CS3216 Homework 3: Labeled Faces in the Wild

Xinyi Xu - A0142487Y Oh Han Gyeol - A0144061U

Choice of Model and Algorithm

Since it is an image classification task, Convolutional Neural Networks was our primary choice given in inherent capability of dealing with data with spatial correlations, specifically images. That being said, we still have attempted to construct other models to tackle this task, but all of which seemed lacking in performance as compared to a CNN model. The reason is that other models treat the individual pixels as separate features independent of each other, which results in the model's inability to correctly capture the underlying relationships among the pixels and thus inferior performances. Additionally, while 50*37 may be a moderately small image for a CNN model, 1850 features are already overwhelmingly large for some models such as SVM, KNN classifiers and Decision Tree. This caused the training time of these models to be significantly longer while still overfitting the data badly.

Our group used Keras Python library to construct the CNN model for this project.

Model Architecture

As mentioned above, the size of the images is 50*37 in addition to the size of the dataset, which is limited to 966 labeled instances. The information captured by this dataset is rather limited, if not very little, to deep CNNs. Therefore, the architecture of the CNN model, including the number of hidden layers, number of neurons per layer and the density of connection are to be carefully calibrated so as to avoid overfitting. In other words, we need to construct a CNN that is relatively shallow in depth and sparse in connections but just complex enough to preserve and propagate the information in the images. By trial and error, we have decided to construct a CNN with 2 convolutional layers that each gives 32 dimensions of output space. And each hidden layer is followed by a max pooling layer of size 3*3. After the convolutional layers, the output is flattened and goes through two dense layers to give out classification predictions, using a softmax function. The figure below demonstrates our model architecture.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_38 (Conv2D)           (None, 48, 35, 32)        608
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 46, 33, 32)        9248
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 15, 11, 32)        0
_________________________________________________________________
dropout_31 (Dropout)         (None, 15, 11, 32)        0
_________________________________________________________________
flatten_16 (Flatten)         (None, 5280)              0
_________________________________________________________________
dense_32 (Dense)             (None, 2048)              10815488
_________________________________________________________________
dropout_32 (Dropout)         (None, 2048)              0
_________________________________________________________________
dense_33 (Dense)             (None, 7)                 14343
=================================================================
Total params: 10,839,687
Trainable params: 10,839,687
Non-trainable params: 0
```

Figure 1. Model architecture

The choice of loss function presented some problems. The intuitive categorical cross-entropy in such a multi-class classification problem was not able to provide as good performance whilst the binary cross-entropy was able to yield better performance during training but tends to cause the model to overfit rather badly.

Optimisation and Regularization Techniques

The limited size of both image and the dataset dictates the need for optimisation and regularization techniques to ensure the model performance. The following are the several approaches that we implemented for such purpose.

1. Hyperparameter Tuning

After choosing the best performing CNN architecture, we used Bayesian Optimization to tune some of the hyperparameters of the CNN model such as the size of each hidden layer, dropout rate, batch size, number of epochs. We divided the dataset into the training set and the validation set, and chose the test accuracy of the validation set as the value to optimize. Using Gaussian Process with Upper Confidence Bound as the acquisition function, we managed to get the the set of parameters that gave us the highest accuracy after going through 100 iterations.

2. Data Preprocessing

This dataset has no missing data point and all data instances, though of low resolution, do not seem to contain heavy noises. Also, there were no apparent anomalies in the data set. Hence, adding noise reduction methods such as applying gaussian blur actually reduced the accuracy of the prediction. In addition, all images are normalised and done by dividing the pixels by 255 since it is the max-min for the pixel value. Such practice can help speed up the training and prevent the weights from growing unnecessarily large.

To mitigate the effect of low resolution, we preprocessed the dataset by applying edge detection on both training and test datasets. We choose Sobel operator for the edge detection method. After applying the horizontal and vertical Sobel filter, we got the intensity change at each pixel by calculating the squared sum of vertical and horizontal changes. We also calculated the direction of pixel intensity change by calculating the inverse tangent of the horizontal and vertical changes. By doing so, the dimension of the image data is increased by one, adding more information for the cnn to utilize. Below is a figure of the applying edge detection by opencv.

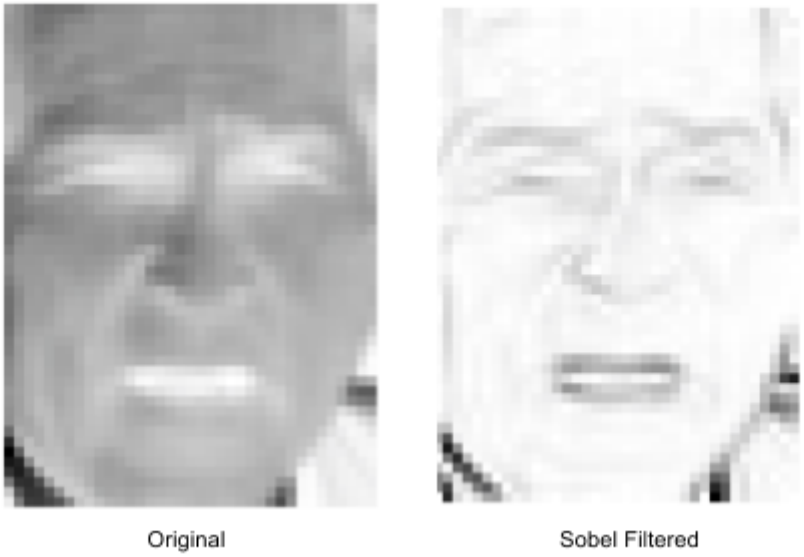

Figure 2. Image preprocessing: edge detection

3. Data Augmentation

Keras provides rather extensive image augmentation techniques and we have utilized several to ‘create’ more data to train our model on. To illustrate, we have included techniques like rotation, width shift, height shift and etc. Below is a comparison between original images and the ‘generated’ images.These techniques are selected since the faces given in the training data are not perfectly aligned in the center, nor are they at the same angle. Hence, these augmentation techniques will allow the model to train on a more varied data in terms of relative positions of the faces in the frame as well as the tilts of the faces. However, to our dismay, this particular implementation provided by Keras was not particularly effective in improving the performance of our model.

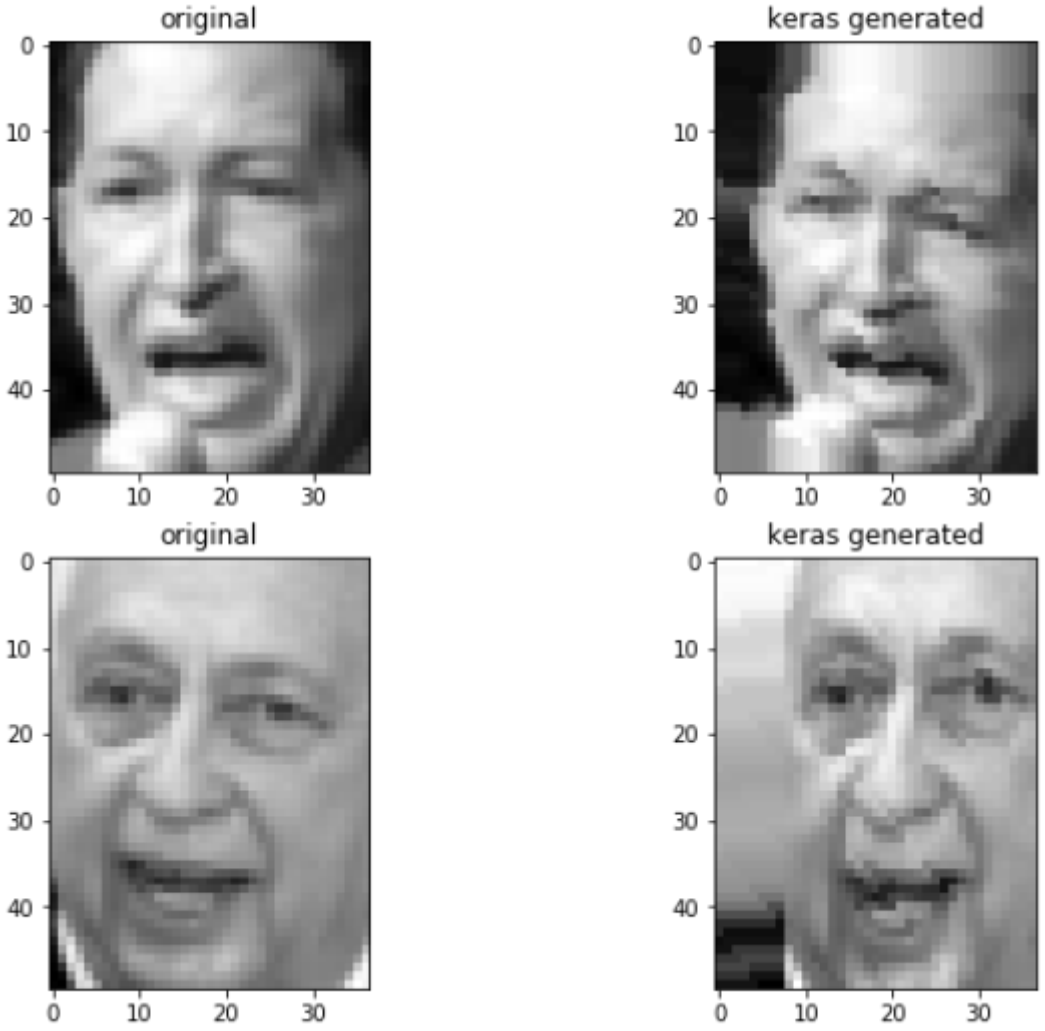

Figure 3. Keras Image Generator

4. Model Pretraining

As discussed above, we ‘created’ more data points by performing some image transformation techniques. We then used these transformed data points for a pre-training stage of the model. During pre-training, the model can learn some underlying structure of the faces and thus initializing and pre-training the weights to be able to capture pixels relevant for face recognition. Subsequently, the model is trained on the original data (normalised) for a more precise and accurate iteration of training of the weights.

We have also attempted to pre-train our model on additional data from publicly available data repos, including CIFAR-10. However, due to the a pre-trained model on CIFAR-10 did not seem to speed up the

training process on the face data nor did it yield better performance. One possible cause for that is that the features that the model learnt during pretraining was not transferrable to the actual training. This is because the classes in CIFAR-10 dataset are considerably different from the faces dataset. Therefore, the weights initialised and updated during pretraining were not able to capture the features in the faces datasets well and thus the less than fruitful attempt at pretraining. Additionally, we did manage to retrieve one external faces dataset. However, unfortunately this particular dataset faced the same problem of be small in size but large in number of classes, where it has 40 classes but only 10 instances of which class. Furthermore, the size of each picture from that external dataset is rather small. Unsurprisingly, by the curse of dimensionality, the pre-training on this dataset did not help with the performance of our model much, if at all.

5. Regularization

Regularization in CNN models is most commonly implemented by using Dropout, where for each training batch, a certain proportion of neurons are 'dropped out' so that the model needs to depend on fewer neurons. This approach is one way to incorporate higher variance to our model. Given N neurons in total, the total theoretical model space is in O(2^N), which correlates to the hypothesis space of our model. Therefore, Dropout allows the model to reduce overfitting by introducing more variance in the hypothesis set. Additionally, separate implementations of L1 and/or L2 regularizers are supported by Keras. Therefore, we have also employed these regularizers on Convolution2D layers and the fully connected layers to both weights of the neurons and the bias term of the layers. The regularization techniques employed were effective as can be demonstrated by the cross-validation during the training stage. The difference in the training accuracy and validation accuracy is significantly reduced to reasonable range of usually less than 5%.

6. Visualization of the dense layer

In order to better understand the performance of our model, we constructed plots of the activation areas of the last dense layer of our model. More specifically, on the sobel-transformed images, we were able to map the activation status of the dense layer of the model. Figures below demonstrate such visualization. The 'vanilla', 'guided' and 'relu' are three ways of registering the activation in the last layer provided by an external library. In comparison, the sobel-transformed images register more distinct activations in the dense layer which we believe is key part why these training on these images led to improvement in the performance.

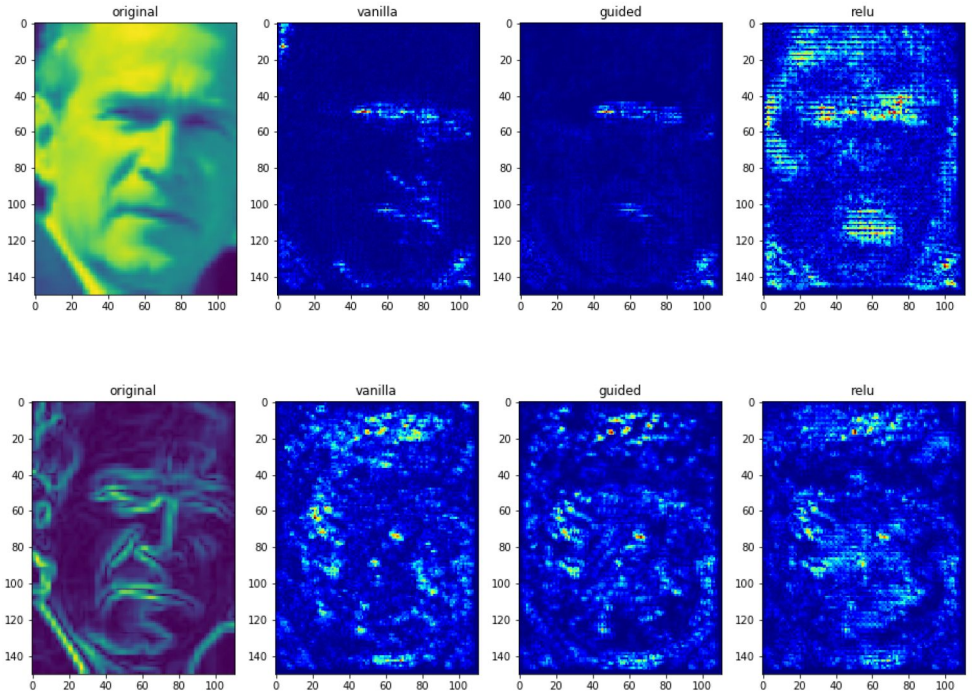

Figure 4. Activation status of dense layer

Model Performance

The model yields 1.00 accuracy when tested on the training dataset, and yields 0.93167 accuracy when test on the Kaggle's test dataset.

File listing for Submission

a. IPython Notebook for constructing and training model and visualization
b. Readme.pdf
c. A zip file including the figures and extra figures obtained but not shown in the report. (figures.zip)
d. y_test.csv (Prediction for Kaggle test set)

Team member's individual contribution

Han (A0144061U)

- Construction of model
- Image preprocessing
- Hyperparameter tuning

Xu Xinyi (A0142487Y)

- Construction of model
- Image preprocessing and augmentation
- Regularization
- Visualization (plotting figures)

Statement of Individual Work

Xu Xinyi

I, A0142487Y, certify that I have followed the CS3244 Machine Learning class guidelines for homework assignments. In particular, I expressly vow that I have followed the Facebook rule in discussing with others in doing the assignment and did not take notes (digital or printed) from the discussions.

Oh Han Gyeol

I, A0144061U, certify that I have followed the CS3244 Machine Learning class guidelines for homework assignments. In particular, I expressly vow that I have followed the Facebook rule in discussing with others in doing the assignment and did not take notes (digital or printed) from the discussions.

References

Cl.cam.ac.uk. (2002). The Database of Faces. [online] Available at: http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html [Accessed 25 Oct. 2017].

Keras: The Python Deep Learning library. (n.d.). Retrieved November 07, 2017, from https://keras.io/

Krizhevsky, A., Nair, V. and Hinton, G. (2009). CIFAR-10 and CIFAR-100 datasets. [online] Cs.toronto.edu. Available at: https://www.cs.toronto.edu/~kriz/cifar.html [Accessed 1 Nov. 2017].