CT5133 / CT5145 Deep Learning (/Online) 2022-2023

Assignment 2

James McDermott

- Student ID(s): 22223696
- Student name(s): Smitesh Nitin Patil

Note : In this notebook, I have add a cell of analysis and understanding at the end of every part. The references used for that part are also mentioned in the same cell

Due date: midnight Sunday 19 March (end Week 10).

Weighting: 20% of the module.

In this assignment the goal is to take advantage of pre-trained NN models to create an embedding with a dataset of movie posters, and demonstrate how to use that embedding.

The dataset is provided, along with some skeleton code for loading it.

The individual steps to be carried out are specified below, with `### YOUR CODE HERE` markers, together with the number of marks available for each part.

- **Topics**: in Part 5 below, students are asked to add some improvement to their models. In general, these improvements will differ between students (or student groups). **The proposed improvement must be notified to the lecturer at least 1 week before submission, and approved by the lecturer**. If working in a group, the two members of the group should not work on different topics in Part 5: they must work on the same topic and submit identical submissions.

- Students are not required to work incrementally on the parts. It is ok to do all the work in one day, so long as you abide by the rules on notifying groups and notifying topics.

- **Groups**: students may work solo or in a group of two. A student may not work together in a group with any student they have previously worked on a group project with, in this module or any other in the MSc programme. **Groups must be notified to the lecturer in writing before beginning work and at least 1 week before submission.** If working in a group, both students must submit and both submissions must be identical. If working in a group, both students may be asked to explain any aspect of the code in interview (see below), therefore working independently on separate components is not recommended. Any emails concerning the project should be cc-ed to the other group member.

- **Libraries**: code can be written in Keras/Tensorflow, or in PyTorch.

- **Plagiarism**: students may discuss the assignment together, but you may not look at another student or group's work or allow other students to view yours (other than within a group). You may use snippets of code (eg 1-2 lines) from the internet, **if you provide a citation with URL**. You may also use a longer snippet of code if it is a utility function, again only with citation. You may not use code from the internet to carry out the core of the assignment. You may not use a large language model to generate code.

- **Submission**: after completing your work in this Jupyter notebook, submit the notebook both in `.ipynb` and `.pdf` formats. The content should be identical.

- **Interviews**: a number of students may be selected for interview, post-submission. The selection will depend on submissions, and random chance may be used also. Interviews will be held in-person (CT5133) or online (CT5145). Interviews will last approximately 10 minutes. The purpose of interviews will be to assess students' understanding of their own submission.

Dataset Credits

The original csv file is from:

https://www.kaggle.com/datasets/neha1703/movie-genre-from-its-poster

I have added the *year* column for convenience.

I believe most of the information is originally from the famous MovieLens dataset:

- https://grouplens.org/datasets/movielens/
- https://movielens.org/

However, I'm not clear whether the poster download URLs (Amazon AWS URLs) which are in the csv obtained from the Kaggle URL above are from a MovieLens source, or elsewhere.

To create the dataset we are using, I have randomly sampled a small proportion of the URLs in the csv, and downloaded the images. I have removed those which fail to download. Code below also filters out those which are in black and white, ie 1 channel only.

Imports

You can add more imports if needed.

In [5]:
```
import numpy as np
import pandas as pd
import math
import os
import random
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial.distance import cdist, pdist, squareform # useful for distances in the embedding
```

In [6]:
```
import tensorflow as tf
import torch
from tensorflow import keras
from keras import layers, models
from keras.applications.vgg16 import VGG16
from keras.applications.vgg19 import VGG19
from keras.applications.resnet_v2 import ResNet152V2
from keras.applications.inception_v3 import InceptionV3
from keras.applications.resnet import ResNet, ResNet50
from keras.applications.vgg16 import preprocess_input
from keras.layers import Flatten, Dense, Embedding, GlobalAveragePooling2D, Conv2D, MaxPooling2D, MaxPool2D
from keras.models import Model, Sequential
from keras.optimizers import Adam, SGD


import os
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
```

Utility functions

These functions are provided to save you time. You might not need to understand any of the details here.

In [7]:
```
# walk the directory containing posters and read them in. all are the same shape: (268, 182).
# all have 3 channels, with a few exceptions (see below).
# each is named <imdbId>.jpg, which will later allow us to get the metadata from the csv.
IDs = []
images = []
for dirname, _, filenames in os.walk('DL_Sample'):
    for filename in filenames:
        if filename.endswith(".jpg"):
            ID = int(filename[:-4])
            pathname = os.path.join(dirname, filename)
            im = Image.open(pathname)
            imnp = np.array(im, dtype=float)
            if len(imnp.shape) != 3: # we'll ignore a few black-and-white (1 channel) images
                print("This is 1 channel, so we omit it", imnp.shape, filename)
                continue # do not add to our list
            IDs.append(ID)
            images.append(imnp)
```

```
This is 1 channel, so we omit it (268, 182) 290031.jpg
This is 1 channel, so we omit it (268, 182) 294266.jpg
This is 1 channel, so we omit it (268, 182) 30337.jpg
This is 1 channel, so we omit it (268, 182) 3626440.jpg
This is 1 channel, so we omit it (268, 182) 50192.jpg
This is 1 channel, so we omit it (268, 182) 54880.jpg
This is 1 channel, so we omit it (268, 182) 57006.jpg
```

In [8]:
```
#converting to numpy arrays
img_array = np.array(images)
#checking images array shape
img_array.shape
```

Out[8]:
```
(1254, 268, 182, 3)
```

In [9]:
```
# read the csv
df = pd.read_csv("Movie_Genre_Year_Poster.csv", encoding="ISO-8859-1", index_col="Unnamed: 0")
```

```
df.head()
```

Out[9]:

	imdbId	Imdb Link	Title	IMDB Score	Genre	Poster	Year
0	114709	http://www.imdb.com/title/tt114709	Toy Story (1995)	8.3	Animation|Adventure|Comedy	https://images-na.ssl-images-amazon.com/images...	1995.0
1	113497	http://www.imdb.com/title/tt113497	Jumanji (1995)	6.9	Action|Adventure|Family	https://images-na.ssl-images-amazon.com/images...	1995.0
2	113228	http://www.imdb.com/title/tt113228	Grumpier Old Men (1995)	6.6	Comedy|Romance	https://images-na.ssl-images-amazon.com/images...	1995.0
3	114885	http://www.imdb.com/title/tt114885	Waiting to Exhale (1995)	5.7	Comedy|Drama|Romance	https://images-na.ssl-images-amazon.com/images...	1995.0
4	113041	http://www.imdb.com/title/tt113041	Father of the Bride Part II (1995)	5.9	Comedy|Family|Romance	https://images-na.ssl-images-amazon.com/images...	1995.0

In [10]:

```
df2 = df.drop_duplicates(subset=["imdbId"]) # some imdbId values are duplicates - just drop
```

In [11]:

```
df3 = df2.set_index("imdbId") # the imdbId is a more useful index, eg as in the next cell...
```

In [12]:

```
df4 = df3.loc[IDs] # ... we can now use .loc to take a subset
```

In [13]:

```
df4.shape # 1254 rows matches the image data shape above
```

Out[13]:

```
(1254, 6)
```

In [14]:

```
years = df4["Year"].values
titles = df4["Title"].values

assert img_array.shape[0] == years.shape[0] == titles.shape[0]
```

In [15]:

```
def imread(filename):
    """Convenience function: we can supply an ID or a filename.
    We read and return the image in Image format.
    """

    if type(filename) == int:
        # assume its an ID, so create filename
        filename = f"DL_Sample/{filename}.jpg"

    # now we can assume it's a filename, so open and read
    im = Image.open(filename)

    return im

def imshow(im):
    plt.imshow(im)
    plt.axis('off')
    plt.show()
```

Part 1. Create embedding [3 marks]

Use a pretrained model, eg as provided by Keras, to create a flat (ie 1D) embedding vector of some size `embedding_size` for each movie poster, and put all of these together into a single tensor of shape `(n_movies, embedding_size)`.

In [16]:

```
#initialising processed_image array for storing the preprocessed inputs
processed_images = []
n_movies = len(np.array(df4.index))

#processing all images before feeding them to network to create embeddings
for image in img_array:
    image = image/255.0
    processed_images.append(preprocess_input(image))

#inilialising network architecture by removing the dense and output layer REF[1]
model = VGG19(include_top = False, input_shape = img_array[0].shape, weights="imagenet")

#model = ResNet50(include_top = False, input_shape = img_array[0].shape, weights="imagenet")
#model = InceptionV3(include_top = False, input_shape = img_array[0].shape, weights="imagenet")

#setting the layers as not trainable as we are not training our model just creating embeddings
```

```
for layer in model.layers:
    layer.trainable = False

# appending a flatten layer that would be the output the embeddings created would be of size of dimension of
# the images
x =  model.output
predictions = Flatten()(x)

# initialising the model with the architecture created
model = Model(inputs = model.input, outputs = predictions)
model.summary()

#creating embeddings
out = model.predict(np.array(processed_images))
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 268, 182, 3)]     0

 block1_conv1 (Conv2D)       (None, 268, 182, 64)      1792

 block1_conv2 (Conv2D)       (None, 268, 182, 64)      36928

 block1_pool (MaxPooling2D)  (None, 134, 91, 64)       0

 block2_conv1 (Conv2D)       (None, 134, 91, 128)      73856

 block2_conv2 (Conv2D)       (None, 134, 91, 128)      147584

 block2_pool (MaxPooling2D)  (None, 67, 45, 128)       0

 block3_conv1 (Conv2D)       (None, 67, 45, 256)       295168

 block3_conv2 (Conv2D)       (None, 67, 45, 256)       590080

 block3_conv3 (Conv2D)       (None, 67, 45, 256)       590080

 block3_conv4 (Conv2D)       (None, 67, 45, 256)       590080

 block3_pool (MaxPooling2D)  (None, 33, 22, 256)       0

 block4_conv1 (Conv2D)       (None, 33, 22, 512)       1180160

 block4_conv2 (Conv2D)       (None, 33, 22, 512)       2359808

 block4_conv3 (Conv2D)       (None, 33, 22, 512)       2359808

 block4_conv4 (Conv2D)       (None, 33, 22, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 11, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 11, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 11, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 11, 512)       2359808

 block5_conv4 (Conv2D)       (None, 16, 11, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 5, 512)         0

 flatten (Flatten)           (None, 20480)             0

=================================================================
Total params: 20,024,384
Trainable params: 0
Non-trainable params: 20,024,384
_________________________________________________________________
40/40 [==============================] - 212s 5s/step
```

In [17]:

```
#creating final tensor with the embeddings and their respective IDS
n_movies = img_array.shape[0]
X = torch.cat((torch.tensor(out), torch.tensor(IDs).unsqueeze(dim=1)), dim=1)
assert len(X.shape) == 2 # X should be (n_movies, embedding_size)
assert X.shape[0] == n_movies
```

Part 1 : Understanding and Analysis

Answer:

The objective of part 1 of this task was to create embeddings for images from imdb movies imageset. To create the image embeddings VGG16 pretrained model was used. VGG16 is a pre-trained convolutional neural network trained on ImageNet dataset to classify images into 1000 object categories[2]. It has the weights adjusted as per the images it learned during its training phase. The convolution layers with tuned weights where used to generate embeddings for each image in the movies dataset . This is done by removing the dense layers from the VGG16 model and adding a flattening layer that converts the final max pooling layer output to a one dimensional vector.

References for part 1

[1]Verma, S. (2021) A simple guide to using Keras pretrained models, Medium. Towards Data Science. Available at: https://towardsdatascience.com/step-by-step-guide-to-using-pretrained-models-in-keras-c9097b647b29 (Accessed: March 25, 2023).

[2] Deep Network designer (no date) VGG-16 convolutional neural network - MATLAB. Available at: https://www.mathworks.com/help/deeplearning/ref/vgg16.html (Accessed: March 25, 2023).

Part 2. Define a nearest-neighbour function [3 marks]

Write a function `def nearest(img, k)` which accepts an image `img`, and returns the `k` movies in the dataset whose posters are most similar to `img` (as measured in the embedding), ranked by similarity.

In [18]:

```
# function to get k posters most similar to the input image
def k_nearest(img_id, k):
    # getting the index from the previous X tensor
    index = X[:, -1]
    # getting the embeddings generated
    vector_embeddings = X[:, :(len(X[0])-1)]

    #getting the embedding of the input image
    image_embedding = [vector_embeddings[i] for i, idx in enumerate(index) if img_id == int(idx)]
    #initialising the cosime_similarity list
    cosine_similarities = []
    #looping through all the embeddings
    for idx, embeddings in enumerate(vector_embeddings):
        #getting the similarity value between each embedding and image embedding
        similarity = cosine_similarity(embeddings.reshape(1,-1), image_embedding[0].reshape(1, -1))
        cosine_similarities.append(similarity)

    # getting the idx of images with cosine similarity among top 1:k+1 values
    similar_images = [int(idx) for i, idx in enumerate(index)
                      if cosine_similarities[i] in sorted(cosine_similarities,
                                                          reverse = True)[1:k+1]]

    print(len(similar_images))

    #orginial image
    print("Image")
    imshow(imread(img_id))

    #similar images found
    print("Similar Images")
    for img in similar_images:
        imshow(imread(img))

    # score of similar images
    print("Image Scores: ", sorted(cosine_similarities, reverse = True)[1:k+1])
```

Part 2 : Understanding and Analysis

Answer:

The embeddings generated in part 1 could be thought of as points existing in a vector space. Embeddings are of size 20480, hence there are 20480 number of dimensions in our feature space. We can calculate how similar one image is to other by using similarity based or distance based metrics like euclidean and manhattan distance or cosine similarity. Cosine simlarity was used here to compute similarity among two embeddings as it judges the orientation of embeddings in the feature space and not its magnitute as is the case with distance measuring metrics [2].

The function takes in two arguments, First, the id of the poster and second is k the number of movies we want to find that are similar to poster mentioned in our first argument

[1] Verma, S. (2021) A simple guide to using Keras pretrained models, Medium. Towards Data Science. Available at: https://towardsdatascience.com/step-by-step-guide-to-using-pretrained-models-in-keras-c9097b647b29 (Accessed: March 25, 2023).

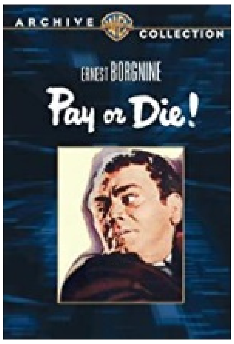
Part 3: Demonstrate your nearest-neighbour function [4 marks]

Choose any movie poster. Call this the query poster. Show it, and use your nearest-neighbour function to show the 3 nearest neighbours (excluding the query itself). This means **call** the function you defined above.

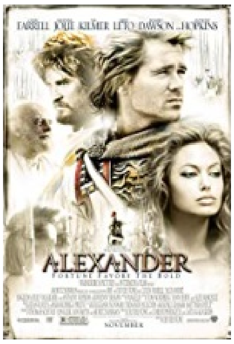
Write a comment: in what ways are they similar or dissimilar? Do you agree with the choice and the ranking? Why do you think they are close in the embedding? Do you notice, for example, that the nearest neighbours are from a similar era?

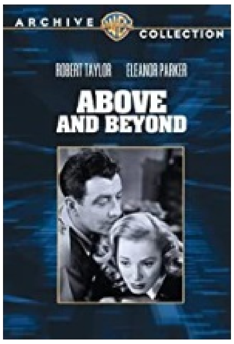
In [19]:
```
### YOUR code HERE
Q_idx = 54164

k_nearest(Q_idx, 3)
```

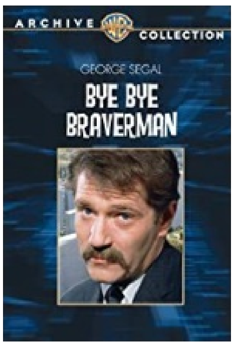
```
3
Image
```




```
Similar Images
```



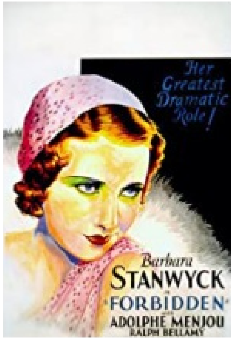





```
Image Scores:  [array([[0.9985604]], dtype=float32), array([[0.9981183]], dtype=float32), array([[0.99810994]], d
type=float32)]
```

```python
Q_idx = 22905 # YOUR VALUE HERE - DO NOT USE MY VALUE

k_nearest(Q_idx, 3)
```

```
3
Image
```




```
Similar Images
```

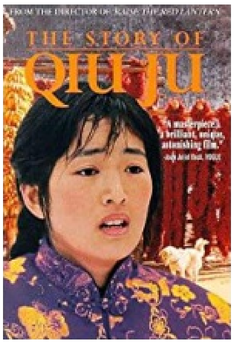

```
Image Scores:  [array([[0.99777174]], dtype=float32), array([[0.99768376]], dtype=float32), array([[0.99740815]], dtype=float32)]
```

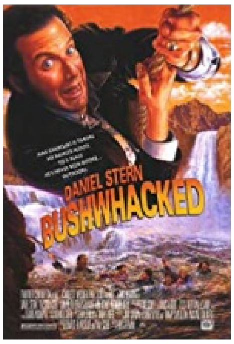
In [21]:

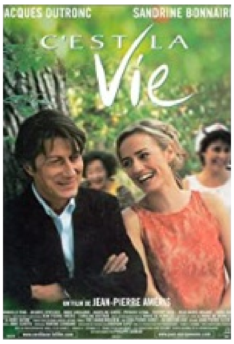
```python
Q_idx = 70463 # YOUR VALUE HERE - DO NOT USE MY VALUE

k_nearest(Q_idx, 3)
```

```
3
Image
```


```
Similar Images
```








```
Image Scores:  [array([[0.9995286]], dtype=float32), array([[0.9994865]], dtype=float32), array([[0.9994724]], dt
ype=float32)]
```

Part 3 : Understanding and Analysis

Answer:

The movies considered for this part are

1. Pay or Die 1960
2. The Man Who Knew Too Much 1956
3. Gernika 1993

Analyis for set 1

Movies found most similar to Pay or Die 1960

1. Alexander 2004
2. Above and Beyond 1952

3. Bye Bye Braverman 1968

It can be noticed for the movie posters as these movie posters have a same jacketed blue border and thus they were found as similar to each other irrespective to the yeat the movie where made also the poster found to be the most similar Alexander (2004) was from a completely different era.

Analysis for set 2

Movies found most similar to Forbidden 1932

1. La Mosquitera 2010
2. Wer Glaubt Wird Selig 2012
3. Je T'aime Je T'aime 1968

For this movie the movies found similar where from completely different eras. A possible reason could be that the movie selected is from 1930's are there are not much movie data present in our dataset. A possible solution could be to train with more data from that era.

Analysis for set 3

Movies found most similar to michel 1971

1. The Story of Qiu Ju 1992
2. Bushwhacked 1995
3. C'est la vie 2001

For this movie, it can be observed that the similar movies are relatively for the similar era. But, their similarity in this case looks to weigh more on the fact that all these movie posters have a human potrait as a central object on the posters.

In conclusion, we can observe that the movie posters generated are similar to the input image, but not just in context of the year the movie was released. Their are various other factors in background affecting their similarity like the format of the poster, the posters that are dominant in a shade of one color, number of movie posters available from the same era.

Part 4: Year regression [5 marks]

Let's investigate the last question ("similar era") above by running **regression** on the year, ie attempt to predict the year, given the poster. Use a train-test split. Build a suitable Keras neural network model for this, **as a regression head on top of the embedding from Part 1**. Include comments to explain the purpose of each part of the model. It should be possible to make a prediction, given a new poster (not part of the original dataset). Write a short comment on model performance: is it possible to predict the year? Based on this result, are there trends over time?

In [18]:

```
#dropping the index
df4 = df4.reset_index()
#selecting the title and year
df4 = df4[['Title', 'Year']]
```

In [19]:

```
#checking if year has any null values
df4['Year'].isna().value_counts()
```

Out[19]:

```
False    1238
True       16
Name: Year, dtype: int64
```

In [20]:

```
#getting the index of null values
index = df4['Year'].index[df4['Year'].apply(np.isnan)]
```

In [21]:

```
df4.iloc[index]
```

Out[21]:

	Title	Year
93	XIII: The Conspiracy	NaN
152	The Last Templar	NaN
189	Carlos	NaN
240	In Two Minds	NaN
290	The Hallelujah Handshake	NaN
359	Dragon Age: Redemption	NaN
416	Black Mirror	NaN
475	The Dust Bowl	NaN

673	And Then There Were None	NaN
683	10.5	NaN
685	Nirvana	NaN
766	North & South	NaN
791	I Hate Christian Laettner	NaN
1081	Holocaust	NaN
1174	Sins	NaN
1196	Step Up Love Story	NaN

In [22]:

```
#droping the indexed value that are NaN
df4 = df4.drop(index)
```

In [23]:

```
# getting embeddings and removing embeddings for which we dont have year data
embeddings = [arr for i, arr in enumerate(X[:, :(len(X[0])-1)]) if i not in index]
```

In [24]:

```
# checking if the len of img_data and yers are equal
assert len(embeddings) == len(df4)
```

In [25]:

```
# appending img_data in dataframe for subsetting
df4['embedding'] = embeddings
```

In [26]:

```
from sklearn.model_selection import train_test_split

# subsetting data into train 70%, test 15%, val 15%
train_df, test_df = train_test_split(df4, train_size=0.7, shuffle=True, random_state=1)
test_df, val_df = train_test_split(test_df, train_size=0.5, shuffle=True, random_state=1)
```

In [27]:

```
# all the data in numpy format for model training
X_train = np.array([t.numpy() for t in train_df['embedding']])
y_train = np.array([int(t) for t in train_df['Year']])
X_val = np.array([t.numpy() for t in val_df['embedding']])
y_val = np.array([int(t) for t in val_df['Year']])
X_test = np.array([t.numpy() for t in test_df['embedding']])
y_test = np.array([int(t) for t in test_df['Year']])

# checking if the data is correct in sizes
assert(len(X_train) == len(y_train))
assert(len(X_val) == len(y_val))
assert(len(X_test) == len(y_test))
```

In [28]:

```
# inilialising the model
model = Sequential([
    #fullu connected layers for our model with relu activations
    Dense(8192,input_shape = (X_train[0].shape[0], ), activation = "relu"),
    Dense(4096, activation = "relu"),
    Dense(2048, activation = "relu"),
    Dense(1024, activation = "relu"),
    Dense(512, activation = "relu"),
    Dense(128, activation = "relu"),
    Dense(64, activation = "relu"),
    Dense(32, activation = "relu"),
    Dense(16, activation = "relu"),
    #output layer with linear function for regression task
    Dense(1, activation = "linear"),
])

# Compile the model

#optimisation and learning using Adam checking loss by checking mean absolute error
model.compile(optimizer = Adam(0.00001), loss='mae', metrics = ['mae'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8192)              167780352

 dense_1 (Dense)             (None, 4096)              33558528

 dense_2 (Dense)             (None, 2048)              8390656
```

```
 dense_3 (Dense)             (None, 1024)              2098176

 dense_4 (Dense)             (None, 512)               524800

 dense_5 (Dense)             (None, 128)               65664

 dense_6 (Dense)             (None, 64)                8256

 dense_7 (Dense)             (None, 32)                2080

 dense_8 (Dense)             (None, 16)                528

 dense_9 (Dense)             (None, 1)                 17

=================================================================
Total params: 212,429,057
Trainable params: 212,429,057
Non-trainable params: 0
_________________________________________________________________
```

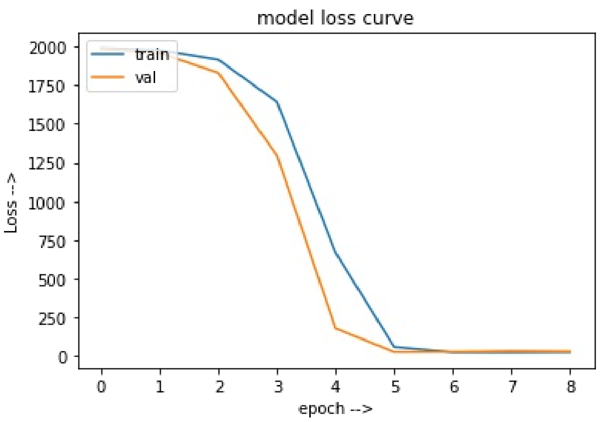
In [29]:

```python
# model training with 50 epochs and stopping early if validation loss increases for 3 consecutive epochs
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs = 50,
        callbacks=[ tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)])
```

```
Epoch 1/50
28/28 [==============================] - 25s 836ms/step - loss: 1989.3162 - mae: 1989.3162 - val_loss: 1984.3684
- val_mae: 1984.3684
Epoch 2/50
28/28 [==============================] - 25s 890ms/step - loss: 1976.2659 - mae: 1976.2659 - val_loss: 1957.7539
- val_mae: 1957.7539
Epoch 3/50
28/28 [==============================] - 26s 908ms/step - loss: 1914.9171 - mae: 1914.9171 - val_loss: 1829.4768
- val_mae: 1829.4768
Epoch 4/50
28/28 [==============================] - 25s 892ms/step - loss: 1642.0304 - mae: 1642.0304 - val_loss: 1296.4419
- val_mae: 1296.4419
Epoch 5/50
28/28 [==============================] - 25s 904ms/step - loss: 668.0460 - mae: 668.0460 - val_loss: 179.9603 - v
al_mae: 179.9603
Epoch 6/50
28/28 [==============================] - 25s 883ms/step - loss: 59.8891 - mae: 59.8891 - val_loss: 27.3022 - val_
mae: 27.3022
Epoch 7/50
28/28 [==============================] - 23s 838ms/step - loss: 25.2242 - mae: 25.2242 - val_loss: 28.1519 - val_
mae: 28.1519
Epoch 8/50
28/28 [==============================] - 23s 832ms/step - loss: 24.1446 - mae: 24.1446 - val_loss: 33.4300 - val_
mae: 33.4300
Epoch 9/50
28/28 [==============================] - 24s 861ms/step - loss: 25.8797 - mae: 25.8797 - val_loss: 30.8750 - val_
mae: 30.8750
```

In [30]:

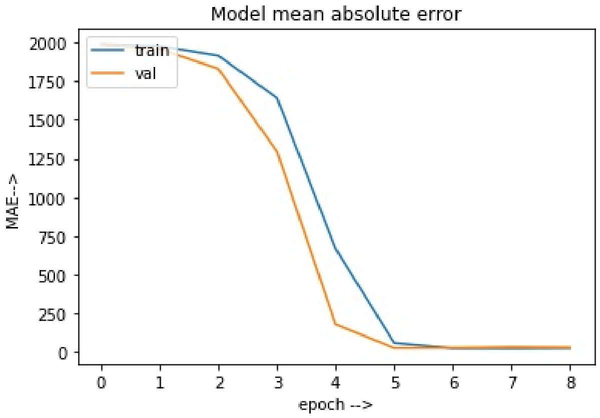
```python
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss curve')
plt.ylabel('Loss -->')
plt.xlabel('epoch -->')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



In [31]:

```python
plt.plot(history.history['mae'])
```

```
plt.plot(history.history['val_mae'])
plt.title('Model mean absolute error')
plt.ylabel('MAE-->')
plt.xlabel('epoch -->')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



Assuming naive baseline for model

created a baseline assuming if we predict mean value of test data every time

In [32]:
```
from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error for baseline model :",mean_absolute_error(y_test, [np.mean(y_test)]*len(y_test)))
```

```
Mean Absolute Error for baseline model : 17.571973638570935
```

In [33]:
```
# predictions
predictions = model.evaluate(X_test, y_test)
```

```
6/6 [==============================] - 1s 153ms/step - loss: 26.9948 - mae: 26.9948
```

In [34]:
```
print("Mean absolute error  value on test_data :"+ str(predictions[1]))
```

```
MAE value on test_data :26.994754791259766
```

Part 4 : Understanding and Analysis

Answer:

Training then model: The embeddings of size 20,480 were trained on a dense, fully connected neural network of 8 layers, early stopping was implemented to prevent. The model loss calculated as mean absolute error keeps on decreasing and it pleateaus around 30. For the baseline model we found that mean absolute error on test data was 17.57. The model we trained gave a mean absolute error value of 26.9948. Thus, it is performing worse than our baseline

From the similar images found in Part 3, It was evident that predicting year from the movie posters could be challenging, in part 5, I have tried to train a new convolutional neural network with around 35000 images from the excel sheet provided

Load Images

Part 5: Improvements [5 marks]

Propose a possible improvement. Some ideas are suggested below. The chosen improvement must be notified to the lecturer at least 1 week before submission and **must be approved by the lecturer to avoid duplication with other students**. Compare the performance between your original and your new model (the proposed improvement might not actually improve on model performance -- that is ok). Some marks will be awarded for more interesting / challenging / novel improvements.

Ideas:

- Try a different pretrained model for creating the embedding
- Alternative ways of reducing the pretrained model's output to a flat vector for the embedding
- Gather more data (see the csv file for URLs)
- Add different architectural details to the regression head
- Fine-tuning
- Training an end-to-end convnet of your own design (no pretraining)
- Improve the embedding by training a multi-headed model, eg predicting both genre and year
- Create a good visualisation of the embedding.

In [35]:
```
### YOUR CODE HERE
#subsetting image url and year
df_5 = df[['imdbId', 'Poster', 'Year']]
```

In [36]:
```
#dropping all null values and reconfirming
df_5 = df_5.dropna()
df_5.isna().value_counts()
```

Out[36]:
```
imdbId  Poster  Year
False   False   False    38882
dtype: int64
```

In [37]:
```
excel = pd.DataFrame(columns = ['ImdbId', 'Address', 'Year'])
```

for downloading images ### ran only once hence commenting[1] import os import sys import warnings warnings.filterwarnings('ignore') #downloading images and storing them as their respective imdbId.jpg import requests i = 1 #getting the url of images, their id and year for img_url, imdbId, year in zip(df_5['Poster'], df_5['imdbId'], df_5['Year']): total = len(df_5) #getting the data img_data = requests.get(img_url).content #intialising empty array for appeding id imdbPresent = [] #only gettings images for status code found(200) if requests.get(img_url).status_code == 200: #writing the image data with open('images/'+str(imdbId)+'.jpg', 'wb') as handler: i = i+1 handler.write(img_data) #appending the imdbid imdbPresent.append(imdbId) #generating address for saved imge address = '/images/'+str(imdbId)+'.jpg' #appending the id, local address and year excel = excel.append({'ImdbId' : imdbId, 'Address' : address, 'Year' : year}, ignore_index = True) #for printing the number of images done sys.stdout.write("\rImages Done: " + str(i)) sys.stdout.flush() #saving to csv excel.to_csv("out.csv", encoding='utf-8', index=False)

In [38]:
```
task5_df = pd.read_csv("out.csv")
```

In [39]:
```
#preprocessing address by removing '/'
task5_df['Address'] = task5_df['Address'].str[1:]
```

In [40]:
```
task5_df
```

Out[40]:

	ImdbId	Address	Year
0	114709	images/114709.jpg	1995.0
1	113497	images/113497.jpg	1995.0
2	113228	images/113228.jpg	1995.0
3	114885	images/114885.jpg	1995.0
4	113041	images/113041.jpg	1995.0
...	...	...	...
35374	98216	images/98216.jpg	1989.0
35375	83291	images/83291.jpg	1981.0
35376	82875	images/82875.jpg	1981.0
35377	815258	images/815258.jpg	2006.0
35378	79142	images/79142.jpg	1979.0

35379 rows × 3 columns

In [41]:
```
# checking if image is valid and generating image list and year list for valid images only

import sys
image_arr = []
image_list = []
year = []
i = 0
j = 0

#zipping year and image address together
```

```
    for address, y in zip(task5_df['Address'].tolist(), task5_df['Year'].tolist()):
        i = i+1
        #checking for valid images
        try:
            im = Image.open(address)
            image_list.append(address)
            year.append(y)
        except:
            j = j+1
        #for printing the number of images done
        sys.stdout.write("\rImages Done: " + str(i)+ "\rTotal Images: "+ str(len(task5_df))+ "\rImproper Images: "+ s
        sys.stdout.flush()
```

```
Improper Images: 29
```

In [42]:

```
#creating a dataframe for valid images and their respective years
df = pd.DataFrame(list(zip(image_list, year)), columns = ["Posters", "Year"])
```

In [43]:

```
#total 35377 movies
len(df)
```

Out[43]:

```
35377
```

In [44]:

```
#generating training, testing dataset
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(df, train_size=0.8, shuffle=True, random_state=1)
```

In [45]:

```
train_df
```

Out[45]:

	Posters	Year
12709	images/64073.jpg	1969.0
22388	images/1411276.jpg	2010.0
7233	images/67333.jpg	1971.0
21065	images/1907614.jpg	2012.0
33380	images/111469.jpg	1994.0
...	...	...
7813	images/323807.jpg	2003.0
32511	images/2043879.jpg	2011.0
5192	images/234215.jpg	2003.0
12172	images/41886.jpg	1949.0
33003	images/12136.jpg	1921.0

28301 rows × 2 columns

In [46]:

```
#using keras preprocessing to generate valid image data as input for our model [2]
import tensorflow as tf
import tf.keras.preprocessing.image.ImageDataGenerator as ImageDataGenerator #[2]
train_generator = ImageDataGenerator(
    #rescaling and generating validations set
    rescale=1./255,
    validation_split=0.2
)

test_generator = ImageDataGenerator(
    rescale=1./255
)
```

In [52]:

```
# generating data for keras model training [3]
train_images = train_generator.flow_from_dataframe(
    #passing the training dataframe
    dataframe=train_df,
    #setting the posters as our data and year as our target variable
    x_col='Posters',
    y_col='Year',
    #mentioning the size of posters
    target_size=(182, 268),
    #three channel image
```

```
    color_mode='rgb',
    class_mode='raw',
    #batch size as 32
    batch_size=32,
    #shuffling the daatset
    shuffle=True,
    #setting seed for random shuffle
    seed=21,
    #training set
    subset='training'
)

val_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Posters',
    y_col='Year',
    #setting image size
    target_size=(182, 268),
    #3 channel image
    color_mode='rgb',
    class_mode='raw',
    #setting batch size 32
    batch_size=32,
    shuffle=True,
    seed=21,
    #validation set
    subset='validation'
)

test_images = test_generator.flow_from_dataframe(
    #taking test dataframe
    dataframe=test_df,
    #x and y variables
    x_col='Posters',
    y_col='Year',
    #image size
    target_size=(182, 268),
    #3 channel image
    color_mode='rgb',
    #class mode raw for regression task
    class_mode='raw',
    batch_size=32,
    #no need to shuffle test set
    shuffle=False
)
```

```
Found 22641 validated image filenames.
Found 5660 validated image filenames.
Found 7076 validated image filenames.
```

In [50]:

```
from sklearn.metrics import r2_score

model1 = Sequential([
    Conv2D(filters=8,input_shape = (182, 268, 3), kernel_size=(3, 3), activation='relu'),
    MaxPool2D(),
    Conv2D(filters=12, kernel_size=(3, 3), activation='relu'),
    MaxPool2D(),
    Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
    MaxPool2D(),
    Flatten(),
    Dense(2048, activation = "relu"),
    Dense(1024, activation = "relu"),
    Dense(512, activation = "relu"),
    Dense(128, activation = "relu"),
    Dense(64, activation = "relu"),
    Dense(32, activation = "relu"),
    Dense(16, activation = "relu"),
    #output layer with linear function for regression task
    Dense(1, activation = "linear")
])

model1.compile(
    optimizer=keras.optimizers.Adam(0.0001),
    loss='mae',
    metrics = ['mae'],
)

model1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_3 (Conv2D)           (None, 180, 266, 8)       224       
```

```
 max_pooling2d_3 (MaxPooling  (None, 90, 133, 8)       0
 2D)

 conv2d_4 (Conv2D)           (None, 88, 131, 12)       876

 max_pooling2d_4 (MaxPooling  (None, 44, 65, 12)       0
 2D)

 conv2d_5 (Conv2D)           (None, 42, 63, 16)        1744

 max_pooling2d_5 (MaxPooling  (None, 21, 31, 16)       0
 2D)

 flatten_2 (Flatten)         (None, 10416)             0

 dense_18 (Dense)            (None, 2048)              21334016

 dense_19 (Dense)            (None, 1024)              2098176

 dense_20 (Dense)            (None, 512)               524800

 dense_21 (Dense)            (None, 128)               65664

 dense_22 (Dense)            (None, 64)                8256

 dense_23 (Dense)            (None, 32)                2080

 dense_24 (Dense)            (None, 16)                528

 dense_25 (Dense)            (None, 1)                 17

=================================================================
Total params: 24,036,381
Trainable params: 24,036,381
Non-trainable params: 0
_________________________________________________________________
```

In [51]:

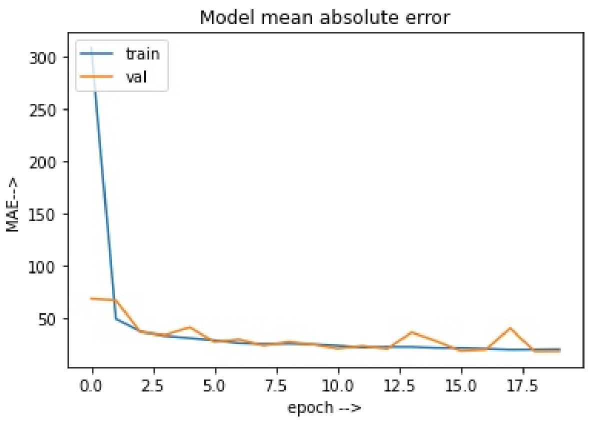
```
history = model1.fit(
    train_images,
    validation_data=val_images,
    epochs=20,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)]
)
```

```
Epoch 1/20
708/708 [==============================] - 414s 582ms/step - loss: 308.9092 - mae: 308.9092 - val_loss: 68.6684 - 
val_mae: 68.6684
Epoch 2/20
708/708 [==============================] - 361s 509ms/step - loss: 49.4072 - mae: 49.4072 - val_loss: 67.1786 - v
al_mae: 67.1786
Epoch 3/20
708/708 [==============================] - 365s 515ms/step - loss: 37.1186 - mae: 37.1186 - val_loss: 36.4979 - v
al_mae: 36.4979
Epoch 4/20
708/708 [==============================] - 372s 526ms/step - loss: 32.8133 - mae: 32.8133 - val_loss: 33.8660 - v
al_mae: 33.8660
Epoch 5/20
708/708 [==============================] - 380s 537ms/step - loss: 30.8883 - mae: 30.8883 - val_loss: 41.0488 - v
al_mae: 41.0488
Epoch 6/20
708/708 [==============================] - 370s 523ms/step - loss: 28.7468 - mae: 28.7468 - val_loss: 27.1664 - v
al_mae: 27.1664
Epoch 7/20
708/708 [==============================] - 377s 533ms/step - loss: 26.0736 - mae: 26.0736 - val_loss: 29.7273 - v
al_mae: 29.7273
Epoch 8/20
708/708 [==============================] - 426s 602ms/step - loss: 25.2707 - mae: 25.2707 - val_loss: 23.6560 - v
al_mae: 23.6560
Epoch 9/20
708/708 [==============================] - 477s 674ms/step - loss: 25.4953 - mae: 25.4953 - val_loss: 27.3945 - v
al_mae: 27.3945
Epoch 10/20
708/708 [==============================] - 466s 658ms/step - loss: 24.9310 - mae: 24.9310 - val_loss: 24.7809 - v
al_mae: 24.7809
Epoch 11/20
708/708 [==============================] - 438s 619ms/step - loss: 23.5768 - mae: 23.5768 - val_loss: 20.6232 - v
al_mae: 20.6232
Epoch 12/20
708/708 [==============================] - 466s 658ms/step - loss: 21.8072 - mae: 21.8072 - val_loss: 23.3612 - v
al_mae: 23.3612
Epoch 13/20
708/708 [==============================] - 482s 681ms/step - loss: 22.4218 - mae: 22.4218 - val_loss: 20.3912 - v
al_mae: 20.3912
Epoch 14/20
708/708 [==============================] - 482s 680ms/step - loss: 22.3255 - mae: 22.3255 - val_loss: 36.3223 - v
```

```
al_mae: 36.3223
Epoch 15/20
708/708 [==============================] - 468s 661ms/step - loss: 21.3807 - mae: 21.3807 - val_loss: 27.9879 - v
al_mae: 27.9879
Epoch 16/20
708/708 [==============================] - 491s 693ms/step - loss: 21.2639 - mae: 21.2639 - val_loss: 18.3972 - v
al_mae: 18.3972
Epoch 17/20
708/708 [==============================] - 437s 617ms/step - loss: 20.5236 - mae: 20.5236 - val_loss: 19.6474 - v
al_mae: 19.6474
Epoch 18/20
708/708 [==============================] - 407s 575ms/step - loss: 19.6217 - mae: 19.6217 - val_loss: 40.2130 - v
al_mae: 40.2130
Epoch 19/20
708/708 [==============================] - 350s 494ms/step - loss: 19.7558 - mae: 19.7558 - val_loss: 17.9433 - v
al_mae: 17.9433
Epoch 20/20
708/708 [==============================] - 346s 488ms/step - loss: 20.0128 - mae: 20.0128 - val_loss: 18.0772 - v
al_mae: 18.0772
```

In [60]:

```
plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.title('Model mean absolute error')
plt.ylabel('MAE-->')
plt.xlabel('epoch -->')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



In [58]:

```
from sklearn.metrics import mean_absolute_error
print("Naive Base Line Mean absolute error : ",mean_absolute_error(list(test_images.labels), [np.mean(list(test_i
```

```
Naive Base Line Mean absolute error :  18.822914805930044
```

In [54]:

```
mae = model1.evaluate(test_images)
print("Mean absolute error : ",mae)
```

```
222/222 [==============================] - 34s 155ms/step - loss: 18.4419 - mae: 18.4419
Mean absolute error :  [18.44194984436035, 18.44194984436035]
```

In [62]:

```
predictions = model1.predict(test_images)
```

```
222/222 [==============================] - 26s 117ms/step
```

Part 5 : Understanding and Analysis

Training on a new model, it is evident from the mean absolute error values that the new model performs only slightly better that our baseline. In conclusion, I think it is not possible to predict year just from the movie posters. But their are some changes that can help achieve us a somewhat similar result.

1. We can convert the problem from a regression to classification: In part 3, we found that many a times images from same era where being group together. Hence, we can convert our years from continuous to discrete decades or eras and and try to predict the decade the

movie came out.

2. Along with movie posters we can use other features like genre (certain movie genres like western were popular at different times), movie cast and directors (certian actors and directors were active during certain periods), box office numbers as time scale increases box office numbers increase as a result of inflation and movies being available for other people overseas.

[1]Su, S. et al. (1962) Python save image from URL, Stack Overflow. Available at: https://stackoverflow.com/questions/30229231/python-save-image-from-url (Accessed: March 26, 2023).

[2] Tf.keras.preprocessing.image.imagedatagenerator : tensorflow V2.12.0 (no date) TensorFlow. Available at: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator (Accessed: March 26, 2023).

[3] Gcdatkin (2021) age prediction from images (CNN regression), Kaggle. Kaggle. Available at: https://www.kaggle.com/code/gcdatkin/age-prediction-from-images-cnn-regression/notebook (Accessed: March 26, 2023).