A Deep Learning Dive into Binary and Multiclass Classification of Images

Bruno Calogero
University College London
Gower Street, London, UK
github.com/brunocalogero/AMLSassignment

Abstract

We explore various Binary and Multi-Class classification problems and determine various methodologies to prepare data for Support Vector Machine, Logistic Regression and Custom literature based CNNs. We also report various experimental metrics regarding accuracies and learning curves, fundamental to the comprehension of overfitting and how to avoid it.

1. Introduction

Training and testing a Deep Neural Network, whether it be a Convolutional Neural Network (CNN) or any other Neural Network architecture, can be a time consuming and expensive task. What we really try to ask ourselves here is whether using complex networks with a large quantity of parameters is really worth it when it comes to varying complexities of datasets, especially when not having access to more powerful cloud instances and other cloud based training approaches (gcloud, AWS EC3..etc).

We are given a dataset with human and cartoon-based frontal-faces[link]. Inside of it we find various outliers and other noisy, distorted images. On top of trying to solve the aforementioned problem, we have the task to correctly classify age (young/old), emotion (smiling/not smiling), glasses(glasses/no glasses), human (human/none human) and hair-colour(bold, blonde, ginger, brown, black, grey).

To be able to express our observations we will need to look at various binary and multi-class classification cases, all of which will be evaluated using three main 'algorithms'/classification models: Support Vector Machines (hinge-loss based), Logistic Regression (more commonly known as Softmax classification due to the entropy based activation function), and, finally, Convolutional Neural Networks (CNNs). Hence, we progressively dive into classification of images using varying degrees of complexity; from simple linear models to more complex higher order systems.

Our metrics will be based on training/validation and inference accuracies/loss, confusion matrix scores, learning times, overfitting (thanks to learning curves), therefore hyper-parameterisation and other methods proper to each 'algorithm' to decrease the overfitting. The goal being to improve our models and report more realistic inference scores.

To end, we will also explore some state of the art pre-trained models from which we can use the weights to perfect our classification/inference scores. Main advantages and disadvantages will also be evaluated as well as a progressive understanding of why certain parameters/hyper-parameters will be used.

For all the the three aforementioned models, we will be separating our dataset into training, validation, and test splits that make sense for the given problem and given dataset. We will be using K-fold cross-validation over 8 and 10 folds. Hyper-parameterisation will first be based on logic, followed by efficient grid-searching for tuning purposes.

1.1. The Dataset

It is important to note that data itself plays one of the most important parts in classification. If the data is noisy, occluded, tilted or even too large in scale, our Neural Networks as well as our more basic classifiers like in the case of multi-class Linear Regression, will have a much harder time learning features and optimizing. Indeed, most pre-trained models in Literature clearly show that it is a wise idea to abide by a certain distribution (data transforms). This can have the benefit of decreasing overfitting by obtaining better training and validation scores, greatly reducing the amount of features (hence parameters) of our raw pixel RGB data, and still give good if not better scores. By reducing the amount of parameters that represent our data we are allowing for faster and more efficient training, this can be seen in our experimental results with the different transformations of the data.

The given dataset is mainly composed of frontal face images. The images are all of the same size (256x256x3) with all Red, Green and Blue channels. This might seem like a small image but in terms of computation this is actually quite large, indeed, this means we have a total of 196608 features per example (per image). So we will definitely have to be careful when delving with our future Neural Networks or Convolutional Neural Networks, some form of re-scaling might have to be taken into consideration if we are to keep the raw pixel image representation.

From the dataset it is clear that we have a few interesting outliers: mainly grey/orange, occasionally mountainous scenery and blue/grey sky. Some images with faces can contain various forms of noise and occlusion such as watermarks, pixelation, noisy backgrounds, distortion, destroyed nasal features and slight tilting. We therefore need to find a methodology to remove these outliers.

1.1.1 Outlier Detection

Since our dataset is mainly composed of frontal faces, it comes to mind to use an efficient face detection method. Indeed, the latter are largely present in the form of pre-trained models or feature extractors in libraries such as 'OpenCV', Davis King's 'dlib' and the more recent 'cvlib'. This is also a great way to start getting familiar with basics behind pre-trained CNNs, feature extraction and SVMs without getting deep into the algorithmic intricacies, but instead understanding how pre-trained models work and how we can leverage them to prepare our data for our own training and testing. The different models used are particularly good with the task at hand: detect faces.

Three different face detectors have been utilized and tested: dlib based Histogram of Gradients with SVM (HOG/SVM)[3], dlib based pre-trained CNN face-detector, OpenCV's Haar-cascade based on Viola and Jones[2][10], a feature extractor based on 'Haar' Faces. The Haar feature-based cascade is used here to benchmark our two other methodologies.

We take advantage of the fact that our dataset is labeled to get a few metrics as can be seen in Table 1. Indeed, outliers have all -1 as labels.

Method	(HOG / SVM)	(HOG / SVM) w/ equal.	CNN	CNN b2b (HOG / SVM)
Outliers Detected	578	609	543	454
false positives	143	174	108	19
Prediction Time	3:58min	3min	55min	55:40min

Table 1. Outlier Detection Methods Used and performance

A Python playbook is provided with an in-depth analysis of this pre-processing on this paper's github. It is clear from the table above that optimal results have been achieved when combining both our CNN (more sensible to face pose variations and position compared to HOG)[12] and HOG/SVM classifiers back to back (b2b), it is therefore chosen as our preferred method to create the new dataset (without the outliers). Same results obtained with optimal four cascaded Haar feature classifiers. The False-positives from Table 1 are images that have been classified as outliers when they shouldn't have. In each case all the main outliers are detected and eliminated, it therefore becomes a question of minimizing the amount of false positive to avoid reducing the size of our dataset and remove useful images.

Histogram equalization of images has also been explored to see if face detection would be improved. With the equalizer we get 609 outliers, so more outliers are detected but these are just cartoons (more false-positives). Hence Histogram equalization does help the classifier, since the latter is trained to detect human based facial landmarks (not cartoon's) so equalization would be useful to use if our dataset was solely based on real human faces, it would classify better and eliminate more noisy none-wanted images. The idea behind using histogram equalization is to avoid intense lighting being an issue in images. For example, a very lightened up face might prove to lose most of its contrast and therefore facial features aren't as apparent, equalization in this case will increase the contrast and reveal useful facial information to the classifier or feature extractor.

From observing the data in each case presented in the table, it is apparent that most, if not all of our false positives are cartoons. We observe that the cartoons that were treated as outliers are mainly the ones with occlusive elements on their face such as glasses or beard, the latter hide the key features that our classifiers are looking for, in our case this might be the jawline. We also notice a prevalent number of dark skin cartoons (mainly black skin). These darker colours are probably not representations or features familiar to our HOG and CNN, because of the way they were initially trained to extract features. This make sense, the CNN weights came from training based off of real human faces and the dataset was mainly based off white Caucasians without necessarily large or any beards at all (since people with beards were probably harder to find and add to the dataset). Hence our results make sense. It seems like our HOG+SVM was slightly less sensible to the beard problem, but still sensible to the skin colour problem. In some sense the CNN is actually doing a great job, because it basically segregated cartoon images with considerable beards which it was never trained to recognize (it did not learn those particular representations when talking in terms of weights), so the weights turned out to not learn that specific feature. We plot the 19 outliers from the CNN b2b HOG/SVM and clearly observe what was mentioned above:

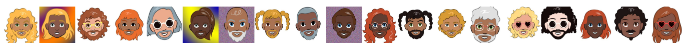


Figure 1. Predominantly Dark-skin, bearded cartoons predicted as outliers

For each classification problem we use three different learning algorithms. For each learning algorithm we will be looking at three different ways of pre-processing our data before it is trained by our algorithms. The idea here is providing our algorithms with the most optimal representation of our data for the given classification problem.

1.1.2 Image Pre-Processing for Model Input: Data Standardization, grayscaling, rescaling

In this first case we want to transform the data such that we zero center it and normalize it. Indeed, it is clear from most literature that many Learning Algorithms, especially the ones used in this paper generally assume features to have a Gaussian form, in other words $mean = 0$, $std = 1$. As seen in [9], this is a 'straightforward way to remove numerical differences originating from faces in different places of the image'. To put it simply if we had a matrix representing all our examples flattened out per row (assuming vector operations, easily done with numpy):

$$I_{new} = \frac{I - I_{mean}}{I_{std}}$$

This is the data standardization step, however we also consider grayscaling and rescaling. Grayscaling is directly dependant on the classification problem as we are fundamentally eliminating our three colour channels. This allows us to reduce the amount of parameters thus dimensionality and therefore parameters or complexity of the problem. since we now only have $256 * 256 = 65536$ parameters. This is not applied for my CNN implementations as I wanted to leverage the features being learnt between the different channels in the 2D convolution, I also ended omitting this from the classification problems since I just ended up rescaling the images to 128x128 (16384 param.), therefore reducing the amount of parameters to be able to run the code in a relatively faster fashion, 256x256 was not optimal as learning would generally require too much computation that was not available.

1.1.3 Image Pre-Processing for Model Input: facial landmarks with augmented features

Here we leverage the pre-trained 68 facial-landmark feature extractor given to us by dlib (as described in the outlier section). The latter basically provides us with 68 landmarks (x-y coordinates) of key features of the face: jawline, nose, eyes, eyebrows. However, we will be augmenting the features a little (we don't only want the coordinate information) instead of destructive normalization[9]. Indeed, to be able to recognize certain elements in a picture such as emotion, we can 'calculate the position of all points relative to each other' (center of gravity) and 'get the position of all points relative to this central point'. To avoid tilted faces confusing our classifiers we also introduce a new feature that is added on top of the two previous: We basically "rotate the vector array so that tilted faces become similar to non-tilted faces with the same expression"[9] and calculate "the angle the nose makes relative to the image" therefore adding a angular offset measure to our features. Thus, with much less features we can obtain similar results than with having all of the heavy RGB pixel data, of course this method cannot be used for our multi-class hair colour recognition problem, however it will allow us to obtain much faster results.

1.1.4 Image Pre-Processing for Model Input: 250 first PCA components

By definition, PCA represents the data in a new coordinate system where basis vectors follow modes of greatest variance in the data. The price to pay however is the initial conversion of the images to their PCA components, which is not very computationally efficient. Here we select the first 250 components which suffice in terms of the main variance presented by the data (sufficient amount of features) and create a new dataset based on the PCA components, to avoid having to recompute the latter every time. Similarly to the landmarks method, this will allow us to train our algorithms much faster because of the lower amount of parameters which still describe the data well, however will be prone to overfitting just like in every case presented.

2. Proposed Algorithms

Support Vector Machines and binary or multi-class Logistic Regression allow us to treat feature vectors as points in higher dimensional space based on the optimization of a set of weights and particular loss functions in each method.

2.0.1 Support Vector Machine (SVM) And Logistic Regression (softmax)

We include both SVM and Softmax in the same section because of their similarities when it comes to classifying images. Most of the observations made are based on Stanford's CS231n [1]. A graph provided by the class will greatly help in building the intuition for the choice of these models and can be found in the supplementary material Section. By simply looking at this figure, it is clear that these methods are ideal for images as they allow us to take a feature vector x_i (with label y_i) - which could be a flattened

RGB image in the form of a single array and apply a set of Weights W and a bias. The main difference between the two comes from the loss function, in other words, a measure of how good the classifier is performing.

In the case of the SVM, for feature vector x_i representing an image, we have the following loss (Hinge-loss):

$$L = \frac{1}{N}\sum_i [\sum_{j\neq y_i} max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

Where w_{y_i} is the row of W corresponding to the weights of the correct, ground truth class and j all the other rows. The last term is L2 Regularization, optional but appealing for max margin property in SVMs.

What we are trying to do here is control Δ and λ to control the trade-off between data loss and regularization loss from the Optimized weights matrix W. What we are saying when setting Δ is to want the correct class to have a score higher than the incorrect classes by a margin Δ. Hence, setting a higher delta in terms of our feature data means that we are more lenient to the difference between the ground truth and the other class scores. Setting this too high might therefore leave too much room for error. Setting this too low might not leave enough room for error or noise to be considered in order to make a correct prediction of say a noisy image, hence also might lead to miss-classification. Of course, optimization here directly influences W and therefore technically has more of a say than Δ, however decreasing λ can control this. With binary classification the idea stays the same but generalizes to a single hyper-parameter C which is inversely proportional to λ and will be seen in our implementation with Sklearn, the simpler one-vs-all SVM is going to be used in practice [7]

In the case of Softmax we have a different decision function and therefore loss, known as cross-entropy loss. Here, the function mapping between our weights W and our feature vector x_i stays unchanged: $f(x_i, W) = Wx_i + bias$ but we now interpret the scores as the un-normalized log probabilities for each class:

$$L = \frac{1}{N}\sum_i [-f(x_i, W)_{y_i} + \log(\sum_j \exp f(x_i, W)_j)] + \lambda R(W)$$

Where R(w) is the L2 Regularization term. It is important to note that exponents are numerically unstable and sometimes too large for computation, hence the use of different normalization tricks. From this, we obtain a set of probabilities for each classes, which gives us a little more intuition as to why an image was labeled as predicted.

In terms of images or feature vectors and also with regards to classification, it is clear that the SVM is looking for a more local objective, it is pretty indifferent to the details of the individual scores, as long as the margins are satisfied it is happy, it does not dwelve into micromanaging the scores beyond the constraint. On the other hand, as mentioned in the Stanford notes, the softmax "unlike the SVM which computes uncalibrated and not easy to interpret scores for all classes, the Softmax classifier allows us to compute probabilities for all labels". Thus more meaningful data.

In general, literature shows that both learning algorithms have similar performance, this will be backed by our experimental results.

Optimization in both cases is mainly based on Stochastic Gradient Descent, CNNs will have us introduce a wide range of optimizers since they play one of the most fundamental roles in back-propagation and weight selection in Neural Networks.

2.0.2 Convolutional Neural Networks (CNNs)

The architecture behind CNNs has shown to be extremely practical for image pixel based feature extraction. The basic idea is that we convolve a set of 'kernels' (2D convolutional layers, that are individually initialized and optimized during back-propagation) to input images or feature vectors describing our problem. This convolution results in a higher dimensional space representation of our individual image. Therefore we can learn much more information of a given image and its particular elements and features. However this can lead to an over-representation of certain unwanted features or certain weights related to certain features that shouldn't be.

It is therefore common to see basic CNN architectures in literature [8], that use additional types of layers such as pooling layers (max or average) , Batch Normalization Layers, dropout layers and finally flattening layers and dense layers followed by an activation function for multi-class or binary prediction. Each layer has its own particular properties. Dropout layers for example, directly complement L1 and L2 regularization and allow for a reduced amount of overfitting by eliminating 'Neural connections' that are not of interest, in general literature shows that it works best after each Convolution. Another important aspect of Convolutional and Dense layers is the fact that we use activation functions at their output that basically weight out the importance of activating a particular link in the network.

Different Activation functions present different advantages, the main ones we will explore in our implementation are the Rectified Linear Unit (Relu), Leaky Relu [11] and softmax activations. The first two are generally used as an activation after fully connected dense layers or Convolutional layers (in the case of RGB pixel data for example). They allow for elementwise activations such as max(0,x) thresholding at zero. This leaves the size of the input volume unchanged unlike the convolutional layer. The advantage of Relu is the great acceleration of convergence in terms of gradient descent and are also very simple to implement. Negatives however include the fact that Relu units can 'die' during training. In essence, a large gradient through a Relu neuron can cause the weights to update in such a way that the neuron will never activate on any datapoint again. This can be caused by a high learning rate for example (when talking about optimization). The Leaky Relu fixes this by using a concept of negative slopes [11].

Pooling is also important in the CNN architecture that has been implemented. We are using max pooling to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, hence it also controls overfitting after our convolutional layers. In general dropout is applied after the latter. "Pooling operates independently on every depth slice of the input and resizes it spatially using the max operator"[1], in general if a stride of 2 is applied we down-sample by 2 along the width and the depth thus reducing the amount of parameters in the system. Doing this discards 75 percent of the activations.

Finally we consider a few different optimizers for our loss, such as the Adam or RMSProp optimizers. These are also known as 'per parameter adaptive learning rate methods' and are extensively described in literature. A general rule of thumb involves using Adam [6].

3. Implementation

The libraries that have been used include: 1) 'OpenCv' and 'pandas' for data processing such as grayscaling, rescaling and reading csv files. 2) 'sklearn' for SVM, Logistic regression, shuffling, train/validation/test splitting, grid-search, learning curves, accuracy scores, Confusion Matrices, PCA calculation. 3) 'numpy' for matrix manipulations and reshaping. 4) 'tensorflow 1.4' and 'tensorflow-gpu 1.4' to implement a raw version of the CNN to run on GPUs with Cuda 8 and CudNN. 5) 'keras' (tensorflow backend) for easier to read CNN implementation and VGG16 net with ImageNet Weights basic implementation.

For each single Binary and Multiclass problem: SVM, Logistic Regression and a basic CNN have been implemented.

For the case of the SVM in the binary problems we feed in three different types of data: 1) normal RGB 128x128x3 scaled data, 2) The augmented facial Landmark feature based data, 3) 250 principal PCA components data (based off the original 256x256x3 pixel data). In the multiclass problem for hair colour, SVM with two types of data are used: 1) normal RGB 128x128x2 scaled data, 2) 250 principal PCA components data

For the case of the Linear Regression in the binary Problems we feed in two different types of data: 1) The augmented facial Landmark feature based data, 2) 250 principal PCA components data. The normal data wasn't used because of time constraints and the more interesting results provided by landmarks and PCA components. In the multiclass problem we only feed the 250 PCA components data.

Finally for the CNN (in every case), we only provide the normal RGB 128x128x3 scaled data, reason being that CNNs benefit from the RGB data and the dimensionality of the latter in coming up with higher dimensional features and therefore being able to detect certain features that landmarks wouldn't. An example of this would be with the eyeglasses. Perhaps the eyeglasses provide some form of shading that is characteristic of the latter, this feature wouldn't be able to be detected with landmarks or would probably not be included in the PCA principle components.

Hence, the problem becomes about not only understanding how well a learning algorithm works but also how it reacts to different types of data that is fed into it and understanding what works best in each case.

Because of the very large amount of classifiers that have been tested, 30 in total, we will not be able to provide all of the learning curves in the report, however we will be focusing on the main observations made per problem to show the main understanding of challenges. All the learning curves (training/validation accuracy and validation loss) are all provided on the github repository for the project. A 10 fold cross-validation has been applied in every case. A train/validation/test split of 60/20/20 (percentages) has been applied respectively. This has been chosen due the relatively large scale of the data. It is important to note however that the test split could have been reduced slightly in favour of a larger training set, especially if a separate inference test set is provided to us after the original dataset. 20 percent validation is used to make sure that our model isn't overfitting and trying to memorize the training data, which is reasonable given the dataset size of about 4000 or so examples after outlier elimination.

In the case of SVMs for the binary and multiclass problems it has been shown through grid-search and reasoning similar to that made in the algorithms section that a basic 'Linear' kernel worked best with a C parameter of 0.01 (somewhat measuring error equivalent to Δ) for the cases of normal, landmarks and PCA based data. This worked better than say an 'RBF' kernel because of the frontal face origin of the data and most likely the cartoonish solid colour data. The stopping criterion is controlled by the tolerance and set to 10^{-3}

In the case of the Logistic Regression for the binary problems, for the landmark data, the optimal solver seems to be the Limited-memory BroydenFletcherGoldfarbShanno (lbfgs) optimization, and a C (Inverse of regularization strength in this case) 0.1 for human problem, 1 for the glasses problem, 0.1 for the emotion problem and 0.01

for the age problem. Therefore the glass problem required less regularization loss compared to the other. In each case a stopping criterion of 15000 iterations had to be used for correct convergence. In the case of PCA, a C value of 0.01 with 5000 max iterations and an lbfgs solver worked best overall for all problems.

Our CNN architecture for both Multiclass and Binary problems is very similar, a full picture is given in section 7.2 of the Supplementary material. Fundamentally it is a cascade of the following form:

2x[CONV2D $\rightarrow$ $BATCHNORM$ $\rightarrow$ $2DMAXPOOL$ $\rightarrow$ $DROPOUT(0.2)$] $\rightarrow$ $FLATTEN() \rightarrow DENSE(size2) \rightarrow SOFTMAX$

The only difference with the multiclass resides in the size of the Dense layer that changes to 6, in other words the number of labels for the multiclass problem.

With this architecture we have a total of 38482 parameters, 32 kernels in the first conv layer and 16 in the second. This is because we want to try and reduce the amount of new features learnt in the deeper layer. All the convolutional layers have a leaky Relu as activation functions. Furthermore, we are training with 10 epochs, with batch sizes of 64 and 0.001 as an optimal learning rate for our Adam optimizer with categorical cross entropy (labels need one hot encoding).

We will now focus on particular learning curves moreso than all of them. We want to be able to recognize cases of overfitting and suggest solutions to the latter as well as recognizing good learning curves.

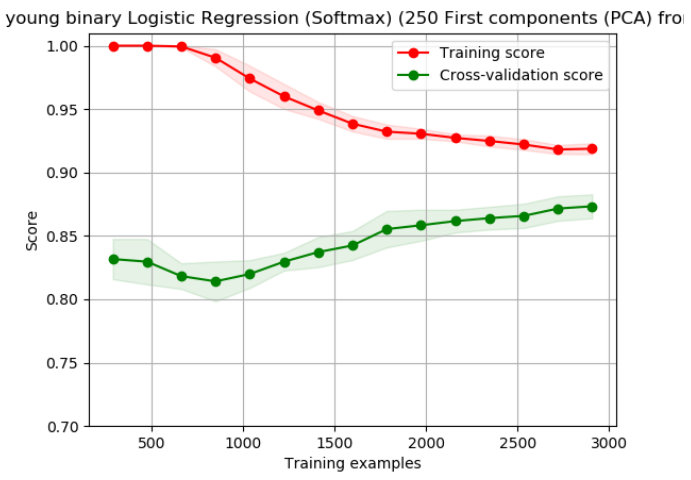


Figure 2. PCA based Logistic Regression for binary age problem Training and Validation accuracies

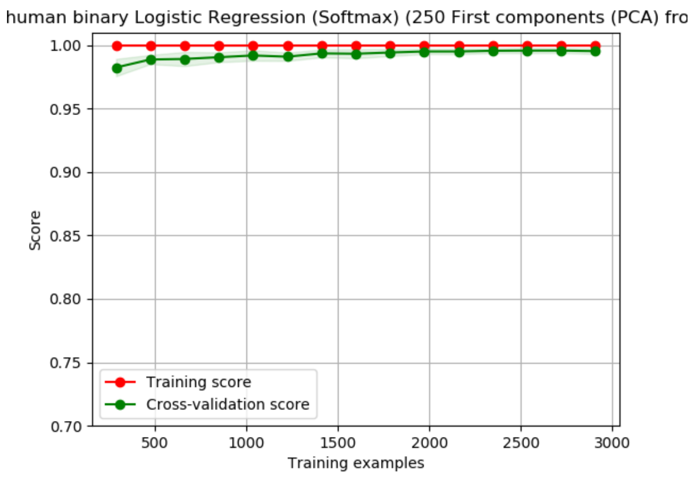


Figure 3. PCA based Logistic Regression for binary human problem Training and Validation accuracies

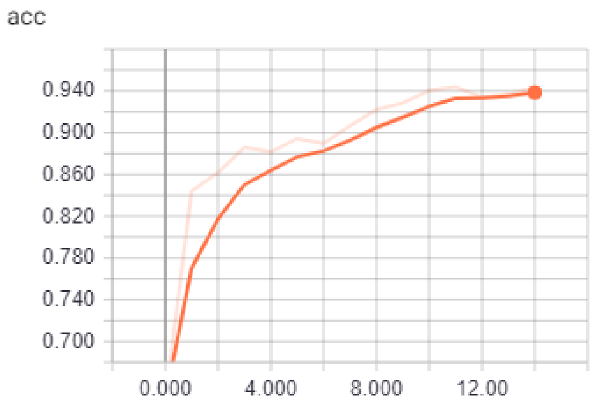


Figure 4. CNN based multiclass hair problem Training accuracy

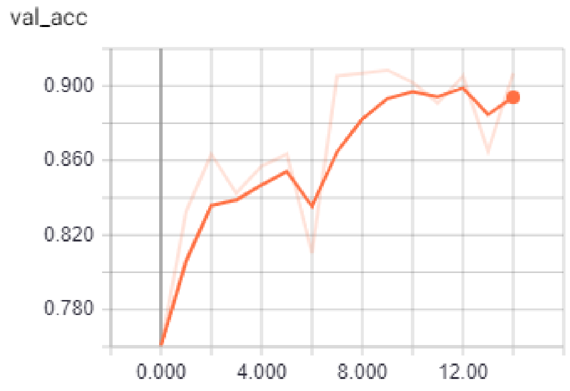


Figure 5. CNN based multiclass hair problem validation accuracy

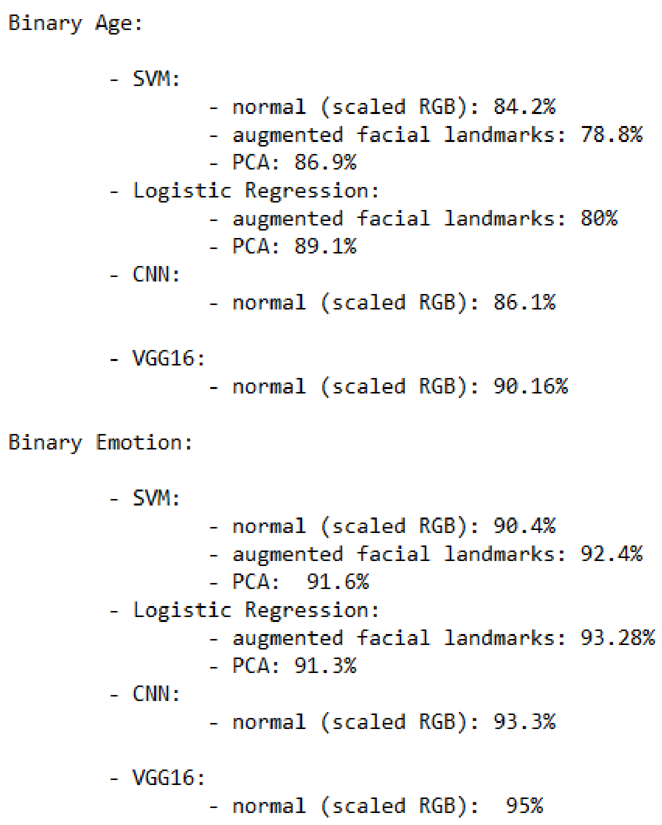
From Figure 2 We can tell that our training score is gradually reducing as we increase the number of training examples. This is actually somewhat of a good sign and it is also observed in the other binary cases for the PCA methodology. Indeed, as we increase the dataset size, it should be harder to fit the training data but this also means that the results generalize better for our test validation data. Hence explaining the shape for the curve. Similar curves are also observed for the SVM case. In Figure 3 however we do have some signs of overfitting for the binary human problem. The training accuracy remains at 100. This might be a sign of us having to increase the amount of parameters in our system or simply increase the training dataset.

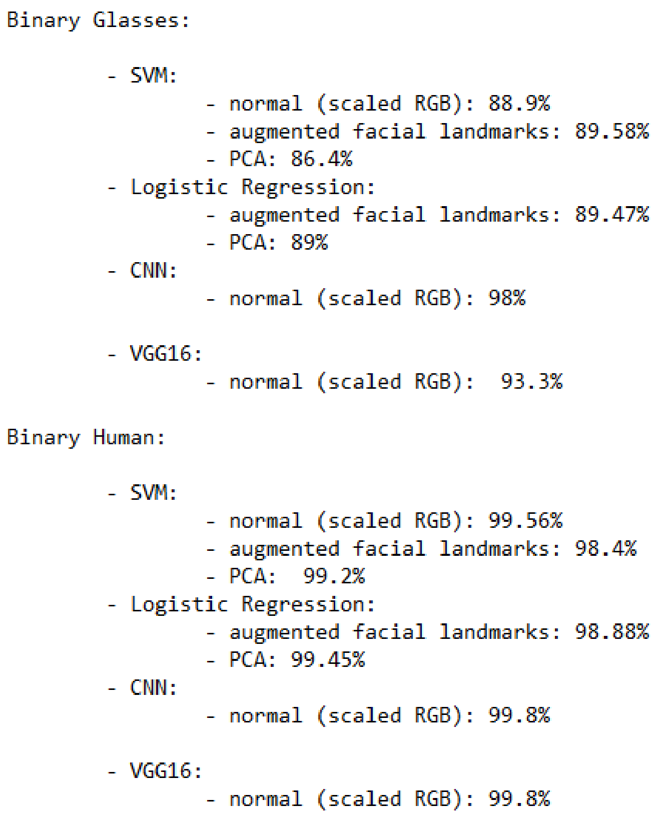
In the Case of figures 4 and 5 we observe interesting overfitting of our validation accuracy over the 15 epochs of training. This can be reduced with perhaps more training data, increased dropout and possibly a third Convolutional layer with Dropout.

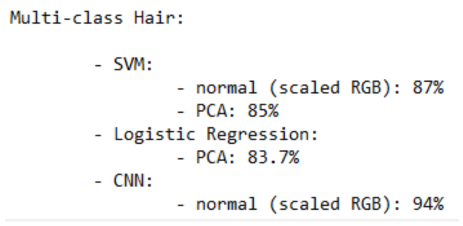
All of the Curves can be found on the github under each problem folder under result-logs as well as some in the supplementary material section. It also includes all the grid search, cross validation and confusion matrix results.

4. Experimental Results

Here we present all of our inference results in the form of tables. From the observations of all of the learning curves it seems that Logistic Regression with PCA presents the least amount of overfitting and therefore the most realistic accuracy scores. Some of the other cases like our SVMs in the case of age for example presents overfitting.


```
Binary Age:

        - SVM:
                - normal (scaled RGB): 84.2%
                - augmented facial landmarks: 78.8%
                - PCA: 86.9%
        - Logistic Regression:
                - augmented facial landmarks: 80%
                - PCA: 89.1%
        - CNN:
                - normal (scaled RGB): 86.1%

        - VGG16:
                - normal (scaled RGB): 90.16%

Binary Emotion:

        - SVM:
                - normal (scaled RGB): 90.4%
                - augmented facial landmarks: 92.4%
                - PCA:  91.6%
        - Logistic Regression:
                - augmented facial landmarks: 93.28%
                - PCA: 91.3%
        - CNN:
                - normal (scaled RGB): 93.3%

        - VGG16:
                - normal (scaled RGB):  95%
```


```
Binary Glasses:

        - SVM:
                - normal (scaled RGB): 88.9%
                - augmented facial landmarks: 89.58%
                - PCA: 86.4%
        - Logistic Regression:
                - augmented facial landmarks: 89.47%
                - PCA: 89%
        - CNN:
                - normal (scaled RGB): 98%

        - VGG16:
                - normal (scaled RGB):  93.3%

Binary Human:

        - SVM:
                - normal (scaled RGB): 99.56%
                - augmented facial landmarks: 98.4%
                - PCA:  99.2%
        - Logistic Regression:
                - augmented facial landmarks: 98.88%
                - PCA: 99.45%
        - CNN:
                - normal (scaled RGB): 99.8%

        - VGG16:
                - normal (scaled RGB): 99.8%
```


```
Multi-class Hair:

        - SVM:
                - normal (scaled RGB): 87%
                - PCA: 85%
        - Logistic Regression:
                - PCA: 83.7%
        - CNN:
                - normal (scaled RGB): 94%
```

The main cases of overfitting are present in the SVM, more particularly the human binary problem and the multi-class hair problem. There is indeed a clear difference between human and cartoons, in all aspects and features (mainly colours and sharp facial features), this is probably why the classification is so good. Looking at the curves however we do notice that it seems like we are just learning by heart and therefore there is some clear overfitting. This can be adjusted by greatly augmenting and increasing the training dataset and observing whether this behaviour shows up again. For the multi-class problem we do seem to be observing a little bit of overfitting, however it does seem that (for both Logistic regression and the SVM case) the training accuracy is slightly decreasing. A clear solution to be tested in that case is to increase the training dataset here as well. This can be compared to the binary glasses SVM learning curves. Here we clearly have overfitting that might be due to the non ideal nature of SVM for this particular problem (unlike LR which proves to be working fine), indeed we see a training accuracy that stays constantly at 1. Another metric to prove that our logistic regression was optimal compared to the rest is the computation/learning time which was in the order of seconds compared to minutes and even hours with our CNN implementation.

5. Related Work

A lot of different related work has been explored, the main one being the exploration of VGGnet using imagenet, Literature [6] [5] [4] has helped in implementing this with keras, we present our results in the Extra material section for different binary cases and the code on github. We have adapted the network to our binary problems by adding an additional Dense and dropout followed by a sigmoid activation and obtained promising none under-fitted / over-fitted learning curves

6. Conclusion

To conclude we have successfully run 30 different classifiers that give us much more intuition about the importance of data in binary and multi-class problems. We have also learnt a lot when it comes to overfitting and understanding how to adapt when it comes to this issue. The importance of pre-processing data to expose features that we want to work with has also proved to be very useful in reducing the complexity of our problems.

Further work includes continued exploration of VGG16 with different initialization weights (something other than 'imagenet', which would make more sense). Further exploration of transfer learning, data augmentation to further avoid overfitting (on top of dropout and regularization) are also logical steps forward.

References

[1] Convolutional neural networks for visual recognition. `http://cs231n.github.io/`.

[2] Opencv: Face detection using haar cascades. `https://docs.opencv.org/3.4/d7/d8b/tutorial_py_face_detection.html`.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. 2005.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[5] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks, 2016.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[7] R. Rifkin and A. Klautau. In defense of one-vs-all classification. 2003.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[9] P. Van Gent. Emotion recognition using facial landmarks. *A tech blog about fun things with Python and embedded electronics*, 2016.

[10] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features, 2001.

[11] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network, 2015.

[12] S. Yang, P. Luo, C. C. Loy, and X. Tang. From facial parts responses to face detection: A deep learning approach, 2015.

7. Extra Material

7.1. Stanford CS231n material

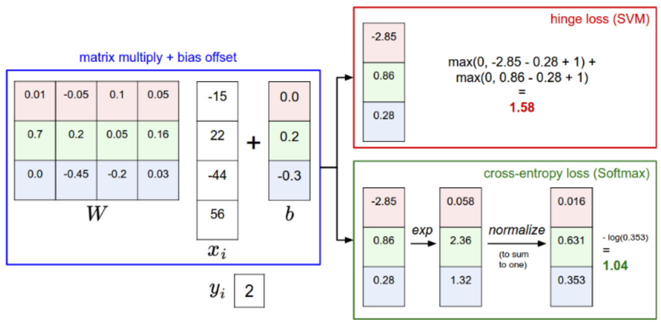
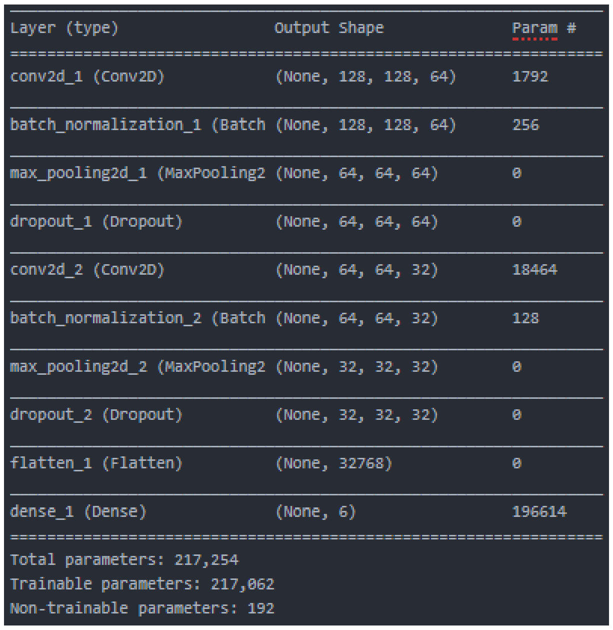


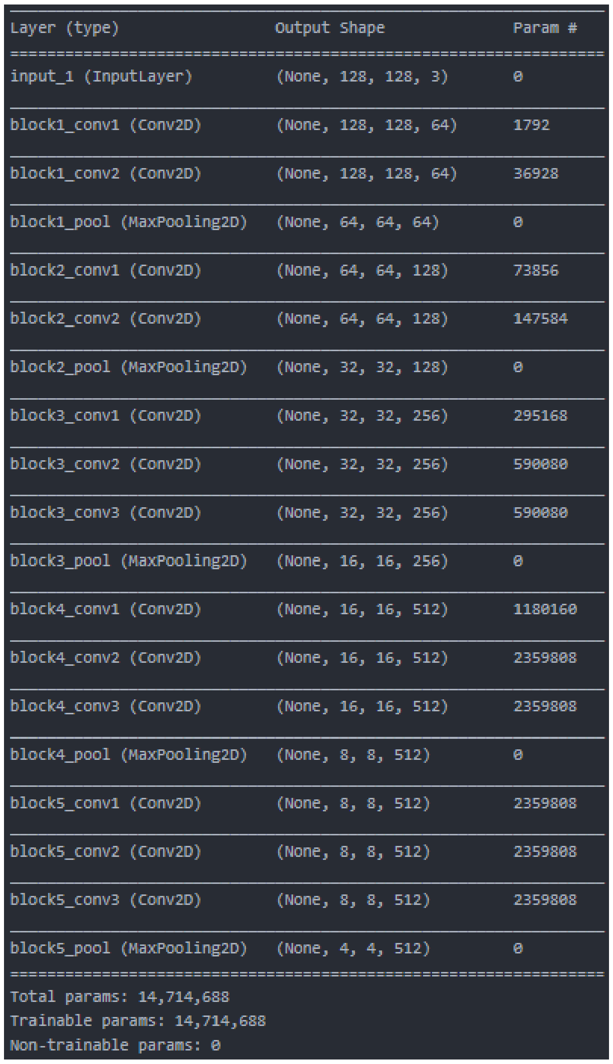
Figure 6. Figure easily explaining the differences between softmax and SVM with feature vector x_i and label y_i

7.2. CNN and VGG16 Keras model Implementations and Parameters



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 64)      1792
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128, 64)      256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 32)        18464
_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 32768)             0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 196614
=================================================================
Total parameters: 217,254
Trainable parameters: 217,062
Non-trainable parameters: 192
```

Figure 7. Keras model Representation of Custom CNN Architecture



```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 128, 128, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 128, 128, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 128, 128, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 64, 64, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 64, 64, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 64, 64, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 32, 32, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 32, 32, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 16, 16, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 16, 16, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 8, 8, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Figure 8. Keras model Representation of VGG16 Architecture

7.3. Binary Age Results

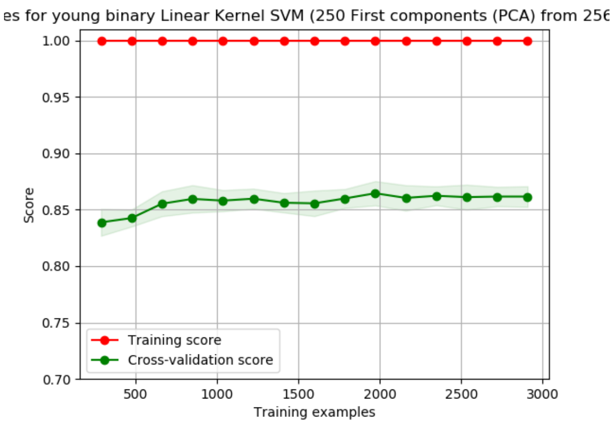


Figure 9. SVM PCA training accuracies on binary age problem

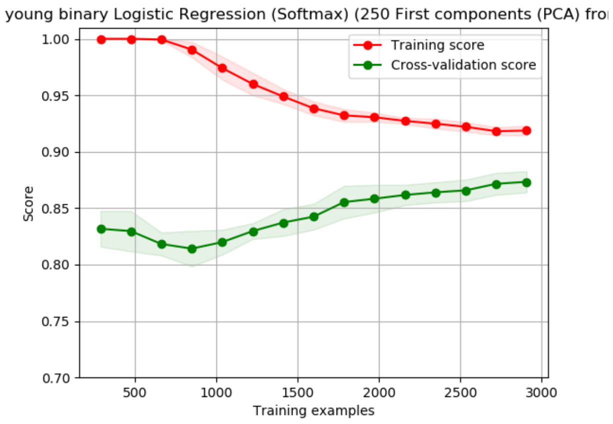


Figure 10. LR PCA training accuracies on binary age problem

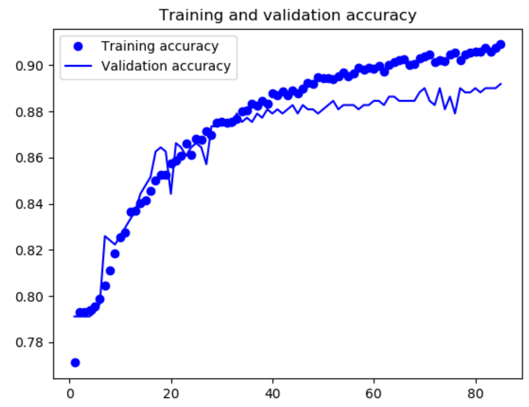


Figure 11. Training Accuracies of VGG16 Architecture on binary age problem

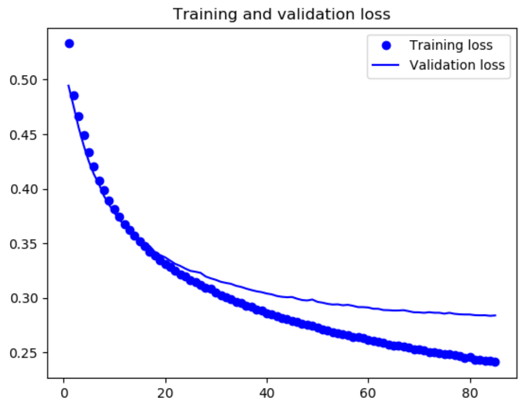


Figure 12. Training loss of VGG16 Architecture on binary age problem

7.4. Binary Emotion Results

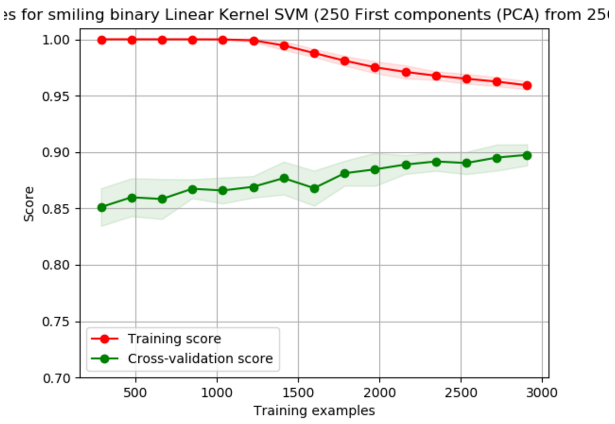


Figure 13. SVM PCA training accuracies on binary emotion problem

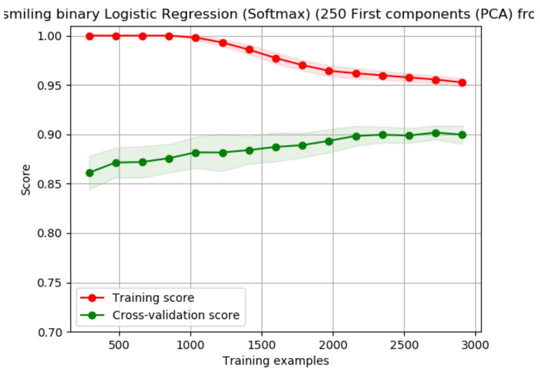


Figure 14. LR PCA training accuracies on binary emotion problem

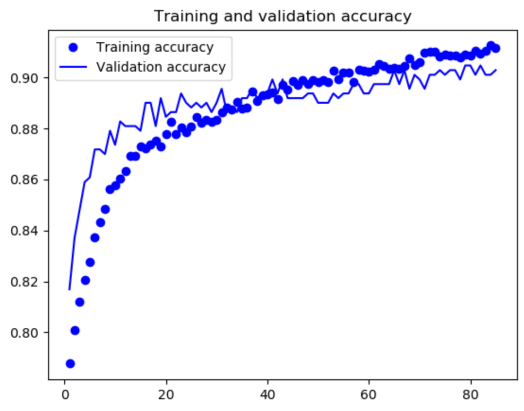


Figure 15. Training Accuracies of VGG16 Architecture on binary emotion problem

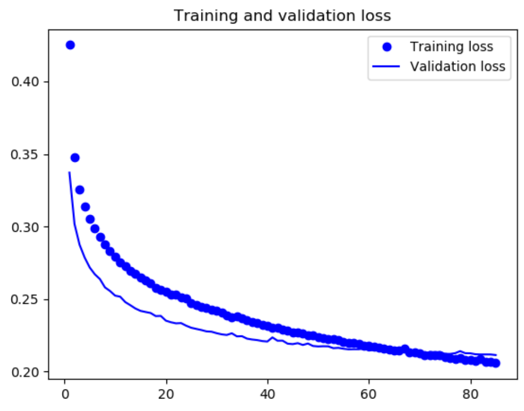


Figure 16. Training loss of VGG16 Architecture on binary emotion problem

7.5. Binary Glasses Results

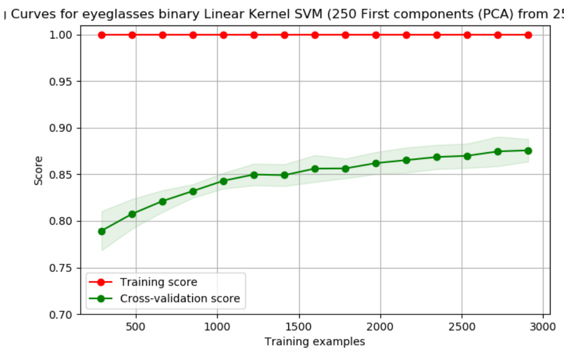


Figure 17. SVM PCA training accuracies on binary glasses problem

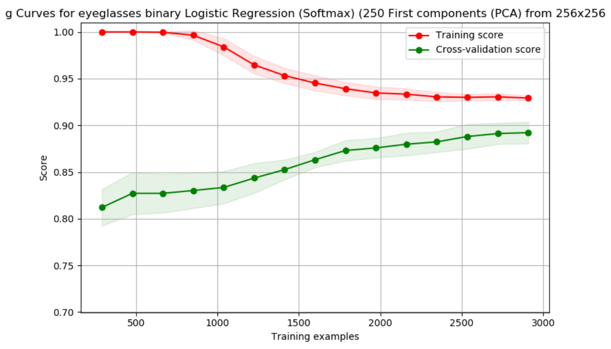


Figure 18. LR PCA training accuracies on binary glass problem

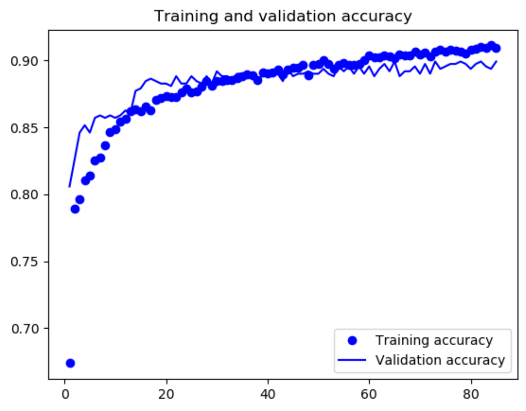


Figure 19. Training Accuracies of VGG16 Architecture on binary glasses problem

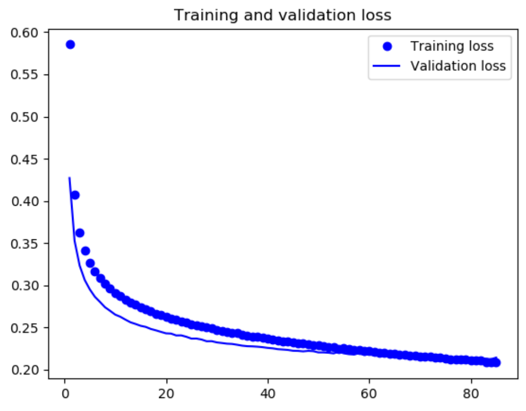


Figure 20. Training loss of VGG16 Architecture on binary glasses problem

7.6. Binary Human Results

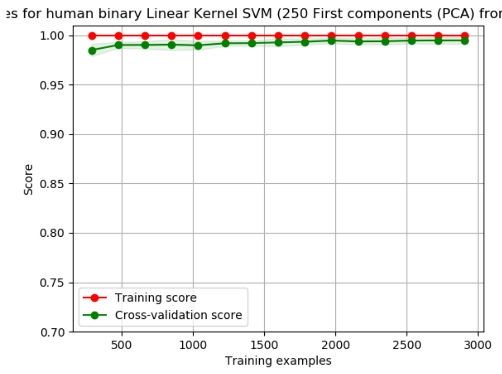


Figure 21. SVM PCA training accuracies on binary human problem

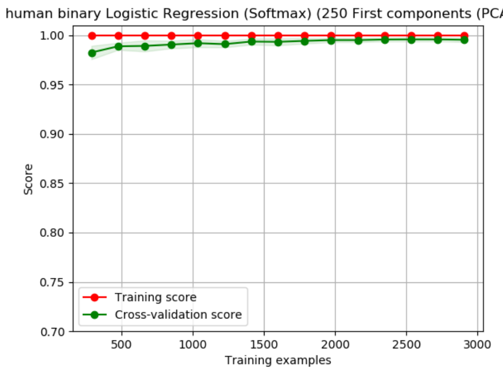


Figure 22. LR PCA training accuracies on binary human problem

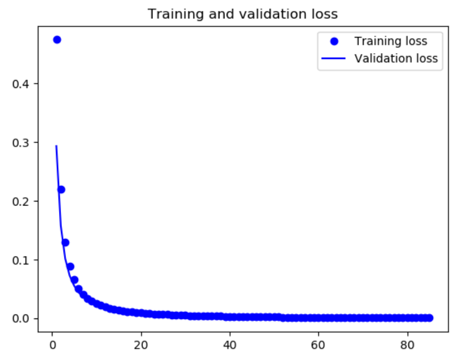


Figure 24. Training loss of VGG16 Architecture on binary human problem

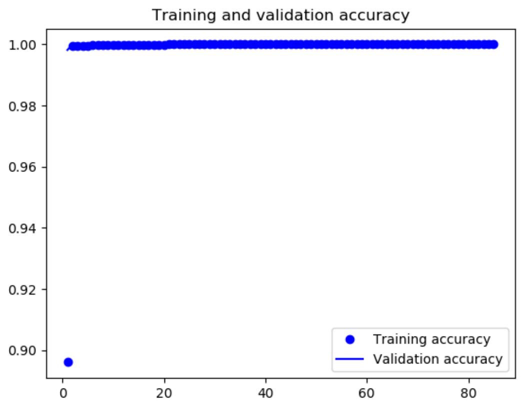


Figure 23. Training Accuracies of VGG16 Architecture on binary human problem

7.7. Multi-class Hair Colour Results

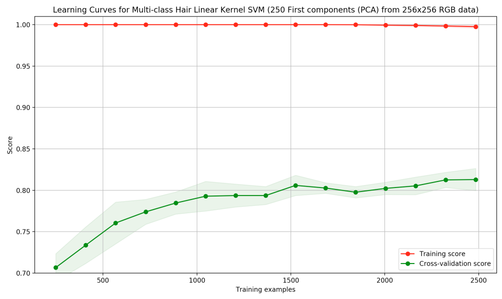


Figure 25. SVM PCA training accuracies on multiclass hair problem

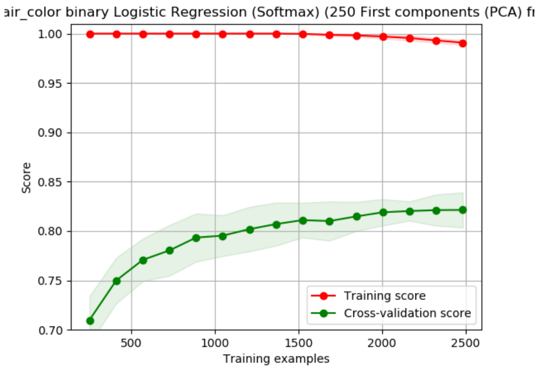


Figure 26. LR PCA training accuracies on multiclass hair problem