CAB420 - Assignment 1C

Callum McNeilage - n10482652

Connor Smith - n9991051

Queensland University of Technology

Problem 1: Clustering and Recommendations

For this task we used (Asad, 2020) as reference.

Data Processing and Handling

For this problem we were given several .csv files; ‘links.csv’, ‘movies.csv’, ‘ratings.csv’ and ‘tags.csv’. In order to improve performance, we limited the ratings dataset to only ratings above 4, movies dataset to the first 200 movies and users dataset to the first 315 users. We chose to limit the ratings to 4 and above as any rating below 4 would likely be unusable by the model as a user who rates a movie a 1, 2 or 3 generally does not like that movie. This also meant we did not have to do any processing of whether a user liked the movie later on as only liked movies are being considered by the model.

```
ratings_4 = ratings[ratings['rating'] >= 4.0]
```

Figure 1: Limit ratings to 4 and above

As the movies dataset contains upwards of 100836 individual movies, this would require a large number of clusters in our KMeans function. As such, we decided to limit the movies to the first 200 movies in the dataset. This allowed us to limit our number of clusters to 82, which was much faster to compute on the hardware available.

```
movies_list = np.unique(ratings['movieId'])[:200]
ratings = ratings.loc[ratings['movieId'].isin(movies_list)]
```

Figure 2: Limit movies to first 200

As only Users 4, 42 and 314 were required for testing the output of our model we decided to also limit our users dataset to the first 315 users to improve compute time.

```
users_list = np.unique(ratings['userId'])[:315]
ratings = ratings.loc[ratings['userId'].isin(users_list)]
```

Figure 3: Limit users to first 315

We then combined these datasets into a single combined dataset for training (See Appendix 1 for sample dataset).

```
users_fav_movies = ratings.loc[:, ['userId', 'movieId']]
users_fav_movies = ratings.reset_index(drop = True)
```

Figure 4: Combine all three datasets into 1

From this processed data, we then created a Sparse Matrix of users against the movies they have rated (See Appendix 2) which was then used in our Elbow Method to find the appropriate number of clusters for our dataset.

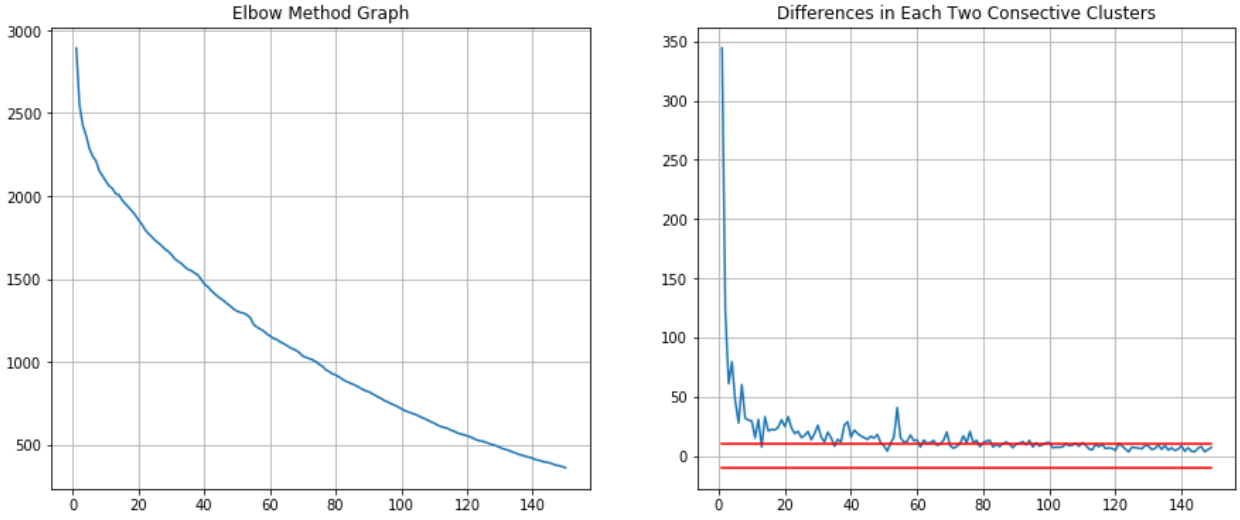


Figure 5: Elbow Method for dataset

Clustering Method

It was determined using the elbow method above that 82 clusters was the optimal for our KMeans model.

	0	1	2	3	4	5	6	7	8	9	...	305	306	307	308	309	310	311	312	313	314
userId	1	3	4	5	6	7	8	9	11	12	...	339	340	341	342	343	344	345	346	347	348
Cluster	13	0	74	77	7	5	68	0	24	0	...	3	9	15	0	5	38	25	32	68	5

Figure 6: Dataframe of clusters

We chose to use KMeans Clustering because the data utilized numerical values, and after or processing, the data contained few dimensions. In addition, we wanted any particular user to only be assigned one cluster.

Results

After clustering, we discovered a large number of clusters contained only a single movie. As the users in these clusters would not receive any recommendations for movies if they remained in their current cluster. As such, we removed the users from those single user clusters and combined them into a single cluster ‘uncategorized’. This ensured that those users could still receive recommendations, although they may not be as reliable as recommendations for users who are still in their original clusters. This allowed us to reduce the number of overall clusters from 82 to 11.

```
Length of total clusters before fixing is ->  82
Max value in users_cluster dataframe column Cluster is ->  81
And dataframe is following
```

	0	1	2	3	4	5	6	7	8	9	...	305	306	307	308	309	310	311	312	313	314
userId	1	3	4	5	6	7	8	9	11	12	...	339	340	341	342	343	344	345	346	347	348
Cluster	13	0	74	77	7	5	68	0	24	0	...	3	9	15	0	5	38	25	32	68	5

Figure 7: Clusters before fixing

```
Length of total clusters after fixing is ->  11
Max value in users_cluster dataframe column Cluster is ->  10
And fixed dataframe is following
```

	0	1	2	3	4	5	6	7	8	9	...	305	306	307	308	309	310	311	312	313	314
userId	1	3	4	5	6	7	8	9	11	12	...	339	340	341	342	343	344	345	346	347	348
Cluster	5	0	8	10	5	3	4	0	5	0	...	1	4	6	0	3	9	7	8	4	3

Figure 8: Clusters after fixing

Recommendations

User 4 Recommendations

```
Movie title:  ['Get Shorty (1995)'] , Genres: [['Comedy|Crime|Thriller'] ]
Movie title:  ['Twelve Monkeys (a.k.a. 12 Monkeys) (1995)'] , Genres:
[['Mystery|Sci-Fi|Thriller'] ]
Movie title:  ['To Die For (1995)'] , Genres: [['Comedy|Drama|Thriller'] ]
Movie title:  ['Seven (a.k.a. Se7en) (1995)'] , Genres: [['Mystery|Thriller'] ]
Movie title:  ['Mighty Aphrodite (1995)'] , Genres: [['Comedy|Drama|Romance'] ]
Movie title:  ['Postman, The (Postino, Il) (1994)'] , Genres:
[['Comedy|Drama|Romance'] ]
Movie title:  ['Nobody Loves Me (Keiner liebt mich) (1994)'] , Genres:
[['Comedy|Drama'] ]
Movie title:  ['Flirting With Disaster (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['NeverEnding Story III, The (1994)'] , Genres:
[['Adventure|Children|Fantasy'] ]
Movie title:  ['Crumb (1994)'] , Genres: [['Documentary'] ]
...
```

Figure 9: Movies watched by User 4

```
Movie title:  ['Taxi Driver (1976)'] , Genres: [['Crime|Drama|Thriller'] , ]
Movie title:  ['Usual Suspects, The (1995)'] , Genres: [['Crime|Mystery|Thriller']
, ]
Movie title:  ['Leaving Las Vegas (1995)'] , Genres: [['Drama|Romance'] , ]
Movie title:  ['City of Lost Children, The (Cité des enfants perdus, La) (1995)'] ,
Genres: [['Adventure|Drama|Fantasy|Mystery|Sci-Fi'] , ]
Movie title:  ['Dead Man Walking (1995)'] , Genres: [['Crime|Drama'] , ]
Movie title:  ['Bottle Rocket (1996)'] , Genres:
[['Adventure|Comedy|Crime|Romance'] , ]
Movie title:  ['Toy Story (1995)'] , Genres:
[['Adventure|Animation|Children|Comedy|Fantasy'] , ]
Movie title:  ['Birdcage, The (1996)'] , Genres: [['Comedy'] , ]
Movie title:  ['Braveheart (1995)'] , Genres: [['Action|Drama|War'] , ]
Movie title:  ['Beauty of the Day (Belle de jour) (1967)'] , Genres: [['Drama'] , ]
Movie title:  ['Jumanji (1995)'] , Genres: [['Adventure|Children|Fantasy'] , ]
Movie title:  ['Clerks (1994)'] , Genres: [['Comedy'] , ]
Movie title:  ["Things to Do in Denver When You're Dead (1995)"] , Genres:
[['Crime|Drama|Romance'] , ]
Movie title:  ['From Dusk Till Dawn (1996)'] , Genres:
[['Action|Comedy|Horror|Thriller'] , ]
Movie title:  ['American President, The (1995)'] , Genres:
[['Comedy|Drama|Romance'] , ]
```

Figure 10: Recommendations for User 4

User 42 Recommendations

```
Movie title:  ['Grumpier Old Men (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['Sabrina (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['GoldenEye (1995)'] , Genres: [['Action|Adventure|Thriller'] ]
Movie title:  ['American President, The (1995)'] , Genres:
[['Comedy|Drama|Romance'] ]
Movie title:  ['Casino (1995)'] , Genres: [['Crime|Drama'] ]
Movie title:  ['Ace Ventura: When Nature Calls (1995)'] , Genres: [['Comedy'] ]
Movie title:  ['Get Shorty (1995)'] , Genres: [['Comedy|Crime|Thriller'] ]
Movie title:  ['Copycat (1995)'] , Genres: [['Crime|Drama|Horror|Mystery|Thriller']
]
Movie title:  ['Seven (a.k.a. Se7en) (1995)'] , Genres: [['Mystery|Thriller'] ]
Movie title:  ['Usual Suspects, The (1995)'] , Genres: [['Crime|Mystery|Thriller']
]
...
```

Figure 11: Movies watched by User 42

```
Movie title:  ['Toy Story (1995)'] , Genres:
[['Adventure|Animation|Children|Comedy|Fantasy'] , ]
Movie title:  ['Twelve Monkeys (a.k.a. 12 Monkeys) (1995)'] , Genres:
[['Mystery|Sci-Fi|Thriller'] , ]
Movie title:  ['Heat (1995)'] , Genres: [['Action|Crime|Thriller'] , ]
Movie title:  ['Clueless (1995)'] , Genres: [['Comedy|Romance'] , ]
Movie title:  ['Jumanji (1995)'] , Genres: [['Adventure|Children|Fantasy'] , ]
Movie title:  ['Sense and Sensibility (1995)'] , Genres: [['Drama|Romance'] , ]
Movie title:  ['From Dusk Till Dawn (1996)'] , Genres:
[['Action|Comedy|Horror|Thriller'] , ]
Movie title:  ['Babe (1995)'] , Genres: [['Children|Drama'] , ]
Movie title:  ['Before Sunrise (1995)'] , Genres: [['Drama|Romance'] , ]
Movie title:  ["Mr. Holland's Opus (1995)"] , Genres: [['Drama'] , ]
Movie title:  ['Smoke (1995)'] , Genres: [['Comedy|Drama'] , ]
Movie title:  ['Mortal Kombat (1995)'] , Genres: [['Action|Adventure|Fantasy'] , ]
Movie title:  ['Taxi Driver (1976)'] , Genres: [['Crime|Drama|Thriller'] , ]
Movie title:  ['Father of the Bride Part II (1995)'] , Genres: [['Comedy'] , ]
Movie title:  ['Billy Madison (1995)'] , Genres: [['Comedy'] , ]
```

Figure 12: Recommendations for User 42

User 314 Recommendations

```
Movie title:  ['Toy Story (1995)'] , Genres:
[['Adventure|Animation|Children|Comedy|Fantasy'] ]
Movie title:  ['Heat (1995)'] , Genres: [['Action|Crime|Thriller'] ]
Movie title:  ['Sabrina (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['GoldenEye (1995)'] , Genres: [['Action|Adventure|Thriller'] ]
Movie title:  ['American President, The (1995)'] , Genres:
[['Comedy|Drama|Romance'] ]
Movie title:  ['Cutthroat Island (1995)'] , Genres: [['Action|Adventure|Romance'] ]
Movie title:  ['Sense and Sensibility (1995)'] , Genres: [['Drama|Romance'] ]
Movie title:  ['Get Shorty (1995)'] , Genres: [['Comedy|Crime|Thriller'] ]
Movie title:  ['Copycat (1995)'] , Genres: [['Crime|Drama|Horror|Mystery|Thriller']
]
Movie title:  ['Assassins (1995)'] , Genres: [['Action|Crime|Thriller'] ]
...
```

Figure 13: Movies watched by User 314

```
Movie title:  ['Dumb & Dumber (Dumb and Dumber) (1994)'] , Genres:
[['Adventure|Comedy'] , ]
Movie title:  ['Jumanji (1995)'] , Genres: [['Adventure|Children|Fantasy'] , ]
Movie title:  ['Ace Ventura: When Nature Calls (1995)'] , Genres: [['Comedy'] , ]
Movie title:  ['Happy Gilmore (1996)'] , Genres: [['Comedy'] , ]
Movie title:  ['From Dusk Till Dawn (1996)'] , Genres:
[['Action|Comedy|Horror|Thriller'] , ]
Movie title:  ['Babe (1995)'] , Genres: [['Children|Drama'] , ]
Movie title:  ['Clerks (1994)'] , Genres: [['Comedy'] , ]
Movie title:  ['Before Sunrise (1995)'] , Genres: [['Drama|Romance'] , ]
Movie title:  ['Smoke (1995)'] , Genres: [['Comedy|Drama'] , ]
Movie title:  ['Taxi Driver (1976)'] , Genres: [['Crime|Drama|Thriller'] , ]
Movie title:  ['Disclosure (1994)'] , Genres: [['Drama|Thriller'] , ]
Movie title:  ['Father of the Bride Part II (1995)'] , Genres: [['Comedy'] , ]
Movie title:  ['First Knight (1995)'] , Genres: [['Action|Drama|Romance'] , ]
Movie title:  ['Billy Madison (1995)'] , Genres: [['Comedy'] , ]
Movie title:  ['Grumpier Old Men (1995)'] , Genres: [['Comedy|Romance'] , ]
```

Figure 14: Recommendations for User 314

As can be seen in Figures 9-14, our model provides relatively accurate recommendations for each user, with at least one of the recommended movie's genres existing in the user's watched movies. We are confident this pattern is followed by all other users in the dataset.

Problem 2: Semantic Person Search

Pre-processing

The data used consisted of a set of png files and a csv containing the labels. The training data and test data was already split 520 images for training and 196 for testing. As the dataset was already really small, we decided to accept the unknown class as its own class rather than missing data. Removing it would limit our data too much. The images were all of different sizes, with the smallest being 108x53 and the largest being 429x267. As all the images need to be the same size for the model, they were all resized to 268x160, the middle value between largest and smallest. This also helps to reduce runtime as the images are quite large. The csv had all irrelevant values removed and the rest was split into their own arrays. The segmentation data was ignored as it wasnt available with the test set.

Description of Approach

For this problem, we created a neural network with 6 different outputs. Each output was each of the traits to be classified; gender, torso clothing type, torso colour, leg clothing type, leg colour and luggage. Originally the model was a small model consisting of 3 convolutions before splitting off into each trait. The full plot for this can be seen in appendix 3. This produced bad results so we added some convolutions after splitting into each trait. This still had poor performance so we used the 3rd example from week 4 example 2. This is a much deeper network with more convolutions and some dropout to help with overfitting. Each trait was also split immediately giving each trait the entire network. This significantly increased training time so it was only trained for 20 epochs. This network can be seen fully in appendix 4.

Evaluation of Performance

Epoch	Gender	Torso Type	Torso Colour	Leg Type	Leg Colour	Luggage
16	17%	80%	45%	13%	15%	17%
17	32%	80%	13%	12%	14%	16%
18	5%	84%	9%	10%	16%	18%
19	1%	84%	9%	14%	14%	17%
20	13%	85%	10%	12%	12%	19%

Figure 15: Training accuracy for last 5 epochs

Epoch	Gender	Torso Type	Torso Colour	Leg Type	Leg Colour	Luggage
16	45%	61%	22%	31%	5%	64%
17	0%	61%	22%	27%	5%	64%
28	5%	61%	22%	26%	5%	64%
19	12%	38%	22%	26%	5%	64%
20	45%	61%	22%	24%	5%	63%

Figure 15: Validation accuracy for last 5 epochs

Overall our model performed poorly for all traits except torso type. The model behaved quite bizarrely for the rest of the traits. For gender, it's quite obviously guessing for both training and validation. With only 3 values to choose from, it explains the mix between high and low values. The images were quite low res and included the whole body. There are also a large amount of shots from behind. This makes it hard to see features that indicate gender, especially the face.

Looking at the data manually, it is hard to determine gender by eye so it makes sense that the model struggled with this category. Torso type is the only trait to have reasonable performance. Finishing at 85% for training and 61% for validation. Most images have a good shot of the torso allowing for features to be extracted. Torso Colour seems to be guessing as well, indicated by the low, varied accuracy. The validation accuracy being a consistent 22% is odd. This may indicate some user error in coding the network. Leg type and leg colour both have similarly bad training performance. It once again indicates that the model is guessing. Most of the images are shot from a higher angle, making the legs harder to see. This may indicate the bad performance in both these traits. The validation accuracy is once again strangely consistent for leg colour, maybe indicating user error. Luggage once again had poor performance in training but had an oddly high and consistent validation accuracy.

This model could be improved in a number of ways. Firstly there were some issues with the data. As all the images were different sizes, resizing them would remove some features, especially for the larger images. The images were also really inconsistent with angles and lighting. The dataset was also really small. A larger more consistent dataset would produce better results, and data augmentation could be used to increase the current dataset a bit. The model could also be adjusted to have both torso traits and both leg traits on the same branch for a bit before splitting off. This would reduce computation time allowing for more epochs to be added. Using a pre-trained model would also help improve accuracy.

References

Asad, S. M. (2020, August 19). *AI Movies Recommendation System with Clustering Based K-Means Algorithm*. Medium. https://asdkazmi.medium.com/ai-movies-recommendation-system-with-clustering-based-k-means-algorithm-f04467e02fcd

Appendix

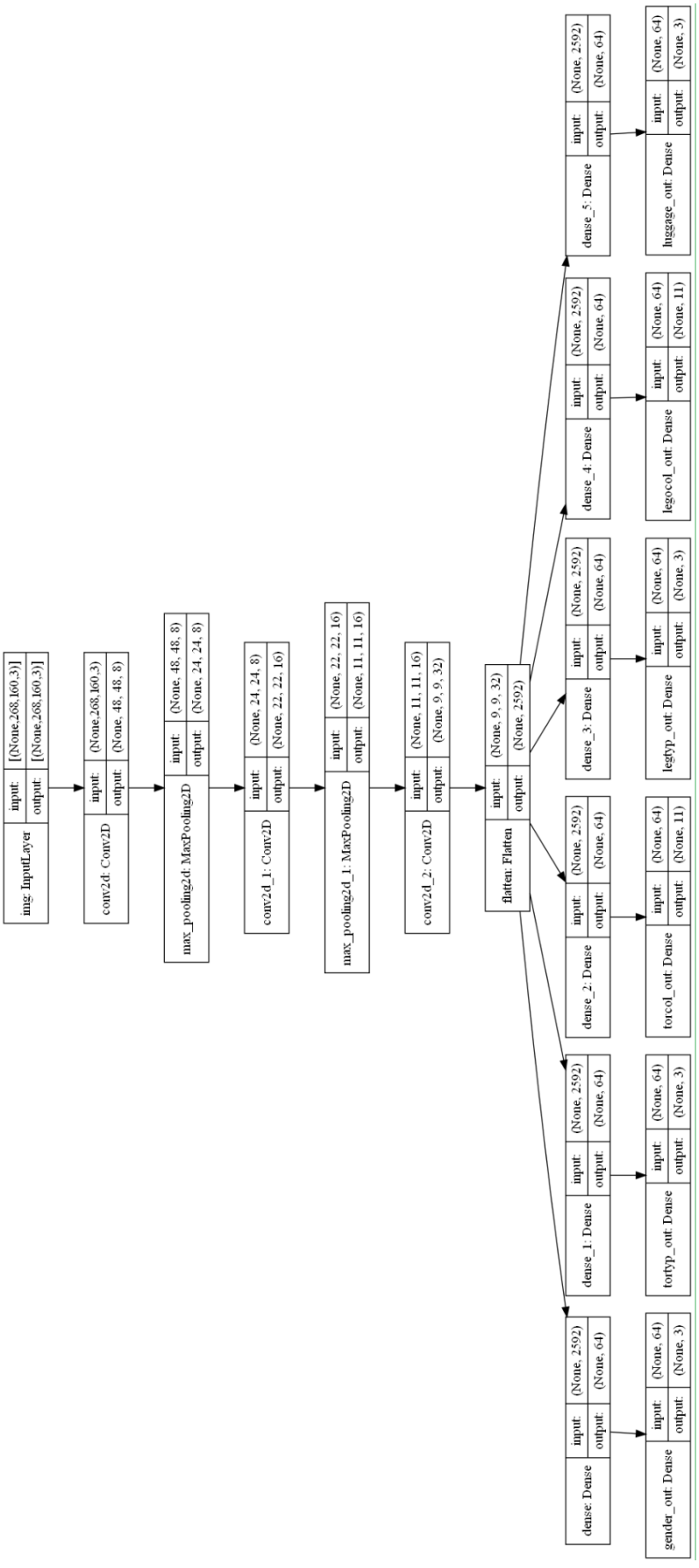
Appendix 1

	0	1	2	3	4	5	6	7	8	9	...
userId	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...
movieId	1.0	3.0	6.0	47.0	50.0	70.0	101.0	110.0	151.0	157.0	...
rating	4.0	4.0	4.0	5.0	5.0	3.0	5.0	4.0	5.0	5.0	...
timestamp	964982703.0	964981247.0	964982224.0	964983815.0	964982931.0	964982400.0	964980868.0	964982176.0	964984041.0	964984100.0	...

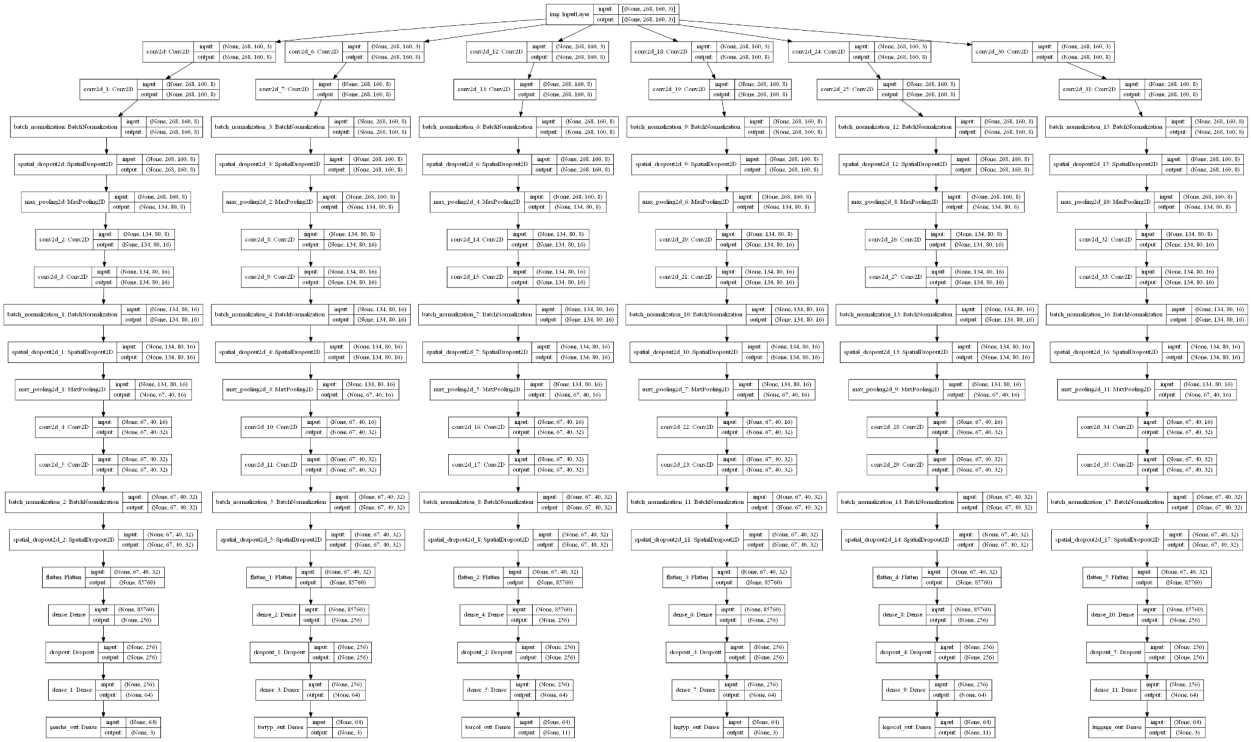
Appendix 2

	1	10	100	101	102	104	105	106	107	11	...	87	88	89	9	92	93	94	95	97	99
1	1	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
6	0	1	1	0	1	1	1	0	0	1	...	1	1	1	0	1	1	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
344	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
345	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
346	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
347	1	1	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
348	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Appendix 3


img: InputLayer
input: [(None, 268, 160, 3)]
output: [(None, 268, 160, 3)]
conv2d: Conv2D
input: (None, 268, 160, 3)
output: (None, 48, 48, 8)
max_pooling2d: MaxPooling2D
input: (None, 48, 48, 8)
output: (None, 24, 24, 8)
conv2d_1: Conv2D
input: (None, 24, 24, 8)
output: (None, 22, 22, 16)
max_pooling2d_1: MaxPooling2D
input: (None, 22, 22, 16)
output: (None, 11, 11, 16)
conv2d_2: Conv2D
input: (None, 11, 11, 16)
output: (None, 9, 9, 32)
flatten: Flatten
input: (None, 9, 9, 32)
output: (None, 2592)
dense: Dense
input: (None, 2592)
output: (None, 64)
gender_out: Dense
input: (None, 64)
output: (None, 3)
dense_1: Dense
input: (None, 2592)
output: (None, 64)
torfyp_out: Dense
input: (None, 64)
output: (None, 3)
dense_2: Dense
input: (None, 2592)
output: (None, 64)
torcol_out: Dense
input: (None, 64)
output: (None, 11)
dense_3: Dense
input: (None, 2592)
output: (None, 64)
legfyp_out: Dense
input: (None, 64)
output: (None, 3)
dense_4: Dense
input: (None, 2592)
output: (None, 64)
legocol_out: Dense
input: (None, 64)
output: (None, 11)
dense_5: Dense
input: (None, 2592)
output: (None, 64)
luggage_out: Dense
input: (None, 64)
output: (None, 3)

Appendix 4



Assignment 1C - Question 1

Clustering and Recommendations

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from scipy.spatial import distance
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering
from matplotlib import cm
from datetime import datetime
from sklearn.feature_extraction.text import CountVectorizer
import pickle
import ast
```

In [2]:

```
links = pd.read_csv('CAB420_Assessment_1C_Data/Data/Q1/links.csv')
movies_metadata = pd.read_csv('CAB420_Assessment_1C_Data/Data/Q1/movies.csv')
ratings = pd.read_csv('CAB420_Assessment_1C_Data/Data/Q1/ratings.csv')
tags = pd.read_csv('CAB420_Assessment_1C_Data/Data/Q1/tags.csv')
```

In [3]:

```
movies_metadata.columns
```

Out[3]:

```
Index(['movieId', 'title', 'genres'], dtype='object')
```

In [4]:

```
print(movies_metadata)
```

```
      movieId                                     title  \
0           1                         Toy Story (1995)
1           2                           Jumanji (1995)
2           3                  Grumpier Old Men (1995)
3           4                 Waiting to Exhale (1995)
4           5       Father of the Bride Part II (1995)
...       ...                                       ...
9737   193581  Black Butler: Book of the Atlantic (2017)
9738   193583              No Game No Life: Zero (2017)
9739   193585                             Flint (2017)
9740   193587       Bungo Stray Dogs: Dead Apple (2018)
9741   193609       Andrew Dice Clay: Dice Rules (1991)

                                       genres
0     Adventure|Animation|Children|Comedy|Fantasy
1                      Adventure|Children|Fantasy
2                                 Comedy|Romance
3                           Comedy|Drama|Romance
4                                         Comedy
...                                          ...
9737              Action|Animation|Comedy|Fantasy
9738                     Animation|Comedy|Fantasy
9739                                       Drama
9740                           Action|Animation
9741                                      Comedy

[9742 rows x 3 columns]
```

Filter ratings for 4+

In [5]:

```
print(ratings)
```

```
        userId  movieId  rating   timestamp
0            1        1     4.0   964982703
1            1        3     4.0   964981247
2            1        6     4.0   964982224
3            1       47     5.0   964983815
4            1       50     5.0   964982931
...        ...      ...     ...         ...
100831     610   166534     4.0  1493848402
100832     610   168248     5.0  1493850091
100833     610   168250     5.0  1494273047
100834     610   168252     5.0  1493846352
100835     610   170875     3.0  1493846415

[100836 rows x 4 columns]
```

In [6]:

```
ratings_4 = ratings[ratings['rating'] >= 4.0]
```

In [7]:

```
movies_list = np.unique(ratings['movieId'])[:200]
ratings = ratings.loc[ratings['movieId'].isin(movies_list)]
print('Shape of ratings dataset is: ',ratings.shape, '\n')
print('Max values in dataset are \n',ratings.max(), '\n')
print('Min values in dataset are \n',ratings.min(), '\n')
```

```
Shape of ratings dataset is:  (6351, 4)

Max values in dataset are
 userId       6.100000e+02
movieId      2.330000e+02
rating       5.000000e+00
timestamp    1.537799e+09
dtype: float64

Min values in dataset are
 userId               1.0
movieId              1.0
rating               0.5
timestamp    828124615.0
dtype: float64
```

In [8]:

```
users_list = np.unique(ratings['userId'])[:315]
ratings = ratings.loc[ratings['userId'].isin(users_list)]
print('Shape of ratings dataset is: ',ratings.shape, '\n')
print('Max values in dataset are \n',ratings.max(), '\n')
print('Min values in dataset are \n',ratings.min(), '\n')
print('Total Users: ', np.unique(ratings['userId']).shape[0])
print('Total Movies which are rated by 100 users: ', np.unique(ratings['movieId']).shap
e[0])
```

```
Shape of ratings dataset is:  (3505, 4)

Max values in dataset are
 userId       3.480000e+02
movieId      2.330000e+02
rating       5.000000e+00
timestamp    1.537158e+09
dtype: float64

Min values in dataset are
 userId               1.0
movieId              1.0
rating               0.5
timestamp    829322340.0
dtype: float64

Total Users:  315
Total Movies which are rated by 100 users:  187
```

In [9]:

```
users_fav_movies = ratings.loc[:, ['userId', 'movieId']]
```

In [10]:

```
users_fav_movies = ratings.reset_index(drop = True)
```

In [11]:

```
users_fav_movies.T
```

Out[11]:

	0	1	2	3	4	5
userId	1.0	1.0	1.0	1.0	1.0	1.0
movieId	1.0	3.0	6.0	47.0	50.0	70.0
rating	4.0	4.0	4.0	5.0	5.0	3.0
timestamp	964982703.0	964981247.0	964982224.0	964983815.0	964982931.0	964982400.0

4 rows × 3505 columns

In [12]:

```
def moviesListForUsers(users, users_data):
    # users = a list of users IDs
    # users_data = a dataframe of users favourite movies or users watched movies
    users_movies_list = []
    for user in users:
        users_movies_list.append(str(list(users_data[users_data['userId'] == user]['movieId'])).split('[')[1].split(']')[0])
    return users_movies_list
```

In [13]:

```
users = np.unique(users_fav_movies['userId'])
print(users.shape)
```

```
(315,)
```

In [14]:

```
users_movies_list = moviesListForUsers(users, users_fav_movies)
print('Movies list for', len(users_movies_list), ' users')
print('A list of first 10 users favourite movies: \n', users_movies_list[:10])
```

```
Movies list for 315  users
A list of first 10 users favourite movies:
 ['1, 3, 6, 47, 50, 70, 101, 110, 151, 157, 163, 216, 223, 231', '31', '2
1, 32, 45, 47, 52, 58, 106, 125, 126, 162, 171, 176, 190, 215, 222, 232',
'1, 21, 34, 36, 39, 50, 58, 110, 150, 153, 232', '2, 3, 4, 5, 6, 7, 8, 10,
11, 13, 15, 16, 17, 19, 21, 22, 24, 25, 26, 27, 31, 32, 34, 36, 41, 43, 4
5, 46, 47, 50, 54, 60, 61, 62, 65, 66, 76, 79, 86, 87, 88, 89, 92, 93, 95,
100, 102, 104, 105, 110, 112, 113, 126, 135, 140, 141, 145, 146, 150, 151,
153, 158, 159, 160, 161, 163, 165, 168, 170, 171, 174, 177, 179, 180, 181,
185, 186, 189, 191, 195, 196, 201, 204, 205, 207, 208, 209, 210, 212, 216,
217, 218, 219, 222, 224, 225, 230, 231', '1, 50, 58, 150, 165', '2, 10, 1
1, 21, 32, 34, 39, 47, 50, 110, 141, 150, 153, 185, 186, 208, 231', '41, 1
87, 223', '6, 10, 36, 44, 95, 110, 150, 153, 165, 170, 208', '39, 168, 22
2']
```

In [15]:

```
def prepSparseMatrix(list_of_str):
    # list_of_str = A list, which contain strings of users favourite movies separate by
comma ",".
    # It will return us sparse matrix and feature names on which sparse matrix is defin
ed
    # i.e. name of movies in the same order as the column of sparse matrix
    cv = CountVectorizer(token_pattern = r'[^\,\ ]+', lowercase = False)
    sparseMatrix = cv.fit_transform(list_of_str)
    return sparseMatrix.toarray(), cv.get_feature_names()
```

In [16]:

```
sparseMatrix, feature_names = prepSparseMatrix(users_movies_list)
```

In [17]:

```
df_sparseMatrix = pd.DataFrame(sparseMatrix, index = users, columns = feature_names)
df_sparseMatrix
```

Out[17]:

	1	10	100	101	102	104	105	106	107	11	...	87	88	89	9	92	93	94	95	97
1	1	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(
4	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	(
5	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(
6	0	1	1	0	1	1	1	0	0	1	...	1	1	1	0	1	1	0	1	(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
344	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(
345	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	(
346	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(
347	1	1	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	(
348	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(

315 rows × 187 columns

In [18]:

```
first_6_users_SM = users_fav_movies[users_fav_movies['userId'].isin(users[:6])].sort_values('userId')
first_6_users_SM.T
```

Out[18]:

	0	13	12	10	9	8
userId	1.0	1.0	1.0	1.0	1.0	1.0
movieId	1.0	231.0	223.0	163.0	157.0	151.0
rating	4.0	5.0	3.0	5.0	5.0	5.0
timestamp	964982703.0	964981179.0	964980985.0	964983650.0	964984100.0	964984041.0

4 rows × 145 columns

In [19]:

```
df_sparseMatrix.loc[np.unique(first_6_users_SM['userId']), list(map(str, np.unique(first_6_users_SM['movieId'])))]
```

Out[19]:

	1	2	3	4	5	6	7	8	10	11	...	217	218	219	222	223	224	225	230	231	232
1	1	0	1	0	0	1	0	0	0	0	...	0	0	0	0	1	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	1
5	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
6	0	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0	1	1	1	1	0
7	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

6 rows × 113 columns

In [20]:

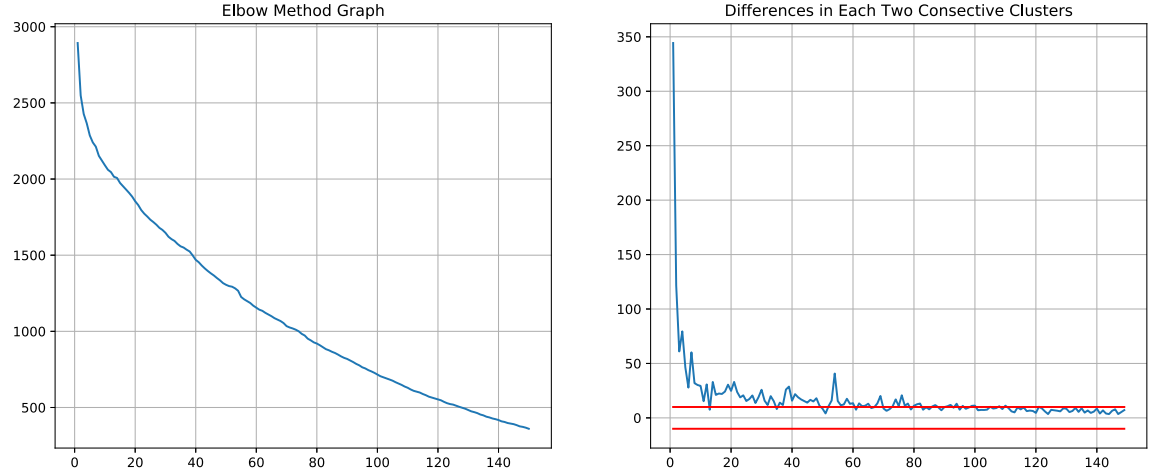
```
class elbowMethod():
    def __init__(self, sparseMatrix):
        self.sparseMatrix = sparseMatrix
        self.wcss = list()
        self.differences = list()
    def run(self, init, upto, max_iterations = 300):
        for i in range(init, upto + 1):
            kmeans = KMeans(n_clusters=i, init = 'k-means++', max_iter = max_iterations
, n_init = 10, random_state = 0)
            kmeans.fit(sparseMatrix)
            self.wcss.append(kmeans.inertia_)
        self.differences = list()
        for i in range(len(self.wcss)-1):
            self.differences.append(self.wcss[i] - self.wcss[i+1])
    def showPlot(self, boundary = 500, upto_cluster = None):
        if upto_cluster is None:
            WCSS = self.wcss
            DIFF = self.differences
        else:
            WCSS = self.wcss[:upto_cluster]
            DIFF = self.differences[:upto_cluster - 1]
        plt.figure(figsize=(15, 6))
        plt.subplot(121).set_title('Elbow Method Graph')
        plt.plot(range(1, len(WCSS) + 1), WCSS)
        plt.grid(b = True)
        plt.subplot(122).set_title('Differences in Each Two Consective Clusters')
        len_differences = len(DIFF)
        X_differences = range(1, len_differences + 1)
        plt.plot(X_differences, DIFF)
        plt.plot(X_differences, np.ones(len_differences)*boundary, 'r')
        plt.plot(X_differences, np.ones(len_differences)*(-boundary), 'r')
        plt.grid()
        plt.show()
```

In [21]:

```
elbow_method = elbowMethod(sparseMatrix)
```

In [22]:

```
elbow_method.run(1, 150)
elbow_method.showPlot(boundary = 10)
```



In [23]:

```
kmeans = KMeans(n_clusters=82, init = 'k-means++', max_iter = 300, n_init = 10, random_
state = 0)
clusters = kmeans.fit_predict(sparseMatrix)
```

In [24]:

```
users_cluster = pd.DataFrame(np.concatenate((users.reshape(-1,1), clusters.reshape(-1,1
)), axis = 1), columns = ['userId', 'Cluster'])
users_cluster.T
```

Out[24]:

	0	1	2	3	4	5	6	7	8	9	...	305	306	307	308	309	310	311	312	3
userId	1	3	4	5	6	7	8	9	11	12	...	339	340	341	342	343	344	345	346	3
Cluster	13	0	74	77	7	5	68	0	24	0	...	3	9	15	0	5	38	25	32	

2 rows × 315 columns

In [25]:

```
def clustersMovies(users_cluster, users_data):
    clusters = list(users_cluster['Cluster'])
    each_cluster_movies = list()
    for i in range(len(np.unique(clusters))):
        users_list = list(users_cluster[users_cluster['Cluster'] == i]['userId'])
        users_movies_list = list()
        for user in users_list:
            users_movies_list.extend(list(users_data[users_data['userId'] == user]['mov
ieId']))
        users_movies_counts = list()
        users_movies_counts.extend([[movie, users_movies_list.count(movie)] for movie i
n np.unique(users_movies_list)])
        each_cluster_movies.append(pd.DataFrame(users_movies_counts, columns=['movieId'
, 'Count']).sort_values(by = ['Count'], ascending = False).reset_index(drop=True))
    return each_cluster_movies
cluster_movies = clustersMovies(users_cluster, users_fav_movies)
```

In [26]:

```
cluster_movies[1].T
```

Out[26]:

	0	1	2	3	4	5	6	7	8	9	...	23	24	25	26	27	28	29
movieId	1	34	231	110	165	153	32	47	10	208	...	173	170	163	161	158	29	3
Count	5	5	5	4	4	4	4	4	4	3	...	1	1	1	1	1	1	1

2 rows × 33 columns

In [27]:

```
for i in range(42):
    len_users = users_cluster[users_cluster['Cluster'] == i].shape[0]
    print('Users in Cluster ' + str(i) + ' -> ', len_users)
```

```
Users in Cluster 0 ->  52
Users in Cluster 1 ->  5
Users in Cluster 2 ->  1
Users in Cluster 3 ->  7
Users in Cluster 4 ->  8
Users in Cluster 5 ->  34
Users in Cluster 6 ->  1
Users in Cluster 7 ->  1
Users in Cluster 8 ->  1
Users in Cluster 9 ->  10
Users in Cluster 10 ->  1
Users in Cluster 11 ->  1
Users in Cluster 12 ->  1
Users in Cluster 13 ->  23
Users in Cluster 14 ->  5
Users in Cluster 15 ->  27
Users in Cluster 16 ->  1
Users in Cluster 17 ->  1
Users in Cluster 18 ->  1
Users in Cluster 19 ->  1
Users in Cluster 20 ->  1
Users in Cluster 21 ->  1
Users in Cluster 22 ->  1
Users in Cluster 23 ->  1
Users in Cluster 24 ->  4
Users in Cluster 25 ->  10
Users in Cluster 26 ->  1
Users in Cluster 27 ->  1
Users in Cluster 28 ->  1
Users in Cluster 29 ->  3
Users in Cluster 30 ->  1
Users in Cluster 31 ->  1
Users in Cluster 32 ->  16
Users in Cluster 33 ->  1
Users in Cluster 34 ->  1
Users in Cluster 35 ->  1
Users in Cluster 36 ->  1
Users in Cluster 37 ->  4
Users in Cluster 38 ->  21
Users in Cluster 39 ->  1
Users in Cluster 40 ->  1
Users in Cluster 41 ->  1
```

In [28]:

```
def getMoviesOfUser(user_id, users_data):
    return list(users_data[users_data['userId'] == user_id]['movieId'])
```

In [29]:

```
def fixClusters(clusters_movies_dataframes, users_cluster_dataframe, users_data, smalle
st_cluster_size = 11):
    # clusters_movies_dataframes: will be a list which will contain each dataframes of
 each cluster movies
    # users_cluster_dataframe: will be a dataframe which contain users IDs and their cl
uster no.
    # smallest_cluster_size: is a smallest cluster size which we want for a cluster to
 not remove
    each_cluster_movies = clusters_movies_dataframes.copy()
    users_cluster = users_cluster_dataframe.copy()
    # Let convert dataframe in each_cluster_movies to list with containing only movies
 IDs
    each_cluster_movies_list = [list(df['movieId']) for df in each_cluster_movies]
    # First we will prepair a list which containt lists of users in each cluster -> [[C
luster 0 Users], [Cluster 1 Users], ... ,[Cluster N Users]]
    usersInClusters = list()
    total_clusters = len(each_cluster_movies)
    for i in range(total_clusters):
        usersInClusters.append(list(users_cluster[users_cluster['Cluster'] == i]['userI
d']))
    uncategorizedUsers = list()
    i = 0
    # Now we will remove small clusters and put their users into another list named "un
categorizedUsers"
    # Also when we will remove a cluster, then we have also bring back cluster numbers
 of users which comes after deleting cluster
    # E.g. if we have deleted cluster 4 then their will be users whose clusters will be
5,6,7,..,N. So, we'll bring back those users cluster number to 4,5,6,...,N-1.
    for j in range(total_clusters):
        if len(usersInClusters[i]) < smallest_cluster_size:
            uncategorizedUsers.extend(usersInClusters[i])
            usersInClusters.pop(i)
            each_cluster_movies.pop(i)
            each_cluster_movies_list.pop(i)
            users_cluster.loc[users_cluster['Cluster'] > i, 'Cluster'] -= 1
            i -= 1
        i += 1
    for user in uncategorizedUsers:
        elemProbability = list()
        user_movies = getMoviesOfUser(user, users_data)
        if len(user_movies) == 0:
            print(user)
        user_missed_movies = list()
        for movies_list in each_cluster_movies_list:
            count = 0
            missed_movies = list()
            for movie in user_movies:
                if movie in movies_list:
                    count += 1
                else:
                    missed_movies.append(movie)
            elemProbability.append(count / len(user_movies))
            user_missed_movies.append(missed_movies)
        user_new_cluster = np.array(elemProbability).argmax()
        users_cluster.loc[users_cluster['userId'] == user, 'Cluster'] = user_new_cluste
r
        if len(user_missed_movies[user_new_cluster]) > 0:
            each_cluster_movies[user_new_cluster] = each_cluster_movies[user_new_cluste
r].append([{'movieId': new_movie, 'Count': 1} for new_movie in user_missed_movies[user_
```

```
new_cluster]], ignore_index = True)
    return each_cluster_movies, users_cluster
```

In [30]:

```
movies_df_fixed, clusters_fixed = fixClusters(cluster_movies, users_cluster, users_fav_
movies, smallest_cluster_size = 6)
```

In [31]:

```
j = 0
for i in range(15):
    len_users = users_cluster[users_cluster['Cluster'] == i].shape[0]
    if len_users < 6:
        print('Users in Cluster ' + str(i) + ' -> ', len_users)
        j += 1
print('Total Cluster which we want to remove -> ', j)
```

```
Users in Cluster 1 ->  5
Users in Cluster 2 ->  1
Users in Cluster 6 ->  1
Users in Cluster 7 ->  1
Users in Cluster 8 ->  1
Users in Cluster 10 ->  1
Users in Cluster 11 ->  1
Users in Cluster 12 ->  1
Users in Cluster 14 ->  5
Total Cluster which we want to remove ->  9
```

In [32]:

```
print('Length of total clusters before fixing is -> ', len(cluster_movies))
print('Max value in users_cluster dataframe column Cluster is -> ', users_cluster['Clus
ter'].max())
print('And dataframe is following')
users_cluster.T
```

```
Length of total clusters before fixing is ->  82
Max value in users_cluster dataframe column Cluster is ->  81
And dataframe is following
```

Out[32]:

	0	1	2	3	4	5	6	7	8	9	...	305	306	307	308	309	310	311	312	
userId	1	3	4	5	6	7	8	9	11	12	...	339	340	341	342	343	344	345	346	
Cluster	13	0	74	77	7	5	68	0	24	0	...	3	9	15	0	5	38	25	32	

2 rows × 315 columns

In [33]:

```
print('Length of total clusters after fixing is -> ', len(movies_df_fixed))
print('Max value in users_cluster dataframe column Cluster is -> ', clusters_fixed['Clu
ster'].max())
print('And fixed dataframe is following')
clusters_fixed.T
```

```
Length of total clusters after fixing is ->  11
Max value in users_cluster dataframe column Cluster is ->  10
And fixed dataframe is following
```

Out[33]:

	0	1	2	3	4	5	6	7	8	9	...	305	306	307	308	309	310	311	312	313
userId	1	3	4	5	6	7	8	9	11	12	...	339	340	341	342	343	344	345	346	347
Cluster	5	0	8	10	5	3	4	0	5	0	...	1	4	6	0	3	9	7	8	4

2 rows × 315 columns

In [34]:

```
print('Users cluster dataFrame for cluster 11 before fixing:')
users_cluster[users_cluster['Cluster'] == 11].T
```

```
Users cluster dataFrame for cluster 11 before fixing:
```

Out[34]:

	195
userId	219
Cluster	11

In [35]:

```
print('Users cluster dataFrame for cluster 4 after fixing which should be same as 11th
 cluster before fixing:')
clusters_fixed[clusters_fixed['Cluster'] == 4].T
```

```
Users cluster dataFrame for cluster 4 after fixing which should be same as
11th cluster before fixing:
```

Out[35]:

	6	11	23	43	53	54	60	75	99	107	...	124	125	132	134	139	162	216
userId	8	14	26	46	56	57	63	81	107	116	...	134	135	142	145	151	179	242
Cluster	4	4	4	4	4	4	4	4	4	4	...	4	4	4	4	4	4	4

2 rows × 22 columns

In [36]:

```
for i in range(len(movies_df_fixed)):
    len_users = clusters_fixed[clusters_fixed['Cluster'] == i].shape[0]
    print('Users in Cluster ' + str(i) + ' -> ', len_users)
```

```
Users in Cluster 0 ->  55
Users in Cluster 1 ->  8
Users in Cluster 2 ->  8
Users in Cluster 3 ->  35
Users in Cluster 4 ->  22
Users in Cluster 5 ->  82
Users in Cluster 6 ->  37
Users in Cluster 7 ->  10
Users in Cluster 8 ->  26
Users in Cluster 9 ->  21
Users in Cluster 10 ->  11
```

In [37]:

```
class saveLoadFiles:
    def save(self, filename, data):
        try:
            file = open('CAB420_Assessment_1C_Data/Data/' + filename + '.pkl', 'wb')
            pickle.dump(data, file)
        except:
            err = 'Error: {0}, {1}'.format(exc_info()[0], exc_info()[1])
            print(err)
            file.close()
            return [False, err]
        else:
            file.close()
            return [True]
    def load(self, filename):
        try:
            file = open('CAB420_Assessment_1C_Data/Data/' + filename + '.pkl', 'rb')
        except:
            err = 'Error: {0}, {1}'.format(exc_info()[0], exc_info()[1])
            print(err)
            file.close()
            return [False, err]
        else:
            data = pickle.load(file)
            file.close()
            return data
    def loadClusterMoviesDataset(self):
        return self.load('clusters_movies_dataset')
    def saveClusterMoviesDataset(self, data):
        return self.save('clusters_movies_dataset', data)
    def loadUsersClusters(self):
        return self.load('users_clusters')
    def saveUsersClusters(self, data):
        return self.save('users_clusters', data)
```

In [38]:

```
saveLoadFile = saveLoadFiles()
print(saveLoadFile.saveClusterMoviesDataset(movies_df_fixed))
print(saveLoadFile.saveUsersClusters(clusters_fixed))
```

```
[True]
[True]
```

In [39]:

```
load_movies_list, load_users_clusters = saveLoadFile.loadClusterMoviesDataset(), saveLo
adFile.loadUsersClusters()
print('Type of Loading list of Movies dataframes of 5 Clusters: ', type(load_movies_lis
t), ' and Length is: ', len(load_movies_list))
print('Type of Loading 100 Users clusters Data: ', type(load_users_clusters), ' and Sha
pe is: ', load_users_clusters.shape)
```

```
Type of Loading list of Movies dataframes of 5 Clusters:  <class 'list'>
and Length is:  11
Type of Loading 100 Users clusters Data:  <class 'pandas.core.frame.DataFr
ame'>  and Shape is:  (315, 2)
```

In [40]:

```
class userRequestedFor:
    def __init__(self, user_id, users_data):
        self.users_data = users_data.copy()
        self.user_id = user_id
        # Find User Cluster
        users_cluster = saveLoadFiles().loadUsersClusters()
        self.user_cluster = int(users_cluster[users_cluster['userId'] == self.user_id][
'Cluster'])
        # Load User Cluster Movies Dataframe
        self.movies_list = saveLoadFiles().loadClusterMoviesDataset()
        self.cluster_movies = self.movies_list[self.user_cluster] # dataframe
        self.cluster_movies_list = list(self.cluster_movies['movieId']) # list
    def updatedFavouriteMoviesList(self, new_movie_Id):
        if new_movie_Id in self.cluster_movies_list:
            self.cluster_movies.loc[self.cluster_movies['movieId'] == new_movie_Id, 'Co
unt'] += 1
        else:
            self.cluster_movies = self.cluster_movies.append([{'movieId':new_movie_Id,
'Count': 1}], ignore_index=True)
        self.cluster_movies.sort_values(by = ['Count'], ascending = False, inplace= Tru
e)
        self.movies_list[self.user_cluster] = self.cluster_movies
        saveLoadFiles().saveClusterMoviesDataset(self.movies_list)

    def recommendMostFavouriteMovies(self):
        try:
            user_movies = getMoviesOfUser(self.user_id, self.users_data)
            cluster_movies_list = self.cluster_movies_list.copy()
            for user_movie in user_movies:
                if user_movie in cluster_movies_list:
                    cluster_movies_list.remove(user_movie)
            return [True, cluster_movies_list]
        except KeyError:
            err = "User history does not exist"
            print(err)
            return [False, err]
        except:
            err = 'Error: {0}, {1}'.format(exc_info()[0], exc_info()[1])
            print(err)
            return [False, err]
```

In [41]:

```
movies_metadata = movies_metadata.loc[
    movies_metadata['movieId'].isin(list(map(str, np.unique(users_fav_movies['movieId'
])))))].reset_index(drop=True)
print('Let take a look at movie metadata for all those movies which we were had in our
 dataset')
movies_metadata
```

```
Let take a look at movie metadata for all those movies which we were had i
n our dataset
```

Out[41]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
182	229	Death and the Maiden (1994)	Drama|Thriller
183	230	Dolores Claiborne (1995)	Drama|Thriller
184	231	Dumb & Dumber (Dumb and Dumber) (1994)	Adventure|Comedy
185	232	Eat Drink Man Woman (Yin shi nan nu) (1994)	Comedy|Drama|Romance
186	233	Exotica (1994)	Drama

187 rows × 3 columns

In [42]:

```
print(list(movies_metadata.loc[movies_metadata['movieId'] == 39]['title']))
```

```
['Clueless (1995)']
```

In [43]:

```
user314Movies = getMoviesOfUser(314, users_fav_movies)
for movie in user314Movies:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['movieId'] == mo
vie]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(end = '\b\b]')
        print('')
```

```
Movie title:  ['Toy Story (1995)'] , Genres: [['Adventure|Animation|Childr
en|Comedy|Fantasy'] ]
Movie title:  ['Heat (1995)'] , Genres: [['Action|Crime|Thriller'] ]
Movie title:  ['Sabrina (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['GoldenEye (1995)'] , Genres: [['Action|Adventure|Thrille
r'] ]
Movie title:  ['American President, The (1995)'] , Genres: [['Comedy|Drama
|Romance'] ]
Movie title:  ['Cutthroat Island (1995)'] , Genres: [['Action|Adventure|Ro
mance'] ]
Movie title:  ['Sense and Sensibility (1995)'] , Genres: [['Drama|Romanc
e'] ]
Movie title:  ['Get Shorty (1995)'] , Genres: [['Comedy|Crime|Thriller'] ]
Movie title:  ['Copycat (1995)'] , Genres: [['Crime|Drama|Horror|Mystery|T
hriller'] ]
Movie title:  ['Assassins (1995)'] , Genres: [['Action|Crime|Thriller'] ]
Movie title:  ['Othello (1995)'] , Genres: [['Drama'] ]
Movie title:  ['Dangerous Minds (1995)'] , Genres: [['Drama'] ]
Movie title:  ['Twelve Monkeys (a.k.a. 12 Monkeys) (1995)'] , Genres: [['M
ystery|Sci-Fi|Thriller'] ]
Movie title:  ['Clueless (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['Richard III (1995)'] , Genres: [['Drama|War'] ]
Movie title:  ['Mortal Kombat (1995)'] , Genres: [['Action|Adventure|Fanta
sy'] ]
Movie title:  ['Seven (a.k.a. Se7en) (1995)'] , Genres: [['Mystery|Thrille
r'] ]
Movie title:  ['Usual Suspects, The (1995)'] , Genres: [['Crime|Mystery|Th
riller'] ]
Movie title:  ['Mighty Aphrodite (1995)'] , Genres: [['Comedy|Drama|Romanc
e'] ]
Movie title:  ['Indian in the Cupboard, The (1995)'] , Genres: [['Adventur
e|Children|Fantasy'] ]
Movie title:  ["Mr. Holland's Opus (1995)"] , Genres: [['Drama'] ]
Movie title:  ['Broken Arrow (1996)'] , Genres: [['Action|Adventure|Thrill
er'] ]
Movie title:  ['City Hall (1996)'] , Genres: [['Drama|Thriller'] ]
Movie title:  ['Muppet Treasure Island (1996)'] , Genres: [['Adventure|Chi
ldren|Comedy|Musical'] ]
Movie title:  ['Braveheart (1995)'] , Genres: [['Action|Drama|War'] ]
Movie title:  ['Flirting With Disaster (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['Birdcage, The (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['Apollo 13 (1995)'] , Genres: [['Adventure|Drama|IMAX'] ]
Movie title:  ['Rob Roy (1995)'] , Genres: [['Action|Drama|Romance|War'] ]
Movie title:  ['Batman Forever (1995)'] , Genres: [['Action|Adventure|Come
dy|Crime'] ]
Movie title:  ['Crimson Tide (1995)'] , Genres: [['Drama|Thriller|War'] ]
Movie title:  ['Desperado (1995)'] , Genres: [['Action|Romance|Western'] ]
Movie title:  ['Die Hard: With a Vengeance (1995)'] , Genres: [['Action|Cr
ime|Thriller'] ]
Movie title:  ['Johnny Mnemonic (1995)'] , Genres: [['Action|Sci-Fi|Thrill
er'] ]
Movie title:  ['Judge Dredd (1995)'] , Genres: [['Action|Crime|Sci-Fi'] ]
Movie title:  ['Lord of Illusions (1995)'] , Genres: [['Horror'] ]
Movie title:  ['Net, The (1995)'] , Genres: [['Action|Crime|Thriller'] ]
Movie title:  ['Species (1995)'] , Genres: [['Horror|Sci-Fi'] ]
Movie title:  ['To Wong Foo, Thanks for Everything! Julie Newmar (1995)']
, Genres: [['Comedy'] ]
Movie title:  ['Waterworld (1995)'] , Genres: [['Action|Adventure|Sci-Fi']
]
Movie title:  ['Boys on the Side (1995)'] , Genres: [['Comedy|Drama'] ]
Movie title:  ['Don Juan DeMarco (1995)'] , Genres: [['Comedy|Drama|Romanc
```

```
e'] ]
Movie title:  ['Drop Zone (1994)'] , Genres: [['Action|Thriller'] ]
```

In [44]:

```
user314Recommendations = userRequestedFor(314, users_fav_movies).recommendMostFavourite
Movies()[1]
for movie in user314Recommendations[:15]:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['id'] == str(mov
ie)]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(']', end = '')
        print()
```

```
Movie title:  ['Dumb & Dumber (Dumb and Dumber) (1994)'] , Genres: [['Adve
nture|Comedy'] , ]
Movie title:  ['Jumanji (1995)'] , Genres: [['Adventure|Children|Fantasy']
, ]
Movie title:  ['Ace Ventura: When Nature Calls (1995)'] , Genres: [['Comed
y'] , ]
Movie title:  ['Happy Gilmore (1996)'] , Genres: [['Comedy'] , ]
Movie title:  ['From Dusk Till Dawn (1996)'] , Genres: [['Action|Comedy|Ho
rror|Thriller'] , ]
Movie title:  ['Babe (1995)'] , Genres: [['Children|Drama'] , ]
Movie title:  ['Clerks (1994)'] , Genres: [['Comedy'] , ]
Movie title:  ['Before Sunrise (1995)'] , Genres: [['Drama|Romance'] , ]
Movie title:  ['Smoke (1995)'] , Genres: [['Comedy|Drama'] , ]
Movie title:  ['Taxi Driver (1976)'] , Genres: [['Crime|Drama|Thriller'] ,
]
Movie title:  ['Disclosure (1994)'] , Genres: [['Drama|Thriller'] , ]
Movie title:  ['Father of the Bride Part II (1995)'] , Genres: [['Comedy']
, ]
Movie title:  ['First Knight (1995)'] , Genres: [['Action|Drama|Romance']
, ]
Movie title:  ['Billy Madison (1995)'] , Genres: [['Comedy'] , ]
Movie title:  ['Grumpier Old Men (1995)'] , Genres: [['Comedy|Romance'] ,
]
```

In [45]:

```
user4Movies = getMoviesOfUser(4, users_fav_movies)
for movie in user4Movies:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['movieId'] == mo
vie]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(end = '\b\b]')
        print('')
```

```
Movie title:  ['Get Shorty (1995)'] , Genres: [['Comedy|Crime|Thriller'] ]
Movie title:  ['Twelve Monkeys (a.k.a. 12 Monkeys) (1995)'] , Genres: [['M
ystery|Sci-Fi|Thriller'] ]
Movie title:  ['To Die For (1995)'] , Genres: [['Comedy|Drama|Thriller'] ]
Movie title:  ['Seven (a.k.a. Se7en) (1995)'] , Genres: [['Mystery|Thrille
r'] ]
Movie title:  ['Mighty Aphrodite (1995)'] , Genres: [['Comedy|Drama|Romanc
e'] ]
Movie title:  ['Postman, The (Postino, Il) (1994)'] , Genres: [['Comedy|Dr
ama|Romance'] ]
Movie title:  ['Nobody Loves Me (Keiner liebt mich) (1994)'] , Genres:
[['Comedy|Drama'] ]
Movie title:  ['Flirting With Disaster (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['NeverEnding Story III, The (1994)'] , Genres: [['Adventure
|Children|Fantasy'] ]
Movie title:  ['Crumb (1994)'] , Genres: [['Documentary'] ]
Movie title:  ['Jeffrey (1995)'] , Genres: [['Comedy|Drama'] ]
Movie title:  ['Living in Oblivion (1995)'] , Genres: [['Comedy'] ]
Movie title:  ['Safe (1995)'] , Genres: [['Thriller'] ]
Movie title:  ['Before Sunrise (1995)'] , Genres: [['Drama|Romance'] ]
Movie title:  ['Circle of Friends (1995)'] , Genres: [['Drama|Romance'] ]
Movie title:  ['Eat Drink Man Woman (Yin shi nan nu) (1994)'] , Genres:
[['Comedy|Drama|Romance'] ]
```

In [46]:

```
user4Recommendations = userRequestedFor(4, users_fav_movies).recommendMostFavouriteMovi
es()[1]
for movie in user4Recommendations[:15]:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['id'] == str(mov
ie)]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(']', end = '')
        print()
```

```
Movie title:  ['Taxi Driver (1976)'] , Genres: [['Crime|Drama|Thriller'] ,
]
Movie title:  ['Usual Suspects, The (1995)'] , Genres: [['Crime|Mystery|Th
riller'] , ]
Movie title:  ['Leaving Las Vegas (1995)'] , Genres: [['Drama|Romance'] ,
]
Movie title:  ['City of Lost Children, The (Cité des enfants perdus, La)
(1995)'] , Genres: [['Adventure|Drama|Fantasy|Mystery|Sci-Fi'] , ]
Movie title:  ['Dead Man Walking (1995)'] , Genres: [['Crime|Drama'] , ]
Movie title:  ['Bottle Rocket (1996)'] , Genres: [['Adventure|Comedy|Crime
|Romance'] , ]
Movie title:  ['Toy Story (1995)'] , Genres: [['Adventure|Animation|Childr
en|Comedy|Fantasy'] , ]
Movie title:  ['Birdcage, The (1996)'] , Genres: [['Comedy'] , ]
Movie title:  ['Braveheart (1995)'] , Genres: [['Action|Drama|War'] , ]
Movie title:  ['Beauty of the Day (Belle de jour) (1967)'] , Genres: [['Dr
ama'] , ]
Movie title:  ['Jumanji (1995)'] , Genres: [['Adventure|Children|Fantasy']
, ]
Movie title:  ['Clerks (1994)'] , Genres: [['Comedy'] , ]
Movie title:  ["Things to Do in Denver When You're Dead (1995)"] , Genres:
[['Crime|Drama|Romance'] , ]
Movie title:  ['From Dusk Till Dawn (1996)'] , Genres: [['Action|Comedy|Ho
rror|Thriller'] , ]
Movie title:  ['American President, The (1995)'] , Genres: [['Comedy|Drama
|Romance'] , ]
```

In [47]:

```
user42Movies = getMoviesOfUser(42, users_fav_movies)
for movie in user42Movies:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['movieId'] == mo
vie]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(end = '\b\b]')
        print('')
```

```
Movie title:  ['Grumpier Old Men (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['Sabrina (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['GoldenEye (1995)'] , Genres: [['Action|Adventure|Thrille
r'] ]
Movie title:  ['American President, The (1995)'] , Genres: [['Comedy|Drama
|Romance'] ]
Movie title:  ['Casino (1995)'] , Genres: [['Crime|Drama'] ]
Movie title:  ['Ace Ventura: When Nature Calls (1995)'] , Genres: [['Comed
y'] ]
Movie title:  ['Get Shorty (1995)'] , Genres: [['Comedy|Crime|Thriller'] ]
Movie title:  ['Copycat (1995)'] , Genres: [['Crime|Drama|Horror|Mystery|T
hriller'] ]
Movie title:  ['Seven (a.k.a. Se7en) (1995)'] , Genres: [['Mystery|Thrille
r'] ]
Movie title:  ['Usual Suspects, The (1995)'] , Genres: [['Crime|Mystery|Th
riller'] ]
Movie title:  ['White Squall (1996)'] , Genres: [['Action|Adventure|Dram
a'] ]
Movie title:  ['Broken Arrow (1996)'] , Genres: [['Action|Adventure|Thrill
er'] ]
Movie title:  ['Happy Gilmore (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['Braveheart (1995)'] , Genres: [['Action|Drama|War'] ]
Movie title:  ['Boomerang (1992)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['Down Periscope (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['Birdcage, The (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['Brothers McMullen, The (1995)'] , Genres: [['Comedy'] ]
Movie title:  ['Apollo 13 (1995)'] , Genres: [['Adventure|Drama|IMAX'] ]
Movie title:  ['Batman Forever (1995)'] , Genres: [['Action|Adventure|Come
dy|Crime'] ]
Movie title:  ['Crimson Tide (1995)'] , Genres: [['Drama|Thriller|War'] ]
Movie title:  ['Desperado (1995)'] , Genres: [['Action|Romance|Western'] ]
Movie title:  ['Die Hard: With a Vengeance (1995)'] , Genres: [['Action|Cr
ime|Thriller'] ]
Movie title:  ['First Knight (1995)'] , Genres: [['Action|Drama|Romance']
]
Movie title:  ['Judge Dredd (1995)'] , Genres: [['Action|Crime|Sci-Fi'] ]
Movie title:  ['Net, The (1995)'] , Genres: [['Action|Crime|Thriller'] ]
Movie title:  ['Nine Months (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['Showgirls (1995)'] , Genres: [['Drama'] ]
Movie title:  ['Something to Talk About (1995)'] , Genres: [['Comedy|Drama
|Romance'] ]
Movie title:  ['Clerks (1994)'] , Genres: [['Comedy'] ]
Movie title:  ['Disclosure (1994)'] , Genres: [['Drama|Thriller'] ]
Movie title:  ['Drop Zone (1994)'] , Genres: [['Action|Thriller'] ]
Movie title:  ['Dumb & Dumber (Dumb and Dumber) (1994)'] , Genres: [['Adve
nture|Comedy'] ]
```

In [48]:

```
user42Recommendations = userRequestedFor(42, users_fav_movies).recommendMostFavouriteMo
vies()[1]
for movie in user42Recommendations[:15]:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['id'] == str(mov
ie)]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(']', end = '')
        print()
```

```
Movie title:  ['Toy Story (1995)'] , Genres: [['Adventure|Animation|Childr
en|Comedy|Fantasy'] , ]
Movie title:  ['Twelve Monkeys (a.k.a. 12 Monkeys) (1995)'] , Genres: [['M
ystery|Sci-Fi|Thriller'] , ]
Movie title:  ['Heat (1995)'] , Genres: [['Action|Crime|Thriller'] , ]
Movie title:  ['Clueless (1995)'] , Genres: [['Comedy|Romance'] , ]
Movie title:  ['Jumanji (1995)'] , Genres: [['Adventure|Children|Fantasy']
, ]
Movie title:  ['Sense and Sensibility (1995)'] , Genres: [['Drama|Romanc
e'] , ]
Movie title:  ['From Dusk Till Dawn (1996)'] , Genres: [['Action|Comedy|Ho
rror|Thriller'] , ]
Movie title:  ['Babe (1995)'] , Genres: [['Children|Drama'] , ]
Movie title:  ['Before Sunrise (1995)'] , Genres: [['Drama|Romance'] , ]
Movie title:  ["Mr. Holland's Opus (1995)"] , Genres: [['Drama'] , ]
Movie title:  ['Smoke (1995)'] , Genres: [['Comedy|Drama'] , ]
Movie title:  ['Mortal Kombat (1995)'] , Genres: [['Action|Adventure|Fanta
sy'] , ]
Movie title:  ['Taxi Driver (1976)'] , Genres: [['Crime|Drama|Thriller'] ,
]
Movie title:  ['Father of the Bride Part II (1995)'] , Genres: [['Comedy']
, ]
Movie title:  ['Billy Madison (1995)'] , Genres: [['Comedy'] , ]
```

In [ ]:

```

```

Assignment 1C - Question 2

Semantic Person Search

In [1]:
```
from keras.models import Sequential
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Activation, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers
import pandas as pd
import numpy as np
import glob
import cv2
import matplotlib.pyplot as plt
import keras
from keras import layers
from PIL import Image

import os
import datetime
import numpy

import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorboard import notebook
from tensorflow.keras.preprocessing.image import Iterator

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

import pydot
import IPython
from IPython.display import SVG
from tensorflow.keras.utils import model_to_dot, plot_model
import imageio
import cv2
```

In [2]:
```
train = pd.read_csv('CAB420_Assessment_1C_Data/Data/Q2/Q2/Train_Data/Train.csv')
test = pd.read_csv('CAB420_Assessment_1C_Data/Data/Q2/Q2/Test_Data/Test.csv')
```

In [3]:
```
train_img = []
gnd = []
files = glob.glob('CAB420_Assessment_1C_Data/Data/Q2/Q2/Train_Data/Originals/*.png')
for myfile in files:
    im = keras.preprocessing.image.load_img(myfile,target_size=(268,160))
    image = keras.preprocessing.image.img_to_array(im)
    gnd.append(myfile[58:])

    train_img.append(image)


train_img = np.array(train_img)
train_img = train_img.astype('float32') / 255.
```

In [4]:

```
print(train_img[0])
```

```
[[[0.27450982 0.2784314  0.24705882]
  [0.27450982 0.2784314  0.24705882]
  [0.27450982 0.2784314  0.24705882]
  ...
  [0.3372549  0.3372549  0.2901961 ]
  [0.34509805 0.34509805 0.29803923]
  [0.34117648 0.34117648 0.29411766]]

 [[0.2784314  0.28235295 0.2509804 ]
  [0.2784314  0.28235295 0.2509804 ]
  [0.2784314  0.28235295 0.2509804 ]
  ...
  [0.3372549  0.3372549  0.2901961 ]
  [0.34901962 0.34901962 0.3019608 ]
  [0.34509805 0.34509805 0.29803923]]

 [[0.2784314  0.28235295 0.2509804 ]
  [0.2784314  0.28235295 0.2509804 ]
  [0.2784314  0.28235295 0.2509804 ]
  ...
  [0.34117648 0.34117648 0.29411766]
  [0.3529412  0.3529412  0.30588236]
  [0.34901962 0.34901962 0.3019608 ]]

 ...

 [[0.30980393 0.30980393 0.2627451 ]
  [0.30588236 0.30588236 0.25882354]
  [0.29803923 0.29803923 0.2509804 ]
  ...
  [0.4509804  0.44313726 0.39215687]
  [0.45490196 0.44705883 0.39607844]
  [0.45490196 0.44705883 0.39607844]]

 [[0.30980393 0.30980393 0.2627451 ]
  [0.30588236 0.30588236 0.25882354]
  [0.29803923 0.29803923 0.2509804 ]
  ...
  [0.4392157  0.44313726 0.3882353 ]
  [0.44705883 0.4509804  0.39607844]
  [0.44705883 0.4509804  0.39607844]]

 [[0.30980393 0.30980393 0.2627451 ]
  [0.30588236 0.30588236 0.25882354]
  [0.29803923 0.29803923 0.2509804 ]
  ...
  [0.44313726 0.44705883 0.39215687]
  [0.45490196 0.45882353 0.40392157]
  [0.4509804  0.45490196 0.4       ]]]
```

In [5]:

```
test_img = []
test_gnd = []
files = glob.glob('CAB420_Assessment_1C_Data/Data/Q2/Q2/Test_Data/Originals/*.png')
for myfile in files:
    im = keras.preprocessing.image.load_img(myfile,target_size=(268,160))
    image = keras.preprocessing.image.img_to_array(im)
    test_gnd.append(myfile[58:])
    test_img.append(image)


test_img = np.array(test_img)
test_img = test_img.astype('float32') / 255.
```

In [6]:

```
train = train.drop(columns=['torcol2','torcol3','tortex','torcol3','legcol2','legcol
```

```
train_gender = train.iloc[:,1]
train_gender = np.asarray(train_gender)
train_tortyp = train.iloc[:,2]
train_tortyp = np.asarray(train_tortyp)
train_torcol = train.iloc[:,3]
train_torcol = np.asarray(train_torcol)
train_legtyp = train.iloc[:,4]
train_legtyp = np.asarray(train_legtyp)
train_legcol = train.iloc[:,5]
train_legcol = np.asarray(train_legcol)
train_luggage = train.iloc[:,6]
train_luggage = np.asarray(train_luggage)
```

In [7]:

```
test = test.drop(columns=['torcol2','torcol3','tortex','torcol3','legcol2','legcol3'

test_gender = test.iloc[:,1]
test_gender = np.asarray(test_gender)
test_tortyp = test.iloc[:,2]
test_tortyp = np.asarray(test_tortyp)
test_torcol = test.iloc[:,3]
test_torcol = np.asarray(test_torcol)
test_legtyp = test.iloc[:,4]
test_legtyp = np.asarray(test_legtyp)
test_legcol = test.iloc[:,5]
test_legcol = np.asarray(test_legcol)
test_luggage = test.iloc[:,6]
test_luggage = np.asarray(test_luggage)
```

In [8]:

```
#fig = plt.figure(figsize=[20, 20])
#for i in range(100):
#    ax = fig.add_subplot(10, 10, i + 1)
#    ax.imshow(train_img[i])
```

In [9]:

```
#,tortyp,torcol,legtyp,legcol,luggage]
#, train_tortyp,train_torcol,train_legtyp,train_legcol,train_luggage]
#, test_tortyp, test_torcol, test_legtyp, test_legcol, test_luggage]
```

In [10]:

```
inputs = keras.Input(shape=(268, 160, 3, ), name='img')


x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='swish')(
x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='swish')(
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='swish')
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='swish')
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='swish')
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='swish')
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='swish')(x)
x = layers.Dropout(0.5)(x)
```

```
x = layers.Dense(64, activation='swish')(x)
gender = layers.Dense(3, name='gender_out')(x)

x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='swish')(
x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='swish')(
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='swish')
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='swish')
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='swish')
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='swish')
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='swish')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(64, activation='swish')(x)
tortyp = layers.Dense(3, name='tortyp_out')(x)

x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='relu')(i
x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='relu')(x
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(64, activation='relu')(x)
torcol = layers.Dense(11, name='torcol_out')(x)

x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='relu')(i
x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='relu')(x
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(64, activation='relu')(x)
legtyp = layers.Dense(3, name='legtyp_out')(x)

x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='relu')(i
x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='relu')(x
```

```
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(64, activation='relu')(x)
legcol = layers.Dense(11, name='legocol_out')(x)

x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='relu')(i
x = layers.Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='relu')(x
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.Conv2D(filters=16, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu')(
x = layers.BatchNormalization()(x)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(64, activation='relu')(x)
luggage = layers.Dense(3, name='luggage_out')(x)

model_cnn = keras.Model(inputs=inputs, outputs=[gender, tortyp, torcol, legtyp, legc


# inputs = keras.Input(shape=(50, 50, 3, ), name='img')
# x = layers.Conv2D(filters=8, kernel_size=(3,3), activation='relu', padding='same')
# x = layers.MaxPool2D(pool_size=(2, 2))(x)
# x = layers.Conv2D(filters=16, kernel_size=(3,3), activation='relu', padding='same'
# x = layers.MaxPool2D(pool_size=(2, 2))(x)
# x = layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', padding='same'
# x = layers.Flatten()(x)
# x = layers.Dense(64, activation='relu')(x)
# outputs = layers.Dense(11, activation='softmax')(x)

# model_cnn = keras.Model(inputs=inputs, outputs=outputs, name='SVHN_CNN_Model')


model_cnn.summary()
```

```
batch_normalization_6 (BatchNor (None, 268, 160, 8)  32          conv2d_13[0][0]
batch_normalization_9 (BatchNor (None, 268, 160, 8)  32          conv2d_19[0][0]
batch_normalization_12 (BatchNo (None, 268, 160, 8)  32          conv2d_25[0][0]
```

```
batch_normalization_15 (BatchNo (None, 268, 160, 8)  32          conv2d_31[0][0]
_____________
spatial_dropout2d (SpatialDropo (None, 268, 160, 8)  0           batch_normalization
[0][0]
_____________
spatial_dropout2d_3 (SpatialDro (None, 268, 160, 8)  0           batch_normalization
_3[0][0]
_____________
spatial_dropout2d_6 (SpatialDro (None, 268, 160, 8)  0           batch_normalization
_6[0][0]
_____________
spatial_dropout2d_9 (SpatialDro (None, 268, 160, 8)  0           batch_normalization
_9[0][0]
_____________
spatial_dropout2d_12 (SpatialDr (None, 268, 160, 8)  0           batch_normalization
_12[0][0]
_____________
spatial_dropout2d_15 (SpatialDr (None, 268, 160, 8)  0           batch_normalization
_15[0][0]
_____________
max_pooling2d (MaxPooling2D)    (None, 134, 80, 8)   0           spatial_dropout2d
[0][0]
_____________
max_pooling2d_2 (MaxPooling2D)  (None, 134, 80, 8)   0           spatial_dropout2d_3
[0][0]
_____________
max_pooling2d_4 (MaxPooling2D)  (None, 134, 80, 8)   0           spatial_dropout2d_6
[0][0]
_____________
max_pooling2d_6 (MaxPooling2D)  (None, 134, 80, 8)   0           spatial_dropout2d_9
[0][0]
_____________
max_pooling2d_8 (MaxPooling2D)  (None, 134, 80, 8)   0           spatial_dropout2d_1
2[0][0]
_____________
max_pooling2d_10 (MaxPooling2D) (None, 134, 80, 8)   0           spatial_dropout2d_1
5[0][0]
_____________
conv2d_2 (Conv2D)               (None, 134, 80, 16)  1168        max_pooling2d[0][0]
_____________
conv2d_8 (Conv2D)               (None, 134, 80, 16)  1168        max_pooling2d_2[0]
[0]
_____________
conv2d_14 (Conv2D)              (None, 134, 80, 16)  1168        max_pooling2d_4[0]
[0]
_____________
conv2d_20 (Conv2D)              (None, 134, 80, 16)  1168        max_pooling2d_6[0]
[0]
_____________
conv2d_26 (Conv2D)              (None, 134, 80, 16)  1168        max_pooling2d_8[0]
[0]
```

```
__________________________________________________________________________________________________
conv2d_32 (Conv2D)              (None, 134, 80, 16)  1168        max_pooling2d_10[0]
[0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 134, 80, 16)  2320        conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 134, 80, 16)  2320        conv2d_8[0][0]
__________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 134, 80, 16)  2320        conv2d_14[0][0]
__________________________________________________________________________________________________
conv2d_21 (Conv2D)              (None, 134, 80, 16)  2320        conv2d_20[0][0]
__________________________________________________________________________________________________
conv2d_27 (Conv2D)              (None, 134, 80, 16)  2320        conv2d_26[0][0]
__________________________________________________________________________________________________
conv2d_33 (Conv2D)              (None, 134, 80, 16)  2320        conv2d_32[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 134, 80, 16)  64          conv2d_3[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 134, 80, 16)  64          conv2d_9[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 134, 80, 16)  64          conv2d_15[0][0]
__________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 134, 80, 16)  64          conv2d_21[0][0]
__________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 134, 80, 16)  64          conv2d_27[0][0]
__________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 134, 80, 16)  64          conv2d_33[0][0]
__________________________________________________________________________________________________
spatial_dropout2d_1 (SpatialDro (None, 134, 80, 16)  0           batch_normalization
_1[0][0]
__________________________________________________________________________________________________
spatial_dropout2d_4 (SpatialDro (None, 134, 80, 16)  0           batch_normalization
_4[0][0]
__________________________________________________________________________________________________
spatial_dropout2d_7 (SpatialDro (None, 134, 80, 16)  0           batch_normalization
_7[0][0]
__________________________________________________________________________________________________
spatial_dropout2d_10 (SpatialDr (None, 134, 80, 16)  0           batch_normalization
_10[0][0]
__________________________________________________________________________________________________
spatial_dropout2d_13 (SpatialDr (None, 134, 80, 16)  0           batch_normalization
_13[0][0]
__________________________________________________________________________________________________
spatial_dropout2d_16 (SpatialDr (None, 134, 80, 16)  0           batch_normalization
_16[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 67, 40, 16)   0           spatial_dropout2d_1
[0][0]
__________________________________________________________________________________________________
```

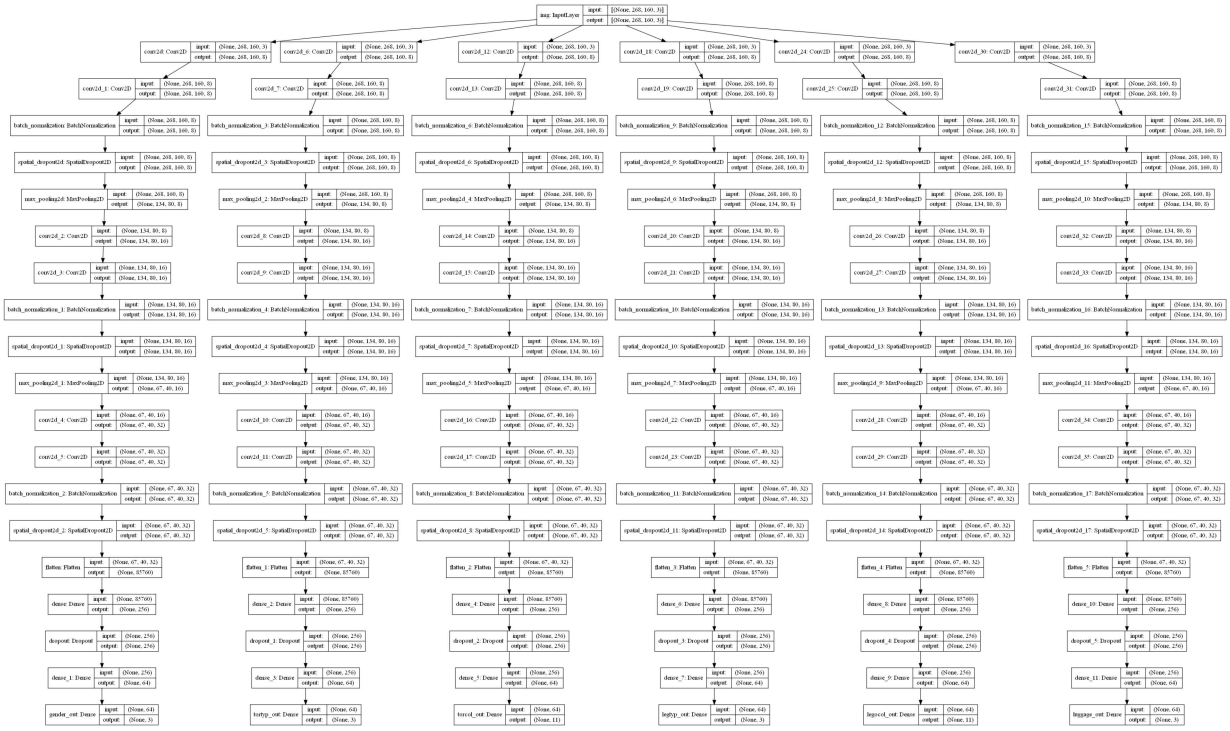
```
max_pooling2d_3 (MaxPooling2D)  (None, 67, 40, 16)   0           spatial_dropout2d_4
[0][0]
__________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)  (None, 67, 40, 16)   0           spatial_dropout2d_7
[0][0]
__________________________________________________________________________________
max_pooling2d_7 (MaxPooling2D)  (None, 67, 40, 16)   0           spatial_dropout2d_1
0[0][0]
__________________________________________________________________________________
max_pooling2d_9 (MaxPooling2D)  (None, 67, 40, 16)   0           spatial_dropout2d_1
3[0][0]
__________________________________________________________________________________
max_pooling2d_11 (MaxPooling2D) (None, 67, 40, 16)   0           spatial_dropout2d_1
6[0][0]
__________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 67, 40, 32)   4640        max_pooling2d_1[0]
[0]
__________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 67, 40, 32)   4640        max_pooling2d_3[0]
[0]
__________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 67, 40, 32)   4640        max_pooling2d_5[0]
[0]
__________________________________________________________________________________
conv2d_22 (Conv2D)              (None, 67, 40, 32)   4640        max_pooling2d_7[0]
[0]
__________________________________________________________________________________
conv2d_28 (Conv2D)              (None, 67, 40, 32)   4640        max_pooling2d_9[0]
[0]
__________________________________________________________________________________
conv2d_34 (Conv2D)              (None, 67, 40, 32)   4640        max_pooling2d_11[0]
[0]
__________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 67, 40, 32)   9248        conv2d_4[0][0]
__________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 67, 40, 32)   9248        conv2d_10[0][0]
__________________________________________________________________________________
conv2d_17 (Conv2D)              (None, 67, 40, 32)   9248        conv2d_16[0][0]
__________________________________________________________________________________
conv2d_23 (Conv2D)              (None, 67, 40, 32)   9248        conv2d_22[0][0]
__________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 67, 40, 32)   9248        conv2d_28[0][0]
__________________________________________________________________________________
conv2d_35 (Conv2D)              (None, 67, 40, 32)   9248        conv2d_34[0][0]
__________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 67, 40, 32)   128         conv2d_5[0][0]
__________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 67, 40, 32)   128         conv2d_11[0][0]
__________________________________________________________________________________
```

```
batch_normalization_8 (BatchNor (None, 67, 40, 32)   128         conv2d_17[0][0]
_____________
batch_normalization_11 (BatchNo (None, 67, 40, 32)   128         conv2d_23[0][0]
_____________
batch_normalization_14 (BatchNo (None, 67, 40, 32)   128         conv2d_29[0][0]
_____________
batch_normalization_17 (BatchNo (None, 67, 40, 32)   128         conv2d_35[0][0]
_____________
spatial_dropout2d_2 (SpatialDro (None, 67, 40, 32)   0           batch_normalization
_2[0][0]
_____________
spatial_dropout2d_5 (SpatialDro (None, 67, 40, 32)   0           batch_normalization
_5[0][0]
_____________
spatial_dropout2d_8 (SpatialDro (None, 67, 40, 32)   0           batch_normalization
_8[0][0]
_____________
spatial_dropout2d_11 (SpatialDr (None, 67, 40, 32)   0           batch_normalization
_11[0][0]
_____________
spatial_dropout2d_14 (SpatialDr (None, 67, 40, 32)   0           batch_normalization
_14[0][0]
_____________
spatial_dropout2d_17 (SpatialDr (None, 67, 40, 32)   0           batch_normalization
_17[0][0]
_____________
flatten (Flatten)               (None, 85760)        0           spatial_dropout2d_2
[0][0]
_____________
flatten_1 (Flatten)             (None, 85760)        0           spatial_dropout2d_5
[0][0]
_____________
flatten_2 (Flatten)             (None, 85760)        0           spatial_dropout2d_8
[0][0]
_____________
flatten_3 (Flatten)             (None, 85760)        0           spatial_dropout2d_1
1[0][0]
_____________
flatten_4 (Flatten)             (None, 85760)        0           spatial_dropout2d_1
4[0][0]
_____________
flatten_5 (Flatten)             (None, 85760)        0           spatial_dropout2d_1
7[0][0]
_____________
dense (Dense)                   (None, 256)          21954816    flatten[0][0]
_____________
dense_2 (Dense)                 (None, 256)          21954816    flatten_1[0][0]
_____________
dense_4 (Dense)                 (None, 256)          21954816    flatten_2[0][0]
_____________
```

```
dense_6 (Dense)                 (None, 256)          21954816    flatten_3[0][0]
__________________________________________________________________________________
dense_8 (Dense)                 (None, 256)          21954816    flatten_4[0][0]
__________________________________________________________________________________
dense_10 (Dense)                (None, 256)          21954816    flatten_5[0][0]
__________________________________________________________________________________
dropout (Dropout)               (None, 256)          0           dense[0][0]
__________________________________________________________________________________
dropout_1 (Dropout)             (None, 256)          0           dense_2[0][0]
__________________________________________________________________________________
dropout_2 (Dropout)             (None, 256)          0           dense_4[0][0]
__________________________________________________________________________________
dropout_3 (Dropout)             (None, 256)          0           dense_6[0][0]
__________________________________________________________________________________
dropout_4 (Dropout)             (None, 256)          0           dense_8[0][0]
__________________________________________________________________________________
dropout_5 (Dropout)             (None, 256)          0           dense_10[0][0]
__________________________________________________________________________________
dense_1 (Dense)                 (None, 64)           16448       dropout[0][0]
__________________________________________________________________________________
dense_3 (Dense)                 (None, 64)           16448       dropout_1[0][0]
__________________________________________________________________________________
dense_5 (Dense)                 (None, 64)           16448       dropout_2[0][0]
__________________________________________________________________________________
dense_7 (Dense)                 (None, 64)           16448       dropout_3[0][0]
__________________________________________________________________________________
dense_9 (Dense)                 (None, 64)           16448       dropout_4[0][0]
__________________________________________________________________________________
dense_11 (Dense)                (None, 64)           16448       dropout_5[0][0]
__________________________________________________________________________________
gender_out (Dense)              (None, 3)            195         dense_1[0][0]
__________________________________________________________________________________
tortyp_out (Dense)              (None, 3)            195         dense_3[0][0]
__________________________________________________________________________________
torcol_out (Dense)              (None, 11)           715         dense_5[0][0]
__________________________________________________________________________________
legtyp_out (Dense)              (None, 3)            195         dense_7[0][0]
__________________________________________________________________________________
legocol_out (Dense)             (None, 11)           715         dense_9[0][0]
__________________________________________________________________________________
luggage_out (Dense)             (None, 3)            195         dense_11[0][0]
==================================================================================
==============
Total params: 131,940,242
Trainable params: 131,939,570
Non-trainable params: 672
__________________________________________________________________________________
```

In [11]:
```
plot_model(model_cnn, to_file='test_keras_plot_model.png', show_shapes=True)
IPython.display.Image('test_keras_plot_model.png')
```

Out[11]:



In [12]:
```
model_cnn.compile(loss=['mean_squared_error', keras.losses.SparseCategoricalCrossent
                  optimizer=keras.optimizers.RMSprop(),metrics=['accuracy'])
```

In [13]:
```
history = model_cnn.fit(train_img, [train_gender, train_tortyp, train_torcol, train_
                        batch_size=64,
                        epochs=20,
                        validation_data=(test_img, [test_gender, test_tortyp, test_torco
```

```
Epoch 1/20
WARNING:tensorflow:Gradients do not exist for variables ['conv2d_12/kernel:0', 'conv
2d_12/bias:0', 'conv2d_18/kernel:0', 'conv2d_18/bias:0', 'conv2d_24/kernel:0', 'conv
2d_24/bias:0', 'conv2d_30/kernel:0', 'conv2d_30/bias:0', 'conv2d_13/kernel:0', 'conv
2d_13/bias:0', 'conv2d_19/kernel:0', 'conv2d_19/bias:0', 'conv2d_25/kernel:0', 'conv
2d_25/bias:0', 'conv2d_31/kernel:0', 'conv2d_31/bias:0', 'batch_normalization_6/gamm
a:0', 'batch_normalization_6/beta:0', 'batch_normalization_9/gamma:0', 'batch_normal
ization_9/beta:0', 'batch_normalization_12/gamma:0', 'batch_normalization_12/beta:
0', 'batch_normalization_15/gamma:0', 'batch_normalization_15/beta:0', 'conv2d_14/ke
rnel:0', 'conv2d_14/bias:0', 'conv2d_20/kernel:0', 'conv2d_20/bias:0', 'conv2d_26/ke
rnel:0', 'conv2d_26/bias:0', 'conv2d_32/kernel:0', 'conv2d_32/bias:0', 'conv2d_15/ke
rnel:0', 'conv2d_15/bias:0', 'conv2d_21/kernel:0', 'conv2d_21/bias:0', 'conv2d_27/ke
rnel:0', 'conv2d_27/bias:0', 'conv2d_33/kernel:0', 'conv2d_33/bias:0', 'batch_normal
ization_7/gamma:0', 'batch_normalization_7/beta:0', 'batch_normalization_10/gamma:
0', 'batch_normalization_10/beta:0', 'batch_normalization_13/gamma:0', 'batch_normal
ization_13/beta:0', 'batch_normalization_16/gamma:0', 'batch_normalization_16/beta:
0', 'conv2d_16/kernel:0', 'conv2d_16/bias:0', 'conv2d_22/kernel:0', 'conv2d_22/bias:
0', 'conv2d_28/kernel:0', 'conv2d_28/bias:0', 'conv2d_34/kernel:0', 'conv2d_34/bias:
0', 'conv2d_17/kernel:0', 'conv2d_17/bias:0', 'conv2d_23/kernel:0', 'conv2d_23/bias:
0', 'conv2d_29/kernel:0', 'conv2d_29/bias:0', 'conv2d_35/kernel:0', 'conv2d_35/bias:
0', 'batch_normalization_8/gamma:0', 'batch_normalization_8/beta:0', 'batch_normaliz
ation_11/gamma:0', 'batch_normalization_11/beta:0', 'batch_normalization_14/gamma:
0', 'batch_normalization_14/beta:0', 'batch_normalization_17/gamma:0', 'batch_normal
ization_17/beta:0', 'dense_4/kernel:0', 'dense_4/bias:0', 'dense_6/kernel:0', 'dense
_6/bias:0', 'dense_8/kernel:0', 'dense_8/bias:0', 'dense_10/kernel:0', 'dense_10/bia
s:0', 'dense_5/kernel:0', 'dense_5/bias:0', 'dense_7/kernel:0', 'dense_7/bias:0', 'd
ense_9/kernel:0', 'dense_9/bias:0', 'dense_11/kernel:0', 'dense_11/bias:0', 'torcol_
out/kernel:0', 'torcol_out/bias:0', 'legtyp_out/kernel:0', 'legtyp_out/bias:0', 'leg
ocol_out/kernel:0', 'legocol_out/bias:0', 'luggage_out/kernel:0', 'luggage_out/bias:
```

```
0'] when minimizing the loss.
WARNING:tensorflow:Gradients do not exist for variables ['conv2d_12/kernel:0', 'conv
2d_12/bias:0', 'conv2d_18/kernel:0', 'conv2d_18/bias:0', 'conv2d_24/kernel:0', 'conv
2d_24/bias:0', 'conv2d_30/kernel:0', 'conv2d_30/bias:0', 'conv2d_13/kernel:0', 'conv
2d_13/bias:0', 'conv2d_19/kernel:0', 'conv2d_19/bias:0', 'conv2d_25/kernel:0', 'conv
2d_25/bias:0', 'conv2d_31/kernel:0', 'conv2d_31/bias:0', 'batch_normalization_6/gamm
a:0', 'batch_normalization_6/beta:0', 'batch_normalization_9/gamma:0', 'batch_normal
ization_9/beta:0', 'batch_normalization_12/gamma:0', 'batch_normalization_12/beta:
0', 'batch_normalization_15/gamma:0', 'batch_normalization_15/beta:0', 'conv2d_14/ke
rnel:0', 'conv2d_14/bias:0', 'conv2d_20/kernel:0', 'conv2d_20/bias:0', 'conv2d_26/ke
rnel:0', 'conv2d_26/bias:0', 'conv2d_32/kernel:0', 'conv2d_32/bias:0', 'conv2d_15/ke
rnel:0', 'conv2d_15/bias:0', 'conv2d_21/kernel:0', 'conv2d_21/bias:0', 'conv2d_27/ke
rnel:0', 'conv2d_27/bias:0', 'conv2d_33/kernel:0', 'conv2d_33/bias:0', 'batch_normal
ization_7/gamma:0', 'batch_normalization_7/beta:0', 'batch_normalization_10/gamma:
0', 'batch_normalization_10/beta:0', 'batch_normalization_13/gamma:0', 'batch_normal
ization_13/beta:0', 'batch_normalization_16/gamma:0', 'batch_normalization_16/beta:
0', 'conv2d_16/kernel:0', 'conv2d_16/bias:0', 'conv2d_22/kernel:0', 'conv2d_22/bias:
0', 'conv2d_28/kernel:0', 'conv2d_28/bias:0', 'conv2d_34/kernel:0', 'conv2d_34/bias:
0', 'conv2d_17/kernel:0', 'conv2d_17/bias:0', 'conv2d_23/kernel:0', 'conv2d_23/bias:
0', 'conv2d_29/kernel:0', 'conv2d_29/bias:0', 'conv2d_35/kernel:0', 'conv2d_35/bias:
0', 'batch_normalization_8/gamma:0', 'batch_normalization_8/beta:0', 'batch_normaliz
ation_11/gamma:0', 'batch_normalization_11/beta:0', 'batch_normalization_14/gamma:
0', 'batch_normalization_14/beta:0', 'batch_normalization_17/gamma:0', 'batch_normal
ization_17/beta:0', 'dense_4/kernel:0', 'dense_4/bias:0', 'dense_6/kernel:0', 'dense
_6/bias:0', 'dense_8/kernel:0', 'dense_8/bias:0', 'dense_10/kernel:0', 'dense_10/bia
s:0', 'dense_5/kernel:0', 'dense_5/bias:0', 'dense_7/kernel:0', 'dense_7/bias:0', 'd
ense_9/kernel:0', 'dense_9/bias:0', 'dense_11/kernel:0', 'dense_11/bias:0', 'torcol_
out/kernel:0', 'torcol_out/bias:0', 'legtyp_out/kernel:0', 'legtyp_out/bias:0', 'leg
ocol_out/kernel:0', 'legocol_out/bias:0', 'luggage_out/kernel:0', 'luggage_out/bias:
0'] when minimizing the loss.
9/9 [==============================] - 38s 4s/step - loss: 685.1964 - gender_out_los
s: 668.8911 - tortyp_out_loss: 16.3054 - gender_out_accuracy: 0.3519 - tortyp_out_ac
curacy: 0.4827 - torcol_out_accuracy: 0.1096 - legtyp_out_accuracy: 0.1115 - legocol
_out_accuracy: 0.1596 - luggage_out_accuracy: 0.1654 - val_loss: 1.3725 - val_gender
_out_loss: 0.6103 - val_tortyp_out_loss: 0.7622 - val_gender_out_accuracy: 0.0000e+0
0 - val_tortyp_out_accuracy: 0.6173 - val_torcol_out_accuracy: 0.0816 - val_legtyp_o
ut_accuracy: 0.0102 - val_legocol_out_accuracy: 0.1071 - val_luggage_out_accuracy:
0.0408
Epoch 2/20
9/9 [==============================] - 33s 4s/step - loss: 138.9341 - gender_out_los
s: 132.3923 - tortyp_out_loss: 6.5418 - gender_out_accuracy: 0.3558 - tortyp_out_acc
uracy: 0.5673 - torcol_out_accuracy: 0.0942 - legtyp_out_accuracy: 0.1500 - legocol_
out_accuracy: 0.1519 - luggage_out_accuracy: 0.1962 - val_loss: 1.3203 - val_gender_
out_loss: 0.5989 - val_tortyp_out_loss: 0.7215 - val_gender_out_accuracy: 0.0000e+00
- val_tortyp_out_accuracy: 0.6173 - val_torcol_out_accuracy: 0.0765 - val_legtyp_out
_accuracy: 0.0714 - val_legocol_out_accuracy: 0.0255 - val_luggage_out_accuracy: 0.2
347
Epoch 3/20
9/9 [==============================] - 32s 4s/step - loss: 48.8480 - gender_out_los
s: 44.2301 - tortyp_out_loss: 4.6179 - gender_out_accuracy: 0.3500 - tortyp_out_accu
racy: 0.5654 - torcol_out_accuracy: 0.1000 - legtyp_out_accuracy: 0.1327 - legocol_o
ut_accuracy: 0.1904 - luggage_out_accuracy: 0.1981 - val_loss: 1.3568 - val_gender_o
ut_loss: 0.6302 - val_tortyp_out_loss: 0.7266 - val_gender_out_accuracy: 0.0000e+00
- val_tortyp_out_accuracy: 0.3827 - val_torcol_out_accuracy: 0.1378 - val_legtyp_out
_accuracy: 0.2143 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.4
847
Epoch 4/20
9/9 [==============================] - 33s 4s/step - loss: 29.0974 - gender_out_los
s: 25.6433 - tortyp_out_loss: 3.4541 - gender_out_accuracy: 0.3538 - tortyp_out_accu
racy: 0.5962 - torcol_out_accuracy: 0.1365 - legtyp_out_accuracy: 0.1096 - legocol_o
ut_accuracy: 0.1635 - luggage_out_accuracy: 0.2019 - val_loss: 1.5976 - val_gender_o
ut_loss: 0.6024 - val_tortyp_out_loss: 0.9953 - val_gender_out_accuracy: 0.1224 - va
l_tortyp_out_accuracy: 0.3827 - val_torcol_out_accuracy: 0.1939 - val_legtyp_out_acc
uracy: 0.3418 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.5969
Epoch 5/20
9/9 [==============================] - 32s 4s/step - loss: 21.0972 - gender_out_los
s: 18.2634 - tortyp_out_loss: 2.8338 - gender_out_accuracy: 0.3269 - tortyp_out_accu
racy: 0.6038 - torcol_out_accuracy: 0.1269 - legtyp_out_accuracy: 0.1404 - legocol_o
ut_accuracy: 0.1577 - luggage_out_accuracy: 0.1981 - val_loss: 1.4471 - val_gender_o
```

```
ut_loss: 0.5811 - val_tortyp_out_loss: 0.8659 - val_gender_out_accuracy: 0.5000 - va
l_tortyp_out_accuracy: 0.3827 - val_torcol_out_accuracy: 0.2245 - val_legtyp_out_acc
uracy: 0.4337 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6327
Epoch 6/20
9/9 [==============================] - 33s 4s/step - loss: 8.9105 - gender_out_loss:
6.6734 - tortyp_out_loss: 2.2371 - gender_out_accuracy: 0.3500 - tortyp_out_accurac
y: 0.6154 - torcol_out_accuracy: 0.1288 - legtyp_out_accuracy: 0.1327 - legocol_out_
accuracy: 0.1692 - luggage_out_accuracy: 0.1904 - val_loss: 1.2542 - val_gender_out_
loss: 0.5466 - val_tortyp_out_loss: 0.7076 - val_gender_out_accuracy: 0.5000 - val_t
ortyp_out_accuracy: 0.3827 - val_torcol_out_accuracy: 0.2245 - val_legtyp_out_accura
cy: 0.4745 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 7/20
9/9 [==============================] - 33s 4s/step - loss: 5.2348 - gender_out_loss:
3.2984 - tortyp_out_loss: 1.9364 - gender_out_accuracy: 0.3000 - tortyp_out_accurac
y: 0.6423 - torcol_out_accuracy: 0.1058 - legtyp_out_accuracy: 0.1519 - legocol_out_
accuracy: 0.1500 - luggage_out_accuracy: 0.2019 - val_loss: 1.1942 - val_gender_out_
loss: 0.5088 - val_tortyp_out_loss: 0.6854 - val_gender_out_accuracy: 0.5000 - val_t
ortyp_out_accuracy: 0.6173 - val_torcol_out_accuracy: 0.2296 - val_legtyp_out_accura
cy: 0.4847 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 8/20
9/9 [==============================] - 33s 4s/step - loss: 3.6165 - gender_out_loss:
1.6301 - tortyp_out_loss: 1.9863 - gender_out_accuracy: 0.3250 - tortyp_out_accurac
y: 0.6385 - torcol_out_accuracy: 0.1231 - legtyp_out_accuracy: 0.1462 - legocol_out_
accuracy: 0.1788 - luggage_out_accuracy: 0.2096 - val_loss: 1.5365 - val_gender_out_
loss: 0.5258 - val_tortyp_out_loss: 1.0107 - val_gender_out_accuracy: 0.5000 - val_t
ortyp_out_accuracy: 0.3827 - val_torcol_out_accuracy: 0.2296 - val_legtyp_out_accura
cy: 0.4694 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 9/20
9/9 [==============================] - 33s 4s/step - loss: 2.3035 - gender_out_loss:
1.1435 - tortyp_out_loss: 1.1599 - gender_out_accuracy: 0.2635 - tortyp_out_accurac
y: 0.6846 - torcol_out_accuracy: 0.1327 - legtyp_out_accuracy: 0.1385 - legocol_out_
accuracy: 0.1288 - luggage_out_accuracy: 0.1750 - val_loss: 2.3796 - val_gender_out_
loss: 1.6809 - val_tortyp_out_loss: 0.6987 - val_gender_out_accuracy: 0.5000 - val_t
ortyp_out_accuracy: 0.3827 - val_torcol_out_accuracy: 0.2245 - val_legtyp_out_accura
cy: 0.4643 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 10/20
9/9 [==============================] - 32s 4s/step - loss: 1.9065 - gender_out_loss:
0.7941 - tortyp_out_loss: 1.1125 - gender_out_accuracy: 0.2365 - tortyp_out_accurac
y: 0.7019 - torcol_out_accuracy: 0.1058 - legtyp_out_accuracy: 0.1212 - legocol_out_
accuracy: 0.1519 - luggage_out_accuracy: 0.1500 - val_loss: 1.4545 - val_gender_out_
loss: 0.7424 - val_tortyp_out_loss: 0.7121 - val_gender_out_accuracy: 0.5000 - val_t
ortyp_out_accuracy: 0.3827 - val_torcol_out_accuracy: 0.2398 - val_legtyp_out_accura
cy: 0.4082 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 11/20
9/9 [==============================] - 33s 4s/step - loss: 1.9955 - gender_out_loss:
0.6594 - tortyp_out_loss: 1.3361 - gender_out_accuracy: 0.1212 - tortyp_out_accurac
y: 0.6827 - torcol_out_accuracy: 0.1154 - legtyp_out_accuracy: 0.1673 - legocol_out_
accuracy: 0.1673 - luggage_out_accuracy: 0.1962 - val_loss: 1.4139 - val_gender_out_
loss: 0.7180 - val_tortyp_out_loss: 0.6960 - val_gender_out_accuracy: 0.5000 - val_t
ortyp_out_accuracy: 0.4031 - val_torcol_out_accuracy: 0.2347 - val_legtyp_out_accura
cy: 0.3776 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 12/20
9/9 [==============================] - 32s 4s/step - loss: 1.7898 - gender_out_loss:
0.4804 - tortyp_out_loss: 1.3095 - gender_out_accuracy: 0.1577 - tortyp_out_accurac
y: 0.6769 - torcol_out_accuracy: 0.1269 - legtyp_out_accuracy: 0.0981 - legocol_out_
accuracy: 0.1269 - luggage_out_accuracy: 0.1769 - val_loss: 1.3780 - val_gender_out_
loss: 0.5771 - val_tortyp_out_loss: 0.8010 - val_gender_out_accuracy: 0.5000 - val_t
ortyp_out_accuracy: 0.3827 - val_torcol_out_accuracy: 0.2347 - val_legtyp_out_accura
cy: 0.3622 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 13/20
9/9 [==============================] - 32s 4s/step - loss: 1.3396 - gender_out_loss:
0.4062 - tortyp_out_loss: 0.9334 - gender_out_accuracy: 0.1673 - tortyp_out_accurac
y: 0.7135 - torcol_out_accuracy: 0.1135 - legtyp_out_accuracy: 0.1442 - legocol_out_
accuracy: 0.1635 - luggage_out_accuracy: 0.1923 - val_loss: 1.1065 - val_gender_out_
loss: 0.3906 - val_tortyp_out_loss: 0.7159 - val_gender_out_accuracy: 0.0000e+00 - v
al_tortyp_out_accuracy: 0.3827 - val_torcol_out_accuracy: 0.2398 - val_legtyp_out_ac
curacy: 0.3469 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 14/20
9/9 [==============================] - 33s 4s/step - loss: 1.1128 - gender_out_loss:
```

```
0.3818 - tortyp_out_loss: 0.7310 - gender_out_accuracy: 0.0596 - tortyp_out_accurac
y: 0.7865 - torcol_out_accuracy: 0.1173 - legtyp_out_accuracy: 0.1346 - legocol_out_
accuracy: 0.1596 - luggage_out_accuracy: 0.2327 - val_loss: 1.0729 - val_gender_out_
loss: 0.4067 - val_tortyp_out_loss: 0.6661 - val_gender_out_accuracy: 0.4592 - val_t
ortyp_out_accuracy: 0.6173 - val_torcol_out_accuracy: 0.2398 - val_legtyp_out_accura
cy: 0.3367 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 15/20
9/9 [==============================] - 33s 4s/step - loss: 0.9705 - gender_out_loss:
0.3300 - tortyp_out_loss: 0.6405 - gender_out_accuracy: 0.0442 - tortyp_out_accurac
y: 0.7885 - torcol_out_accuracy: 0.0904 - legtyp_out_accuracy: 0.1519 - legocol_out_
accuracy: 0.1827 - luggage_out_accuracy: 0.1846 - val_loss: 1.0626 - val_gender_out_
loss: 0.3667 - val_tortyp_out_loss: 0.6959 - val_gender_out_accuracy: 0.4592 - val_t
ortyp_out_accuracy: 0.3878 - val_torcol_out_accuracy: 0.2296 - val_legtyp_out_accura
cy: 0.3214 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 16/20
9/9 [==============================] - 32s 4s/step - loss: 0.9160 - gender_out_loss:
0.3211 - tortyp_out_loss: 0.5949 - gender_out_accuracy: 0.0173 - tortyp_out_accurac
y: 0.8058 - torcol_out_accuracy: 0.1308 - legtyp_out_accuracy: 0.1346 - legocol_out_
accuracy: 0.1538 - luggage_out_accuracy: 0.1788 - val_loss: 1.0612 - val_gender_out_
loss: 0.3662 - val_tortyp_out_loss: 0.6950 - val_gender_out_accuracy: 0.4592 - val_t
ortyp_out_accuracy: 0.6173 - val_torcol_out_accuracy: 0.2296 - val_legtyp_out_accura
cy: 0.3112 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 17/20
9/9 [==============================] - 33s 4s/step - loss: 0.9245 - gender_out_loss:
0.3254 - tortyp_out_loss: 0.5991 - gender_out_accuracy: 0.0327 - tortyp_out_accurac
y: 0.8077 - torcol_out_accuracy: 0.1308 - legtyp_out_accuracy: 0.1231 - legocol_out_
accuracy: 0.1442 - luggage_out_accuracy: 0.1673 - val_loss: 1.1279 - val_gender_out_
loss: 0.4397 - val_tortyp_out_loss: 0.6883 - val_gender_out_accuracy: 0.0000e+00 - v
al_tortyp_out_accuracy: 0.6173 - val_torcol_out_accuracy: 0.2296 - val_legtyp_out_ac
curacy: 0.2755 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 18/20
9/9 [==============================] - 32s 4s/step - loss: 0.8435 - gender_out_loss:
0.4218 - tortyp_out_loss: 0.4218 - gender_out_accuracy: 0.0538 - tortyp_out_accurac
y: 0.8404 - torcol_out_accuracy: 0.0904 - legtyp_out_accuracy: 0.1096 - legocol_out_
accuracy: 0.1654 - luggage_out_accuracy: 0.1865 - val_loss: 1.0542 - val_gender_out_
loss: 0.3888 - val_tortyp_out_loss: 0.6654 - val_gender_out_accuracy: 0.0561 - val_t
ortyp_out_accuracy: 0.6173 - val_torcol_out_accuracy: 0.2296 - val_legtyp_out_accura
cy: 0.2653 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 19/20
9/9 [==============================] - 34s 4s/step - loss: 0.8214 - gender_out_loss:
0.3026 - tortyp_out_loss: 0.5188 - gender_out_accuracy: 0.0173 - tortyp_out_accurac
y: 0.8404 - torcol_out_accuracy: 0.0981 - legtyp_out_accuracy: 0.1404 - legocol_out_
accuracy: 0.1481 - luggage_out_accuracy: 0.1731 - val_loss: 1.0614 - val_gender_out_
loss: 0.3431 - val_tortyp_out_loss: 0.7184 - val_gender_out_accuracy: 0.1276 - val_t
ortyp_out_accuracy: 0.3827 - val_torcol_out_accuracy: 0.2296 - val_legtyp_out_accura
cy: 0.2602 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6429
Epoch 20/20
9/9 [==============================] - 33s 4s/step - loss: 0.7890 - gender_out_loss:
0.2794 - tortyp_out_loss: 0.5097 - gender_out_accuracy: 0.0135 - tortyp_out_accurac
y: 0.8558 - torcol_out_accuracy: 0.1096 - legtyp_out_accuracy: 0.1288 - legocol_out_
accuracy: 0.1288 - luggage_out_accuracy: 0.1904 - val_loss: 1.0545 - val_gender_out_
loss: 0.3832 - val_tortyp_out_loss: 0.6713 - val_gender_out_accuracy: 0.4541 - val_t
ortyp_out_accuracy: 0.6173 - val_torcol_out_accuracy: 0.2245 - val_legtyp_out_accura
cy: 0.2449 - val_legocol_out_accuracy: 0.0051 - val_luggage_out_accuracy: 0.6378
```

In [ ]: