Age Prediction and Gender Classification from Facial Images

Group V

Computer Science and Engineering Department
Indian Institute of Information Technology
December 5, 2020

1 Abstract

Automatic age and gender classification has become relevant to an increasing amount of applications, particularly since the rise of social platforms and social media. Nevertheless, performance of existing methods is still significantly lacking, especially when compared to the tremendous leaps in performance recently reported for the related task of face recognition. In this paper, we started experimenting with traditional machine learning algorithms like KNN, MLP then moved to basic versions of CNN and finally to modern deep-convolutional neural networks (CNN), to match the best accuracy that is available on the simplified version of UTK data set available on Kaggle.

2 Introduction

Age and gender play fundamental roles in social interactions. Languages reserve different salutations and grammar rules for men or women, and very often different vocabularies are used when addressing elders compared to young people. Despite the basic roles these attributes play in our day-to-day lives, the ability to automatically estimate them accurately and reliably from face images is still far from meeting the needs of commercial applications. This is particularly perplexing when considering recent claims to super-human capabilities in the related task of face recognition.[4]

3 Problem Formulation

Age and gender classification has wide range of application especially in tackling some real world problems like security and video surveillance, social media platforms and also for better add targeting. We have trained several machine learning algorithm on the UTK dataset for gender classification and age prediction. Our aim was to see the performance of several machine learning algorithm for the task and to figure out a way to beat the the performance of the existing models by using modern deep convolutional neural network that is under our capacity.

4 Data Analysis:

The dataset was collected from kaggle's Gender Classification of Facial Images.The dataset contains 23,705 images having 5 attributes describing the ages ranging from 1 to 116 years, ethnicity(0 for white, 1 for black, 2 for Asian, 3 for Indian, 4 for Other), gender(0 for male, 1 for female).The dataset is normalized using min-max normalization. The age distribution is as follows,

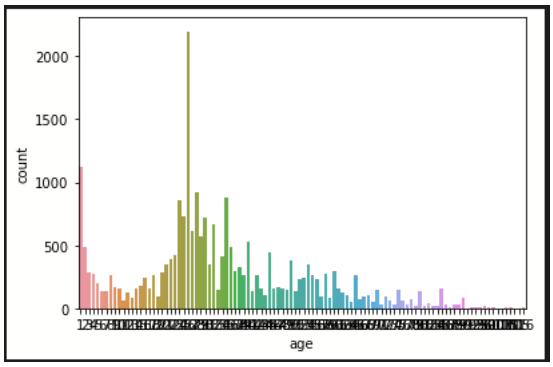


Figure 1. Age Distribution

Gender Distribution is shown below:

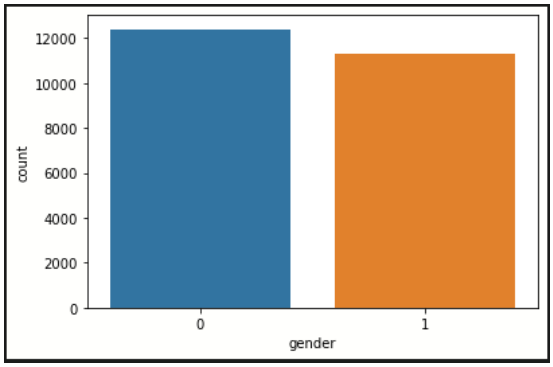


Figure 2. Gender Distribution

5 Experiments

5.1 Intuitions

The most important thing the team focused on was to use per pixel regressors or classifiers. It is already known that feature engineering on a dataset, like this, didn't work up to the expectations from previous works. The baseline MAE for the regression task is 15.3 considering model just predicts the mean

of the ages. The baseline accuracy for the classification task is 52.2 % considering the model predicts only the majority class.

The first consideration was to use a linear classifier and linear regressor.

5.2 Linear Regression

5.2.1 Algorithm

Algorithm 1. Standard linear regression

Input: data set $\boldsymbol{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_n, y_n)\}$ and threshold t
Output: $\boldsymbol{\theta} = [w_1, w_2, \cdots, w_d, b]$
1: Define the loss function $J(\boldsymbol{\theta}) = (\frac{1}{2n}) \sum_{i=1}^{n} (\boldsymbol{\theta}^T \boldsymbol{x}_i - y_i)^2$
2: Generating the $\boldsymbol{\theta}^0$ randomly
3: **repeat**
4: $\quad \boldsymbol{\theta}^k = \boldsymbol{\theta}^{k-1} - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$, where $\boldsymbol{\theta}^k$ is the value of k^{th} iteration and α is the iteration step.
5: **until** $J(\boldsymbol{\theta}^{k+1}) - J(\boldsymbol{\theta}^k) < t$
6: **return** $\boldsymbol{\theta}$

Figure 3. Linear Regression Algorithm

5.2.2 Observation

Model	Accuracy(MAE)
Linear Regression	11.0999
Linear Regression with l1 regularization	11.7482
Linear Regression with l2 regularization	10.9368

5.3 Single-Layer Perceptron(SLP)

5.3.1 Algorithm

1. Initialize weights at random
2. For each training pair/pattern(x,y_{actual})
→ compute $y_{prediction}$
→ compute error, δ =(y_{actual}- $y_{prediction}$)
→ use the error to update weights, $W_{new} = W_{old} - \eta * \delta * x/m$
where, η is called the learning rate or step size and it determines how smoothly the learning process is taking place.
3. Repeat 2 until convergence.(i.e. error δ is zero)

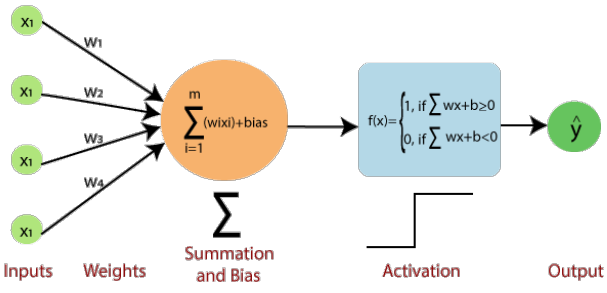


Figure 4. Single Layer Perceptron

5.3.2 Intuition

As SLP makes a Linear boundary to classify a linearly separable dataset. So if the classes male and feamale are linearly separable the SLP will give very high accuracy. Therefore SLP have been used on the dataset.

5.3.3 Observation

Model	Accuracy
SLP	76.19%
SLP with l1 regularization	85.11%

5.3.4 Inference

Using SLP gave an accuracy of 76.19% which implies that a large number of datapoints are linearly separable. When l1 regularization is used with SLP the accuracy improved by a substantially to 85.11% as l1 regularization have shrinked the less important feature's coefficient to zero thus, removing some feature altogether. Hence only those features are selected across which the classes are more linerly separable i.e have less noise.

5.4 K-Nearest Neighbour(KNN)

5.4.1 Algorithm

1. A positive integer k is specified, along with a new sample.
2. Select k entries in our database which are closest to the new sample.
3. Find the most common classification of these entries.
4. This is the classification we give to the new sample

The algorithm is based on feature similarity, i.e., the closeness of our out-of-sample features resemblance in our training set that determines how we classify a given data point:

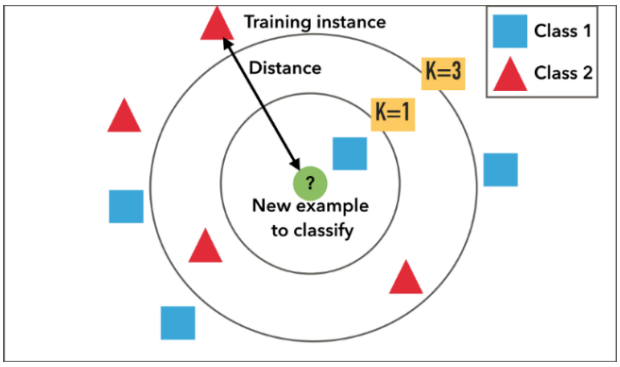


Figure 5. The test sample (inside circle) should be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 (outside circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If, for example k = 5 it is assigned to the first class (3 squares vs. 2 triangles outside the outer circle).

5.4.2 Intuition

As KNN algorithm is widely used for classification. In our case each pixel of the image will be considered as feature by the KNN algorithm. KNN was select as CNN also uses pixels for extracting feature for classification.

5.4.3 Observation

The accuracies for different K values are tabulated below:

K values	Accuracy(Euclidean)	Accuracy(Manhattan)
K = 1	71.95%	72.04%
K = 2	70.26%	70.73%
K = 3	72.41%	73.63%
K = 4	71.91%	72.96%
K = 5	73.21%	74.35%
K = 6	72.83%	73.97%
K = 7	73.59%	75.49%
K = 8	72.83%	73.93%
K = 9	73.34 %	75.07%
K = 10	73.80%	74.65%
K = 11	73.80%	74.98%

Also, the root of training sample size was used as the value of K and the accuracies are as follows:

K values	Accuracy(Euclidean)	Accuracy(Manhattan)
K = 146	72.12%	72.75%

5.4.4 Inference

As we could see that Manhattan distance has performed better than Euclidean distance as there is high dimensionality in the data. Moreover, KNN is sensitive to spatial shifts and hence it didn't perform well on the dataset.

5.5 Multi-layer Perceptron(MLP)

5.5.1 Algorithm

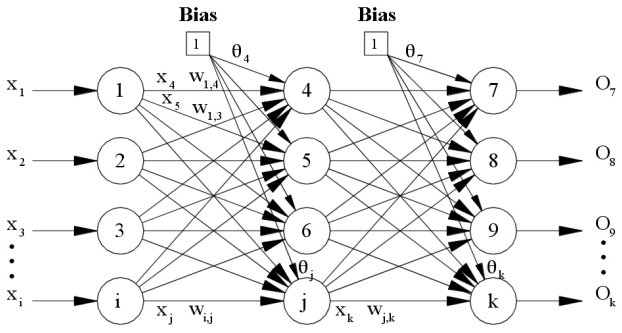


Figure 6. Multi-layer Perceptron(MLP)

5.5.2 Intuition

SLP gave an accuracy of 76% which implies that the classes are not linearly seprable. Therefore MLP has been used to find a non linear boundry which can give better accuracy compared to SLP

5.5.3 Observation

The results of age prediction using MLP are as follows:

Age-prediction Model	mse	mae
MLP with 1 layer	159.51	9.14
MLP with 2 layer	144.96	8.46
MLP with 3 layer	141.96	8.41
MLP with 4 layer	155.46	8.89

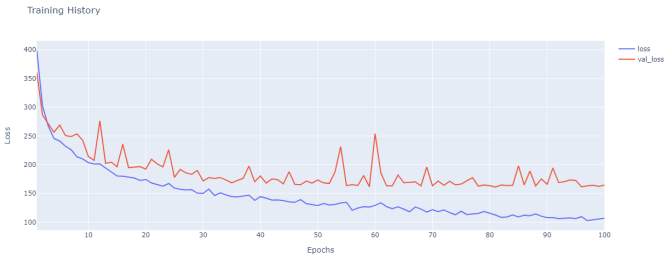


Figure 7. MLP for regression using 1 hidden layer

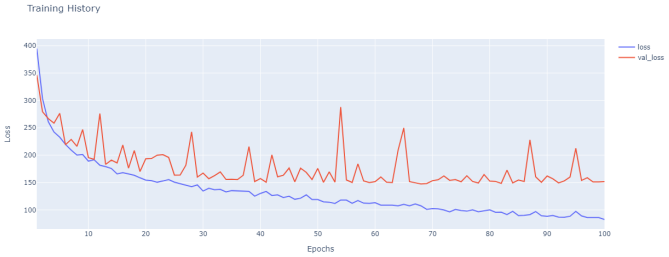


Figure 8. MLP for regression using 2 hidden layer

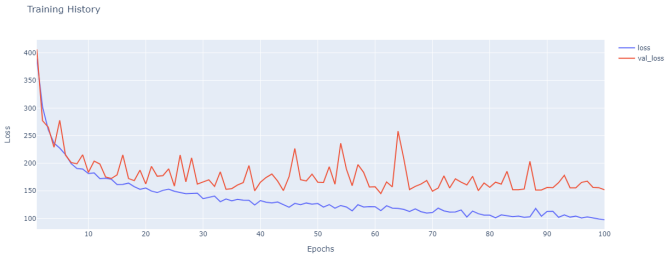


Figure 9. MLP for regression using 3 hidden layer

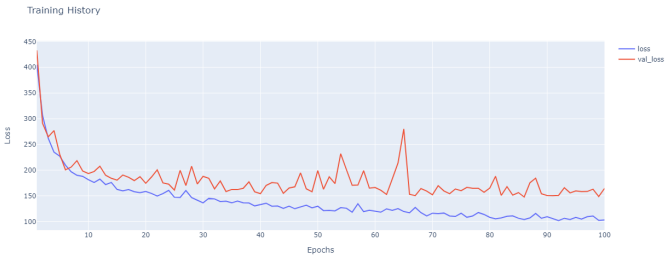


Figure 10. MLP for regression using 4 hidden layer

The results of gender prediction using MLP are as follows:

Gender-prediction Model	test-loss	Accuracy
MLP with 1 layer	0.30	89.05%
MLP with 2 layer	0.30	87.02%
MLP with 3 layer	0.37	85.53%
MLP with 4 layer	0.33	87.32%

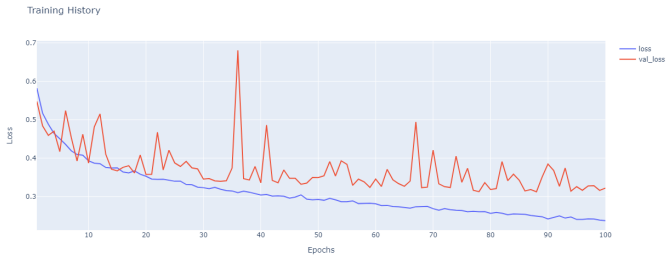


Figure 11. MLP for classification using 1 hidden layer

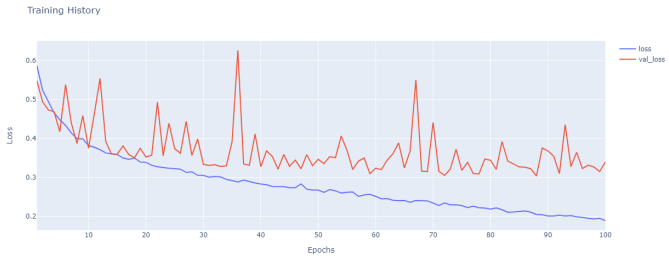


Figure 12. MLP for classification using 2 hidden layer

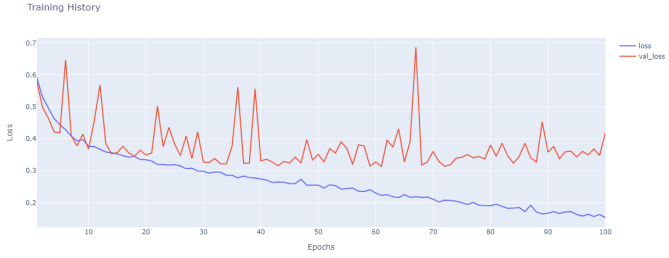


Figure 13. MLP for classification using 3 hidden layer

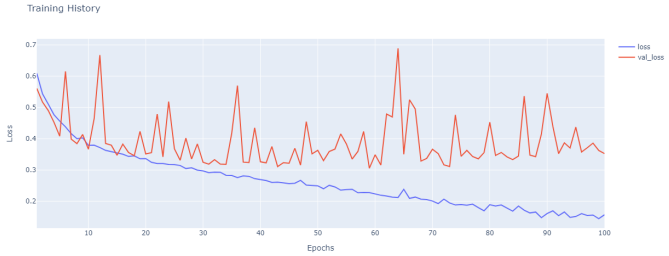


Figure 14. MLP for classification using 4 hidden layer

5.5.4 Inference

MLP didn't perform that well on this image dataset,as it takes flattened image as an input for which spatial information is lost. Moreover it overlooks the fact of translation invariance, for which MLP became sensitive to spatial shifts. Also, our model started to become over-parameterized on increasing hidden layers, so accuracy decreased.

5.6 Convolution Neural Network (CNN)[7]

5.6.1 Model Architecture

We have used keras with tensorflow for making our model. Here is the Network Architecture of our model:

1. 96 filters of size 7×7 pixels are applied to the input in the first convolutional layer, followed by a rectified linear operator (ReLU), a max pooling layer taking the maximal value of 2 × 2 regions with stride size equal to pool size.[10]

2. So, the 21 x 21 x 96 output of the previous layer is processed by the second convolution layer,containing filters of size 5×5 pixels, followed by a rectified linear operator (ReLU), a max pooling layer taking the maximal value of 2 × 2 regions with stride size equal to pool size .[6]

3. Then the 8 x 8 x 256 output of the previous layer is processed by the third convolution layer,containing filters of size 3×3 pixels, followed by a rectified linear operator (ReLU), a max pooling layer taking the maximal value of 2 × 2 regions with stride size 2.

The following fully connected layers are then defined by:

4. A first fully connected layer that receives the output of the third convolutional layer and contains 512 neurons, followed by a ReLU.

5. A second fully connected layer that receives the 512-dimensional output of the first fully connected layer and again contains 512 neurons, followed by a ReLU.

6. Finally, output layer contains 1 neuron with sigmoid or relu as activation function based on the task.

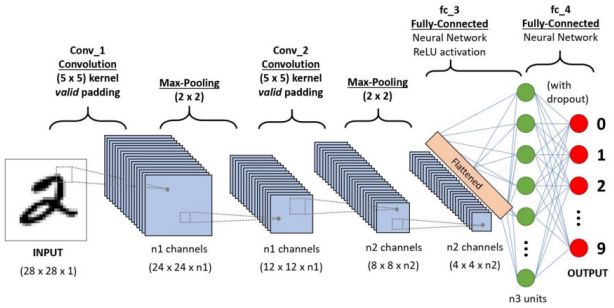


Figure 15. CNN architecture

5.6.2 Pooling layer:[8]

A common CNN architecture contains convolution layer and pooling layer staked one after another. In pooling layer we achieve the spatial invariance by minimizing the resolution of feature map. As it reduces the dimension of the feature map, the number of parameters are reduced and so the computations to be performed in the network. Moreover, by summarizing the features present in a particular region of the feature map, our model becomes flexible to any change in the position of the features in the input image.

Suppose, we have n x n window size, window can be of any arbitrary size and can be overlapping. Now, there are different types of pooling operations:

- Max-pooling.
- Average-pooling

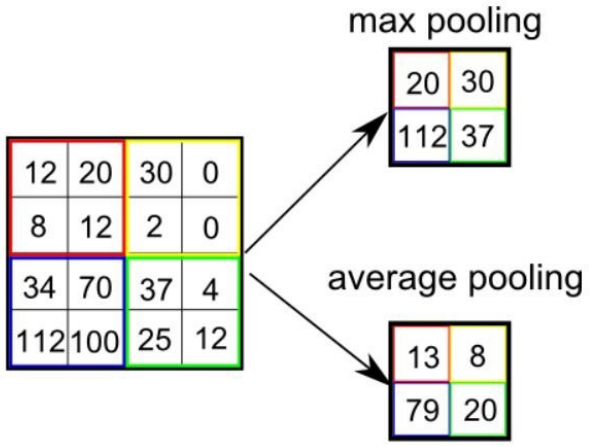


Figure 16. Max-pooling and avg-pooling

5.6.2.1 Max-pooling:

The max-pooling function takes the maximum out of the window applied on the feature map, so the output contains only the prominent features of the previous feature map. The function is as follows,

$$a_j = \underset{\mathbf{NXN}}{max}(a_i^{nXn} u(n,n)) \quad (1)$$

5.6.2.2 Average-pooling:

The average-pooling function takes the average out of the window applied on the feature map, so that the output contains only the average features of the previous feature map.The function is as follows,

$$a_j = \underset{\mathbf{NXN}}{avg}(a_i^{nXn} u(n,n)) \quad (2)$$

Evaluation of max-pooling and average-pooling on classification and regression task: We have implemented max-pooling and avg-pooling on gender prediction task and it is observed that after 10 epochs validation loss and training loss started to diverge(Figure).

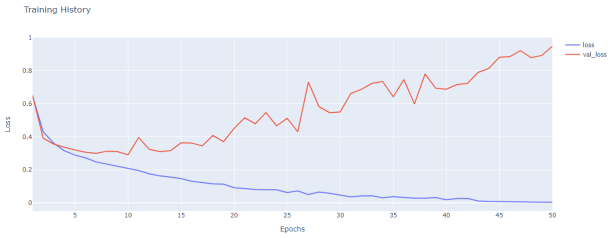


Figure 17. Max-pooling in classification task

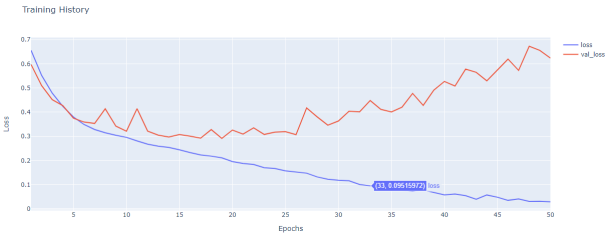


Figure 18. Average-pooling in classification task

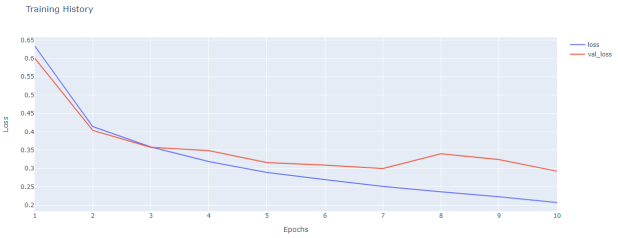


Figure 19. Max-pooling in classification using callback

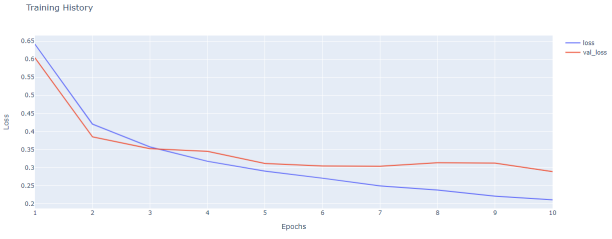


Figure 20. Average-pooling in classification using callback

For classification problem test accuracies are tabulated below:

Task	Method	Accuracy
Gender prediction(Classification)	Max-pooling	89.20%
	Avg-pooling	87.96%
	Max-pooling using callback	88.20%
	Avg-pooling using callback	88.24%
Task	Method	Mean absolute error
Age prediction(Regression)	Max-pooling	6.71
	Avg-pooling	6.83

Table 1. Variation in test-accuracies in max and average pooling

from the above table, it is clear that in classification problem max-pooling performs better than avg-pooling when callback is not used,but when training is stopped after 10 epochs, avg pooling performed a little better.Similarly, in case of regression max-pooling outperforms avg-pooling.

5.6.3 Learning Rate decay

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process.

The challenge of training deep learning neural networks involves carefully selecting the learning rate. It may be the most important hyperparameter for the model.

One solution is to use learning rate decay in our mode. With learning rate decay, the learning rate η is calculated each update (e.g. end of each mini-batch) as follows:

Where η is the learning rate for the current epoch, $\eta_{initial}$ is the learning rate specified as an argument to SGD, decay is the decay rate which is greater than zero and iteration is the current update number.

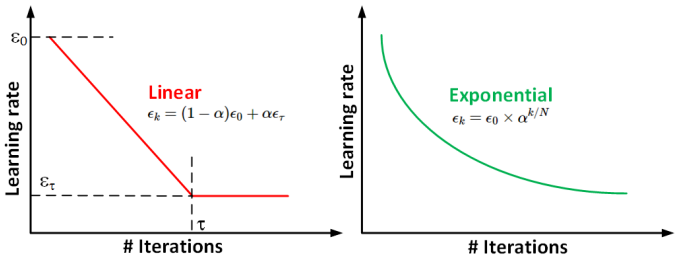


Figure 21. Linear and Exponential Learnin rate decay

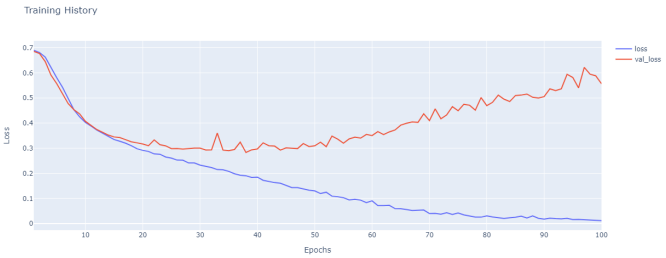


Figure 22. With exponential learning rate decay

5.6.4 Batch Normalization

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it no- toriously hard to train models with saturating non linearities. This phenomenon is referred as internal covariate shift.[5]
This problem is solved using Batch Normalization.Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

It does this scaling the output of the layer, specifically by standardizing the activations of each input variable per mini-batch, such as the activations of a node from the previous layer. Recall that standardization refers to rescaling data to have a mean of zero and a standard deviation of one, e.g. a standard Gaussian.

Batch normalization can be implemented during training by calculating the mean and standard deviation of each input variable to a layer per mini-batch and using these statistics to perform the standardization.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 23. Batch Normalization algorithm

We trained our model using first without batch normalization and then using batch normalization and noted down the

observation.

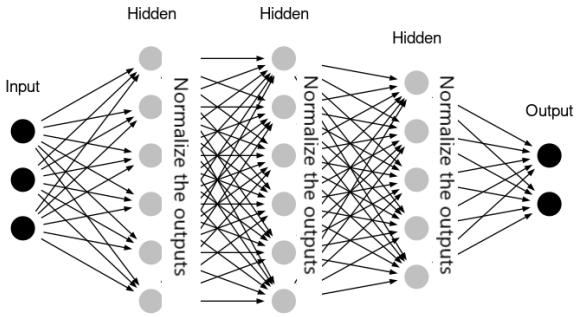


Figure 24. Model with BN after every layer

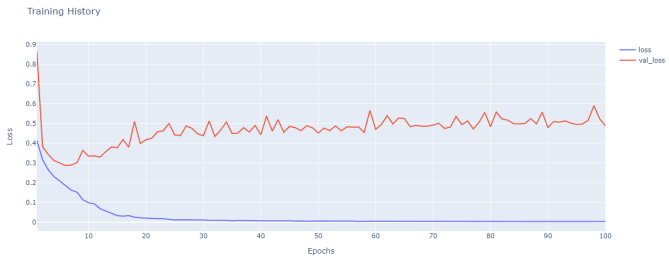
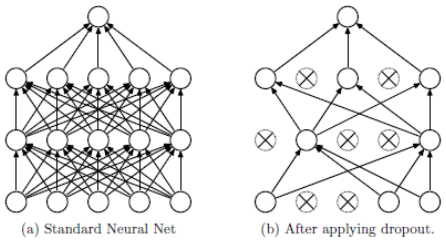


Figure 25. Using only batch normalisation

5.6.5 Dropout:

In deep neural network we generally have a large number of parameters, which leads to the problem of *"Overfitting"*. These types of networks contain multiple numbers of non-linear hidden layers, for which the model learns very complicated relationships between input and output. With a small training sample, many of these relationships can be the result of sampling noise which will only exist in training sample but not in test dataset even if it is drawn from the same distribution. [9]

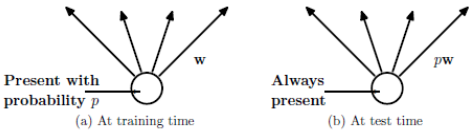
Now to address this problem of *"overfitting"*, we have a standard method called *"Dropout"*. In this method we simply drop some of the units (hidden and visible) in the neural network temporarily along with all its incoming and outgoing connections as shown in Figure 1.



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Now, the choice of dropping the units is random. In simple case, each neuron is retained with a fixed probability p independent of other neurons or dropout ratio p which depends on the validation set or simply set to 0.5. However for input layer, the optimal probability of retention of a neuron is close to 1 rather than 0.5.



(a) At training time (b) At test time

Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Dropout can be used in any layer i.e. input layer, convolution layer, output layer. After applying dropout we get a *"thinned"* network which contains all such neurons which survived dropout. Suppose we have a network with n number of nodes, then total number of possible *thinned* networks will be 2n. As they share weights, the number of parameters are still O(n^2) . So, training a neural network with dropout can be seen as training of 2^n collection of networks with extensive weight sharing, where each *thinned* network gets trained very rarely.

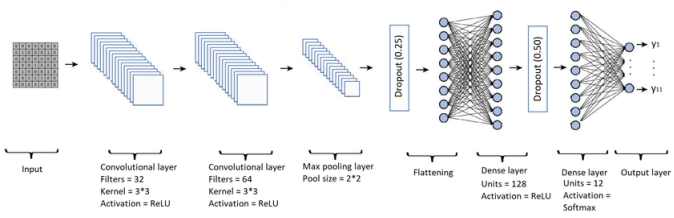


Figure 26. CNN architecture with dropout

At test time it is unfeasible to calculate the average of predictions from exponentially many *thinned* network. During training if a neuron is retained with a probability p the outgoing weights are multiplied by p at the time of testing in Figure 2, which ensures that the predicted output is same as the actual output at the test time.

Observation

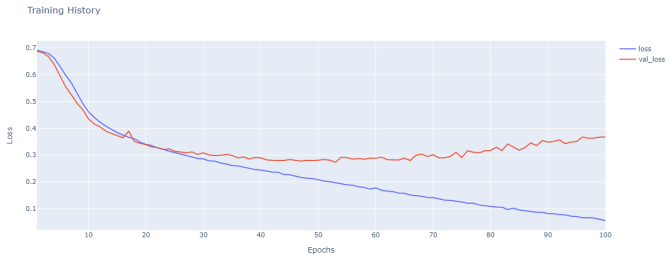


Figure 27. Using dropout with learning rate decay

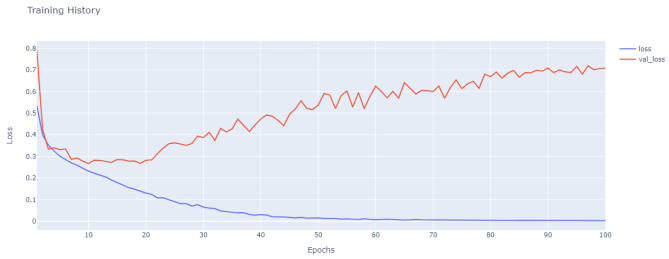


Figure 28. Using BN, dropout and Learning rate decay

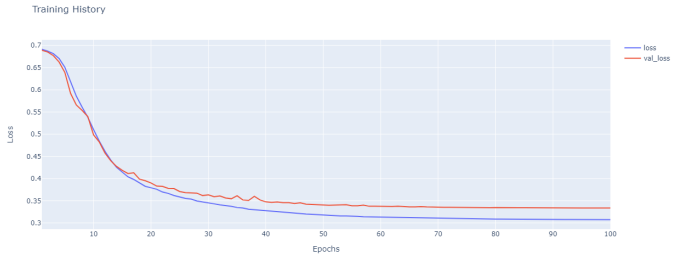


Figure 29. Using learning rate decay

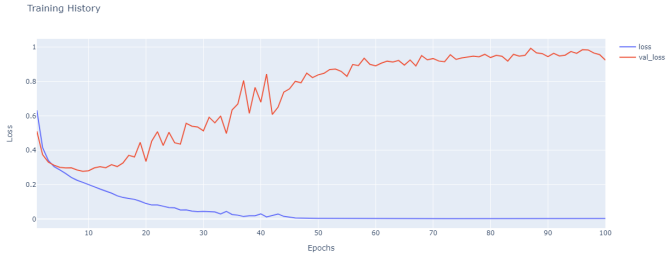


Figure 30. Without using BN, learning rate decay and dropout

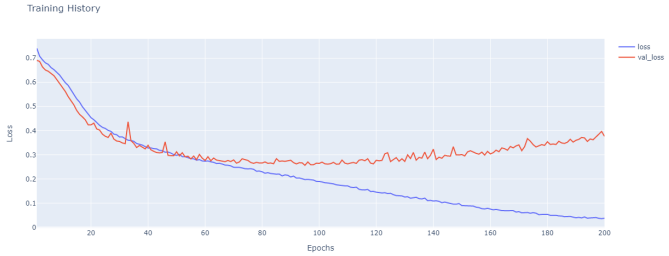


Figure 31. Using sigmoid as activation function

Model	Test Accuracy
BN + Dropout	87.98%
LRD	88.90%
BN + LRD	89.79%
Dropout + LRD	89.04%
BN + Dropout + LRD	90.17%
Without BN + Dropout + LRD	89.13%
BN	89.79%
with sigmoid	89.01%

5.6.6 Gradient Descent

Gradient descent is a way to minimize an objective function J(θ) parameterized by a model's parameters $\theta \in R^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla\theta$ J(θ) w.r.t. to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum.

5.6.6.1 Stochastic Gradient Descent:

This is the basic algorithm responsible for having neural networks converge, i.e. we shift towards the optimum of the cost function.

The equation for SGD is used to update parameters in a neural network – this equation is used to update parameters in a backwards pass, using backpropagation to calculate the gradient ∇

We can regard Stochastic Gradient Descent as another version of Gradient Descent(GD). We need to know that Gradient Descent algorithm update the parameters in using all samples. However, SGD just randomly choose one sample to update the parameter. Therefore, when there are a lot of samples in our dataset, if we use GD to update parameters, then the computation cost can be very very very large. However, if we use SGD, the computation cost will be very low. Thus, SGD is always faster than GD. Because SGD algorithm has this characteristic, it is widely used to train some machine learning model such as graphical models where the amount of data is usually very large.

$\theta = \theta$ - η $\nabla\theta$J(θ;x,y)

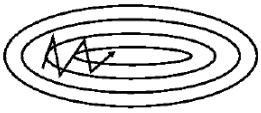


Figure 32. SGD with momentum

Figure 33. SGD without momentum

5.6.6.2 Momentum:

For each time we roll the ball down the hill (for each epoch), the ball rolls faster towards the local minima in the next iteration. This makes us more likely to reach a better local minima (or perhaps global minima) than we could have with SGD.

Momentum is where we add a temporal element into our equation for updating the parameters of a neural network – that is, an element of time.

This time element increases the momentum of the ball by some amount. This amount is called gamma γ , which is usually initialized to 0.9. But we also multiply that by the previous update vt

Function for momentum is basically the same as SGD, with an extra term:

$\theta = \theta - \eta \nabla J(\theta;x,y) + \gamma vt$

5.6.7 Adam

Adaptive Moment Estimation (Adam) is the next optimizer, and probably also the optimizer that performs the best on average. Taking a big step forward from the SGD algorithm to explain Adam does require some explanation of some clever techniques from other algorithms adopted in Adam, as well as the unique approaches Adam brings.

Adam uses Momentum and Adaptive Learning Rates to converge faster.

Adaptive Gradient Algorithm (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).

Root Mean Square Propagation (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

As for the Gradient Descent and Stochastic Gradient Descent algorithm we mentioned above, they have a common characteristic which is that we need to set the learning rate of these algorithms by ourselves and the learning rate is constant as we run the algorithm. Adam algorithm improves this situation. Specifically, this algorithm allows each independent variable in the objective function to have its own learning rate. Besides, along with the increase in the number of iterations, these learning rates will be constantly adjusted. We find that Adam is more flexible than SGD and GD. It utilized more information coming from dataset. This algorithm is widely used in deep learning.

5.6.8 Age Classification

Data Cleaning:

Kids of age less than 5 have been removed to get proper distribution of data as the number of datapoints for kids below 10 was very high.
Old people of age greater 90 has been removed as to get properly distributed data and also the datapoint for them is very less to train the model.
Model architecture used from [1]

Observation:

Accuracy before data cleaning was 90.17%. Where as after data cleaning accuracy reached 94.22% as a part of the missclassification was removed from the dataset.

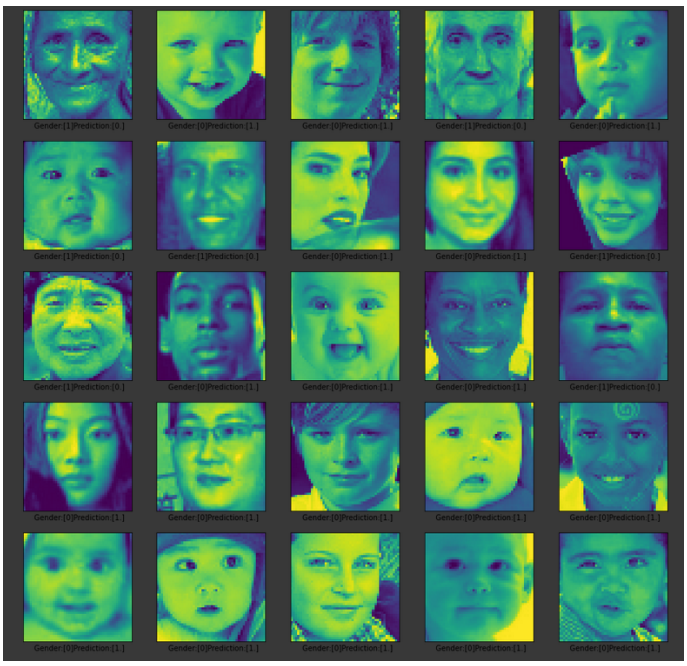


Figure 34. Missclassified images

Ethnicity wise classification:

There are 5 ethnicities in the data set. So we split the dataset ethnicity wise then trained the model on it. After that we trained 5 models.

Observation:

Model	Accuracy
Combined Prediction	88.60%
Best Prediction	90.02 %

6 Final Results:

For classification task:

Model name	Best accuracy
KNN	75.49%
SLP	85.11%
MLP	89.05%
CNN	90.17%
CNN after data-cleaning	94.22%

For regression task:

Model name	Best mae
KNN	12.01
MLP	8.41
CNN	6.78

7 Findings:

While exploring the data, it became quite evident that age and gender are quite ethincity dependent and a model that could learn simultaneously on these three tasks would perform quite better. Through this idea, the team came across MTCNN architecture. Though we were not able to implement it as it requires very long GPU training hours.

8 Future Work

The works that we carried out were in a short span and on our limited knowledge. There were other pretrained models like FaceNet and also the MTCNN architecture that demanded a thorough study. The work was carried on grayscale version of UTK Face dataset. The color version of the dataset would have helped to perform better on this task. The latest papers on this dataset have come in 2019 and 2020 like one with continual life-long learning[3] and deep ordinal regression[2] and due to our limited resources, which have performed considerably better on this dataset, will be the next focus of the team.

9 Acknowledgements

This project was a great opportunity for all of us. Though we were not habituated to this type of teaching process, but eventually we learnt a lot while working on this paper. Despite of having difficulties in working on such a paper eventually we completed the task and learned a lot in the process. We thank you very much for this wonderful learning opportinity.

References

[1] Octavio Arriaga, Matias Valdenegro-Toro, and Paul Plöger. *Real-time Convolutional Neural Networks for Emotion and Gender Classification*. 2017. arXiv: 1710.07557 [cs.CV].

[2] Axel Berg, Magnus Oskarsson, and Mark O'Connor. *Deep Ordinal Regression with Label Diversity*. 2020. arXiv: 2006.15864 [cs.LG].

[3] Wenzhi Cao, Vahid Mirjalili, and Sebastian Raschka. "Rank consistent ordinal regression for neural networks with application to age estimation". In: *Pattern Recognition Letters* 140 (Dec. 2020), pp. 325–331. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2020.11.008. URL: http://dx.doi.org/10.1016/j.patrec.2020.11.008.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[5] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[6] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[7] Gil Levi and Tal Hassner. "Age and Gender Classification Using Convolutional Neural Networks". In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) workshops*. June 2015. URL: %5Curl%7Bhttps://osnathassner.github.io/talhassner/projects/cnn_agegender%7D.

[8] Dominik Scherer, Andreas Müller, and Sven Behnke. "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition". In: Jan. 2010, pp. 92–101. ISBN: 978-3-642-15824-7. DOI: 10.1007/978-3-642-15825-4_10.

[9] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[10] J. Wu. "Introduction to Convolutional Neural Networks". In: 2017.