Project 15: Visual Search using Deep Embeddings

1st Matt Clifford
Dept. of Computer Science
University of Bristol
Bristol, UK
matt.clifford@bristol.ac.uk

2nd Daniel Collins
Dept. of Computer Science
University of Bristol
Bristol, UK
daniel.collins@bristol.ac.uk

3rd Jonathan Erskine
Dept. of Computer Science
University of Bristol
Bristol, UK
jonathan.erskine@bristol.ac.uk

4th Xuyang Fang
Dept. of Computer Science
University of Bristol
Bristol, UK
xf16910@bristol.ac.uk

***Abstract*—In this paper we create a data pipeline that uses the Snap Vision dataset to investigate deep embeddings as a means of correctly identifying similar images. We implemented a simple statistical model as a baseline embedding model as well as two simple neural networks and the inception implementation of FaceNet [3]. Our smallest network performs the best, likely due to not overfitting and limits on our training program. We discuss the results of our embeddings qualitively and quantively, with visual inspection of embeddings provided via t-sne. We discuss potential future improvements, and explore the concept of manually generated image embeddings, with promising results.**

***Index Terms*—deep embeddings, triplet loss, visual similarity**

I. Introduction

In 2020, more than half of the UK population purchased clothing online. In January 2022, 28% of total UK retail sales of textile products, clothes and shoes were made online. This figure peaked at roughly 60% in February 2021, coinciding with the 3rd national Covid-19 lockdown. [1]

Snap Vision, a visual search technology company, aims to "make the online shopping experience as satisfying as shopping in the real world" by developing a platform through which customers can discover and purchase suggested items. These suggestions are based on the item(s) they are currently viewing, and the user can provide feedback on the quality of these suggestions.

Fundamentally, suggesting similar images to a user requires a target image to be processed and some embedded detail extracted from the image; this is typically a vector which attempts to capture semantic-level information of the image, with the nature of information dependent on the embedding technique used. Embedding techniques can be simple, such as extracting image size to group images of a similar resolution or calculating mean pixel value to group by brightness or colour scheme, but they can also be more complex, such as those learnt using deep learning techniques trained on similar images. Regardless of embedding process, similar images can consequently be retrieved by comparing two or more image embeddings using a similarity measure, such as cosine distance.

An e-commerce fashion web page will typically display several images of an item, varying aspects such as perspective (field of view, distance) or the pose of the model wearing the item, or emphasising features such as logos, patterns or functional features of interest to the customer; Figure 1 provides some examples. This can result in the generation of embeddings which group objects by peripheral features such as the object perspective or by the pose of the model, instead of by object similarity. The objective of the Snap Vision platform is to generate embeddings of items of clothing which disregard these peripheral features, therefore enabling multiple images of the same item of clothing to be correctly clustered together.

Beyond comparing visually similar images, Snap Vision aims to enable users to find items which compliment a target item (i.e. shoes that would "go with" a particular dress), or simply to find other items of the same clothing type. Such affiliations are dependent on rich, descriptive embeddings which capture contextual information about the image such as clothing type, brand or styles. These embeddings can aid visual search in other domains (e.g. facial recognition) which are susceptible to many of the same problems.

This project aims to investigate "deep-embedding" techniques, which have shown promise when maintaining similarity across multiple views of a target object. From a review of current methods [2] [3], we use FaceNet for many of our model and training algorithms inspirations. We investigate multiple embedding techniques by developing a robust, modular pipeline which can automatically create and evaluate image embeddings.

To aid in pipeline development and establish a baseline for model performance with which to compare deep embedding techniques, we first implemented a simple embedding model. The pipeline is designed for ease of use, and as such was developed to be deployed within minimal set-up and modification. We attempted to achieve flexibility through a modular design which allows users to "plug-and-play" with new models, training methods or evaluation techniques without having to modify the project pipeline.

II. Data

Snap Vision provides a dataset comprising of 6411 images of 1500 individual items of clothing. The dataset is structured into separate folders of images for each item of clothing; each folder contains between 2-5 different images of the item, with a mean of 4.274 images and mode of 5 images per item.

Figure 1 illustrates two image sets; a typical set contains at least one image of (1) the item of clothing in isolation, (2) the item being worn by a model, and (3) a close-up showing the texture or a key-feature of an item in greater detail. There is a divergence in the dataset where images of items of clothing in

isolation typically feature a white background, whereas images containing a model feature a background with visible shadows and corners. This is an example of a peripheral image feature which should be ignored when generating embedding.

Fig. 1: Typical example image sets for an item of clothing, showing the item in isolation (left), the item worn by a model from different angles (middle three images), and a close up showing the pattern, texture or key detailing of the item (right).

Exploring the dataset revealed several cases where image embedding models would likely struggle to correctly capture relevant information about the target object. A folder may contain several images of a model wearing the target item, as well as another visually prominent item of clothing. In some cases the target item is largely covered up by another item of clothing, such as the jacket and jeans in Figure 2 (Top). Furthermore, it was assumed that the non-target items of clothing worn by models (e.g. the jacket and jeans) are not items which have their own folder within the data-set. This challenge was not addressed in our investigation, and presents an opportunity for further investigation. For the purposes of the Snap Vision application, the function of the visual search algorithm would be to find associated items of clothing for a user-selected reference image, and we would assume that the user would select an image within which the desired target item was clearly visible.

Figure 3 highlights cases where small accessories are likely to be associated only when viewed in isolation due to the significant difference in background visual information, and smaller amounts of salient visual information. For example, images of different types of sunglasses are all visually similar. It is likely to be challenging to create a visual search model capable of distinguishing between sets of very similar coloured small objects. Figure 4 illustrates further challenging cases. Sets are observed to contain images which do not feature the target item, contain different variants of an item, or show an item contained within a case which bears very little or no visual similarity with the product, as well as items that are seen in a different context (such as a watch face with and without a cover). We do not investigate the statistics regarding these undesirable images, and it might be that they make up a statistically small amount of our training data. If they are significant in number then this may introduce a level of aleatoric uncertainty into our results.

Fig. 2: (Top) Image set for a dark shirt. One photograph (second from the left) shows the target item obscured by a light coloured jacket. (Bottom) Image set for a patterned skirt, second image from left shows model wearing a lightly colored denim jacket. A visual search on this image would ideally return the jacket or the skirt, however a naive model would likely return other images of models based on the total visual information in the image.

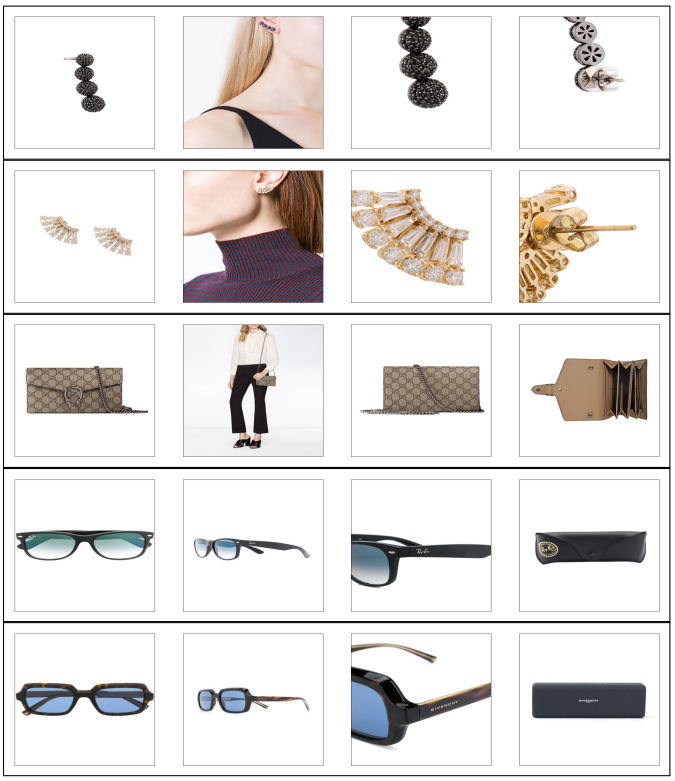

Fig. 3: Small accessories make up a very small amount of the total visual information when worn by a human model (second image from the left). Images of sunglasses are presented similarly, with very similar colours and features.

III. Related Work

There is recent state of the art work into visual representation learning in an unsupervised setting such as [7], as well as in a self-supervised setting such as [8] and [9].

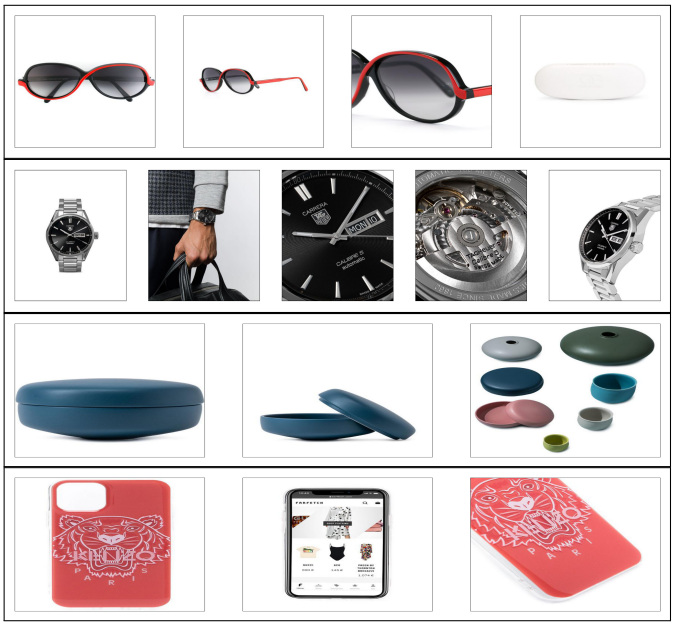

Fig. 4: (Top) The rightmost image shows a case, which cannot be visually associated with the item. (Upper Middle) The watch is shown with and without the face-plate, changing the appearance significantly. (Lower Middle) Multiple colour variants of the item are shown in the rightmost image. (Bottom) The central image does not show the target item, and contains within it images of other items of clothing.

These constrastive learning techniques of embeddings involve augmenting the image data via cropping or occlusions to learn meaningful embeddings. Learning is taken place by a triplet loss [6] of some kind. Where for a given embedding of an image (anchor) during training, the loss function is also given the embedding of a 'similar' image (positive) and of a 'dissimilar' image (negative). The overall objective is to penalise positives from being far away from anchor and also penalise negative from being close to the anchor. This concept is explained in more detail in section IV-C.

For our data we do not need to augment images to obtain 'similar' images since we have 'similar' images already provided and labeled as the ones that contain the same specific item of clothing. This is a similar scenario to FaceNet [3], where they use deep convolutional network to create unified embeddings for face recognition and clustering. Their problem scenario is similar to ours, but with mutliple views of faces rather than items of clothing. As such, we take a lot of inspiration from their methods.

A 2019 paper [4] addresses some of the challenges associated with modeling fashion product compatibility from real world images, i.e. images containing background information, such as scenes, other items of clothing, and being worn by a human as opposed to images showing only the item of clothing. In real-world images, complexity, visual clutter, variations in lighting and pose complicate the problem of modeling compatibility for product recommendation. In the paper, a set of four real-world images were given as examples for qualitative analysis, each showing people wearing different items of clothing, or rooms containing different items of furniture. The example images were displayed alongside images of the top three most and least compatible products as given by the model. The authors proceed to use this figure as a basis for discussing whether the recommended products could be considered reasonably compatible for the example scenes, and discuss factors influencing this such as the color of the products, the design/style (modern, minimalist), and differences from the least compatible items. This method of qualitative visual evaluation is useful for this type of problem, since "compatibility" in this context is somewhat subjective and difficult to define. We do not attempt to implement this functionality in the pipeline, but we recognise this work as an essential 'next step' for assessing whether clothes are compatible as far as styling and aesthetics.

IV. Models

A. Simple Model

As a baseline performance to compare more complicated model against we first created a simple model. This model created 6 embeddings: The mode and mean value of each colour channel of the image (red, green, blue).

B. Neural Networks

We created three neural network models to create 'deep' embeddings. First we created two simple 'vanilla' convolutional neural networks, one inputting small image sizes and the other large input image sizes. Vanilla neural network architectures are shown in tables I and II.

We also implemented the inception [5] version of the FaceNet which is depicted in Table 2 of their paper [3].

All neural network models are implemented in the popular pytorch library since it provides easy access to efficient C++ and Cuda programs for running networks as well as automatic gradient calculation, back propagation and optimisation algorithms used for training.

Layer	size-in	size-out	kernel	function after
Conv1	256x256x3	252x252x32	5x5x3, 1	PReLU
Pool1	252x252x32	126x126x32	2x2, 2	Dropout(p=0.3)
Conv2	126x126x32	122x122x64	5x5x32, 1	PReLU
Pool2	122x122x64	61x61x64	2x2, 2	Dropout(p=0.3)
Reshape	61x61x64	238144x1		
fc1	238144x1	512x1		PReLU
fc2	512x1	64x1		

TABLE I: Small Vanilla Convolution Neural Network Architecture. The input and output sizes are described as rows x cols x no. filters. The kernel is specified as rows x cols, stride.

Layer	size-in	size-out	kernel	function after
Conv1	512x512x3	506x506x16	7x7x3, 1	PReLU
Pool1	506x506x16	253x253x16	4x4, 2	Dropout(p=0.3)
Conv2	253x253x16	246x246x32	5x5x16, 1	PReLU
Pool2	246x246x32	123x123x32	2x2, 2	Dropout(p=0.3)
Conv3	123x123x32	118x118x64	5x5x32, 1	PReLU
Pool3	118x118x64	59x59x64	2x2, 2	Dropout(p=0.3)
Conv4	59x59x64	56x56x128	5x5x64, 1	PReLU
Pool4	56x56x128	28x28x128	2x2, 2	Dropout(p=0.3)
Reshape	28x28x128	100352x1		
fc1	100352x1	256x1		PReLU
fc2	256x1	128x1		

TABLE II: Bigger Vanilla Convolution Neural Network Architecture. The input and output sizes are described as rows x cols x no. filters. The kernel is specified as rows x cols, stride.

C. Neural Network Training

To train neural network models we use a triplet loss function as the training signal [6]. To calculate the triplet loss, first the embeddings of: the image of interest $\mathbf{e_a}$ (anchor), similar image $\mathbf{e_p}$ (positive) and not similar image $\mathbf{e_n}$ (negative) must be obtained from the same network. We want the embedding of the positive example to be closer to the anchor than the embedding of the negative example. Which we use the Euclidean distance metric

$$d(\mathbf{e_0}, \mathbf{e_1}) = \|\mathbf{e_0} - \mathbf{e_1}\|_2^2 \tag{1}$$

to formulate our objective:

$$d(\mathbf{e_a}, \mathbf{e_p}) < d(\mathbf{e_a}, \mathbf{e_n}). \tag{2}$$

Subject to

$$\forall \mathbf{e} \in \mathbf{E} : \mathbf{E} = f(\mathbf{T}). \tag{3}$$

Where $\mathbf{T}$ is the training set of images and f is the neural network embedding function. From this we can formulate our loss

$$L(\mathbf{e_a}, \mathbf{e_p}, \mathbf{e_n}) = max\left[d(\mathbf{e_a}, \mathbf{e_p}) - d(\mathbf{e_a}, \mathbf{e_n}) + m, 0\right] \tag{4}$$

where m is the margin, which is the minimum distance that the negative must be further away than positive. For our training we fix $m = 1$.

Positive examples are randomly picked. To select negative examples, we use both randomly selected negative examples as well as picking negative where

$$d(\mathbf{e_a}, \mathbf{e_p}) > d(\mathbf{e_a}, \mathbf{e_n}). \tag{5}$$

To determine distances we precompute all the embeddings at the start of each epoch offline. A randomly selected negative example which satisfies equation 5 is then chosen.

We use the ADAM optimiser [14] to update weights, using a learning rate of 0.0001 which was found to yield good training via a grid search of learning rates. We also use a learning rate decay of 0.98 applied after every epoch to focus on refinement updates after a good area of the parameter space is initially found.

During training, images are randomly cropped to help with robustness of the model by preventing over-fitting to training set.

Models are saved after every 20 epochs and if training is stopped it can be picked up from a saved model checkpoint to avoid unnecessary retraining. Saved models are also used for evaluation and inference purposes.

With a batch size of 64, the small vanilla network uses 11.089GB, FaceNet uses 11.439GB and the bigger vanilla network used 20.479GB of GPU VRAM.

V. Pipeline

We created a robust, modular pipeline which facilitates easy training and evaluation of models. An overview of the pipeline can be viewed at our GitHub repository [1]. All parts of the pipeline are designed to run on any compute system, but it is recommended that neural network training is performed using an Nvidia GPU. To ensure proper and efficient installation of Python dependencies, all external python packages were added to a requirements.txt file.

A. Dataset Acquisition

To easily get the pipeline up and running from any computer, the automatic download and unzipping of the snap vision dataset is required. If the dataset is not found on the local system it is downloaded using wget [2]. Wget is chosen over other popular python libraries such as urllib since it comes built in with a download progress bar which is essential when downloading large files such a datasets. The dataset is stored on google drive in a zip format which is unzipped using the zipfile standard python package.

The images in the dataset are stored in reasonably high resolution. As most of our models require that this resolution be reduced, we also create a lower resolution version of the dataset where all the images are halved in size. While storing a separate dataset necessitates larger project storage, this is deemed acceptable due to the significant reduction in the computational effort when downsizing an image every time that is it loaded; this is especially useful when training neural networks for multiple epochs.

B. Wrangling

To be able to easily access and manipulate the Snap Vision dataset we created a database of metadata about each image. The metadata is stored in a tabular comma-separated value (csv) format for easy read and write access from the pandas python library. The csv format also benefits from being easily and quickly human-readable, which supports developing and debugging.

The metadata about each image that we store is: (1) its absolute file path on the system, (2) the unique clothing item number and (3) the file paths of all images that contain the same item of clothing which are denoted as 'similar' images. The metadata database is split into training and evaluation sets with an 80/20% split.

To be able to intuitively access the dataset, we created a Python class that handles reading the metadata base, iterating over all the entries and loading images from file paths.

[1] https://github.com/mattclifford1/snap_vision_ads

[2] https://www.gnu.org/software/wget/

C. Data Loading

When training neural networks it is important to make the most of available resources. Reading and preprocessing image data can present a bottleneck in the training routine, especially with the relatively small neural networks we will use. To reduce the image loading and preprocessing bottleneck, we implemented the data loading process across multiple CPU process. Multi core data loading allows preproccessing of images to happen in parallel and reduces wait time for IO bottlenecks. This significantly decreases the time taken to load and prepossess images. We also further increase performance by prefetching batches of images before they are needed in an asynchronous manner. This allows CPU and GPU resources to be utilised simultaneously during training.

We implemented the asynchronous, multicore data loader by extending the Python class that we made in the data wrangling section into a pytorch DataLoader class [3] which handles the parallel processing, asynchronous queues, prefetching data and dataset shuffling for us.

We observed more than a ten-fold increase in speed when loading and preproccesing (image resizing to 256x256) the whole training dataset, taking 520 seconds single threaded and 45 seconds using all 12 cores available on our CPU. This directly equates to the reduction in training time, taking 521 seconds per epoch with a single threaded data loader and 51 seconds with the multi core version (numbers are for the small 'vanilla' neural network described below).

D. Evaluation

To automate the evaluation of embeddings we compare whether 'similar' image embeddings are found in the closest k embeddings.

$$score(V, f, k) = \frac{\sum_{v \in V} c(v, V, f, k)}{length(V)} \tag{6}$$

where V is the validation set, f is the embedding function and $c(v, V, f, k) = 1$ if the embedding of the image, $f(v_i)$ contains a 'similar' image (contains the exact same item of clothing) within the closest k embeddings and $c(v, V, f, k) = 0$ otherwise. Closeness is defined in terms of the Euclidean distance of each embedding pair.

For our analysis we use values of $k = 1$ and $k = 5$. To find the closest neighbours, we adapted the use of the k nearest neighbours classification algorithm from scikit-learn [4].

We also implemented a method to visually inspect the closest k embeddings of an input images. Although manual visual inspection can be time consuming, it is an important part of the evaluation for this type of problem, due to the subjective nature of what "similarity" can represent. PNG images of all the closest embeddings from the cases where $c(v, V, f, k) = 1$ (pass) and $c(v, V, f, k) = 0$ (fail) are automatically generated and can be viewed on our GitHub [5].

[3]https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader

[4]https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[5]https://github.com/mattclifford1/snap_vision_ads/tree/main/evaluation/eval_figs

For our neural network models described in Section IV; if the model weights are not stored locally, which occurs when the computer is different to the machine that trained the model, the weights are automatically downloaded from cloud storage. This simplifies the evaluation process so that it can be done by any person regardless of whether they trained the neural network or not.

VI. Dimensionality Reduction of Embeddings

The goal of this section is to visualise the relationships between the embeddings in a human friendly format. For example, we would hope that embeddings of different dresses (different in terms of shapes or colours) are "close" to each other, but embeddings of dresses and shoes are "far apart" from each other.

To view a high dimension embedding in 2 or 3 dimension we need to reduce the size of the embeddings. Dimensional reduction will come with a loss of information, so the approach used needs to be selected carefully. The most popular dimensionality reduction techniques are Principal Component Analysis (PCA) [10] and T-distributed Stochastic Neighbour Embedding (t-sne) [11]. PCA is better at keeping global information, whilst t-sne is better as recovering local information. We are interested in how close the local information is between embeddings so t-sne is the most suitable approach

Naively computing t-sne is expensive ($O(N^2)$ complexity) since it requires checking the distances between each and every other embedding available. However, current software packages of t-sne such as sklearn relies on Barnes-Hut approximation which reduces the complexity to $O(Nlog(N))$ [13]. Making it feasible to run on a dataset of our size.

A. Method

The pipeline stores embeddings of the Snap Vision image sets as csv files, which are then loaded into a Jupyter Notebookas we didn't have time to ingerate it fully into the pipeline. The perplexity of t-sne is a hyper-parameter that further reduces the complexity of the t-sne model. It is set to 30 as default. Before the training starts, the low dimensional embeddings are initialised using PCA and different learning rates, a number of iterations and early exaggerations are tried. This is because the loss function of t-sne has many local minima, and by tuning the learning rates, number of iterations and early exaggerations, it is possible to enable the t-sne model to not be stuck at a local minima.

After the dimensions of the embeddings are reduced to 2-D, each point is assigned with a colour that uniquely represents one category of clothing e.g. dresses are coloured blue, shoes are coloured green. Therefore, if the points clustered together have the same colour (with only 1 or 2 points that are exceptions), then it means embeddings of the clothes of the same category are close together. However, this requires the test data to be classified/labelled with categories of clothes prior to the visualisation.

The labelling process is conducted manually in a csv file. This is because of the lack of available pre-trained clothes

classifiers that are trained on similar dataset with a lot of different styles/categories of clothing. The ones are available are usually pre-trained on the fashion-MNIST [15] which only contains 10 categories of clothing. There is an abundance of literature available that records the training of deep Convolutional Neural Networks on large datasets that happen to be similar to the dataset that we were given [16]. However, the authors did not provide their pre-trained models, and training such models from scratch is beyond the scope of this project. However, the manual labelling process we employed is exhausting and not scalable if the snap vision dataset was larger.

VII. Results

A. Network Training

Figure 5 shows the training loss of the neural network during training, and figure 6 shows the evaluation score during training.

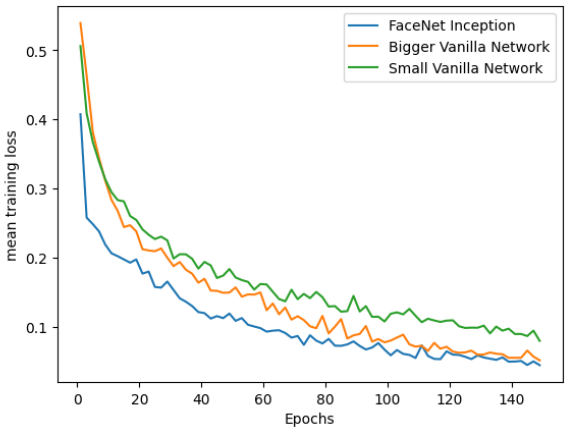

Fig. 5: Average loss per epoch during training. Trained using adam optimiser with 0.0001 learning rate, learning rate decay of 0.98 and batch size of 64.

B. Comparing all models

Model	'similar' is closest (%)	'similar' in closest 5 (%)
Simple	12	23
Small Network	46	63
Bigger Network	43	60
FaceNet	36	58

TABLE III: Comparison of models showing the percentage of times that a similar image is the closest neighbour as well as if a similar image is in the top 5 closest neighbours. Neural networks are all trained for 150 epochs using adam optimiser with 0.0001 learning rate, learning rate decay of 0.98 and batch size of 64.

C. Visual Inspection of Closest Embedding Images

Closest embeddings for target images produced by each model were inspected on a test cases similar to those seen

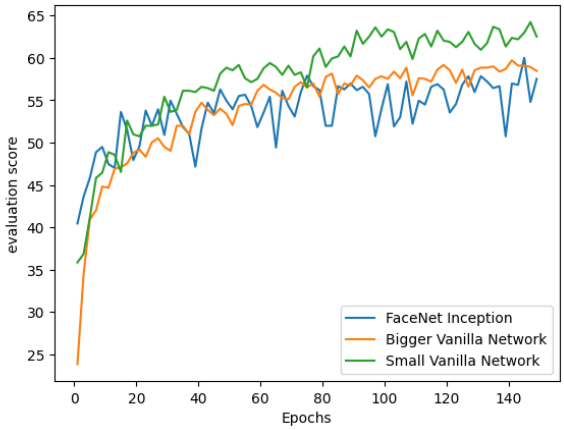

Fig. 6: Score on the evaluation set using $k = 5$ at the end of each epoch. Trained using adam optimiser with 0.0001 learning rate, learning rate decay of 0.98 and batch size of 64.

in 1, in which the human-model is wearing only one item of clothing. For the first target item (a single dress with distinct colour and pattern), the simple baseline model was only able to associate two of the five images, both of which show the a human model wearing the target item from different angles 7 (top). As anticipated, the image of the item in isolation is associated with other isolated images for different items, and the close-up image of the item is associated with other close-up images of different items.

The simple neural network model demonstrates significantly improved performance. Target images are correctly for four of the five images, and the embeddings correctly associate the isolated image of the item with an image of item worn by a human model. Further, the closest embeddings which do not correspond to the target item are more aesthetically similar than those presented by the simple model, and may be considered qualitatively to be more appropriate suggestions for a real-world fashion visual search algorithm. The neural network model still fails on the close-up image 7 (bottom), and only associates with other close-up images. The Facenet model and large-image neural network models also fail to perform well on the isolated and close-up image cases. Visual inspection was not persued for these models since their performance was poorer than the small-image neural network model III.

Considering only the images of a human model wearing the target item, the simple neural network appears to perform well at associating images of the target item regardless of the pose of the model. Another example is seen in 8.

Images of small accessories were examined for the neural network model. Isolated images of accessories were typically associated with other small items of the same colour 9. None of the examined isolated images of accessories were correctly associated with images of the item being worn by a human.

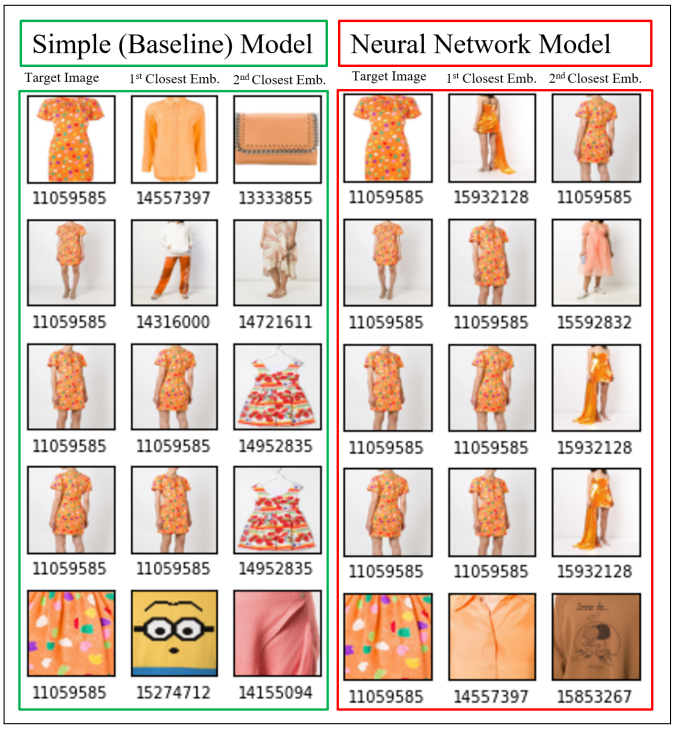

Fig. 7: (Top) Simple model and (Bottom) simple neural network model performance. The first column contains the target images, and subsequent columns display the images corresponding with their closest embedding.

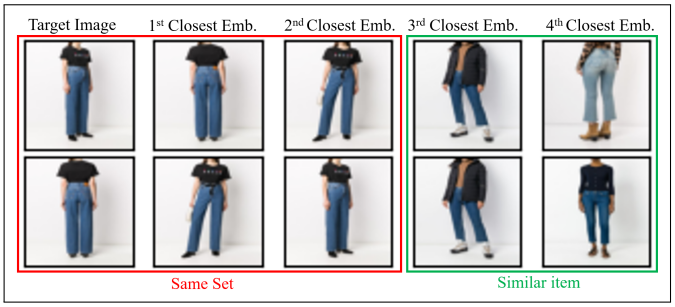

Fig. 8: (Top) Simple model and (Bottom) simple neural network model performance. The first column contains the target images, and subsequent columns display the images corresponding with their closest embedding.

D. T-SNE visualisation of embeddings

Figures 10 to 13 show the results of tsn-e reduced embeddings on different item catergories

VIII. Discussion

All networks managed to obtain a better score than the simple baseline model. However, we randomly picked the positive and negative examples that satisfied equation 5. This is a naive approach which might have inhibited the training process. This is because it would be better to train the network on triplet examples that are the argmin and argmax of the distances as this would provide a stronger loss signal to train

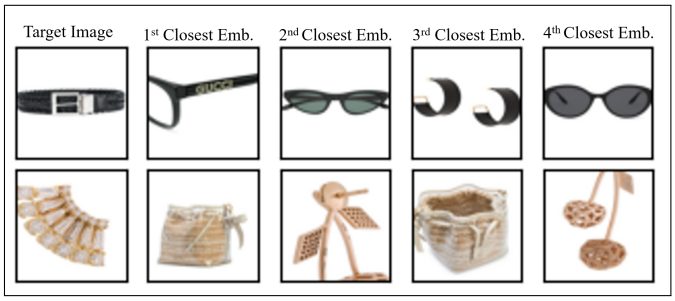

Fig. 9: Images of small accessories were not well associated, but were linked to other small accessories of similar colour.

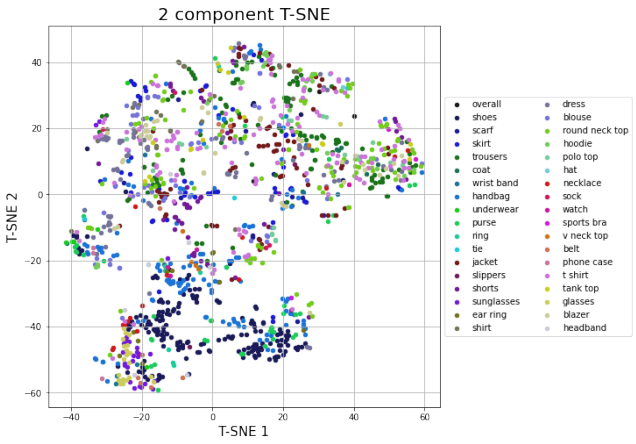

Fig. 10: learning-rate-200-exaggeration-20

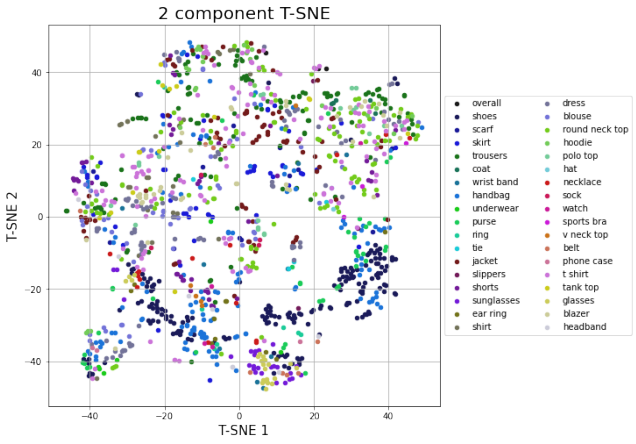

Fig. 11: learning-rate-500-exaggeration-20

on. How we would improve on this is discussed in more detail in the further work section.

Figures 5 and 6 show that training is fast initially and then after around 10 - 20 epoch begins to slow. this is probably due to the aforementioned issue with triplet selection during training. However, although slow, training continues to improve the evaluation score right up until 150 epochs for the small vanilla network, this suggests that it could continue to be trained further to improve the performance.

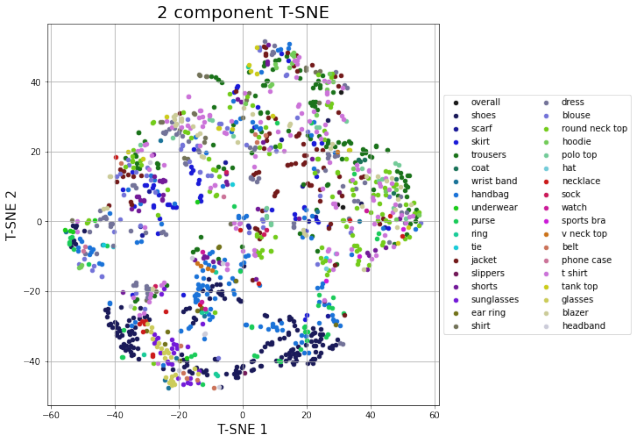

Fig. 12: learning-rate-auto-exaggeration-12

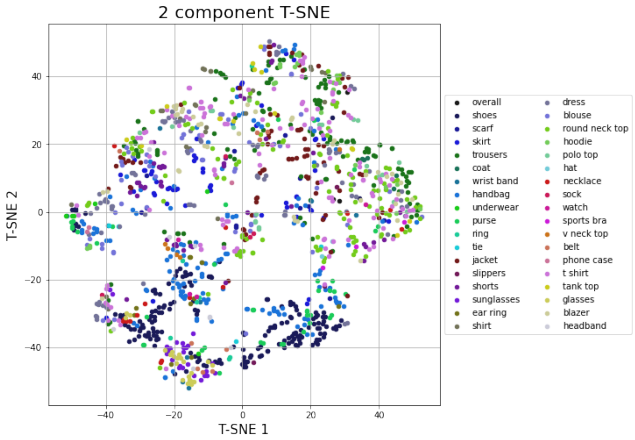

Fig. 13: learning-rate-auto-exaggeration-20

FaceNet and the bigger vanilla neural network also have lower training loss during training but also lower evaluation score. This indicates that they are overfitting to the training set more than the small vanilla neural network. However, more investigation into this would be to be done by training for longer to see if the evaluation score decreases.

Despite the flaws in neural network training, they still appear have learnt meaningful and relevant embeddings. This can be shown in the visual inspection of the closest embeddings.

Figure 7 illustrates that both the simple baseline model and neural network model are able to leverage visual information to suggest aesthetically similar items, with marked improvement in the neural network model compared to the baseline model. Sets of images showing a human-model wearing a prominent item of clothing from different angles are well associated by the neural network model, demonstrating some degree of pose invariance. This can also be seen in 8 with a slightly more complex case, containing more redundant visual information. However, neither model was able to match items from a "close-up" target image, likely due to the lack of white-space in the image. The models may be primarily leveraging the amount of white-space in the image, causing different but aesthetically similar items to be associated. This is further evidenced by Figure 9, in which different small accessory items of the same colour are matched to one another.

Figures 10 to 13 show observable clusters for shoes, especially at the bottom left, bottom middle and bottom right of the image. It would be better if the t-sne method could merge the separated clusters of shoes into one cluster. Therefore, the learning rates and the early exaggerations are both increased in order to avoid t-sne getting stuck in local minima. Despite the tuning of those two hyper-parameters, the separated clusters of shoes cannot be merged together. This indicates that the emergence of separate clusters of the same class of clothing is due to the qualities of the embeddings produced instead.

It can also be noted that the dresses are scattered with small clusters, each containing 3 or 4 dots. This is due to the triplet sampling procedure during the training of the neural network models. After an anchor image is randomly selected, the corresponding positive image is randomly selected out of the same folder as the anchor image. The negative image is then randomly selected out of all the images that are not stored the same folder as the anchor image. This ensures that the parameters in the trained deep models map the images in the same folder to embeddings that are "close" to each other, but images in different folders are further "apart" from each other. This means, if the goal is to ensure the points corresponding to the same class of clothing are close together in one unique cluster (with few exception points), then the training data must be labelled. The anchor will still be randomly chosen, but the positives will be sampled out of images in the same class, whilst the negatives will be sampled out of images in the different classes (which is what previous papers did). However, it was decided that manually labelling 6411 images would require too much work and we did not elect to pursue this method.

We did not focus on refining the network architectures or training routines since in this project we focused more on the creation of our robust and efficient data pipeline as a whole rather than getting the best performing models possible.

IX. Conclusion

We created a robust and modular pipeline to create and evaluate image embeddings. Although there are flaws in the model training and evaluation process, our aim was not to create the best performing model possible, it was to create a pipeline that is effiecient and flexible so that changes to any models, training or evaluation can be easily implement in the future if desired.

X. Future Work

A. Cloud integration

Parts of the pipeline such as dataset and model storage are currently integrated into cloud storage. It would be preferable if model uploading was integrated into the pipeline after a neural network model is successfully trained. This is more challenging since cloud account passwords or keys are required to either be input by the user or stored locally

in order to write to online storage. It would also be nice to fully integrate the whole pipeline into a cloud compute environment such as AWS. This would allow easy access to train models and perform inference. However, this is a challenge due to the potential cost and of AWS services. A Google Colab notebook might overcome these issues with their free tier of GPU access, but this would be a much more restrictive cloud compute environment. Nevertheless, a Google Colab embeddings inference notebook would make a great addition to the GitHub page, allowing users to quickly generate embeddings from images other than the ones found in the snap vision dataset.

B. Manual Models

During the project, we performed an investigation into manually generated embeddings. Originally this was to generate a simple model with which we could perform pipeline validation activities. However, we decided to further explore the manual generation of image statistics to determine whether any performance gains could be made. One technique was applied whereby we calculated the most dominant colours in an image, and performed an optimisation to match the dominant colours with a comparison image.

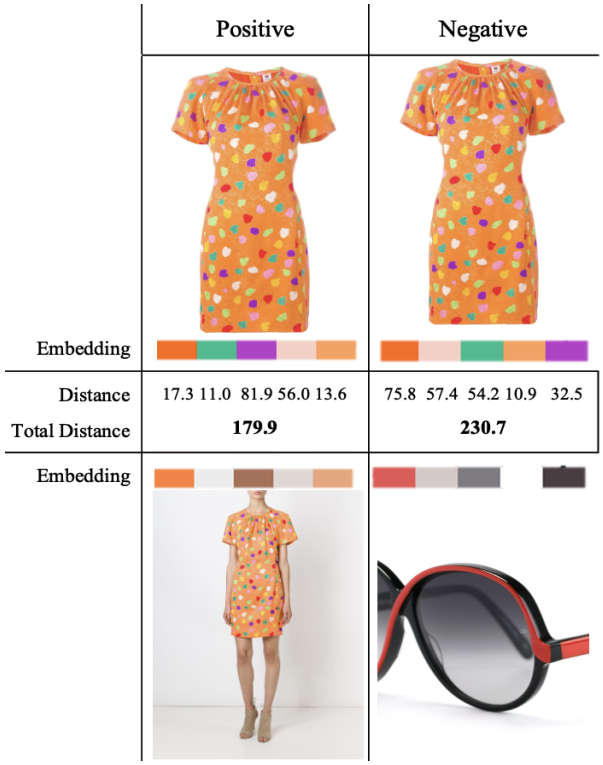

Fig. 14: Detecting dominant colours in images for comparison

Figure 14 shows one such result, with a higher distance metric (the euclidean distance between two colours CIELAB colour space[6]) between the negative example and positive example than between the two positive examples. This method showed very promising results, achieving a similar image in the top 5 returned images 100% of the time, and a neighbouring similar image 83%. While these results are impressive when compared to the deep embeddings, they are not quite embeddings but rather statistical information about the image. The comparison step required a linear sum assignment optimization step which presented a significant challenge for pipeline implementation, and ultimately this model was not implemented into the pipeline and validated with our verified evaluation method. In addition this model would not be resilient to the challenges discussed in Section II.

Aside from implementation into the pipeline for evaluation, future work could further investigate manual embeddings alongside deep embeddings to better understand where image embeddings might be interpreted via statistically generated counterparts, or to create additional features on which to train deep embeddings to improve robustness.

C. Neural Network Training

We could improve training by not randomly selected a negative that satisfies equation 5 but select the argmin of $d(\mathbf{e_a}, \mathbf{e_n}$ and the argmax of $d(\mathbf{e_a}, \mathbf{e_p}$. This is called a hard example. Using subset of the training set to produce semi-hard triplet examples is claimed also to improve training in the FaceNet paper.

To improve the training of neural networks we would need to implement the **online** version of semi-hard negative picking for the triplet loss as described in FaceNet [3]. This would hopefully improve the training as it ensures that bad negative examples are chosen for the latest model weights making the training more efficient and relevant. However, there is a significant computational overhead with online negative picking, even when reducing the possible examples into smaller batches rather than the whole training set.

It would also be required, given the computational resources, to train more of the same neural network models with different initial weights to get a grasp of the variance within the same model.

D. Evaluation

It would help understanding how well the embeddings are performing by not only evaluating whether a 'similar' image is in the k closest embeddings, but also to evaluate how close any 'similar' images are in the embedding space. This would provide a way of automatically evaluating the embeddings in more detail. This 'cluster' distance of embeddings could be performed on a per clothing item or category of clothing explained in Section VI.

E. Image Processing

It was noted that the models presented in this report were likely to associate images based on the background visual information, which was predominantly white or off-white. Whilst accessories, such as different types of sunglasses, appeared to be successfully associated by the models, there was

[6]https://en.wikipedia.org/wiki/CIELAB_color_space

evidence to suggest visual search models were leveraging the amount of white-space in the images rather than meaningful features of the images. A preliminary method for addressing this issue was implemented into the pipeline. Training and evaluation images were masked to remove near-white pixels, Figure 15. This method was not fit for evaluation, as it does not address the issue in this state (background pixels are simply replaced with white or empty pixels). Further, using fixed threshold values for the mask means that the method is not appropriate for all images. In Figure 16, the top and middle examples show reasonable masking, which could help to associate between isolated and human-modelled items. However, in the bottom examples, based on images of a white skirt, the target item is largely masked from the image, and unimportant visual information is preserved.

F. Ethical Considerations and Implications

Visual search algorithms are susceptible to bias. The results of this work have demonstrated how naive models can associate fashion images based on the appearance of the human model wearing the item of clothing (an image of a model wearing a particular target item may be associated with another image of a model wearing different clothing, rather than with a different image of the target item). In a real world commercial application, this would need to be addressed carefully to avoid racial biasing, particularly if the platform is designed to allow users to search based on images they have taken themselves, of themselves or people they know. Whilst it proved to be outside of the scope of this project (See X-E), it is possible that such issues could be addressed by removing visual information from the images in the training data that is associated with human appearance e.g. by masking and removing visible skin, faces and extremities.

Fig. 15: Automatic masking was applied using OpenCV package in Python. (Left to Right) The image was converted to grey-scale with pixel values from 0 (black) to 255 (white). A threshold mask was applied to remove near-white values between 200 and 255. Morphological structuring was then applied to remove high spatial frequency details of the mask. The mask was then applied to the image to remove light background details. The colour has been inverted to visual clarity.

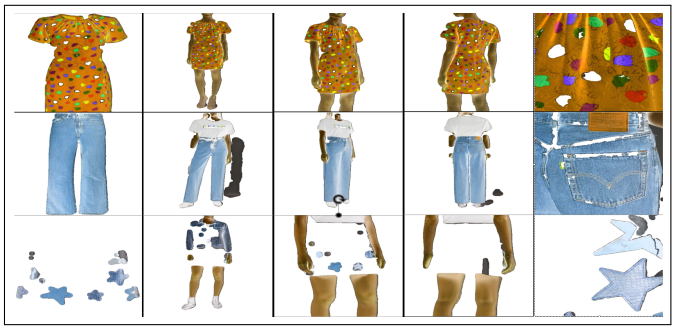

Fig. 16: Basic masking applied to three images sets. The original images for these sets can be seen in Figures 1 and 2

REFERENCES

[1] D. Tighe, “UK: Change in online shopping since COVID-19;2020-2021“, Statista, 2021

[2] C. Ho and P. Morgado and A. Persekian and N. Vasconcelos, “PIEs: Pose Invariant Embeddings“ CVPR, 2022.

[3] F. Schroff and D. Kalenichenko and J. Philbin, “FaceNet: A Unified Embedding for Face Recognition and Clustering,” CVPR, 2015.

[4] W. Kang et. at., “Complete the Look: Scene-based Complementary Product Recommendation”, CVPR, 2019.

[5] C. Szegedy et. al., “Going Deeper with Convolutions,” CVPR, 2014.

[6] E. Hoffer and N. Ailon, “Deep metric learning using Triplet network,” SIMBAD, 2015.

[7] K. He et. al. “Momentum Contrast for Unsupervised Visual Representation Learning”, CVPR 2020.

[8] T. Chen et. al. “A Simple Framework for Contrastive Learning of Visual Representations”, ICLR 2020.

[9] J. Grill et. al. “Bootstrap your own latent: A new approach to self-supervised Learning”, NIPS 2020.

[10] I. Jolliffe and J. Cadima, “Principal component analysis: a review and recent developments”, Trans. R. Soc, 2016.

[11] L. van der Maaten and G. Hinton, “Visualizing Data using t-SNE”, Journal of Machine Learning Research, 2008.

[12] G. Hinton and S. Roweis, “Stochastic Neighbor Embedding”, Journal of Machine Learning Research, 2002.

[13] L. van der Maaten, “Accelerating t-SNE using Tree-Based Algorithms”, Journal of Machine Learning Research, 2014.

[14] D. Kingma, J. Ba, “Adam: A Method for Stochastic Optimization”, Conference for Learning Representations, San Diego, 2015.

[15] H. Xiao, K. Rasul, R. Vollgraf, “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”, arXiv:1708.07747, 2017

[16] W. Wang, Y. Xu, J. Shen, S. Zhu, “Attentive Fashion Grammar Network for Fashion Landmark Detection and Clothing Category Classification”, CVPR, 2018