ENGR 691 Deep Learning
Project 2 Report
By: Babita Pradhan, Xing Yin

Objective:

- Design neural network architecture and implement it using tool-kits such as Pytorch, and Keras that predicts the age using face images.
- Use IMDB/WIKI dataset (faces only) for training, testing, and validation of network by splitting it into training/validation/testing sets.
- Evaluate the performance of each designed neural network.

Processing Raw Data:

The raw data for this project is taken from [IMDB-WIKI - 500k+ face images with age and gender labels (ethz.ch)](), that contains the images of faces along with the metadata containing the information as date of birth, the year when the photo was taken, the path of the file, gender and others. The data downloaded from the website was processed stepwise as follows:

1. **Age calculation:** For calculating the age of the image, we use the date of birth and the time when the photo was taken. The *.mat* file was read in MatLab and age was calculated using: [age,~]=datevec(datenum({imagesource}.photo_taken,7,1)-{imagesource}.dob);
   where image source can be either wiki or IMDB. The full_path and age information is then saved on a CSV file for further processing. This process is coded in the *imdbwiki_age.m* file and the CSV is saved as i*mdb_wiki.csv*.

2. **Data filtering:** The data are filtered to remove the outliers as images with negative ages and ages greater than 100. We are only using images between the ages of 10 to 100 for the project. This filtering reduces the data size from 523,051 to 516,255. Also, images with sizes greater than or equal to 64X64 are filtered reducing the dataset size to 507,747. This process is coded under the code file *Data_filtering.ipynb* that produces CSV *data_filtersize.csv*.

3. **Data preprocessing and splitting:** Here, we resized all images to 64X64 and converted them to grayscale, and saved these converted images to the *data_preprocessed* folder. This process is coded on *data_preprocessing.ipynb* python file. Once all images have the same sizes and are converted to grayscale, these images are split as training, validation, and testing data with 60, 20, and 20 % of total images respectively. This is coded on a python file named *data_splitting.ipynb*.

Table 1: IMDB_WIKI data size in after processing and splitting

IMDB_WIKI data	Data after filtering age and size	Training Data	Testing data	Validation Data
523,051	507,747	304,648	101,550	101,549

NN architecture and results:

Three different neural networks are designed and implemented for the age prediction. All architecture takes the grayscaled input image of size 64X64. Those three NN architectures with the results are described below:

1. **Network architecture 1**
   This neural network uses two convolutional layers followed by two linear layers and predicts the age of the input image through regression. The detailed CNN architecture is provided in

table 2 with other parameters such as learning rate, optimizer, and batch size. Both age and image values are normalized to get the values between 0 and 1. The training and validation loss observed for this architecture is provided in figure 1. Only a slight decrease in validation loss can be observed. Some examples with true and predicted age for testing data along with corresponding images are shown in fig. 2. Fig. 3 shows the true age and predicted age curve for 100 testing data.

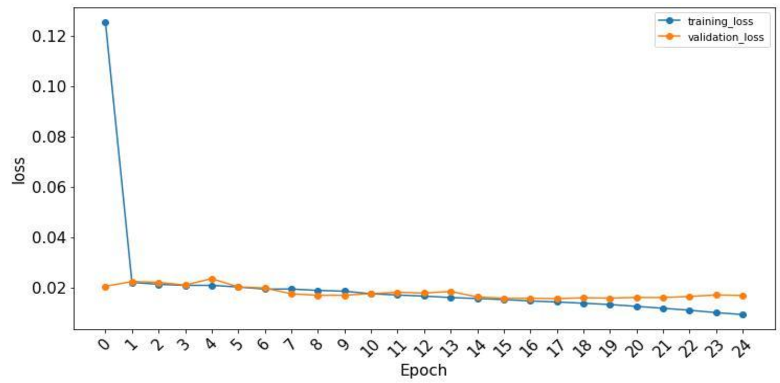
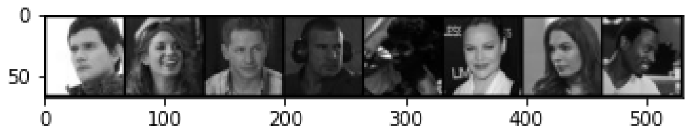


Fig 1: Training and validation loss using architecture 1



```
True Age   : 34.00, 18.00, 32.00, 43.00, 38.00, 28.00, 33.00, 27.00,
Predict Age: 37.73, 29.25, 44.77, 38.95, 35.64, 26.66, 41.34, 34.97,
```

Fig2: True and predicted age with input images using architecture 1

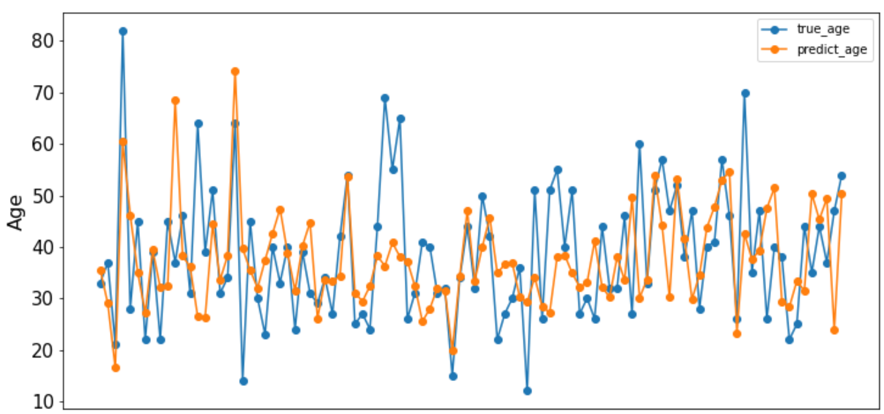


Fig3: True and predicted age curve obtained using architecture 1

2. **Network architecture 2**

   This neural network uses two convolutional layers followed by one linear layer, with different kernel size and stride size and predicts the age of the input image through classification. The detailed CNN architecture is provided in table 2. Both age and image values are normalized to get the values between 0 and 1. The loss function of the architecture is cross-entropy. The loss obtained in each epoch and the CNN layers used for this network are shown below.

```
Epoch :  1   loss : tensor(4.7743, grad_fn=<NllLossBackward>)
Epoch :  2   loss : tensor(14.8594, grad_fn=<NllLossBackward>)
Epoch :  3   loss : tensor(27.0080, grad_fn=<NllLossBackward>)
Epoch :  4   loss : tensor(28.8070, grad_fn=<NllLossBackward>)
Epoch :  5   loss : tensor(22.7597, grad_fn=<NllLossBackward>)
Epoch :  6   loss : tensor(13.5484, grad_fn=<NllLossBackward>)
Epoch :  7   loss : tensor(4.4982, grad_fn=<NllLossBackward>)
Epoch :  8   loss : tensor(4.4393, grad_fn=<NllLossBackward>)
Epoch :  9   loss : tensor(4.4263, grad_fn=<NllLossBackward>)
Epoch :  10      loss : tensor(4.4052, grad_fn=<NllLossBackward>)
```

```
Net(
  (cnn_layers): Sequential(
    (0): Conv2d(1, 4, kernel_size=(2, 2), stride=(2, 2))
    (1): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(4, 4, kernel_size=(2, 2), stride=(2, 2))
    (5): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (linear_layers): Sequential(
    (0): Linear(in_features=1024, out_features=100, bias=True)
  )
```

3. **Network architecture 3**

This network architecture is similar to the 2 except change in kernel size from 2X2 to 3X3. For other parameters used refer to table 2. The loss obtained in each epoch and the CNN layers used for this network are shown below.

```
Epoch :  1   loss : tensor(4.7139, grad_fn=<NllLossBackward>)
Epoch :  2   loss : tensor(12.6895, grad_fn=<NllLossBackward>)
Epoch :  3   loss : tensor(22.5013, grad_fn=<NllLossBackward>)
Epoch :  4   loss : tensor(25.2296, grad_fn=<NllLossBackward>)
Epoch :  5   loss : tensor(22.1341, grad_fn=<NllLossBackward>)
Epoch :  6   loss : tensor(19.9133, grad_fn=<NllLossBackward>)
Epoch :  7   loss : tensor(15.1316, grad_fn=<NllLossBackward>)
Epoch :  8   loss : tensor(8.8675, grad_fn=<NllLossBackward>)
Epoch :  9   loss : tensor(4.7501, grad_fn=<NllLossBackward>)
Epoch :  10      loss : tensor(4.3009, grad_fn=<NllLossBackward>)
```

```
Net(
  (cnn_layers): Sequential(
    (0): Conv2d(1, 4, kernel_size=(3, 3), stride=(2, 2))
    (1): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(4, 4, kernel_size=(2, 2), stride=(2, 2))
    (5): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (linear_layers): Sequential(
    (0): Linear(in_features=900, out_features=89, bias=True)
  )
```

Table 2: Network architecture description

	Architecture 1	Architecture 2	Architecture 3
Prediction model	Regression	Classification	Classification
CNN Architecture	**Convolution layer 1** Conv-64,kernel 5X5 BatchNorm Relu Maxpool:kernel 2X2 Dropout(0.5) **Convolution layer 2** Conv-128,kernel 5X5 BatchNorm Relu Maxpool:kernel 2X2 Dropout(0.5) **Linear layer 1** linear(21632, 4096) Relu **Linear layer 2** linear(4096,1)	**Convolution layer 1** Conv2d(1,4,kernel_size=(2,2), stride=(2, 2)) BatchNorm ReLU MaxPool2d(kernel_size=2,stride=2) **Convolution layer 2** Conv2d(4,4,kernel_size=(2, 2), stride=(2, 2)) BatchNorm ReLU MaxPool2d(kernel_size=2,stride=2) **linear_layer** Linear(in_features=1024, out_features=100)	**cnn_layer 1:** Conv2d(1,4, kernel_size=(3,3), stride=(2, 2)) BatchNorm ReLU MaxPool2d(kernel_size=3,stride=2) **Convolution layer 2** Conv2d(4,4,kernel_size=(2,2), stride=(2, 2)) BatchNorm ReLU MaxPool2d(kernel_size=2,stride=2) **linear_layer** Linear(in_features=900,out_features=89)
Loss function	MSE	Cross Entropy	Cross Entropy
Optimizer	Adam	Adam	Adam
Learning rate	0.0005	0.07	0.07
Batch size	100	100	100
Max. Epoch	25	25	25
Convergence tolerance	0.00001	0.00001	0.00001
Training, validation, Testing dataset	80K, 10K, and 10K	80K, 10K, and 10K	80K, 10K, and 10K
Library and platform	PyTorch in Kaggle GPU	PyTorch with CPU +GPU	PyTorch with CPU+GPU

Result Discussion:
The accuracy obtained in each CNN network is as shown in table 3. For architecture 1, the accuracy is computed with the threshold of 5 i.e. for the true age of 30 if the predicted age falls between 25-35, it is assumed to be correct. Whereas for networks 2 and 3 only exact matches are taken.

Table 3: Test accuracy of CNN

	Architecture 1	**Architecture 2**	**Architecture 3**
Accuracy	**31.93 %**	**5 %**	**5 %**

For all three architectures, the accuracy of the network is comparatively low. The reasons for this low accuracy in age prediction might be due to the following reasons:

- Only the outliers on age and image size were removed from the data. There might be other outliers such as blurred images, images with more than one face, and others, which were not removed and might decrease the efficiency of the model.
- Some of the true labels of age in the image might be incorrect which might affect the model.
- All input images are grayscaled and their size is reduced to 64X64 in order to make the computation easy and fast. This reduction in size and channel on input images may cause significant loss of information decreasing the efficiency and affecting the accuracy.
- The even number of kernel size and stride size will provide more efficiency and accuracy in performance, compared with odd numbers.

Work division:

Babita Pradhan	Xing Yin
Processing the raw data Design, test and implement architecture 1	Design, test, and implement architecture 2 and architecture 3