# Data Preparation

We were given 452,261 images from the Imdb data set. These images have been cropped to create images of people’s head with 40% margin. A large portion of the images had multiple people present, and therefore resulting in wrongly labelled cropped images as each image created multiple cropped headshots.

The data labels were given in a Matlab file. We parsed the content using Scipy library, and stored in a Pandas dataframe.

We first ignored any cropped images with multiple people present, as their labels may be wrong. We then noticed there are a number of wrongly cropped images with no faces obviously present, we removed these from our data set as well, leaving 181,626 images in the dataset. However at this point, there was more male than female data points. Using random sampling, we further discarded a number of male data points to eliminate sample bias.

Our final data set has a total of 163,336 images, the data set is perfectly balanced with male and female genders each having 81,668 samples.

To be able to experiment our models in stages in a time-efficient manner, we created smaller datasets. We created dataframes containing 1 percent, 5 percent, 10 percent, and 20 percent of the final dataset. For each of the dataset, we further divided it into training set and test set at 95%-5% split.

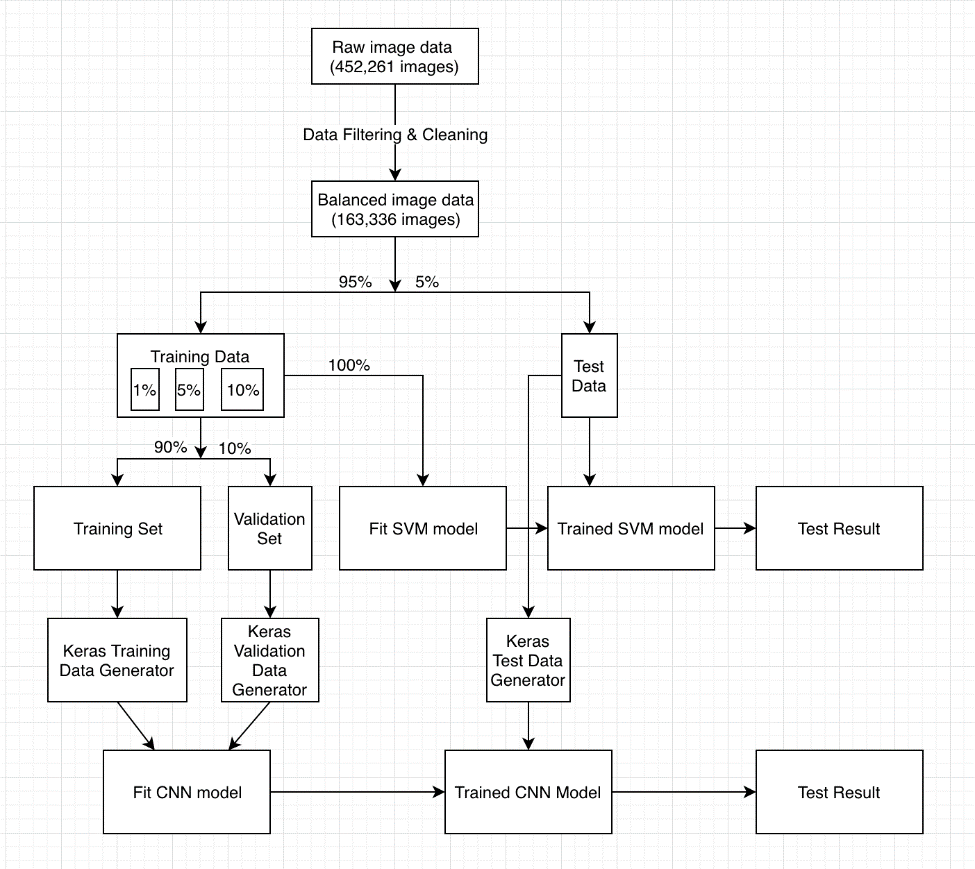
The dataframes contain the following columns:

* ‘path’: relative file path to the jpg image file
* ‘id’: an unique id associated with the person
* ‘name’: name of the person
* ‘dob’: date of birth
* ‘gender’: the person’s biological gender
* ‘score1’: face score of the person
* ‘score2’: face score of a second person if present
* ‘pic\_date’: date the picture was taken
* ‘region’: the region on the original image where the cropped image comes from
* ‘age’: calculated age of the person at the time the photo was taken

We are only interested in the ‘path’ and ‘gender’, but we kept the other information in the dataframes for future studies involving the same dataset.

# Data Preprocessing

We used a data generator from Keras library to feed images into our training model. For most of the training experiments, the images were resized to 100 pixel by 100 pixels, gray scale, and normalized. The batch size we used for training was 64. The training data set was further divided into training set and validation set with a 90%-10% split. The chart below highlights how data is used in the training and testing process.

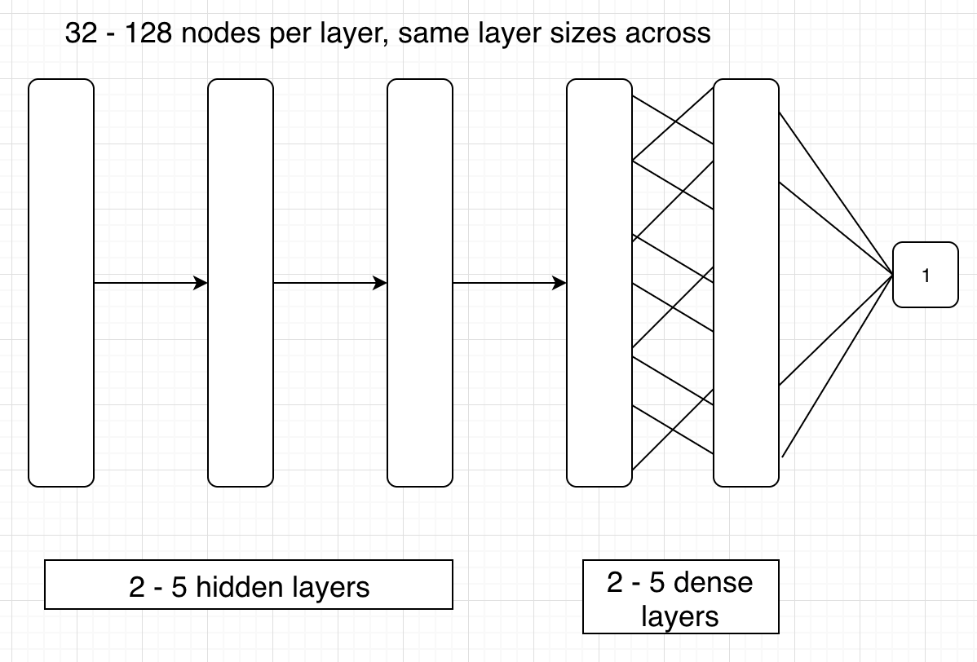


## Image tensorization for traditional machine learning

The images were converted greyscale and then to 2-dimensional tensors (matrices), where each pixel is a feature of the resulting dataset. The images were also resized so that all the images were the same size. The resulting dataset was also normalized with respect to intensity by dividing each intensity by 255.

# Convolutional Neural Networks

We used Keras library to construct the neural network models, all layers are standard layers provided by the Keras library, as well as implementation of the activation functions, loss function, optimizer, and kernels.



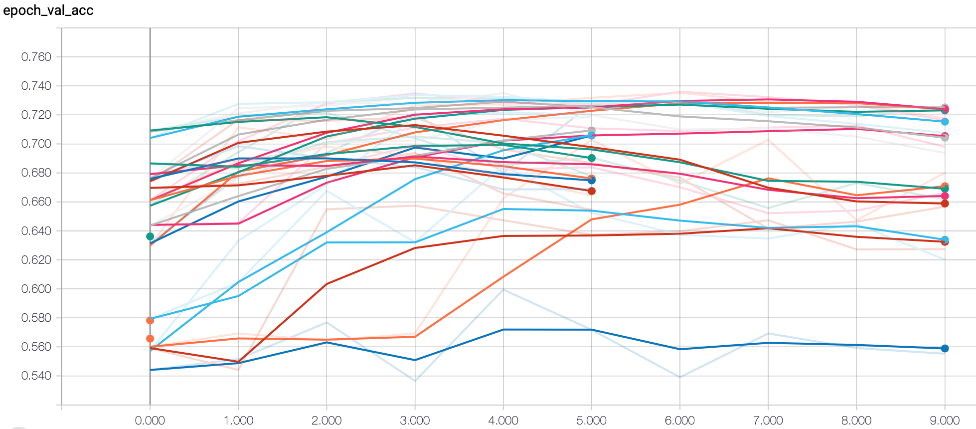
We started with a relatively simple convolutional neural network architecture where layers of identical sizes are used. The networks consist of 2 to 5 hidden layers, followed by 2 to 5 dense layers, followed by an output layer. Except for the output layer, each of the layers either have 32, 64, 96, or 128 nodes per layer, and have the same number of nodes across all layers.

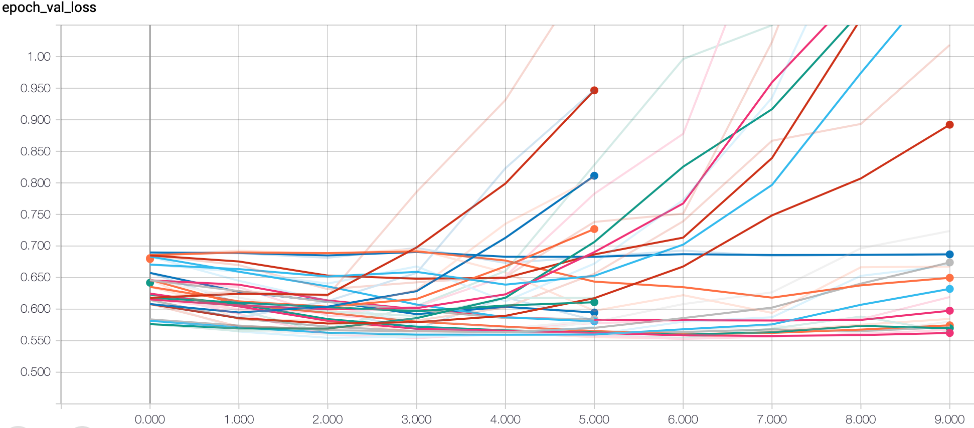
The hidden layers was padded to ensure size does not change, the hidden layers had 3X3 kernel, and using relu activation function. A 2X2 max pooling kernel was used after each hidden layer.

The dense layers also uses relu activation function. On the final output layer, sigmoid activation function was used as the age classification was treated as a binary classification problem.

We used Adam optimizer with a constant default learning rate of 0.001, and validation accuracy as training metrics. Binary cross-entropy was used as loss function.

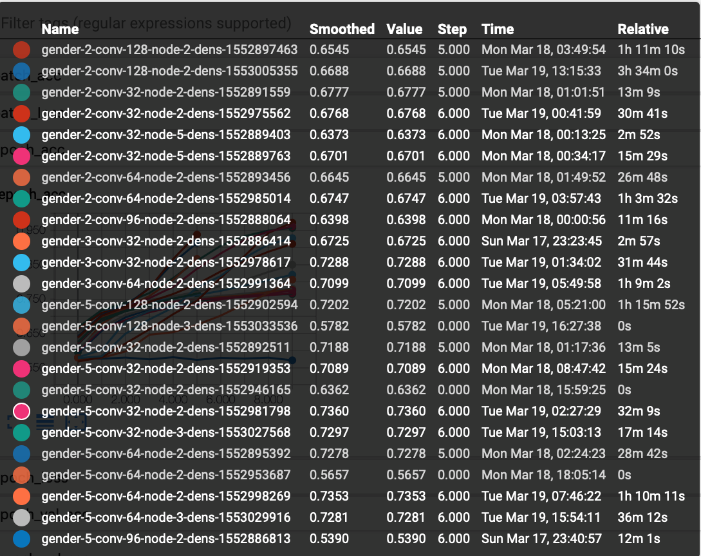
A combination of hyperparameters were tested with this simple architecture, the validation accuracy and loss for each trial are shown in the graphs below.



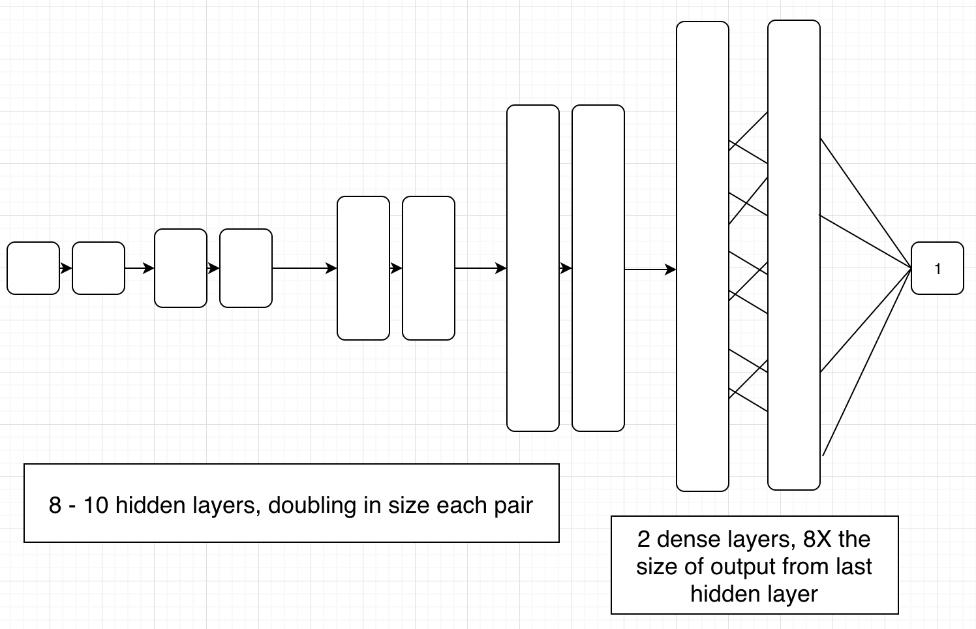


As shown, we first ran 5 epochs, and selected the models with relatively high validation accuracies and low losses, from there we trained these select models again to up to 10 epochs to see if they continue to improve.

Disappointingly, all of these models started to show performance degradation after 6 epochs. The table below shows all of the trials we have ran with this architecture and the validation accuracy at 6th epoch. The best validation accuracy achieved was 73.6% using 5 hidden layers, 2 dense layers, and each layer has 32 nodes.

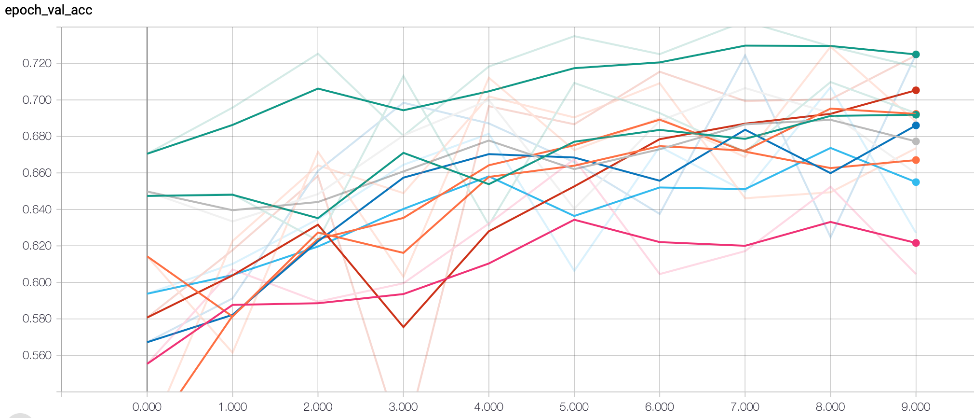


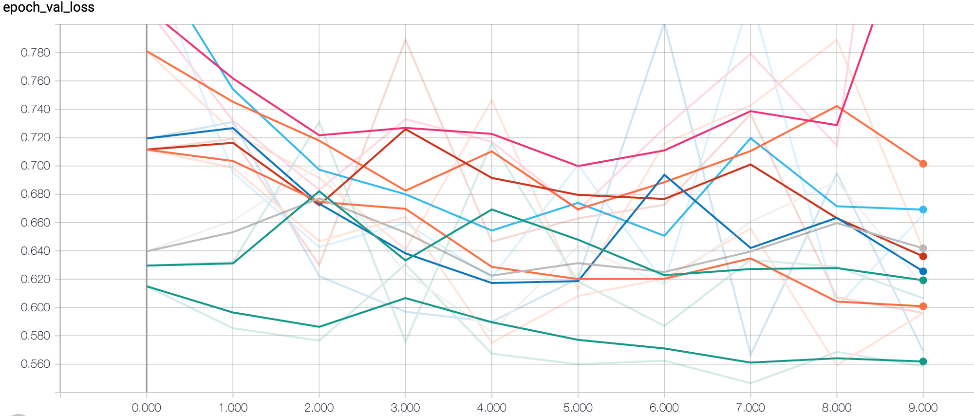
Next, we modified our networks architecture in an attempt to capture more info from the training images, so that the networks do not saturate after only 6 epochs. We came up with an architecture demonstrated in the image below.

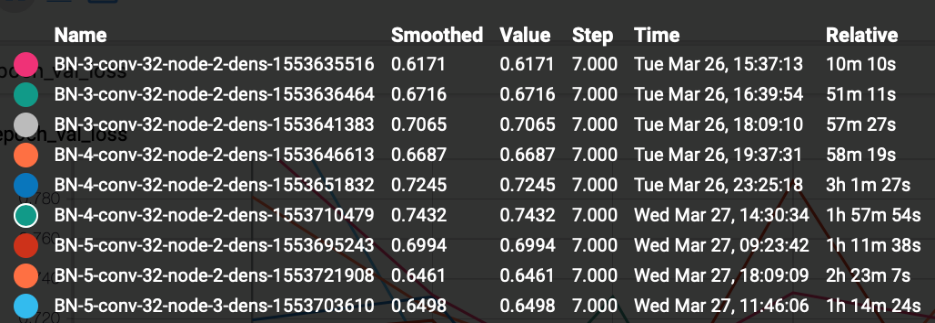


The original paper authors used a pre-trained VGG-16 model, we took some inspiration from the VGG-16 architecture and incorporated into our design. Our architecture has the following distinct properties:

* The VGG-16 starts with small hidden layer, and doubles the layer size every other layer. We took this feature and incorporated into our design, as you can see that we start with two hidden layers with 32 nodes, and then doubling the number of nodes every pair thereafter until we get to the dense layers.
* We do not have as many layers in our design as the VGG-16 architecture. The VGG-16 was developed as a classifier with 1000 classes (1000 node soft-max output layer), we do not believe our network needs quite as much complexity since our goal is binary classification. After some experimentation, we narrowed down to 8-10 hidden layers structure, with 2 dense layers, and a single node sigmoid output layer at the end.
* The dense layers would be considerably larger than the hidden layers. After some testing, we settled on having the dense layer’s size being 4 times the size as the preceding hidden layer, or 8 times the size as the output from the preceding hidden layer.
* In the implementation of VGG-16, we did not see batch normalization and drop out layers, they may have been removed after training. We added batch normalization between each of the layers, which seem to make this architecture much easier to train.
* We also added a drop out layer after each pair of hidden layer, with a setting of 0.25. A single drop out layer at the end of the dense layer was used, with a setting of 0.5. Several trials were run and the results are shown in the graphs below.







The most significant trend was that most of these models have not shown performance peaking after 10 epochs, and we believe they will continue to improve with more training on larger data set. We incrementally carried out each of these trials, testing different hyperparameters along the way. The following highlights how we compared the hyperparameters and how we found our final model.

**Larger dataset improves the overall performance of the networks**

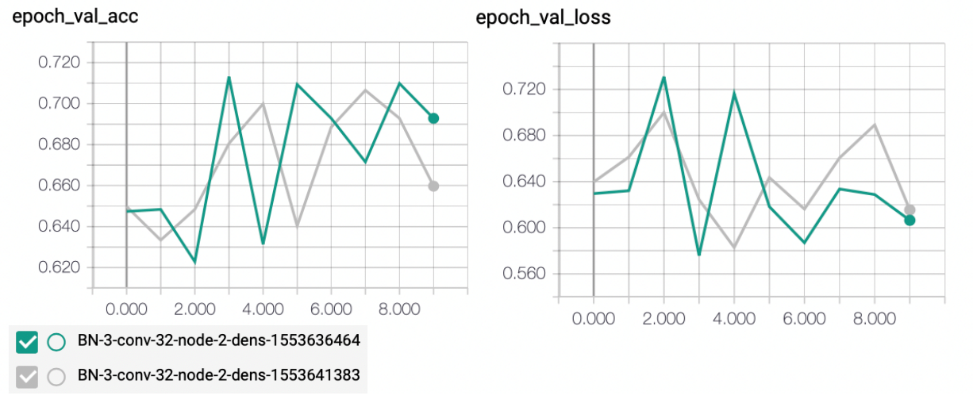
BN-3-conv-32-node-2-dens-1553635516 vs BN-3-conv-32-node-2-dens-1553636464: same hyperparameters were used in both models, but the second model is trained on 5 times the image set. The first model was trained on only 1 percent of the dataset, the latter trained on 5 percent of the dataset.



For the subsequent trials, we will continue to use the larger 5 percent dataset.

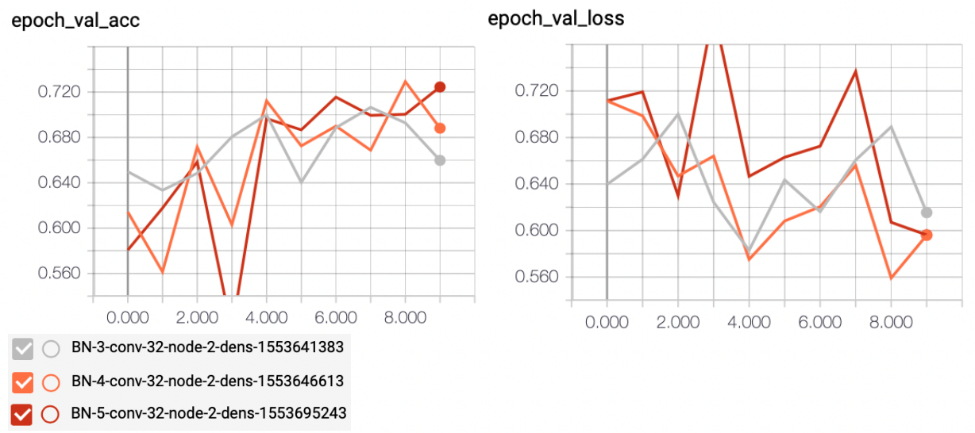
**Dense layer having 8X of size of the output of the preceding hidden layer seems to work**

BN-3-conv-32-node-2-dens-1553641383 vs BN-3-conv-32-node-2-dens-1553636464: The first model follows a “convention” of using a dense layer that is 8 times the size of the output from the preceding hidden layer, the latter is using a dense layer that is 16 times. The latter showed lower accuracy and higher loss, although not significantly.



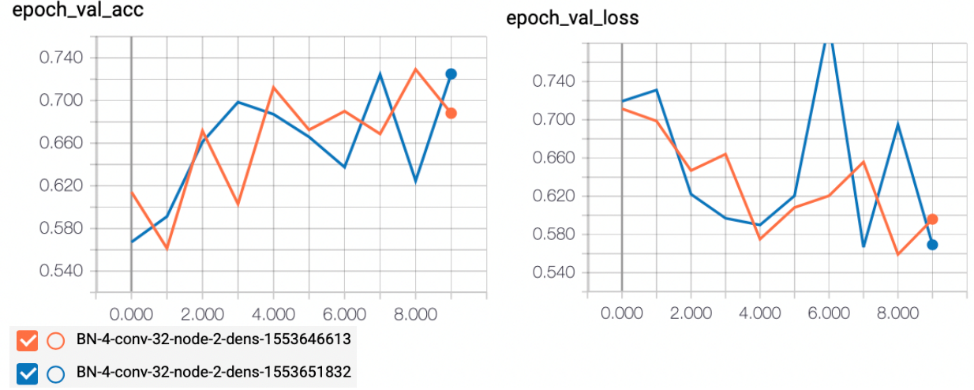
**Increasing the number of hidden layers improves the model**

BN-3-conv-32-node-2-dens-1553641383 vs BN-4-conv-32-node-2-dens-1553646613 vs BN-5-conv-32-node-2-dens-1553695243: They each use 3, 4, and 5 pairs of hidden layers. Because of we are sticking to the convention of having dense layers that are 8 times the sizes of the output of the preceding hidden layer, the latter models also have twice as large dense layers. The resulting validation accuracy and loss shown improvements.



**Increasing size of images for training improves the model but at expensive the training time**

While the size of the image matter depending on the feature and goal of the models, having very large image does not necessarily improve performance of the models. BN-4-conv-32-node-2-dens-1553646613 vs BN-4-conv-32-node-2-dens-1553651832: the latter was trained on images with size of 224 pixels instead of 120 pixels, the overall accuracy and loss improved very slightly but took much longer to train.



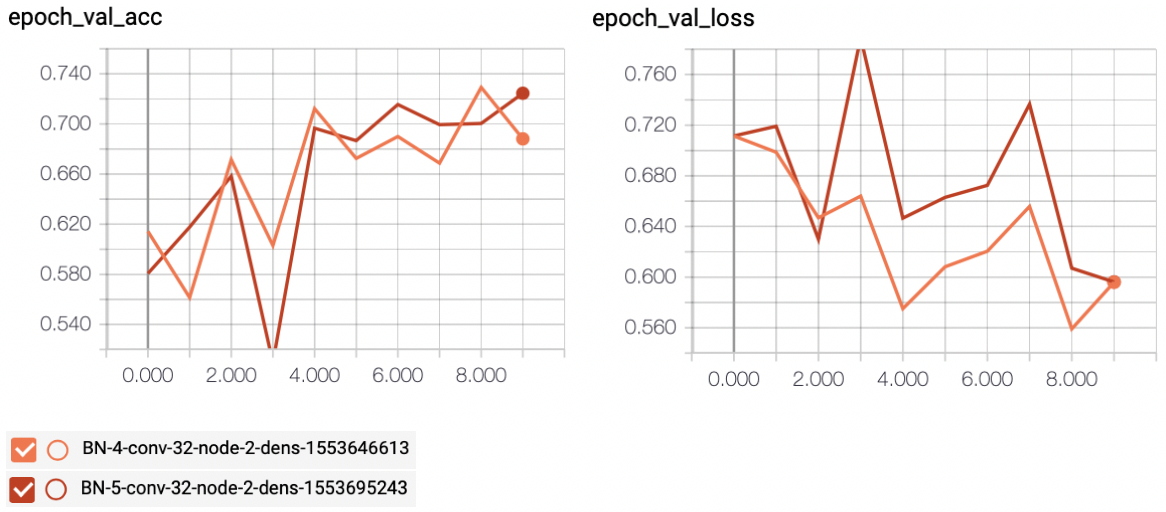
**Increasing the number of dense layers did not improve the model**

BN-5-conv-32-node-2-dens-1553695243 vs BN-5-conv-32-node-3-dens-1553703610: Same settings except the latter model had an extra dense layer, the resulting validation accuracy and loss are slightly worse.



**The effects of training data set size and number of epochs on complexity of the networks explored**

So far, the best trials were BN-5-conv-32-node-2-dens-1553695243 and BN-4-conv-32-node-2-dens-1553646613. The only difference between these two are the number of hidden layers.



They both showed good performance overall and continued trend of improvement after 10 epochs. These two models have very similar design, except that the first one has 4 pairs of hidden layers, with dense layers having size of 1024 neurons; and the latter has 5 pairs of hidden layers, with dense layers having size of 2048 neurons.

We wanted to see if the two networks would start to differentiate with larger dataset. The next two trials were using larger 10 percent data set on these same two networks for comparison.

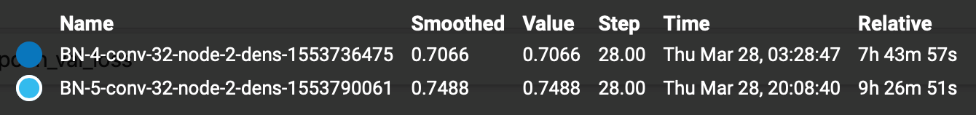


It seems to indicate that the larger and more complex model performed a bit worse with the increased training set size. However we are not convinced. The next trial is a comparison of the same architecture, but training for much longer up to 30 epochs.



As the graphs have shown here, even though in the first 10 epochs, the smaller less complex network performed better, as more repeated training took place, the larger more complex network is able to continue to improve its parameters in prolong epochs while the simpler network started to deteriorate after 12 epochs. We believe the larger network having more parameters take longer to train.

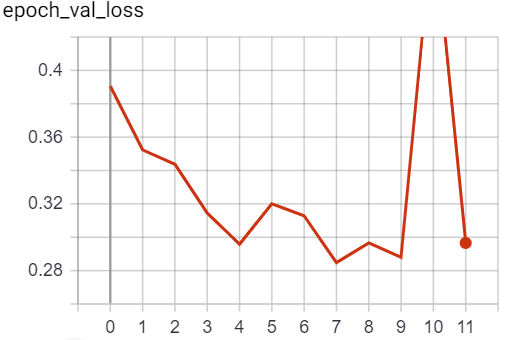
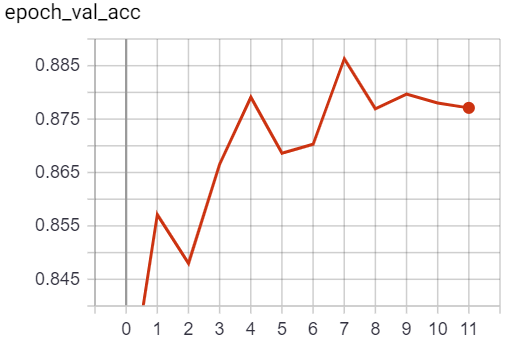
The smaller network with 4 pairs of hidden layers and 1024 neurons in dense layers appear to have saturated at 11th epoch and shown a degradation in validation accuracy and loss thereafter. The larger network with 5 pairs of hidden layers and 2048 neurons in dense layers appear to have continued to improve gradually all through 30 epochs. Their validation accuracy is shown below.

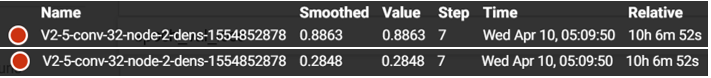


We believe the latter design is appropriate for training on larger dataset, which is the design we used for subsequent training to produce a final model. At this point, we implemented some transfer training features from the Keras library, and we are able to save weights from the best epoch and resume training with different hyperparameters. We eventually slowed down learning rate to 0.0001 (from 0.001), increased the hidden layer dropout to 0.5 (from 0.25), increased the dense layer dropout to 0.7 (from 0.5), and were able to sustain further training to gain another 1% improvement in validation accuracy. In the end, we decided on the following parameters to train our model:

* 10 hidden layers in total (or 5 pairs)
* Hidden layer start with 32 nodes in the first two layers, doubling in sizes every second hidden layer, up to 512 in the last two hidden layers.
* 2 densely connected layers in total, having 2048 nodes each.
* 1 output layer with 1 node.
* Batch normalization between each of the hidden layer.
* All of the hidden layers use 3X3 convolutional kernel, with padding that ensure the convolution does not change size.
* The first hidden layer uses 3X3 max pooling, with all subsequent hidden layers having 2X2 max pooling.
* Relu activation function is used in all of the hidden layers and dense layers. Sigmoid activation is used in the output layer.
* We used drop out of 0.35 after each pair of the hidden layer. We used a drop out of 0.6 at the end of 2nd dense layer.
* We used Adam optimizer with constant learning rate of 0.0008, binary cross-entropy loss function, and accuracy as metrics.

The validation accuracy and loss for the training is shown in the graphs below.





We took the weights from the 7th epoch where the model had highest validation accuracy of 0.886, with the lowest validation loss of 0.285, and compiled the final model.

# Traditional machine learning classifiers

The following classifiers were used for this project:

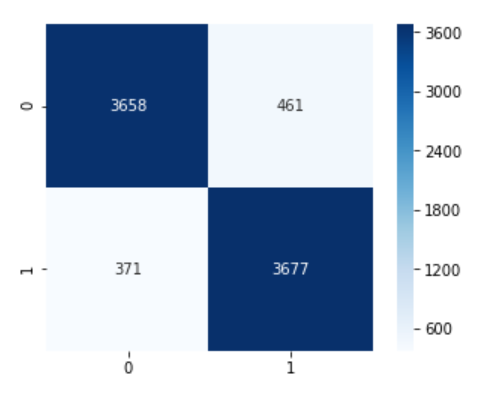
* Linear Discriminant Analysis
* Multinomial Logistic regression
* Gaussian Naïve Bayes classifier
* K-Nearest Neighbour
* Support Vector Machine

The classifiers are a mixture of parametric and non-parametric types.

# Evaluation

We now use the test data set, which is not part of the training set or validation set, for the purpose of evaluation.

A total of 8,167 images were used for testing, the confusion matrix is illustrated in the graph below.



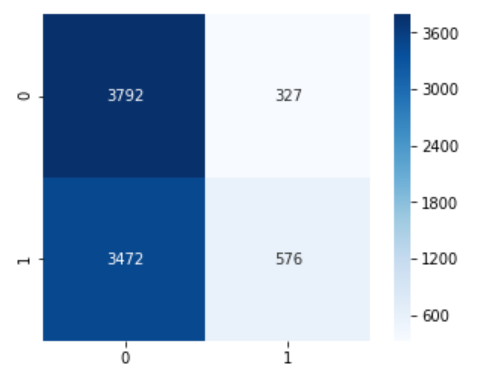
The evaluation metrics are as the follow:

* Specificity: 0.89
* Recall: 0.91
* Precision (Male): 0.89
* Precision (Female): 0.91
* Accuracy: 0.90

# Final CNN Model Trained with Smaller Training Sets

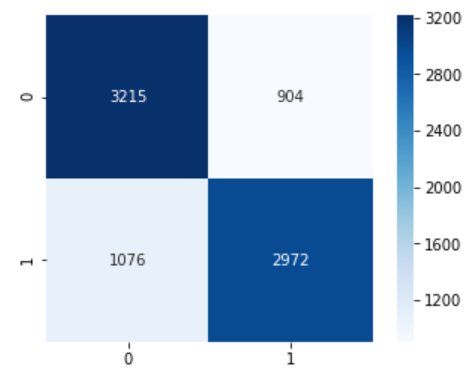
To demonstrate the effect of training sample set on the CNN model, we trained multiple instances of the model from scratch using different number of images.

**CNN model trained with ~1500 images (1% of filtered data set):**



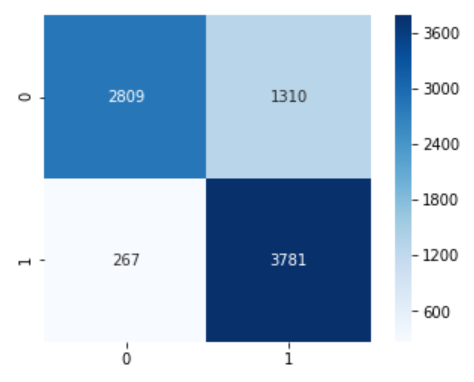
* Specificity: 0.92
* Recall: 0.14
* Precision (Male): 0.64
* Precision (Female): 0.52
* Accuracy: 0.53

**CNN model trained with ~7800 images (5% of filtered data set):**



* Specificity: 0.78
* Recall: 0.73
* Precision (Male): 0.77
* Precision (Female): 0.75
* Accuracy: 0.76

**CNN model trained with ~15000 images (10% of filtered data set):**



* Specificity: 0.68
* Recall: 0.93
* Precision (Male): 0.74
* Precision (Female): 0.91
* Accuracy: 0.81

**Comparison of Results**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Training Set Size | Model | Specificity | Recall | Precision (Male) | Precision (Female) | Overall  Accuracy |
| 1,500 | Convolutional Network | 0.92 | 0.14 | 0.64 | 0.52 | 0.53 |
| Support Vector Machine |  |  |  |  |  |
| Logistic Regression |  |  |  |  |  |
| K-Nearest Neighbor |  |  |  |  |  |
| Linear Discriminant Analysis |  |  |  |  |  |
| Gaussian Naïve Bayes |  |  |  |  |  |
| Random Forest |  |  |  |  |  |
| 7,800 | Convolutional Network | 0.78 | 0.73 | 0.77 | 0.75 | 0.76 |
| Support Vector Machine |  |  |  |  |  |
| Logistic Regression |  |  |  |  |  |
| K-Nearest Neighbor |  |  |  |  |  |
| Linear Discriminant Analysis |  |  |  |  |  |
| Gaussian Naïve Bayes |  |  |  |  |  |
| Random Forest |  |  |  |  |  |
| 15,000 | Convolutional Network | 0.68 | 0.93 | 0.74 | 0.91 | 0.81 |
| Support Vector Machine |  |  |  |  |  |
| Logistic Regression |  |  |  |  |  |
| K-Nearest Neighbor |  |  |  |  |  |
| Linear Discriminant Analysis |  |  |  |  |  |
| Gaussian Naïve Bayes |  |  |  |  |  |
| Random Forest |  |  |  |  |  |
| 150,000 | Convolutional Network | 0.89 | 0.91 | 0.89 | 0.91 | 0.90 |