|  |  |
| --- | --- |
| COMP-1804-Applied Machine Learning  Coursework Report | Abstract  This report documents the details and vital processes involved in training 5 DNNs for 5 different classification tasks.  Abdullah Bin Zubair  001126108 | az5645a@gre.ac.uk |

Table of Contents

[Introduction 2](#_Toc69431870)

[Related Work 2](#_Toc69431871)

[Dataset Preparation 3](#_Toc69431872)

[Machine Learning method 4](#_Toc69431873)

[Evaluation 6](#_Toc69431874)

[Future Work 7](#_Toc69431875)

[References 8](#_Toc69431876)

## Introduction

Machine learning is the technique used to make machine perform specific tasks. In most cases it makes use of Neural Networks and train them so that they can solve a particular task. In this report we will train 5 different Deep Neural Networks (DNN) to solve 5 different classification tasks. We are given images as our dataset and so we will be using different python libraries all along the work flow to convert images to RGB, to implement complex architectures and to use Google Drive as our file directory. We will be going through the whole process of data annotation, data import and data preprocessing before feeding it to our neural networks. We will also discuss our choice of DNN and why we are training using that particular model given the dataset and resources available. Finally we will evaluate the performance of our models and comment how they can be further developed to give better accuracy on unseen data.

## Related Work

The field of computer vision has been around for around 50 years and it is linked with Artificial Intelligence (AI) from as early as 1960. However the most prominent decade for Machine Learning (ML) and all its related fields such as AI and Computer Vision has to be the latest one from 2010 to 2020. In 2009, the ImageNet data was released which is a large annotated dataset (14 million images) for computer vision research. This was made open source to promote research and development of improved methods for computer vision. These photos were annotated by humans using crowd sourcing platforms. From 2010 to 2017 an annual competition called ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was held to benchmark the state of the art models. Tasks included “image classification” and “object detection and localization”. In 2012, AlexNet, an 8-layer CNN, won the ImageNet Challenge by a phenomenally large margin (error 15.3%; the runner up was not a deep learning method: error 26.2%; prior competitors’ error was 25.7% and 28.2%). This network showed, for the first time, that the features obtained by learning can transcend manually-designed features, breaking the previous paradigm in computer vision. While AlexNet offered empirical evidence that deep CNNs can achieve good results, it did not provide a general template to guide subsequent researchers in designing new networks. The idea of using blocks first emerged from the Visual Geometry Group (VGG) at Oxford University, in their eponymously-named VGG network. VGG was the first runner-up in image classification and the winner in object localization at the ILSVRC 2014. It achieved error 7.3% (vs 15.3% of AlexNet). VGG has 16 or 19 layers and is deeper than AlexNet (8 layers. However, VGG consists of 138 million parameters (AlexNet consists of 61 million parameters). More parameters means more amount of time required for training. In our classification tasks, we will be using VGG network consisting of 16 layers with a slight modification in order to compensate the provided dataset and input size. Further discussion on the model used will later come under heading “ML method”.

**Environment used:** Google Colab notebooks for running python scripts and Google Drive as file directory.

## Dataset Preparation

We were given an unlabeled dataset in form of images. There were a number of processes involved in order to convert those images in the form which was acceptable for the model to ingest. The detail of each process is explained below:

**Data Annotation**

The annotation tool was provided by the module instructor and it was used to annotate data for our models. The provided dataset comprised of 2000 unlabeled images. From those, only 1000 images were annotated properly due to time constraint. To maintain the credibility of model results, random annotation was not used for remaining 1000 images. Due to this our models were only fed 1000 images for training. Steps followed for annotation are listed below:

* Downloading the images dataset
* Running the ‘merge\_images\_to\_video.py’ script to merge all the images in dataset to one single video file. Note that this script was also provided by the instructor. Along the whole coursework, only this script was run on local machine for creating the video.
* Converting the created video file to 720p using native or any other video player/editor.
* Opening the annotation tool in browser and importing the higher quality video file.
* Annotating each image one by one in video using the purpose-oriented interface. As discussed above, only 1000 images were annotated for the models.
* Exporting the annotated file created by the tool. File name: ‘AU\_video.txt’
* Mounting the file directory as we used Google Colab. For file directory we used Google Drive and mounted it with our Colab notebook.
* Running the ‘processing\_of\_output.py’ script and specifying the name and path of target file which was ‘AU\_video.txt’. The script produces a classic 2D array of annotated labels (array name: ‘outputs’). Notice that this script was made part of ‘training\_code.ipynb’. Also note that for this implementation the annotated file ‘AU-video.txt’ was uploaded to a specific folder in Google Drive (file path: ‘/content/drive/MyDrive/AU\_video.txt’) in order to import it into our Colab notebook.

**Data Compilation and Pre-Processing**

This process involves the creation of main dataframe which contains all the information we will be needing to train our model and then test it too. This process also includes the splitting and normalization of data. The steps involved in the creation of this dataframe are listed below:

* Continuing with the ‘outputs’ array of labels produced by the ‘processing\_of\_output.py’ script (refer to last point in section ‘Data Annotation’). Converting that array into Pandas dataframe. The name of this dataframe is ‘df\_all’ in our implementation.
* Adding the ‘file\_name’ column which contains the file names for which labels are entered. This was given by the instructor as well. This was amended to cater 1000 images only with correct mapping with the labels.
* Images dataset import: This process involves importing all the images into python notebook for use in training. The steps involved in importing the images dataset are listed below:

1. Importing the images dataset using python open-cv library. Converting them to RGB values. Note that in this implementation the folder which had all the images was named ‘images’ and it was uploaded to a specific location in Google Drive (folder path : ‘/content/drive/MyDrive/images/’). Also note that in this step the order of images was carefully maintained. As ‘file\_name’ column was populated first, images were also taken from ‘images’ folder by referring the file name of the image.
2. Resizing each image to 96 x 96 as this is the input size chosen for all of our models.
3. Appending each image to an image list in the correct order.

* Adding ‘images’ column to main dataframe with data from the newly created image list. The completion of this step marks the compilation of complete dataset (df\_all) we will be using for training and testing.
* Using pickle library to export the main dataframe (’df\_all’) so that we would not have to go through all the data creation steps every time we want to use this data. (This step was only performed when main dataframe was created for the first time)
* Using pickle again to import the dataset. File name: ‘df\_all.pkl’ (This step was performed whenever the already created dataset was needed for training or testing)

The following steps were performed before training each of the 5 models:

* Data was split into 80/20 ratio for training and validation respectively. This step takes specified information from our main dataframe (‘df\_all’) and creates classic X\_train, X\_test, y\_train and y\_test variables for training and testing. This was done using sklearn’s ‘train\_test\_split’ function.
* Data had to be manually converted to the shape which was acceptable to the model. This code snippet was also taken from the ‘Sample ML workflow’ notebook/Git repository provided by instructor.
* Data was normalized. X\_train and X\_test values were divided by 255 in order to make them in the range from 0 to 1.
* Data was augmented: For each model data augmentation layer was implemented. There were three types of augmentations performed – Random Flip, Random Rotation and Random Zoom. This was done using python Keras library which is a wrapper for Tensorflow and Theano platform.

## Machine Learning method

For these classification tasks given to us in coursework specification where the model has to pick right class for each image, we will be using a state-of-the-art Deep Neural Network (DNN) called VGG16 but with a little modification in its architecture. There are major reasons for using this network:

* This network has won the 2014 ILSVRC challenge with a marginal improvement from previous years,
* VGG convolutional filters are small (3x3) which makes it more efficient to pick details from image,
* VGG has very reasonable amount of layers for training in regard of time and memory required
* VGG has showed better accuracy on CIFAR dataset and on the provided dataset for coursework tasks as well. The selection of DNN to be used was followed by trying different DNNs on CIFAR and given dataset for classification tasks. Other DNNs tested includes: ResNet50, VGG19 and ResNet101
* VGG is easily implemented using Keras and it can provide the trained weights for bottom layers on ImageNet dataset

**Transfer Learning**

In order to make full use of Keras, we downloaded all the pre-trained weights of VGG16 obtained through training on ImageNet dataset. This saves us time as we do not have to train the network from the scratch and learned weights can then be altered in order to modify the network to perform the tasks given in coursework specification.

**Modification**

To align our network with the input size of our data (96 x 96), we have to include another block on the top consisting of 3 convolutional layers and 1 max pooling layer. This makes the network consistent with our input size. After this block, the input was flattened and 2 dense layers were added as usual before the output layer.

**Fine Tuning**

For these tasks we will not be fine tuning/freezing our bottom layers. This decision is backed by the practice of training the network with fine tuning and not getting better results when compared with training with bottom layers further learning. Freezing the layers means you are not letting the parameters of bottom layers to further train/change which makes it less favorable for our classification task accuracy.

**Training parameters given**

There are a number of parameters involved in training the model. The value for each parameter is given/decided through many experiments and testing the network with different set of values and thus selecting the one which gave best accuracy on validation dataset. Parameters include:

* Input size: (96 x 96 x 3)
* Learning rate: 000.1
* Loss: Binary Cross Entropy (Binary-classification – Wrinkles, Freckles), Sparse Categorical Cross Entropy (Multi-class classification – Hair top, Hair color, Glasses)
* Optimizer: Adam
* Activation function of output layer nodes: Softmax for Multi-class and Sigmoid for Binary class classification
* Epochs: 10 epochs were given to each model. The epochs were not increased because model was not showing any accuracy gains in successive epochs
* Batch Size: 16 images were given as one batch. This is a reasonable number as we tested 32, 256 and 512 as well but 16 gave the best results
* Steps per epoch: Size of training data/ batch size
* Checkpoint parameters: For every model there are different checkpoint settings but mostly every 5 epochs the model/model weights are saved to file directory.

## Evaluation

First we will be mentioning that we have to take into account the 2 major **limiting factors** for the accuracy which are: (1) Only 800 images (as we had split 80/20 for train/test in our training\_code.ipynb notebook) were used to train the network and (2) There was unequal class distribution for all the 5 attributes/features in the training data given to model.

As we have 5 different models, we will discuss the evaluation report for each of them separately:

1. Wrinkles: For this attribute our DNN has given 67.50 % accuracy on validation set. This means that our model is giving this much accuracy on the images which it has not seen before. When this model was tested against new images in ‘testing\_code.ipynb’ notebook, it gave the precision of 0.7 which is fairly good score when we consider the limiting factors. We used micro average for precision and all other scores in all models because we have unequal distribution of classes in all our features that’s why micro averages are more suitable than macro in these given circumstances. The difference between macro and micro averaging for performance metrics (such as the F1-score or Precision) is that macro weighs each class equally whereas micro weights each sample equally. If the distribution of classes is symmetrical (i.e. you have an equal number of samples for each class), then macro and micro will result in the same score.
2. Freakles: For this attribute our DNN has given 73.00 % accuracy on validation set. And when this was tested against the new images it gave precision score of 0.6 and accuracy of 60 % in ‘testing\_code.ipynb’ notebook. Now this is pretty average accuracy and precision score but again we have to take into account the limiting factors, especially the size of images in this case as for freckles, it can be interpreted that the model will take significantly more amount of labelled images to capture the edges/traits of freckled person as they are not easily detectable.
3. Glasses: For this attribute our DNN has given 88.00 % accuracy on validation set. This is very good but we also have to consider that this feature is relatively easy to detect as it represents a significant change in the pattern of RGB values. This model gave us precision of 0.8 when tested against new images in ‘testing\_code.ipynb’. This is the best performing model from all 5.
4. Hair Color: For this attribute we had a very poor accuracy of 32.5% on validation set. This is below average and this model is nowhere near to be put into application. It needs a lot of work and effort before it is capable to predict better than what it has shown now. The poor performance is backed by the unequal samples for different classes. One of the class (brown) has the most amount of occurrences and thus the weights leading to output for reaming classes suffer heavily because of this. The data given to model for training is also very less in size thus effecting the performance of model badly. This model gave us the precision score of 0.4 when tested against new images in ‘testing\_code.ipynb’ notebook.
5. Hair Top: For this attribute our DNN has given 64.50 % accuracy on validation set. This again is very average and this network also needs more training on labelled data. However when tested against the new images, this model gave precision score of 0.7 and approximately 70% accuracy. Now this can be taken as luck but it not too far from what it gave on validation set. The training data given to this model had better distribution of classes then all the other models.

Note that each model has not been further trained if there was no increase in validation accuracy seen in successive epochs. This was ensured to prevent overfitting because training more would have only caused model to fit the training data vigorously if validation accuracy was not increasing.

## Future Work

There is further potential in these models. These accuracy scores should just be taken as initial foundation for these tasks. Firstly the network should be trained using more amount of annotated/labeled images. Now when we say ‘more’, then it does not mean that it has a roof limit, you can give 1000 images or 10,000 images or even more, it will only do better. The more amount of accurately labeled images, the more accuracy. Another thing which must be done is to maintain the equal distribution of each class in the sample dataset. This is very important because nobody wants their model to be bias and predict wrong results. If availability of resources is not going to be a problem for someone who wants to further work on these tasks then denser, more heavily layered, structures like ResNet152V2, DenseNet201 etc. should be experimented with low learning rate and more amount of epochs. There are other things which can be done in order to improve the performance but the most important ones are that each class should have same number of samples in the training dataset and the training dataset should be accurately increased if future work is to be done on these models.

Note: The new images folder is uploaded in ‘others’ section on moodle page.

## References

* <https://datascience.stackexchange.com/questions/45974/micro-f1-and-macro-f1-are-equal-in-binary-classification-and-i-dont-know-why>
* <https://github.com/mahtabhossain/comp1804-pca-image-dataset/blob/main/pca_logistic_regression_courswork_image_sample.ipynb>
* <https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/>
* <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html>
* <https://datatofish.com/export-dataframe-to-csv/>

Note: Other than these references, only the course material was used for whole implementation of this coursework.