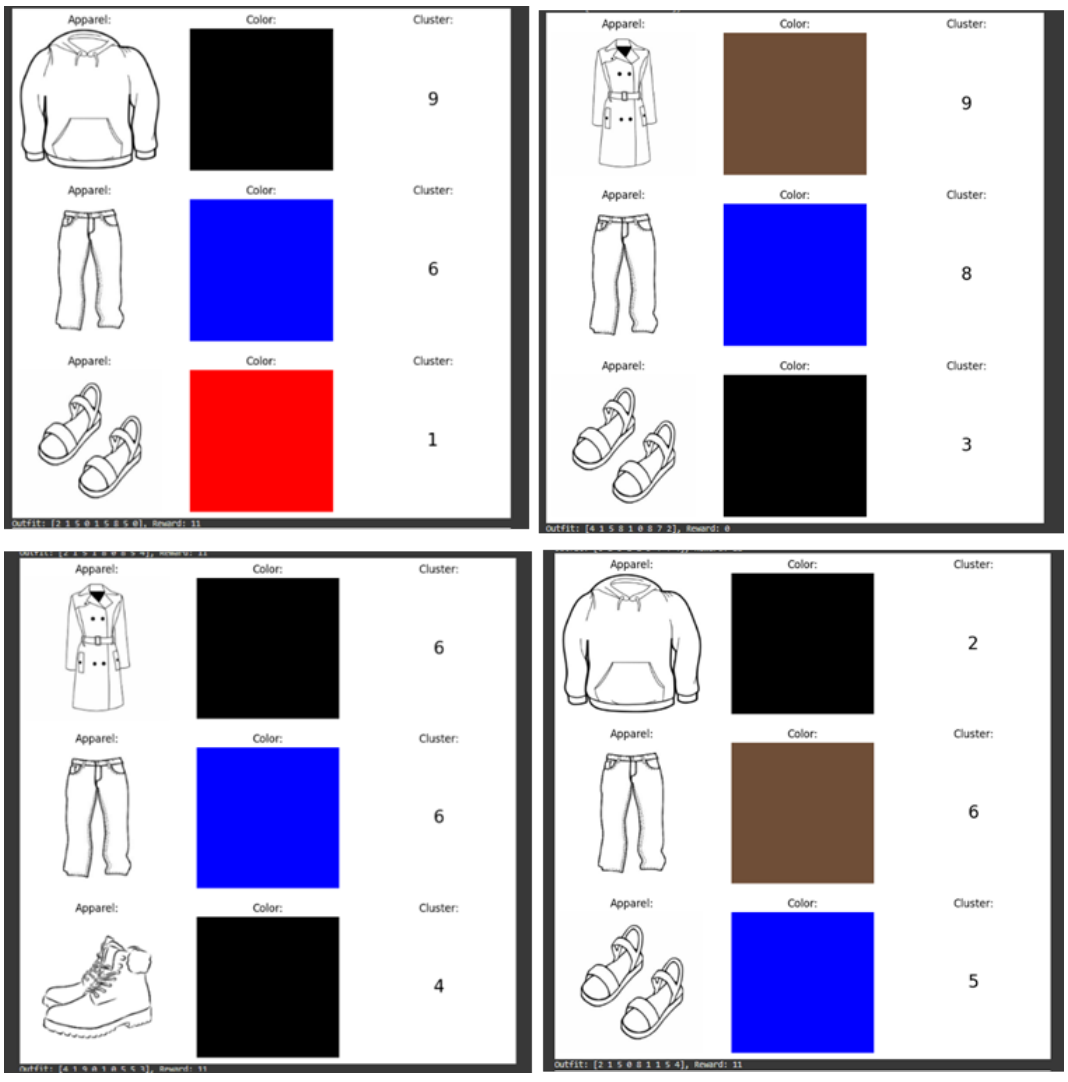


Figure 14: Model Performance Output

Subsequently, for the training process, it is defined a log path for storing training-related information and creates a DummyVecEnv wrapper, adapting the environment to be compatible with the stable baselines PPO (Proximal Policy Optimization) algorithm. The PPO algorithm is then employed to create a model using a Multi-Layer Perceptron (MLP) policy. The training process commences with the model.learn() function, which iterates over a specified number of total timesteps (300,000 in this instance) to train the agent within the fashion environment. The training is carried out with verbosity enabled, providing detailed information about the training progress. This part of the code snippet represents the training pipeline for the reinforcement learning agent, allowing it to learn optimal strategies for outfit generation within the given environment.

After training the model, it was necessary to check the model's performance. As observed, it was used 20 episodes, in each episode, the environment is reset, and the model predicts actions, representing clothing, color, and cluster choices. The environment responds to these actions, and the chosen outfits are visually displayed. Information about each step, including the chosen outfit and the associated reward, is printed. This process allows an assessment of the model's performance in generating outfits, providing insights into its decision-making and the rewards obtained. The evaluation concludes after 20 episodes, and the environment is closed. Overall, it was analyzed the trained model's ability to create well-coordinated outfits within the specified fashion environment.

## 7. Conclusion and Future Steps.

In conclusion, this project has demonstrated the potential of advanced machine learning techniques to address the complex challenges of personalized fashion. By leveraging supervised, unsupervised, and reinforcement learning, we have made significant progress in identifying in-trend apparel, revealing implicit patterns in fashion styles, and crafting personalized outfit combinations. The utilization of diverse datasets, data augmentation, and data preprocessing has laid the foundation for robust and effective model training.

Looking ahead, the future steps for the project involve further refinement of the machine learning models, exploration of additional datasets to capture a wider range of fashion styles, and the integration of user feedback to continually improve the personalized fashion recommendations. Additionally, the project aims to contribute to the creation of a more personalized and accessible fashion landscape, where technology and style converge to shape the transformative future of fashion.

## 8. Github Link

The following is the link to access the code: Click here

## References

[1] Abid Ali Awan (2022, November 23). *A Complete Guide to Data Augmentation.* Retrieved from: https://www.datacamp.com/tutorial/complete-guide-data-augmentation

[2] Mandal, M. (2021, May). *Introduction to Convolutional Neural Networks (CNN).* InfoWorld. Retrieved from: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

[3] Built In. (2019). *A Step-by-Step Explanation of Principal Component Analysis (PCA).* Retrieved from: https://builtin.com/data-science/step-step-explanation-principal-component-analysis

[4] Ramírez, L. (2023, January 5). *Algoritmo k-means: ¿Qué es y cómo funciona?.* Retrieved from: https://www.iebschool.com/blog/algoritmo-k-means-que-es-y-como-funciona-big-data/

[5] OpenAI. (2017). *Proximal Policy Optimization.* Retrieved from: https://openai.com/research/openai-baselines-ppo

[6] A. K. Singh, S. Kumar, and S. K. Singh. (2021). *A Comparative Study of Data Visualization Tools for Machine Learning.*International Conference on Cloud Computing, Data Science  Engineering (Confluence), 2021, pp. 1-6.