PROJECT-4 REPORT

Introduction to Machine Learning - CSE574

Jayant Solanki

UBIT Name: jayantso
Person Number: 50246821
Fall 2017 CSE

Swati Nair

UBIT Name: swatishr
Person Number: 50246994
Fall 2017 CSE

INTRODUCTION

This report contains the implementation details of Project 4 i.e. determining whether the person in the portrait image is wearing glasses or not, using Convolutional Neural Network.

OBJECTIVES ACHIEVED

We have trained our classification model on Celeb dataset using convolutional neural network and predicted the labels of the test celeb images i.e. whether the person is wearing eyeglasses or not.

After training the model using CNN, we achieved an accuracy of approximately 95% on the test images.

IMPLEMENTATION

1. **Data Extraction:**
    a. We extracted the entire celeb dataset (images and labels) by reading the and then for training the model, we chose different sizes of dataset. Initially, just for testing purpose, we used 1000 data samples for training, then we trained the model on 20K training samples, 70K samples and finally on 2L dataset. We partitioned the dataset into training, validation and test set.
    b. Below steps were followed:
        i. Labels for eyeglasses were first extracted from the list_attr_celeba.txt (column 16th). Converted the -1s label to 0s depicting absence of glasses,1st depicted presence of glasses.
        ii. Read the images from img_align_celeba folder and converted the RGB images into grayscale. This was done as we only have to determine eyeglasses and it can be achieved even with a grayscale image.
        iii. Resized the image to a smaller dimension (tried 28x28 and 32x32)
        iv. Saved the image names, image pixels data and labels corresponding to each data in an npz file on the disk.
        v. Standardized the image pixel values to be in a range of 0 to 1

vi. Flattened each image into 1D array

C. Also, we tried a different variant of data extraction for 70K data samples.

i. In this, all the 202599 labels were read and the ones having eyeglasses were extracted.
ii. The total count of such images was 13193.
iii. Then, we extracted 56807 non-eyeglass images and combined both the sets, shuffled the data and created a new dataset of size 70K.
iv. This was done because the ratio of eyeglass images to non-eyeglass images is too low in the current celeb dataset. So, in order to train the model effectively, we needed a good proportion of eyeglasses images in the dataset. This data extraction method helped us to achieve it.

2. Training and testing the CNN model:

a. Tried different variants of model as a part of hypertuning and selected the optimum one which gives minimum validation error. We have attached the dataset as the table in the hyperparameters tuning section.
b. The model was first implemented, trained and tested using generic Tensorflow libraries. In an effort to reduce the training time, tf.estimator was used which improved the speed to a large extent.
c. The model consists of 4 layers: Convolutional layer 1 (convolution with 32 features applied on 5x5 patch of image + 2D max pooling), Convolutional layer 2 (convolution with 64 features applied on 5x5 patch of image + 2D max pooling), fully connected layer with 1024 neurons and ReLU activation function, logit layer with 2 units corresponding to 2 labels
d. In fully connected layer, few neuron outputs are dropped out to prevent overfitting. The no_drop_prob placeholder makes sure that dropout occurs only during training and not during testing
e. Tuned the model by varying different hyperparameters like dropout rate, number of hidden layers, number of nodes in hidden layers, optimizer type (GradientDescentOptimizer, AdamOptimizer) with learning rate set to **0.001** and varying number of steps on input batches of size 100 and chose that model which has the minimum cross-entropy error on validation set

f. Then, ran the selected model on test celeb images to get test accuracy.

TASKS COMPLETED:

1. **Extracted feature values and labels from the data:** Since extracting the features in original resolution severely clogged the system's RAM, we downsampled the images.
2. **Reduce the resolution of the original images:** We worked with images resized to dimension, 28x28
3. **Reduce the size of training set:** Initially according to the the mentioned paper we took 20000 training sets.
4. **Data Partition**: We partitioned the dataset into training, validation and test sets after shuffling the data
5. **Apply dropout or other regularization methods:** We experimented with different dropout values, Details have been provided in the table in Hyperparameters tuning section.
6. **Train model parameter:** We used SGD and AdamOptimizer with mini batches. AdamOptimizer was found to be much faster and achieved the higher accuracy in lower number of epochs
7. **Tuned hyper-parameters:** we used the automate.py script to create the grid map for different values of the dropout and hidden layers and hidden nodes.
8. **Retrain the model using higher resolutions:** We used 32x32 image resolution. The performance improved as we increased the resolution
9. **Use bigger sizes of the training set:** We augmented the training set to 50K, 70K and original dataset count.

IMPROVEMENTS

1. The labels and images were read and stored in the file system as numpy files and were later reloaded every time whenever needed. This saved us from unnecessary data read time.
2. Similar to data augmentation, we tried to achieve better and real accuracy by selecting all the eyeglasses images from entire dataset (13193 out of 202599) and combining it with 56807 non-eyeglass images to generate a dataset of size 70K. This dataset was then shuffled, and partitioned into training set(50400), validation set (5600) and test set (14000). The pros of this method was that the dataset now had enough eyeglasses and non-eyeglasses images to be trained on, thus

improving the generalization of the model.

CODE LAYOUT

Below are the files present in proj4code.zip:

1. **Main.py**: main code file
2. **lib.py**: contains helper functions
3. **Automate.py:** used for hyperparameter tuning

HYPERPARAMETER TUNING:

Tuning of number of layers in CNN:

*Number of layers: **2** [Convolutional layers with max-pooling, 2 fully connected layers]*
*Dropout rate: **0.3, 0.4, 0.5**,*
*learning rate: **0.001**,*
*image dimension: **28x28**,*
*number of nodes in the 3 convolutional layers: **32, 64***
*DataSet: **70000***
Best Accuracy obtained is highlighted in green

Sr. No	Resolution	Dropout rate	Number of Convolutional Layers	Number of nodes	Train Accuracy	Validation Accuracy	Test Accuracy
Trained using SGD (Epoch 10000)							
1	28x28	0.3	2	32, 64	0.90134919	0.89892858	0.90742856
2	28x28	0.3	2	64, 128	0.89954364	0.89910716	0.90721428
3	28x28	0.3	2	128, 256	0.90037698	0.89928573	0.9069286
4	28x28	0.4	2	32, 64	0.89515871	0.89285713	0.90135711
Trained using AdamOptimiser (Epoch 1000)							
5	**28x28**	**0.4**	**2**	**32, 64**	**0.94714284**	**0.94410712**	**0.94035715**
6	28x28	0.4	2	64, 128	0.93579364	0.93410712	0.93457144

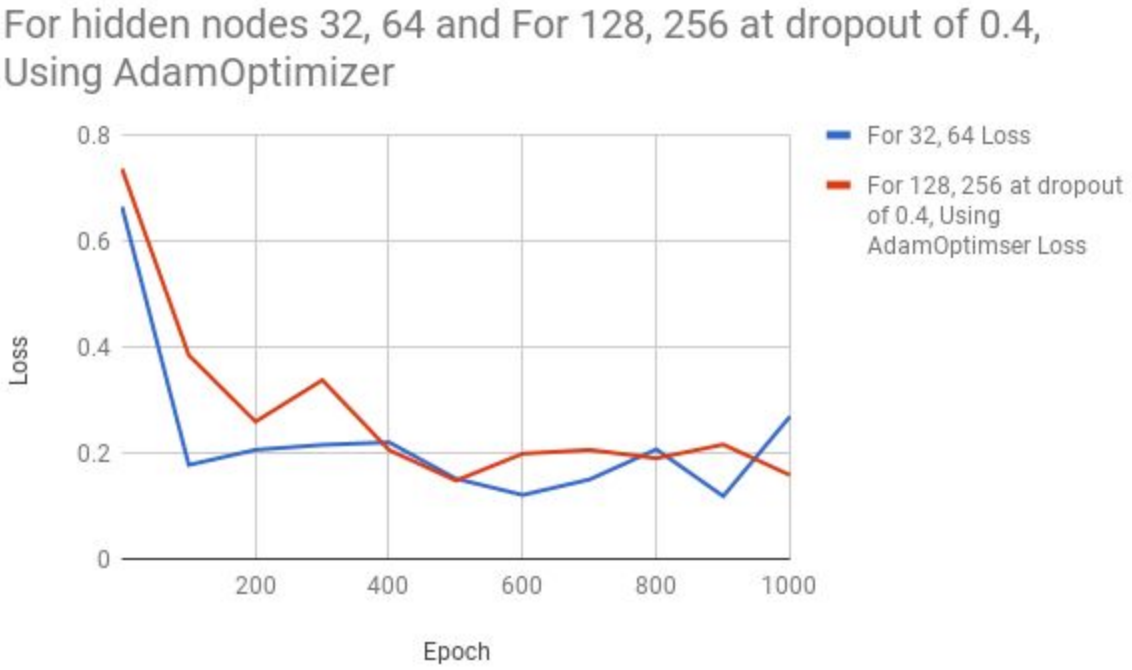
7	28x28	0.4	2	128, 256	0.92555553	0.92214286	0.92307144
8	28x28	0.5	2	32, 64	0.93267858	0.93339288	0.93142855
9	28x28	0.5	2	64, 128	0.94003969	0.93892854	0.93785715
10	28x28	0.5	2	128, 256	0.93531746	0.93464285	0.93321431

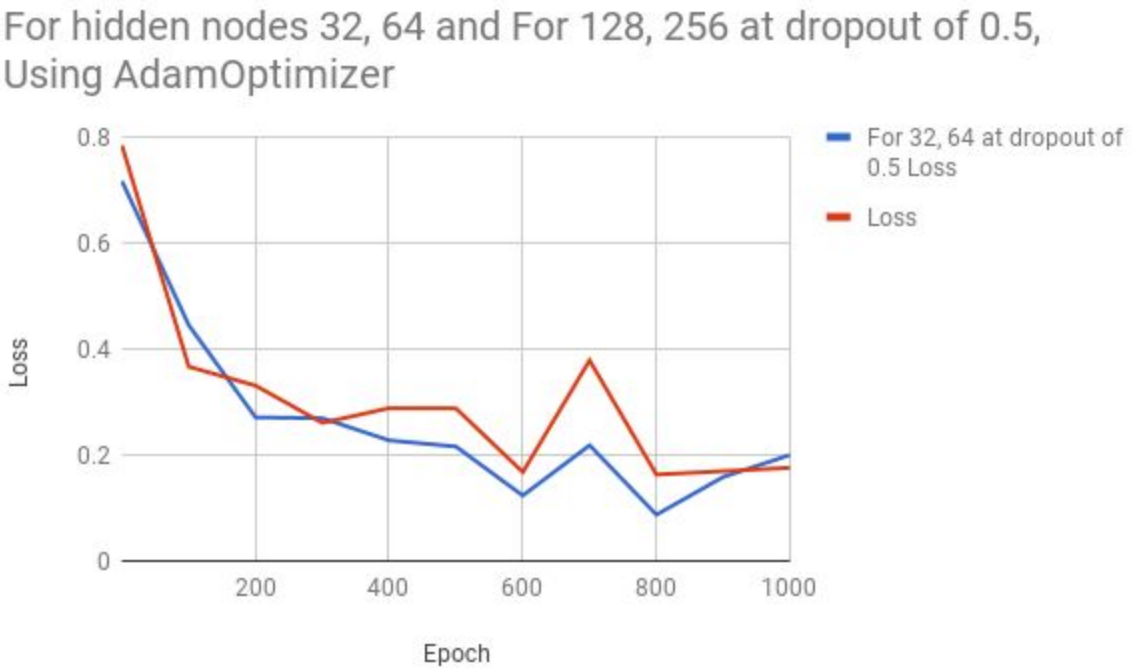
*Number of layers: **3** [Convolutional layers with max-pooling, 2 fully connected layers]*
*Dropout rate: **0.5**,*
*learning rate: **0.001**,*
*image dimension: **32x32**,*
*number of nodes in the 3 convolutional layers: **32, 64, 128***
*DataSet**: 70000***

Sr. No	Epochs	Optimizer Used	Train Accuracy	Validation Accuracy	Test Accuracy
1	10000	GradientDescentOptimizer	0.9	0.89	0.897
2	10000	AdamOptimizer	0.987	0.946	0.948
3	1000	AdamOptimizer	0.94	0.939	0.942

GRAPHS

Loss over Epoch


For hidden nodes 32, 64 and For 128, 256 at dropout of 0.4, Using AdamOptimizer
0.8
0.6
0.4
0.2
0
Loss
200
400
600
800
1000
Epoch
For 32, 64 Loss
For 128, 256 at dropout of 0.4, Using AdamOptimser Loss


For hidden nodes 32, 64 and For 128, 256 at dropout of 0.5, Using AdamOptimizer
0.8
0.6
0.4
0.2
0
Loss
200
400
600
800
1000
Epoch
For 32, 64 at dropout of 0.5 Loss
Loss

RESULTS

Using SGD
For dropout 0.30
32 64
Training set accuracy {'accuracy': 0.90134919, 'loss': 0.26699808, 'global_step': 10000}
validation set accuracy {'accuracy': 0.89892858, 'loss': 0.26727545, 'global_step': 10000}
Test accuracy {'accuracy': 0.90742856, 'loss': 0.25485015, 'global_step': 10000}
64 128
Training set accuracy {'accuracy': 0.89954364, 'loss': 0.26833287, 'global_step': 10000}
validation set accuracy {'accuracy': 0.89910716, 'loss': 0.26699775, 'global_step': 10000}
Test accuracy {'accuracy': 0.90721428, 'loss': 0.25618285, 'global_step': 10000}
128 256
Training set accuracy {'accuracy': 0.90037698, 'loss': 0.26520491, 'global_step': 10000}
validation set accuracy {'accuracy': 0.89928573, 'loss': 0.26478675, 'global_step': 10000}
Test accuracy {'accuracy': 0.9069286, 'loss': 0.25473234, 'global_step': 10000}
For dropout 0.40
32 64
Training set accuracy {'accuracy': 0.89515871, 'loss': 0.27771208, 'global_step': 10000}
validation set accuracy {'accuracy': 0.89285713, 'loss': 0.27534744, 'global_step': 10000}
Test accuracy {'accuracy': 0.90135711, 'loss': 0.26559976, 'global_step': 10000}

Using AdamOptimiser

For dropout 0.40
32 64
Training set accuracy {'accuracy': 0.94714284, 'loss': 0.13676977, 'global_step': 2000}
validation set accuracy {'accuracy': 0.94410712, 'loss': 0.15439203, 'global_step': 2000}
Test accuracy {'accuracy': 0.94035715, 'loss': 0.15354592, 'global_step': 2000}
64 128
Training set accuracy {'accuracy': 0.93579364, 'loss': 0.16461349, 'global_step': 1000}
validation set accuracy {'accuracy': 0.93410712, 'loss': 0.17125739, 'global_step': 1000}
Test accuracy {'accuracy': 0.93457144, 'loss': 0.16714543, 'global_step': 1000}
128 256
Training set accuracy {'accuracy': 0.92555553, 'loss': 0.19548431, 'global_step': 1000}
validation set accuracy {'accuracy': 0.92214286, 'loss': 0.20418282, 'global_step': 1000}
Test accuracy {'accuracy': 0.92307144, 'loss': 0.20004663, 'global_step': 1000}

For dropout 0.50
32 64
Training set accuracy {'accuracy': 0.93267858, 'loss': 0.16984062, 'global_step': 1000}
validation set accuracy {'accuracy': 0.93339288, 'loss': 0.17521785, 'global_step': 1000}
Test accuracy {'accuracy': 0.93142855, 'loss': 0.17155841, 'global_step': 1000}
64 128
Training set accuracy {'accuracy': 0.94003969, 'loss': 0.15690202, 'global_step': 1000}

validation set accuracy {'accuracy': 0.93892854, 'loss': 0.16877748, 'global_step': 1000}
Test accuracy {'accuracy': 0.93785715, 'loss': 0.1625112, 'global_step': 1000}
128 256
Training set accuracy {'accuracy': 0.93531746, 'loss': 0.18713054, 'global_step': 1000}
validation set accuracy {'accuracy': 0.93464285, 'loss': 0.19151181, 'global_step': 1000}
Test accuracy {'accuracy': 0.93321431, 'loss': 0.19136761, 'global_step': 1000}

DATASET:

Datasheet on https://goo.gl/3z5BsL

SOFTWARE/HARDWARE USED

- Sublime Text 3,
- Python 3 Environment based on Anaconda,
- TensorFlow,
- Ubuntu 16 System, Intel core i3 processor.
- Windows 10, Intel core i7 8th gen processor., 16GB RAM
- Python libraries: numpy, sklearn

REFERENCES

1. Ublearns
2. Stackoverflow.com
3. Python, Numpy and TensorFlow documentations
4. https://cs231n.github.io/convolutional-networks/
5. Liu, Ziwei; Luo, Ping; Wang, Xiaogang; Tang, Xiaoou. “Deep Learning Face Attributes in the Wild”