**Fashion Product attribute determination using Tensorflow:**

**A statue of a person wearing a suit and tie

Description automatically generated**

**Business Case:**

Fashion E Commerce websites have a significant impact on business costs and productivity. E-Commerce has a chance to be widely adopted due to its simple applications. Thus it has a large economic impact. Electronic Commerce provides the capability of buying and selling products and information on the internet and other on-line service. They are a boon for the sellers selling product online. Where they can advertise their products online. But what about the E commerce firms how would they determine if the image uploaded by the seller about the product matches the description mentioned or not. If there is a mismatch there might be negative review for the website and might affect the customer base in a long term. So ability to have a systematic andquantitative characterisation of our products is key for the company to make data-driven decisionsacross a set of problems, includingpersonalization, demand forecasting, range planningand logistics. We need a solution how to predict a consistent and complete set of product attributes, and illustrate how this enables us to personalise the customer experience by providing more relevant product recommendations.

**Solution:**

Build an Automatic Fashion product attribute Identifier using Image processing and compare the attributes with the attributes mentioned by the seller. Our model extracts attribute values from product images and textual descriptions. To do this, we rely on the success that Deep Neural Networks have in classifying text and images. Below, we briefly discuss the main model components.

* Type: Upper wear / bottom wear
* Clothing item type: ex. T-shirt, Shirt, jeans
* Length: Half sleeve or full sleeve
* Color: Multi colored or single colored
* Pattern: Solid or printed

**Image processing:**

For each product, four high-quality images are produced by the studio team. As fashion is predominantly a visual business, visual features are at the core of many data science products. Training good image models from scratch can take weeks. As we use image features for many applications, to minimise computational costs we have implemented a centralised visual feature generation pipeline. This uses pre-trained [Convolutional Neural Networks](https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050) (CNNs) to extract product representations from images.

**Process:**

1. Scrap data ~ 10K -15K for each type of clothing.
2. Clean the data using data wrangling techniques.
3. Covert you data into vectors using Keras Image generator and covert the data into train and validation set.
4. Build a 4D CNN model and train your model for each attribute.
5. Build a pipeline with the above to be used as web interface.
6. Build a simple web interface to access the functionality using Flask framework and HTML
7. Deploy the above in GCP cloud.

**Data Scraping:**

We can scrap the data form multiple Fashion ecommerce websites. We need around 10-25K images with attributes. So we will be using **scrapy** and **selenium** for this. Please refer to the Web scarping notebook for the code.

**Data Wrangling:**

Data wrangling refers to the process of cleaning, restructuring and enriching the raw data available into a more usable format. Organizing and cleaning data before analysis has been shown to be extremely useful and helps the firms quickly analyses larger amounts of data. Data Acquisition. Identify and obtain access to the data within your sources. So we will be performing the below operation.

1. Joining data. Combine the edited data for further use and analysis.
2. Data cleansing. Redesign the data into a usable and functional format and correct/remove any bad data.

Please refer to the Data extraction and wrangling notebook for the code.

**Building the CNN model and training the model:**

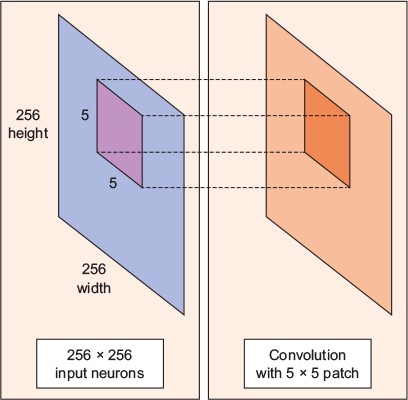
**Image Classification:**

The problem of Image Classification goes like this: Given a set of images that are all labelled with a single category, we are asked to predict these categories for a novel set of test images and measure the accuracy of the predictions. There are a variety of challenges associated with this task, including viewpoint variation, scale variation, intra-class variation, image deformation, image occlusion, illumination conditions, background clutter etc.

How might we go about writing an algorithm that can classify images into distinct categories? Computer Vision researchers have come up with a data-driven approach to solve this. Instead of trying to specify what every one of the image categories of interest look like directly in code, they provide the computer with many examples of each image class and then develop learning algorithms that look at these examples and learn about the visual appearance of each class. In other words, they first accumulate a training dataset of labeled images, then feed it to the computer in order for it to get familiar with the data.

Convolutional Neural Networks (CNNs) is the most popular neural network model being used for image classification problem. The big idea behind CNNs is that a local understanding of an image is good enough. The practical benefit is that having fewer parameters greatly improves the time it takes to learn as well as reduces the amount of data required to train the model. Instead of a fully connected network of weights from each pixel, a CNN has just enough weights to look at a small patch of the image. It’s like reading a book by using a magnifying glass; eventually, you read the whole page, but you look at only a small patch of the page at any given time.

Consider a 256 x 256 image. CNN can efficiently scan it chunk by chunk — say, a 5 × 5 window. The 5 × 5 window slides along the image (usually left to right, and top to bottom), as shown below. How “quickly” it slides is called its stride length. For example, a stride length of 2 means the 5 × 5 sliding window moves by 2 pixels at a time until it spans the entire image.



A **convolution** is a weighted sum of the pixel values of the image, as the window slides across the whole image. Turns out, this convolution process throughout an image with a weight matrix produces another image (of the same size, depending on the convention). Convolving is the process of applying a convolution.

The sliding-window shenanigans happen in the convolution layer of the neural network. A typical CNN has multiple convolution layers. Each convolutional layer typically generates many alternate convolutions, so the weight matrix is a tensor of 5 × 5 × n, where n is the number of convolutions.

As an example, let’s say an image goes through a convolution layer on a weight matrix of 5 × 5 × 64. It generates 64 convolutions by sliding a 5 × 5 window. Therefore, this model has 5 × 5 × 64 (= 1,600) parameters, which is remarkably fewer parameters than a fully connected network, 256 × 256 (= 65,536).

The beauty of the CNN is that the number of parameters is independent of the size of the original image. You can run the same CNN on a 300 × 300 image, and the number of parameters won’t change in the convolution layer.

## ****Data Augmentation:****

Image classification research datasets are typically very large. Nevertheless, data augmentation is often used in order to improve generalisation properties. Typically, random cropping of rescaled images together with random horizontal ﬂipping and random RGB colour and brightness shifts are used. Different schemes exist for rescaling and cropping the images (i.e. single scale vs. multi scale training). Multi-crop evaluation during test time is also often used, although computationally more expensive and with limited performance improvement. Note that the goal of the random rescaling and cropping is to learn the important features of each object at different scales and positions. Keras does not implement all of these data augmentation techniques out of the box, but they can easily implemented through the preprocessing function of the ImageDataGenerator modules.

**Training the model:**

• Split the original training data into 80% training (48,000 images) and 20% validation optimize the classifier, while keeping the test data to finally evaluate the accuracy of the model on the data it has never seen. This helps to see whether I’m over-fitting on the training data and whether I should lower the learning rate and train for more epochs if validation accuracy is higher than training accuracy or stop over-training if training accuracy shift higher than the validation.

• Train the model for 100 epochs with batch size of 60 and with 60 validation set compiled with categorical\_crossentropy loss function and Adam optimizer.

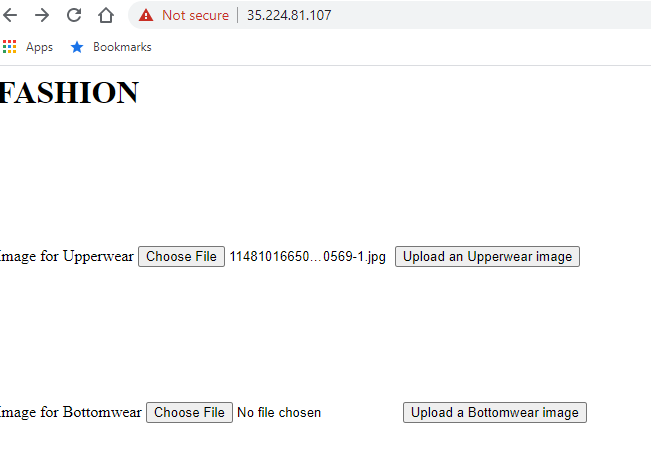
• data augmentation, which generates new training samples by rotating, shifting and zooming on the training samples.

Evaluate the model for each attribute after each attribute trained and try to check if the model accuracy can be increased with the same data.

Refer to the Model Training and testing Notebook for code.

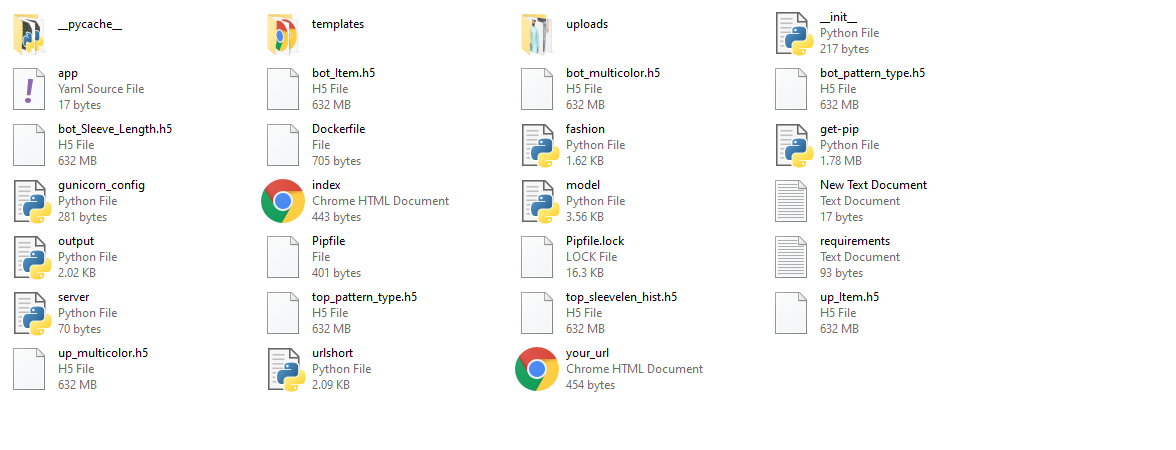
**Creating the UI part:**

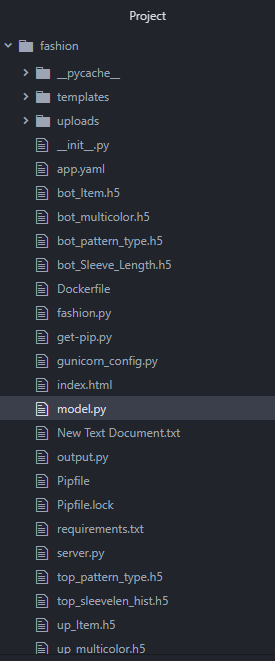
We will be using Flask framework for building a basic user interface to interact with our model. We need to upload and image and derive its attribute. We will be having two forms each for upperwear and bottom wear clothing. Below is the Ui interface for this project.



**Testing the application:**

We can test the application local by running the final python file. Below is the format where how the file is organized.

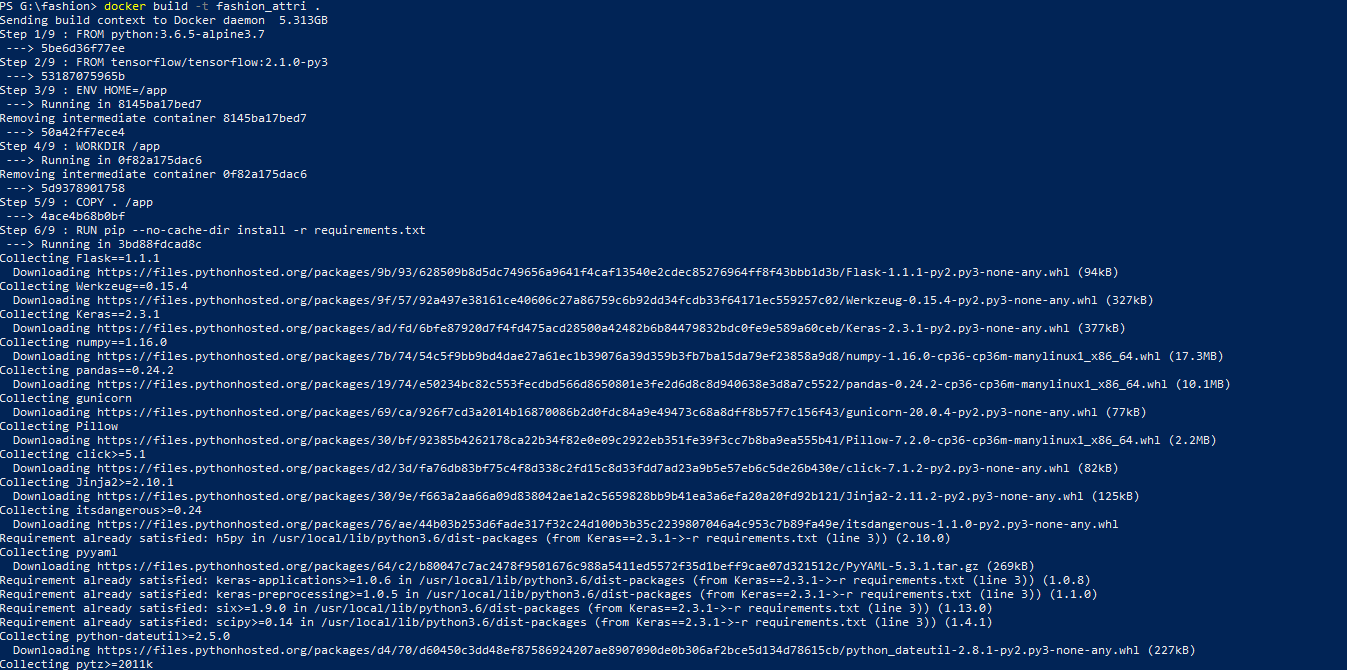




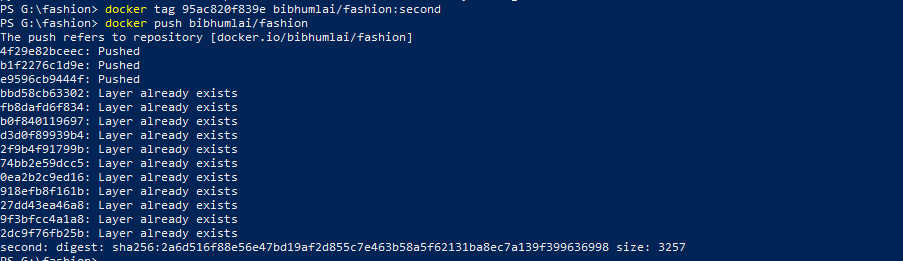
**Creating a docker file:**

Docker is an application that allows you to package your application in a container with all your dependencies so you can run them anywhere. The advantage of using docker in this process is to ensure that our application works the way we want it to work, ensure scalability by starting up new containers as we get more requests, and maintaining fault tolerance by being able to replace faulty containers with new ones in no time.

In the Dockerfile above, we are using tensorflow2.11 as as our base image with alpine python, then we copy the contents of our current directory into the “/app” directory in the image. Then, we set the “/app” directory as our working directory. Meaning, all the commands we run in our image will be run from that location. We go ahead to use the RUN command to install our dependencies and then we use gunicorn to start our app inside a module called fashion.py.



Once Our docker Image is ready. We need to push the docker image to dockerhub repository.

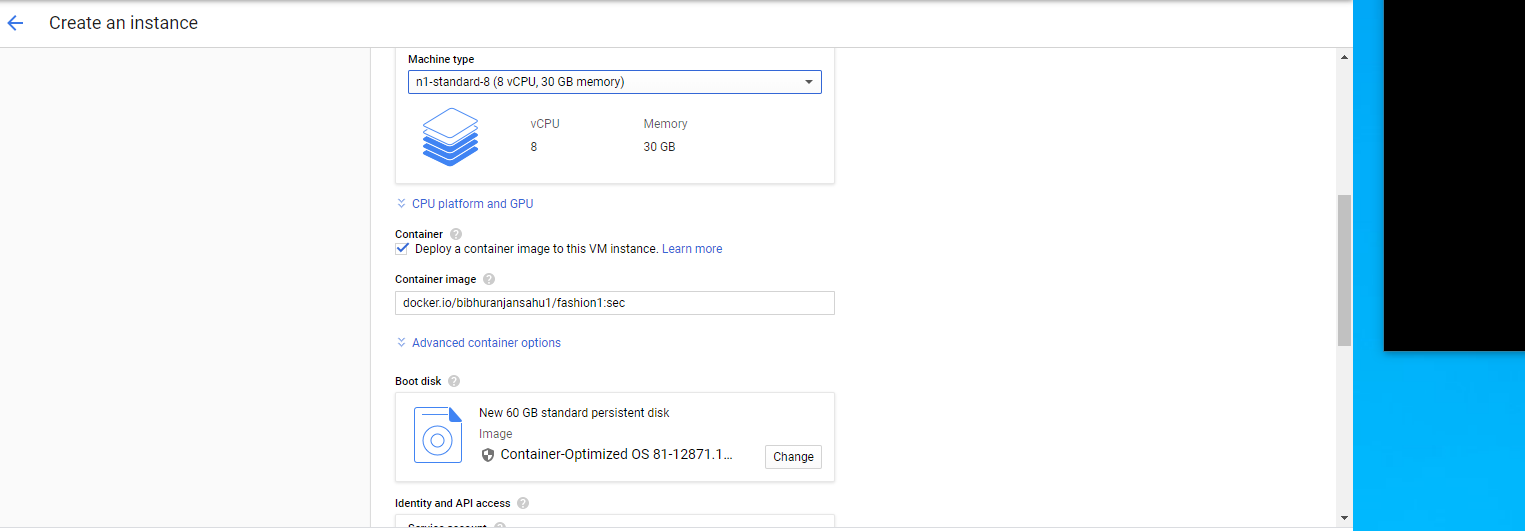


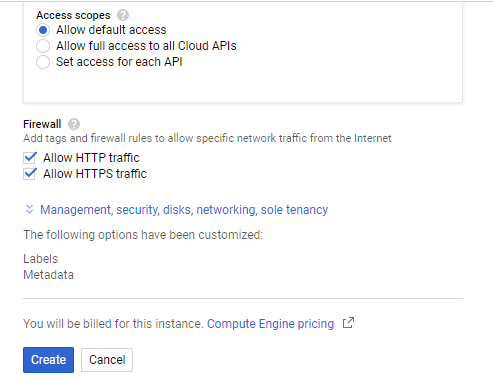
Once our Docker Image file is ready, we run our image by using the following command:



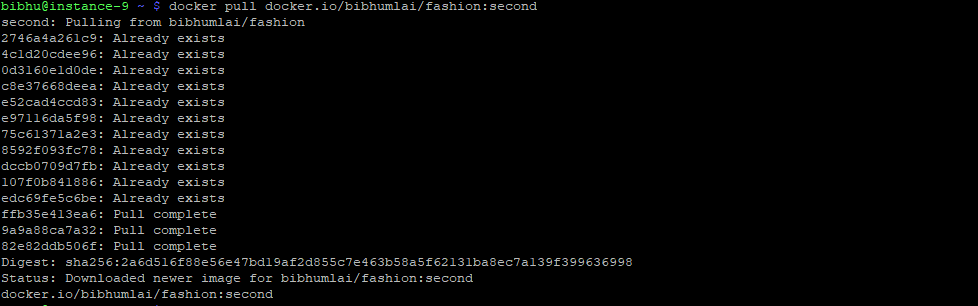
**Deploying project to production:**

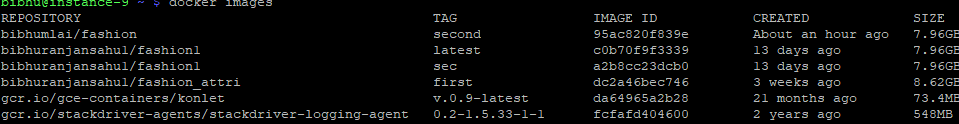
We will be using Google cloud platform compute engine to deploy our docker and host our flask application. We will be using 30 GB RAM memory with 8 CPU and container optimized Chrome OS for our project with 60 GB Memory. We need to make sure we allow all HTTP and HTTPS traffic access. Below are the steps to be followed.





Once compute engine VM is created. Open the GCP SDK terminal and launch the VM. Pull the docker form dockerhub repository.

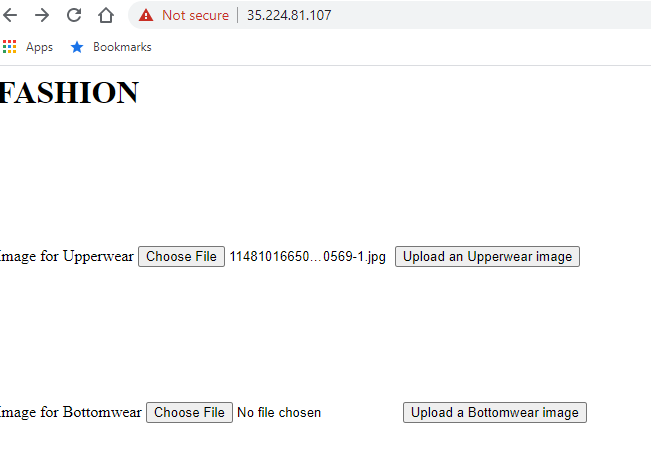




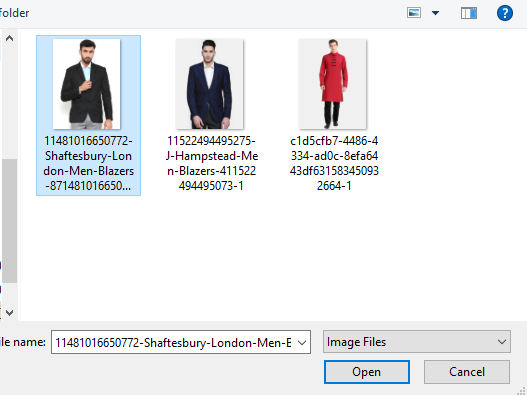
Then Run the docker image.



Open the external exposed url in your browser.



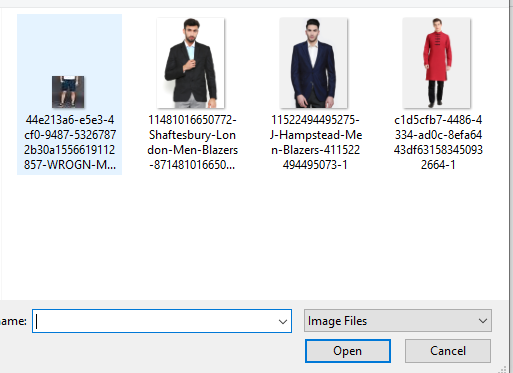
Upload the image for upper wear



Below is the output



Now lets upload an image for buttom wear



Below is the the output

