MSc in Business Analytics

Big Data Content Analitics
Reverse Image Search Engine

Written by

Eirini Anagnou P2821801

Konstantinos Giorgas P2821807

Description

The main idea behind this project was to create an algorithm that given a photo it would be able detect any clothing, if present, classify it to one of the following clothing categories:

Topwear, Shoes, Bags, Bottomwear, Watches, Innerwear, Jewellery, Eyewear and Socks.

Afterwards, it will recommend similar apparels based on image proximity.

In other words, our algorithm is a reverse image search regarding clothing making it more convenient for the users to discover the apparels of their interest based directly on its appearance.

Also, by enabling the user to perform a more detailed search (such as image search) could lead to a higher conversion rate.

This project could be a part of a mobile application or embedded in a site of a clothing brand, in which the user can capture or upload clothing pictures in order to find similar products of this brand. On a larger scale, this application by companies such as Skroutz or Bestprice enabling the users to find similar or even the same product with the lowest price, or in the nearest store.

Real Life Scenarios:

- A user could capture a photo of a shop window, then find similar products through our application.
- A user could take a photo of a well-dressed celebrity and then be able to copy their style.

Mission

Up until now, there was a gap on how the user described a cloth of his liking and the keywords the developer or the marketing specialists attributed to such products. As a result by using the existing filters of a site a user might not be able to find the desired product, thus diminishing the possibility of conversion.

A reverse image search is already used by Google, but such an application is not yet utilized specifically for the fashion industry.

Our scope is to try to bridge the gap between the way a user describes a cloth and the categorization made by the specialist of a certain brand by enabling users to search directly through an image and not by typing keywords which eventually might be off the topic or nonexistent.

Data

Finding the proper data was an easy task as the internet is full of clothing images. The dataset we finally chose can be found here.

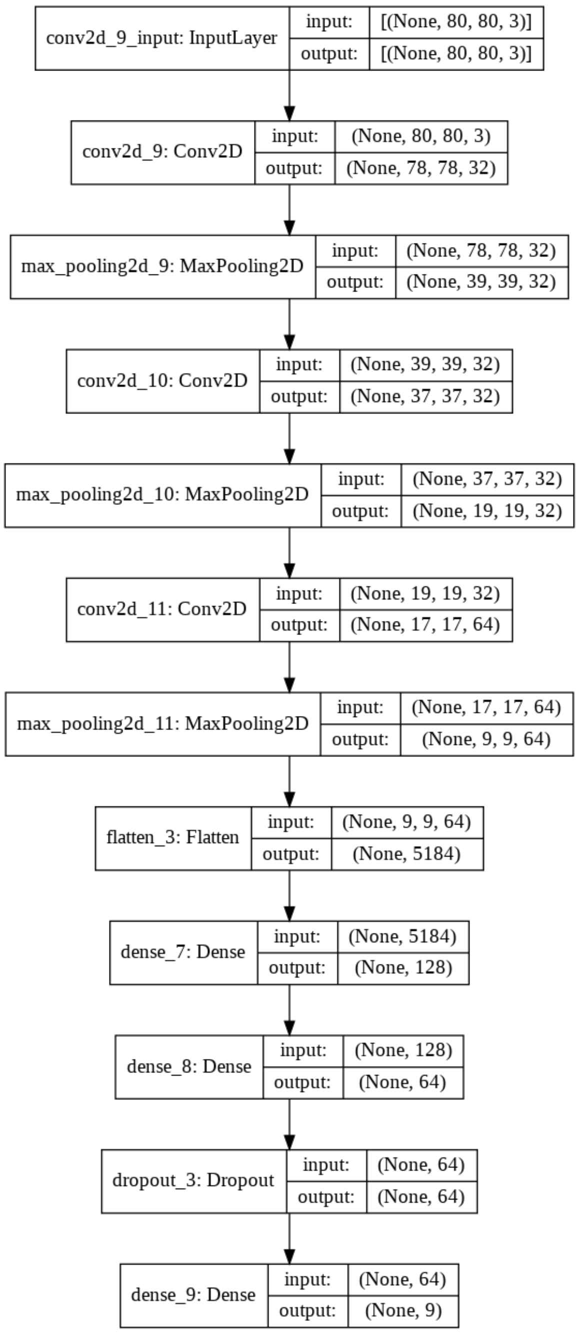
The initial data consisted of a folder with 44441 images and a csv file. Each row of the csv file represents an image and each column a categorization in which every image falls to. For our project we only used the top 9 categories of the subcategory column.

The final dataset consisted of 34931 images which represent the 80% of our initial dataset.

Subcategory	count
Topwear	15402
Shoes	7343
Bags	3055
Bottomwear	2694
Watches	2542
Innerwear	1808
Jewelry	1079
Eyewear	1073
Socks	698

Methodology

Since it is an image classification task, the most appropriate model to use is Convolutional Neural Network (CNN). The architecture of the model can be se en below:



conv2d_9_input: InputLayer | input: [(None, 80, 80, 3)] | output: [(None, 80, 80, 3)]
conv2d_9: Conv2D | input: (None, 80, 80, 3) | output: (None, 78, 78, 32)
max_pooling2d_9: MaxPooling2D | input: (None, 78, 78, 32) | output: (None, 39, 39, 32)
conv2d_10: Conv2D | input: (None, 39, 39, 32) | output: (None, 37, 37, 32)
max_pooling2d_10: MaxPooling2D | input: (None, 37, 37, 32) | output: (None, 19, 19, 32)
conv2d_11: Conv2D | input: (None, 19, 19, 32) | output: (None, 17, 17, 64)
max_pooling2d_11: MaxPooling2D | input: (None, 17, 17, 64) | output: (None, 9, 9, 64)
flatten_3: Flatten | input: (None, 9, 9, 64) | output: (None, 5184)
dense_7: Dense | input: (None, 5184) | output: (None, 128)
dense_8: Dense | input: (None, 128) | output: (None, 64)
dropout_3: Dropout | input: (None, 64) | output: (None, 64)
dense_9: Dense | input: (None, 64) | output: (None, 9)

Basically, our model requires an 80x80 RGB image. The image passes through three filters. The first two filters are convolutional layers of 32 neurons with a relu activation function and the max pooling layers with a size of 2x2.
The third filter consist of a convolutional layer with 64 neurons with the relu activation function and a 2x2 max pooling layer having set the padding='same'.
'Relu' has much faster run time from other functions since it is just max(0,x). Also makes it possible to add as many layers we like since its derivative is 0 or 1 so by multiplying the gradients will neither vanish nor explode making our model hard to train (as it would have happened with a sigmoid since the product of many smaller than 1 values goes to 0 quickly).
We have set padding='same'. The main reason behind it is that after augmenting our data, a lot of images were zoomed so we do not want to miss any behavior occurring at the edges of our image. We decided not to use stride at all since it would diminish the size of our already small images.

We then proceeded on flattening the output of the third filter, passing it to a dense layer of 128 neuros and a relu activation function. Afterwards we pass it through a dense layer of 64 neurons whose output will serve as a 'feature' array of each image to use it in our recommendation script.

In order to reduce the chance of some neurons to get over activated and our model overfitting the training data we added a drop-out layer which drops out neurons with a 50% possibility. Although adding such layer at later stages of a model may not be a good idea since the model does not have the time to make the required corrections, it seems to have worked well in our case. The output layer of our model consists of 9 neurons representing each one of our 9 clothing categories.

The main idea behind building our CNN model was to use layers with less neurons at the beginning to detect low level features, which will be combined to form more complex shapes in later stages as we are utilizing more complex layers with more neurons and provide more separation between the classes.
By inspecting our dataset one can understand why such a simple model was able to work so effectively. The classification we wanted to achieve was between shoes and tshirts or jewelry and bottom wear for example, which can be distinguished by merely using their outer shape. We cannot be sure of how the model classifies the apparels and we do not claim the previous to be the case but it is a plausible explanation. Also the fact that no picture contained any background noise may have contributed to the case.

All this lead to a model of 701.161 trainable parameters, training accuracy of 88,03%, training loss of 48,36%, validation loss of 26,04% and validation accuracy of 92,61%.

Regarding the hyper-parameters of our model:

- Apart from the input and output layers, it contains 10 hidden ones.
- After many trials we set the drop-out to 50%.
- In the output layers we used the softmax activation function since we are making multi-class predictions.
- We set a learning rate of 0,001. Although we made the learning process slower, our model converged smoothly and the magnitude of converging to a sub optimal point would be less.
- The number of epochs used was 50. This is explained below.
- We set the batch size, due to system limitations, to 500 images to pass simultaneously through our model.

We used adam optimizer instead of classical stochastic gradient descent, in order to improve the training speed.

Since it is a multi-class classification task, we used a categorical crossentropy and our targets are one-hot-encoded, having as evaluation metric the validation accuracy.

To ensure that our model would stop the training process early and not proceed on overfitting if no actual improvement on accuracy was made through the epochs, we set a call back. What we basically established was that if for 12 epochs, the chance in validation loss was 0, the training process would stop.

As we stated before the goal of this assignment was not to just create a model that would classify images but to create an algorithm that would recommend similar apparels to the input picture. To achieve that we requested two outputs of our model. One output would be the class - category of the cloth (deriving from the output layer) serving as a filter and the other one would be a 64 number array "describing" the apparel, deriving from the final hidden layer before the output layer.
To successfully orchestrate our intentions we created a Mongo Cluster using Mongo Atlas and we made it publicly accessible. Inside this cluster we created a collection were one document per apparel was stored. More precisely, each document had the following format:

{

_id: <An object id automatically allocated by mongo>,

Category: <The category id of 1 out of the 9 possible categories>,

id: <The id of the image>,

vector:[<The array of 64 numbers describing the apparel]

}

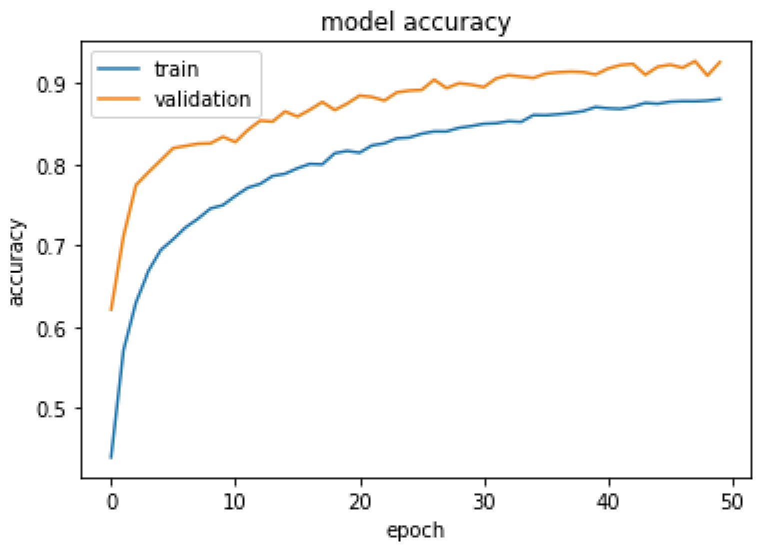
We first passed all the pictures of our set through our model and stored the previous fields in mongo. The mongo created automatically an index to the Category field so we do not needed to create on to make our queries faster.

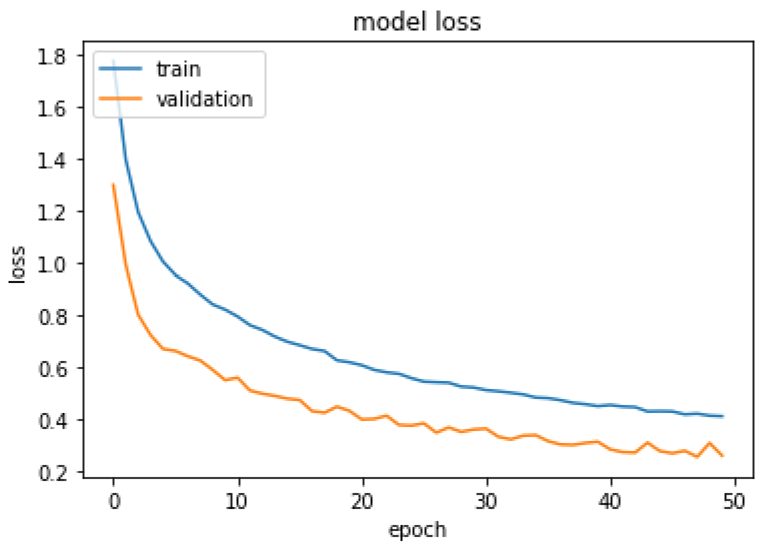
The recommendation process goes as follows:

1. The user passes an image through our algorithm.
2. Based on the last hidden and the output layer the model generates an array of 64 length and based on the output layer the model predicts the category of the apparel.
3. The algorithm queries mongo, fetching all arrays from said category with their ids.
4. Then utilizing Nearest Neighbors algorithm on the vectors the ids of the 10 most similar apparels are returned.
5. The ids are pasted with the image repository path and returned to the user as output.

Results

As you can see from both the model accuracy and loss distributions, as we proceed through the epochs, the increasing accuracy (or the decreasing loss) changes at a slower pace. During the final epochs the curve is almost parallel to the horizontal axis which means that the training process stopped at the optimal time. The train and validation curves follow a similar pattern and there is not a big gap between them which means that our model did not overfit on our data.





Problems:

1. The main problem we encountered was that our laptop's GPUs were not enough for the training of our model.For this reason we used Google Colab's resources, with GPU runtime serving as accelerator.
2. At first we created a very complicated model with multiple layers whose training far exceeded the ten hour window of training time provided on Google Colab. We could set a call back and save our model at an earlier training state (before Google Colab run time disconnects) and reload our model to proceed the training . Instead, we used a much model which proved to be equally accurate to our first, most complicated one.
3. Initially we used a batch size of 100 which proved to be too much for Google Colab to handle as it was constantly throwing errors. We used a smaller batch size to be able to complete the training.
4. At the beginning, we used a layer of 128 neurons to form the feature vector of each image. Performing the nearest neighbor algorithm on such vectors proved to be too slow, so we finally used a layer with 64 neurons to produce the image vectors.

Time:

Below you can the documented time of the training history.

```
model_88
Epoch 1/50
100/100 [==============================] - 129s 1s/step - loss: 1.7765 -
acc: 0.4395 - val_loss: 1.2993 - val_acc: 0.6217
Epoch 2/50
```

```
100/100 [==============================] - 117s 1s/step - loss: 1.3972 -
acc: 0.5714 - val_loss: 0.9932 - val_acc: 0.7120
Epoch 3/50
100/100 [==============================] - 117s 1s/step - loss: 1.1948 -
acc: 0.6302 - val_loss: 0.8023 - val_acc: 0.7745
Epoch 4/50
100/100 [==============================] - 119s 1s/step - loss: 1.0832 -
acc: 0.6685 - val_loss: 0.7239 - val_acc: 0.7903
Epoch 5/50
100/100 [==============================] - 107s 1s/step - loss: 1.0046 -
acc: 0.6944 - val_loss: 0.6704 - val_acc: 0.8051
Epoch 6/50
100/100 [==============================] - 125s 1s/step - loss: 0.9534 -
acc: 0.7077 - val_loss: 0.6627 - val_acc: 0.8201
Epoch 7/50
100/100 [==============================] - 116s 1s/step - loss: 0.9223 -
acc: 0.7224 - val_loss: 0.6423 - val_acc: 0.8228
Epoch 8/50
100/100 [==============================] - 116s 1s/step - loss: 0.8794 -
acc: 0.7330 - val_loss: 0.6259 - val_acc: 0.8255
Epoch 9/50
100/100 [==============================] - 115s 1s/step - loss: 0.8412 -
acc: 0.7456 - val_loss: 0.5899 - val_acc: 0.8260
Epoch 10/50
100/100 [==============================] - 114s 1s/step - loss: 0.8217 -
acc: 0.7497 - val_loss: 0.5504 - val_acc: 0.8340
Epoch 11/50
100/100 [==============================] - 117s 1s/step - loss: 0.7955 -
acc: 0.7610 - val_loss: 0.5600 - val_acc: 0.8276
Epoch 12/50
100/100 [==============================] - 117s 1s/step - loss: 0.7619 -
acc: 0.7712 - val_loss: 0.5104 - val_acc: 0.8424
Epoch 13/50
100/100 [==============================] - 117s 1s/step - loss: 0.7434 -
acc: 0.7759 - val_loss: 0.4982 - val_acc: 0.8539
Epoch 14/50
100/100 [==============================] - 119s 1s/step - loss: 0.7181 -
acc: 0.7857 - val_loss: 0.4898 - val_acc: 0.8528
Epoch 15/50
100/100 [==============================] - 116s 1s/step - loss: 0.6982 -
acc: 0.7884 - val_loss: 0.4792 - val_acc: 0.8653
Epoch 16/50
100/100 [==============================] - 116s 1s/step - loss: 0.6848 -
acc: 0.7952 - val_loss: 0.4740 - val_acc: 0.8590
```

```
Epoch 17/50
100/100 [==============================] - 118s 1s/step - loss: 0.6702 -
acc: 0.8005 - val_loss: 0.4312 - val_acc: 0.8674
Epoch 18/50
100/100 [==============================] - 117s 1s/step - loss: 0.6623 -
acc: 0.8000 - val_loss: 0.4250 - val_acc: 0.8770
Epoch 19/50
100/100 [==============================] - 116s 1s/step - loss: 0.6268 -
acc: 0.8137 - val_loss: 0.4484 - val_acc: 0.8670
Epoch 20/50
100/100 [==============================] - 118s 1s/step - loss: 0.6193 -
acc: 0.8166 - val_loss: 0.4325 - val_acc: 0.8749
Epoch 21/50
100/100 [==============================] - 117s 1s/step - loss: 0.6077 -
acc: 0.8144 - val_loss: 0.3993 - val_acc: 0.8849
Epoch 22/50
100/100 [==============================] - 118s 1s/step - loss: 0.5899 -
acc: 0.8234 - val_loss: 0.4011 - val_acc: 0.8832
Epoch 23/50
100/100 [==============================] - 116s 1s/step - loss: 0.5803 -
acc: 0.8259 - val_loss: 0.4137 - val_acc: 0.8787
Epoch 24/50
100/100 [==============================] - 117s 1s/step - loss: 0.5745 -
acc: 0.8321 - val_loss: 0.3779 - val_acc: 0.8888
Epoch 25/50
100/100 [==============================] - 117s 1s/step - loss: 0.5579 -
acc: 0.8331 - val_loss: 0.3757 - val_acc: 0.8910
Epoch 26/50
100/100 [==============================] - 112s 1s/step - loss: 0.5446 -
acc: 0.8378 - val_loss: 0.3842 - val_acc: 0.8918
Epoch 27/50
100/100 [==============================] - 125s 1s/step - loss: 0.5423 -
acc: 0.8406 - val_loss: 0.3484 - val_acc: 0.9045
Epoch 28/50
100/100 [==============================] - 117s 1s/step - loss: 0.5405 -
acc: 0.8406 - val_loss: 0.3685 - val_acc: 0.8940
Epoch 29/50
100/100 [==============================] - 113s 1s/step - loss: 0.5257 -
acc: 0.8450 - val_loss: 0.3515 - val_acc: 0.9000
Epoch 30/50
100/100 [==============================] - 118s 1s/step - loss: 0.5221 -
acc: 0.8472 - val_loss: 0.3607 - val_acc: 0.8983
Epoch 31/50
```

```
100/100 [==============================] - 115s 1s/step - loss: 0.5119 -
acc: 0.8501 - val_loss: 0.3638 - val_acc: 0.8953
Epoch 32/50
100/100 [==============================] - 119s 1s/step - loss: 0.5074 -
acc: 0.8507 - val_loss: 0.3325 - val_acc: 0.9060
Epoch 33/50
100/100 [==============================] - 117s 1s/step - loss: 0.5018 -
acc: 0.8532 - val_loss: 0.3226 - val_acc: 0.9100
Epoch 34/50
100/100 [==============================] - 116s 1s/step - loss: 0.4952 -
acc: 0.8523 - val_loss: 0.3369 - val_acc: 0.9083
Epoch 35/50
100/100 [==============================] - 117s 1s/step - loss: 0.4836 -
acc: 0.8608 - val_loss: 0.3389 - val_acc: 0.9067
Epoch 36/50
100/100 [==============================] - 118s 1s/step - loss: 0.4813 -
acc: 0.8605 - val_loss: 0.3156 - val_acc: 0.9120
Epoch 37/50
100/100 [==============================] - 120s 1s/step - loss: 0.4734 -
acc: 0.8618 - val_loss: 0.3030 - val_acc: 0.9136
Epoch 38/50
100/100 [==============================] - 120s 1s/step - loss: 0.4629 -
acc: 0.8634 - val_loss: 0.3011 - val_acc: 0.9144
Epoch 39/50
100/100 [==============================] - 120s 1s/step - loss: 0.4579 -
acc: 0.8657 - val_loss: 0.3087 - val_acc: 0.9138
Epoch 40/50
100/100 [==============================] - 121s 1s/step - loss: 0.4504 -
acc: 0.8707 - val_loss: 0.3134 - val_acc: 0.9108
Epoch 41/50
100/100 [==============================] - 121s 1s/step - loss: 0.4546 -
acc: 0.8691 - val_loss: 0.2840 - val_acc: 0.9182
Epoch 42/50
100/100 [==============================] - 121s 1s/step - loss: 0.4488 -
acc: 0.8685 - val_loss: 0.2735 - val_acc: 0.9225
Epoch 43/50
100/100 [==============================] - 121s 1s/step - loss: 0.4471 -
acc: 0.8711 - val_loss: 0.2717 - val_acc: 0.9239
Epoch 44/50
100/100 [==============================] - 121s 1s/step - loss: 0.4301 -
acc: 0.8758 - val_loss: 0.3099 - val_acc: 0.9105
Epoch 45/50
100/100 [==============================] - 121s 1s/step - loss: 0.4309 -
acc: 0.8747 - val_loss: 0.2774 - val_acc: 0.9204
```

```
Epoch 46/50
100/100 [==============================] - 121s 1s/step - loss: 0.4299 -
acc: 0.8772 - val_loss: 0.2690 - val_acc: 0.9229
Epoch 47/50
100/100 [==============================] - 117s 1s/step - loss: 0.4184 -
acc: 0.8779 - val_loss: 0.2787 - val_acc: 0.9192
Epoch 48/50
100/100 [==============================] - 126s 1s/step - loss: 0.4215 -
acc: 0.8778 - val_loss: 0.2554 - val_acc: 0.9274
Epoch 49/50
100/100 [==============================] - 120s 1s/step - loss: 0.4137 -
acc: 0.8786 - val_loss: 0.3086 - val_acc: 0.9095
Epoch 50/50
100/100 [==============================] - 120s 1s/step - loss: 0.4117 -
acc: 0.8803 - val_loss: 0.2604 - val_acc: 0.9261
```

Total Training Time : 5908 seconds (1,64 hours).

6.Roles

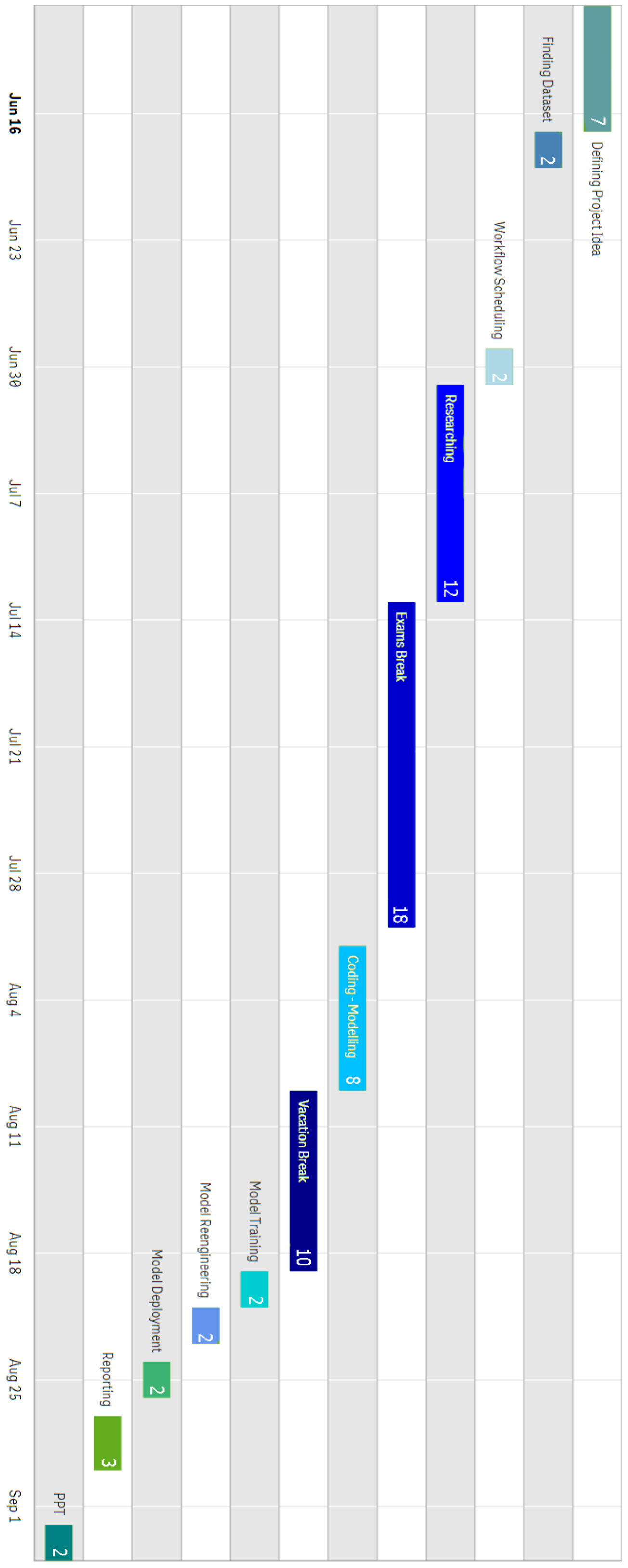
Our team consists of 2 members:

Members	Role	Studies	Responsibilities
Eirini Anagnou	Business Intelligence Engineer	Mechanical Engineer | NTUA	1. Finding the dataset. 2. Initiating a Mongodb cluster and created the process and schema in which the image features would be saved. 3. Creating the image recommendation flow using nearest neighbors algorithm. 4. Writing the final report of the assignment.
Konstantinos Giorgas	Machine Learning Engineer	Department of Management Science and Technology | AUEB	1. Cleaning the data and preparing it for training process, organizing it in folders. 2. Training a Convolutional Neural Network model. 3. Deploying the CNN model in flask. 4. Scripting the web application that would recommend images. 5. Writing the final report of the assignment..

Bibliography

- https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d
- https://blog.griddynamics.com/reverse-image-search-with-convolutional-neural-networks/
- https://medium.com/intuitive-deep-learning/build-your-first-convolutional-neural-network-to-recognize-images-84b9c78fe0ce
- https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-satellite-photos-of-the-amazon-rainforest/
- https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/
- Σημειώσεις του μαθήματος Big Data Content Analytics

Time Plan


Defining Project Idea 7
Finding Dataset 2
Workflow Scheduling 2
Researching 12
Exams Break 18
Coding - Modelling 8
Vacation Break 10
Model Training 2
Model Reengineering 2
Model Deployment 2
Reporting 3
PPT 2
Jun 16
Jun 23
Jun 30
Jul 7
Jul 14
Jul 21
Jul 28
Aug 4
Aug 11
Aug 18
Aug 25
Sep 1

Contact Person

Contact Person: Konstantinos Giorgas
Telephone: 6944596721
Email: konstantinosgiorgas@gmail.com

Comments / Problems

Comments:

- At first, the processing power of our laptops was not enough to undergo the training of our model. To deal with this issue we used resources found on Google Colab.
- Secondly, the initial dataset had a size of 12 GB meaning that we were unable to upload it on Google Drive and process it with Google Colab.
- Luckily, we didn't have to do any resizing ourselves since we found the same dataset on a smaller version of 280 MB.
- Inspired by the VGG16 model, we tried to create something similar with too many convolutional layers but Google Colab's RAM was not enough to support our model's training.

Improvements:

- For logged-in users, our application would combine the given image with previous browsing and purchased data and recommend apparels more similar to the user's taste.
- We could implement style transfer capabilities to our algorithm so it can recommend apparels of other categories matching the same style of the uploaded apparels (e.g match this dress with that jewelry).
- We could also implement reinforced capabilities to our model so it would improve by whether the user clicked or not in our recommended images.
- Although our model did not overfit and indeed was able to classify images from the same image repository that was not used in the training process, it performed poorly on images with background different than a white one. In a sense, the model "overfitted" to the distribution of the image files we provided. So with more appropriate training data (and maybe a few more extra layers), our algorithm could learn to classify apparels deriving from various environment.