**BJIT Limited**

*Project Report*

# Image Similarity Based Fashion Recommendation


**Team Members**

Muttakin Islam, Senior Software Engineer (AI)

Md. Rashedul Hasan Safa, Senior Software Engineer (AI)

Md. Shohag Mia, Senior Software Engineer (AI)

**Mentor**

Javed Hasan, SVP (Head of SD1)

**Submitted To**

Javed Hasan, SVP (Head of SD1)


Project Timeline: 01 February 2020 To 30 April 2020

# Objective

The goal of the project is to develop an intelligent fashion recommender- the system will take an image of any fashion product and will be able to recommend similar fashion products available in the inventory based on a similarity ranking.

In addition it will consider some user specific data for personalization purposes. For example, if a user doesn't like the color yellow then don't recommend a product of that color.

# Related Works

We studied a lot of papers, books, documents, blogs and online courses. Those are given at the end of this section.

**Existing systems and research:**

- Recommends items based on visual similarity using CNN model.
    - In this approach models are trained on specific dataset for classification.
    - Here classes are different target classes of fashion items.
    - Features of the images are extracted from CNN model and get embedding of them.
    - Then calculate the similarity of images applying distance calculating methods(cosine, hamming distance etc) on extracted features of images from CNN model.
    - Return the most similar images based on calculated similarity score.
- Recommendation based on embedding and calculating distance among them
    - Make the layers of ImageNet based Keras pre-trained model non-trainable and feed images for feature extraction.
    - Get the embeddings of all images from model.
    - Calculate distance of all images from embedding applying distance calculation methods.
    - Return the most similar images.
- Applying transfer learning on pre-trained models.
    - Apply transfer learning on image classification models with own dataset.
    - Classify the images based on own defined classes of images.
    - Return the images of the identified class.
- Recommendation based on image triplets image search technique.
    - Apply triplets of image for image searching.
    - Find out the similar images based on prediction score.
    - Return the matched similar images.

**Study Approaches:**

- One member studied horizontally(that is screening), another went vertically(in depth) through the topic selected by the first member.
- We matched and compared the study results against our collected dataset.
- Then we decided which ones were potential good resources to serve our purposes.

**Pros of study:**

- Faster and effective. Less chances of missing a useful material .

The studied materials are given below:

- Transfer learning with a pre trained ConvNet
- Image similarity using deep ranking(2)
- Building a recommendation using CNN
- Fashion Image Retrieval and Label Prediction
- ImageNet: VGGNet, ResNet, Inception, and Xception with Keras
- Keras Tutorial : Using pre-trained ImageNet models

# Proposed Approach

Briefly describe the approach. Emphasize on why you selected this approach among different competing alternatives.--

We make 2 types of recommendations in our system:

- In inventory images
- For New(outside image uploaded by user) images.

In our project, we intended to use keras pretrained model for feature extraction and used cosine similarity method for calculating similarity of the images. We stored the features in a variable.
**For inventory images**, we calculate the cosine similarity of the embeddings of the features of the images to get the top most similar images in order.
**For unknown images,** first we classify the image to get it's subcategory, then we use another classifier to get it's type(gender and articleType) for filtering. Then we use the keras model to get the feature embedding and apply cosine similarity to get the top most images from the filtered ones. We mainly do 2 part classification to reduce the inference time. At first when we weren't using any classifiers, it took about 43 seconds. Then we added the subCategory classifier to reduce the number of images. Then the system took 24 seconds for the results.Then we added a 2nd classifier to reduce the number of images further. Then it was giving results within seconds. For inventory images, we get the recommendations within seconds. Additional filters can also be provided for better recommendations.
**Reasons to choose the approach:**

- We chose Keras pre-trained model as we have only 44k data. Transfer learning helps us to get expected level of accuracy with low amount of data.
- Pre-trained model performs better in feature extraction and embedding.
- We used two classifiers to reduce the time complexity of similarity calculation to make the system real time.

# Method

## System Overview

How different components interact. You can add some diagrams so it is easier to get an overall picture.--

In our project, we have two types of recommendation.They are for inventory images and non inventory images. First we used the Resnet50 model for feature extraction. Then for inventory images we calculate the similarity with cosine similarity with all the images and show the best results. We can also add some filters. If filters are added then the number of images are reduced. For unknown images we add two classifiers for subCategory and derivedType features. And then we extract the image features using Resnet50 Models. Then we  calculate the similarity of the images using cosine similarity and show the results. The following images briefly shows how the different components interacts with each other

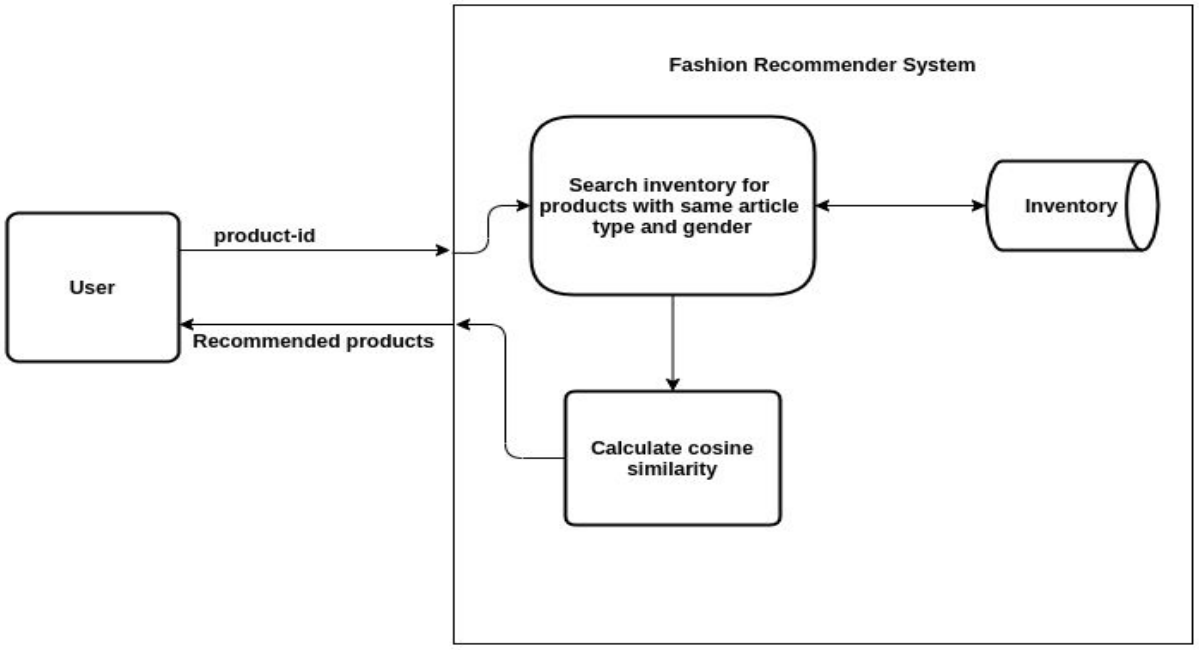


Fig-1: Recommendation for inventory items

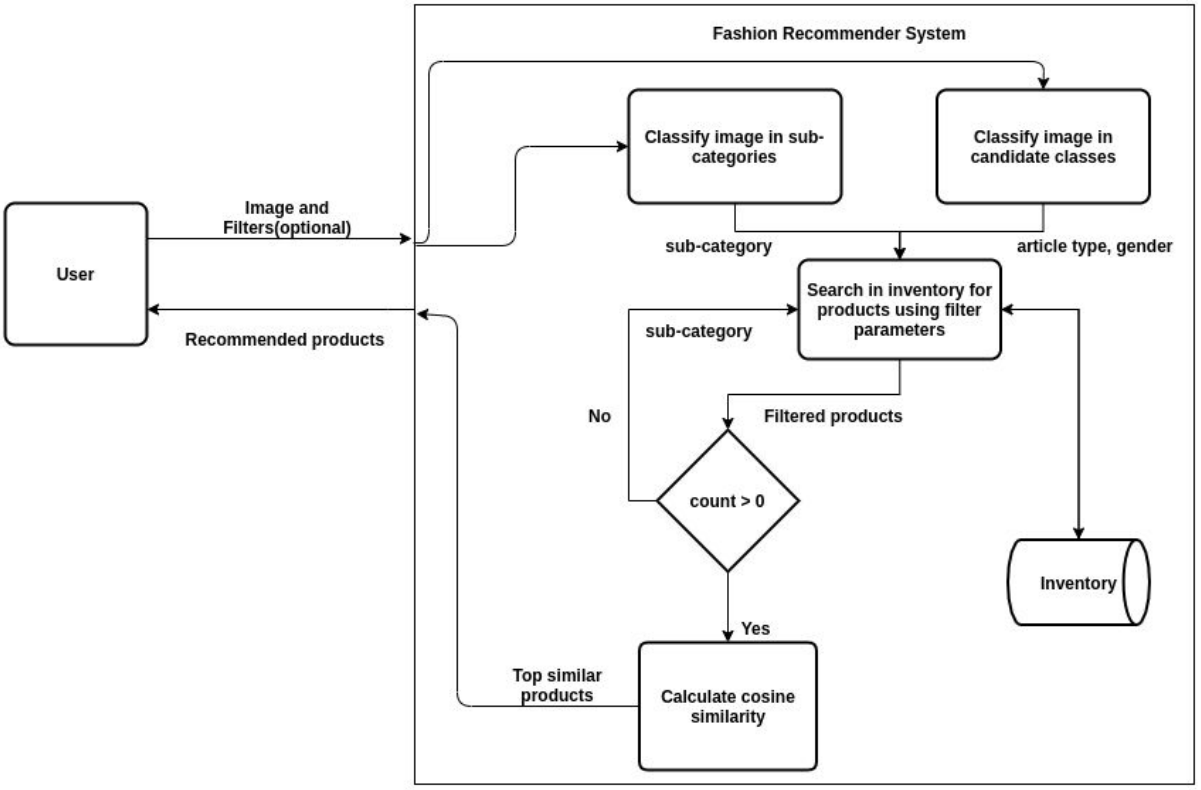


Fig-2: Recommendations for unknown items (search by image)

## Neural Network Architecture/Machine Learning Approach

For inventory image recommendations, we got the embeddings of all images from the features that were extracted with the Resnet50 model. The feature extraction method is explained in the Feature Extraction section. We then stored the embeddings. Users can choose features to filter out the results.Though this is optional. If a user chooses to add filters then the system calculates the cosine similarity of the embeddings of all the filtered images to get the top similar images as recommendations.If a user chooses not to add filters then the system calculates the cosine similarity of the embeddings among 44419 images to get the top similar images as recommendations. For example, suppose a user chooses Blue as the baseColor for filtering and there are 5000 image of baseColor Blue and chooses an item, then the system will calculate the cosine similarity of the items embedding with the 5000 image's embeddings with baseColor Blue and gives the top similar images as recommendations.

For unknown images we first classify the image by its subcategory, then by its derivedType which is a combined feature added by us and then extracted the features and used those features for embedding and calculate the cosine similarity with subCategory and derivedType as default filters. Other filters may be added if the user chooses some. The filters work as it does for the inventory images. Our derivedType feature is composed of gender and articleType. An example for the derivedType may be "Men T-shirt". Suppose a user uploads an image and it's subcategory is topwear. Let's assume that the Topwear subcategory has 12000 images. Then the derivedType classifier is run on the 12000 images. Let's also assume that the image is classified as a Men T-shirt and this feature has 800 images. Then the system will calculate cosine similarity of the image's

embedding that was found from its feature with the 800 image's embeddings and gives the top similar images as recommendations. It will be explained in detail in the next section.

From the features mentioned in the Dataset, we use subCategory as a feature for one classification and also use gender and articleType combinedly as derivedType as a feature for the 2nd classification. For subCategory classification we used the keras classifier model, Resnet50. For derivedType classifiers, we calculate the similarity with a "candidate image" of each derivedType. There are 280 unique derivedTypes. So we just need to compare the image with 280 images. For this candidate image selection at first we extract all images features for embedding. Then we calculate cosine similarity with all the images. After that we calculate summation of all similarity scores for every image in every derivedType group. Then for the 280 groups, we select an image of every group with the highest score as that group's candidate.
When we classify an unknown item, first we apply model to classify it against subCategory. Then we classify the item further with the candidate classifier. Here cosine similarity of the unknown item is calculated against all candidate items of that subCategory. From the highest similarity score, we find the closest derivedType for the unknown item and this is the derivedType of the unknown item detected by candidate classifier.
Afterwards we calculate similarity if the unknown items against that identified derivedType group and recommend the most similar items for the unknown item.

## Feature Extraction

For feature extraction, we used the Resnet50 pretrained model on ImageNet. We just used the layers that are used for feature extraction. We froze all layers of the model and used GlobalMaxPooling for feature extraction. Then get embedding of all images from these extracted features from the model.

## Augmentation

For Inventory image recommendation,
We didn't use any image augmentation technique as it's not needed here.

For Unknown(without any filter parameter) image recommendation,
We used model level default augmentation which is performed online(on the fly) during model training for subCategory classification.

## Preprocessing

Fashion product image dataset consists of 44419 images with metadata. These metadata describes product id, product type, gender etc. We delete incomplete data and derive some new features(columns) for computational, performance and accuracy for the predicted similar products for recommendation.
We preprocess images online while feeding those to Keras model that is done by the model. And we resize images to fit for the model.

# Dataset

Provide details of all used datasets. If the data was prepared internally, describe in detail the process. If it is taken from another source, reference them. Mention dataset volumes (amount of samples, hours). Describe how different splits(train/validation/test) were prepared and amount of data in them.--

We've collected a dataset from kaggle. It contains about 44424 fashion product images and metadata of all the images in a csv file.From 44424 data, 5 images were not found, so we have 44419 images. The csv file is well filled with corresponding values in each column. It has 10 columns of features. They are id,gender,masterCategory,subCategory,articleType,baseColor,season,year,usage and productDisplayName. The images are very clean and have a nice and clear background. Most of the images are 2400x1800 in size. Some of them are 1600x1200. The link for the data is given below:

- Fashion Dataset

Model was trained in 2 ways for 2 different purposes:

- **Inventory items(known) recommendation**: In this case we train the model for image feature extraction to get embedding of them, so the whole dataset was used to train the model with no different train, test and validation split of the dataset.
- **Unknown(new/no filter) item recommendatio**n: In this case we train keras pre-trained model ResNet50 with total image = 7905 and total class=15. We split train, test and validation as test=0.2%, validation = 2% of (total-test) and train = total - test - validation. Here we freeze 170 layers of ResNet50.

# Experiment

In this section, you need to provide details of how experiments were conducted, results obtained and their implications. Following subsections must be included in all reports.

## Training Procedure

For feature extraction model selection, we experimented with images of articleType with 500 or more images. The number of images we used was 34719. We used a slightly smaller data than the original because when we took all the images memory ran out. We used pairwise cosine similarity for all the images. This gave us a ***34719 X 34719*** similarity matrix. For a single image we checked the top 10 most similar products and compared their article type and gender. If it matches with the current products article type and gender then it's a good recommendation. We did experiments on **Inception**, **Xception**, **NasnetLarge**, **InceptionResnet**, **Resnet50**, **Resnet50V2**, **Resnet152** as they give

exceptionally good results for image similarity detection. We took top 10 recommendations with error thresholds of 0, 0.1, 0.2.

For selecting a proper pre-trained keras model for image classification we also had done some experiments. We had compared the performance of several models including **ResNet50** and **Xception**. We also ran the same model with different hyperparameters such as learning rate, batch-size, data augmentation etc. Also took different amounts of data to train, changed the class and number of classes. All those experiments showed that our selected model **ResNet50** gives the best result on classifying the images into sub-categories with training and validation accuracy **90%** and **85%** respectively. The description of the trained model and it's hyper-parameter are pictured below-

```
# base model
restnet = ResNet50(include_top=False, weights='imagenet', input_shape=(IMG_HEIGHT,IMG_WIDTH,3))

# all 170 layers among 175 are non-trainable
# top 5 layers will be trainable
for layer in restnet.layers[:170]:
    layer.trainable = False

base_model = restnet
x = base_model.output


x = Flatten()(x)

# a dense layer for prediction
predictions = Dense(NUM_CLASSES, activation='softmax')(x)

# compile the model
# loss = 'categorical_crossentropy'
# optimizer = SGD
# learning rate = 0.00005
# metrics = 'accuracy'
model_finetuned = Model(inputs=base_model.input, outputs=predictions)
model_finetuned.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(lr=0.00005),
              metrics=['accuracy']
```

Fig-3: ResNet50 customization for sub-category classification

## Evaluation Criteria

In a recommendation system, no wrong prediction should be allowed even if some good recommendations are omitted. So we needed to be precise with the recommendation. We also measured the performance of the feature extractor model after calculating similarity between products. We then took top 10 similar products into consideration and compared their article type and gender. If they are equal we counted it as a true positive. We calculated accuracy with this value. The formula is -

*accuracy = count(true_positive) / 10*

So the error of the model for the current product is

*error = 1 - accuracy*

If the error of the current model for that particular product is less or equal to the threshold then count it as TRUE POSITIVE. Following this step for every other selected(34719) product we calculated TRUE POSITIVE. Then calculated the model performance as below -

*performance = count(TRUE_POSITIVE) / 34719*

We mainly focused on the article type and gender as our evaluating feature. If any of the recommendations gender or article type is wrong then that recommendation is wrong.

For the sub-category classifier model we considered the train, validation and test accuracy. Train and validation accuracy(as well as loss) are calculated when the model is trained. After running the trained model on test data, we calculated train accuracy manually. For all three cases the formula is same,

*accuracy = count(true_prediction) / total sample*

# Results & Discussion

We had 3 levels of result evaluation and decision making for 3 different parts of our project. *In every case, we use **Precision** as our accuracy metric as a single wrong item in recommendation shouldn't be tolerated. That is; if a single item is wrong in 5/6/10 items in recommendation, then we treat the whole recommendation as wrong. There is no way to measure how similar the recommended items, except involvement of human judgement(that is also not accurate). So we consider a recommendation as wrong if its gender or articleType is wrong. But in most wrong recommendations, a human can't say it's wrong as he/she can't differentiate among these type of wrong recommendations.*

***Example**: It's quite impossible to differentiate between boys t-shirt and girls t-shirt, male sport shoes and female ones etc. Only humans can judge it when it's said the gender of the items. Similarly our model makes the same mistakes in some cases. Afterall we consider these as wrong recommendations. If we don't consider these as wrong, then the accuracy could be very high, like 95% or above.*

3 levels of result evaluation and verdict are below:

**Model selection:**

For model selection based on accuracy of image(no filter parameter) recommendation, we got the best result for Resnet50 and Resnet 152. The following table shows the results for both:

| Model | TopN | Feature | Tolerance | Performance |
|---|---|---|---|---|
| | | articleType | .2 | 82.21% |
| | | articleType | .1 | 76.73% |
| | | articleType | 0 | 67.84% |
| | | gender | .2 | 81.02% |

| | | | | |
|---|---|---|---|---|
| Resnet50 | 10 | gender | .1 | 74.24% |
| | | gender | 0 | 63.67% |
| | | gender and articleType | .2 | 67.50% |
| | | gender and articleType | .1 | 58.92% |
| | | gender and articleType | 0 | 47.03% |
| Resnet152 | 10 | articleType | .2 | 82.99% |
| | | articleType | .1 | 77.44% |
| | | articleType | 0 | 68.66% |
| | | gender | .2 | 81.39% |
| | | gender | .1 | 74.75% |
| | | gender | 0 | 63.94% |
| | | gender and articleType | .2 | 68.48% |
| | | gender and articleType | .1 | 59.98% |
| | | gender and articleType | 0 | 47.80% |

The plot for precision - tolerance for article type as feature used for calculating precision is given below:

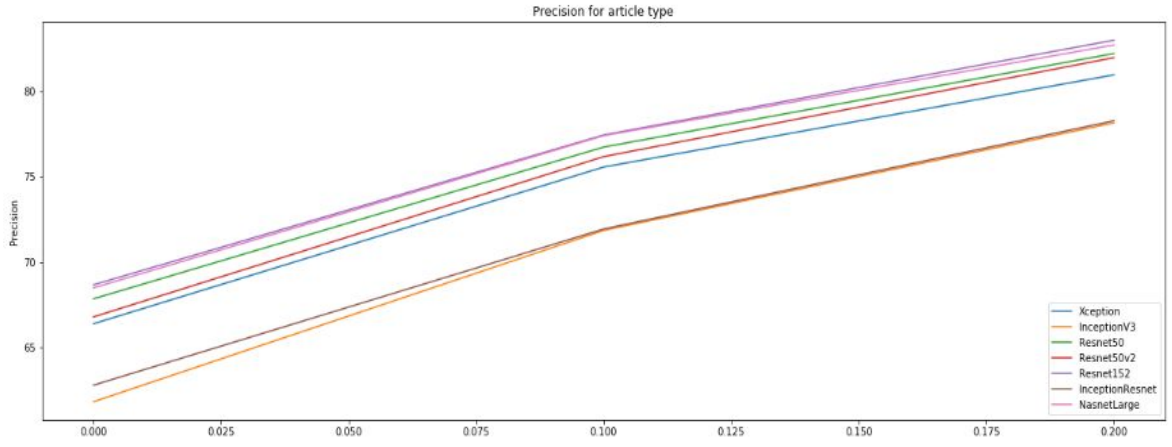

Fig-4: Feature extractor model comparison for article type

The plot for precision - tolerance for gender as feature used for calculating precision is given below:

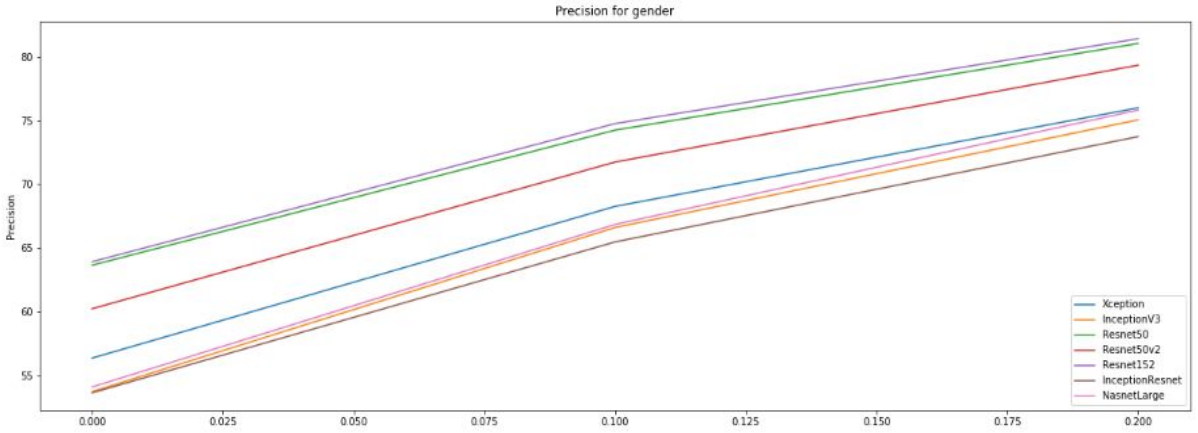


Fig-5: Feature extractor model comparison for gender

The plot for precision - tolerance for both articleType and gender as feature used for calculating precision is given below:

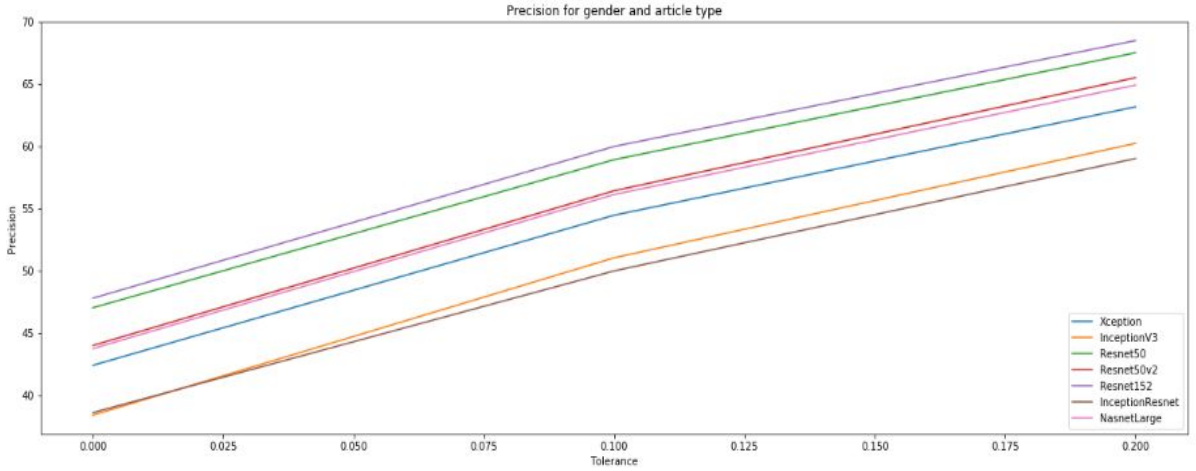

Fig-6: Feature extractor model comparison for article type and gender

As we can clearly see from the above plots, if we used the **Resnet152** model for feature extraction, it gives the best results for all the important features chosen but the size of this model is 232MB. On the other hand, we can also see that **Resnet50** is close enough to **Resnet152** but its size is just 98MB, which is almost 125% lesser than Resnet50. So **Resnet50** is lighter and faster while extracting features from images.

So, we used ***Resnet50*** for feature extraction.

**Classifier**(for subCategory):
The training result of the classifier model is given below-

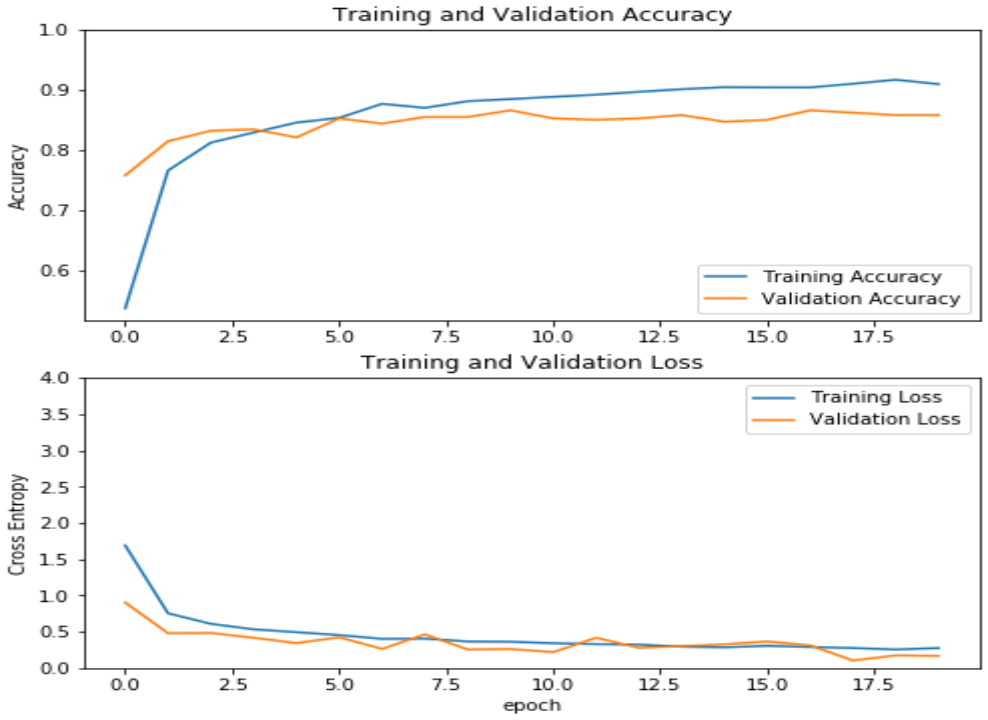


Fig-7: Training history of custom classifier on sub-category

**Train accuracy - 90%**
**Train loss - 0.27**
**Validation accuracy -85%**
**Validation loss - 0.16**
**Test accuracy - 86.46% ( 1367 / 1581 )**

**Classifier**(for articleType/derivedType):
After an image is classified in subCategory, we classify it further in the next level for identifying its derived type; that is gender and articleType. In this case use a candidate based classifier that works on calculating similarity among candidate images of every group. Here we also take Precision as an accuracy metric. That is: if any of the recommended items is wrong by gender or articleType, then the whole recommendation is treated as wrong. In this case for 5000 images,
***Precision*: 75%**

# Lessons Learned

We have learned a few things while working on the project. They are-

- Practically working with a Deep learning model.
- How to work with TensorFlow and Keras.
- Working experience with ImageNet based pre-trained keras models.
- How to transfer learning on pre-trained models.
- Ways to image embedding and applying them for similarity calculation.
- How to leverage Kaggle and Colab.
- Problem with downloading dataset from Kaggle and solved it with moving dataset to Colab and then downloaded with commands and UI together.
- Model loading takes much time and solved it with saving the model in file for next loading.

# Future Work

**Limitations:**

- We only built, tested and ran it on pc. Mobile platforms aren't considered here.
- There might be some way to improve accuracy and inference time for unknown item recommendations.

**Future work:**

- Build it for mobile platforms.
- We've planned to improve accuracy and inference time for unknown item recommendations.
- Extract current fashion trends from social media and apply them in fashion recommendation.