APPLIED MACHINE LEARNING SYSTEM ELEC0132 20/21 REPORT

SN: 17006378

ABSTRACT

The purpose of this project is to study and practise the implementation of various machine learning methodologies to solve the following real world problems: Gender, Emotion, Face Shape and Eye Colour Classification. Datasets used involve the celebrity A dataset and the Cartoon dataset. The models chosen are convolutional neural networks (CNN) and support vector machines (SVM) as well as a pretrained facial landmark detector based on an ensemble of regression trees. For the binary classification problems, the best performing model for gender classification, the CNN model, achieved a testing accuracy of 94%, while the best model for emotion classification, the SVM model, achieved a testing accuracy of 88%. While for the multiclass classification problems, CNNs proved to be the best model for both the face shape and eye colour classification, achieving 99% and 88% testing accuracy respectively.[1]

1. INTRODUCTION

Facial recognition has become a key area of research in the field of computer vision and machine learning techniques have played a vital role. This report will discuss the application of various machine learning methods used to solve four different classification problems. The first two are binary classification problems on gender and emotion identification on the "CelebA" dataset. The second two are multiclass classification problems on face shape and eye colour recognition and will be carried out on the "Cartoon Set" dataset.

Two prosepective approaches were used to solve the gender classification task on the celebrity dataset including passing the faces into a convolutional neural network (CNN), performing both the feature extraction and classification in the same network. The second makes use of a facial landmark detector that creates an active shape model (ASM) for each image with 68 coordinates corresponding to locations of identified facial features. This data is then used to train a support vector machine (SVM).

The task of emotion recognition required the use of the facial landmark detector which identifies and crops the faces to just the mouth before being passed to a CNN that is trained to detect whether a face is smiling or not. Similarly, the matrix of facial landmarks is also used to train a SVM model as a comparison.

Similarly to the gender classification task, the approach to classifying face shapes on the cartoon dataset involves the use of a CNN to perform feature extraction and classification. The facial landmark detector again is used in conjunction with a SVM model to build an alternative model.

Finally for the eye colour classification problem, the facial landmark detector is used to perform feature extraction to crop to the left eye of the face. Noise in the form of the face wearing sunglasses are removed before being passed to a CNN for training.

The machine learning models used in these exercises will be explained in detail as well as how they are implemented and executed to solve the provided problems. Results on their performances will be analysed and conclusions will be made on the whole learning process of this project.

2. LITERATURE SURVEY

The use of Support Vector Machines have shown to perform well for pattern recognition problems. The following paper [1] proposes the use of an SVM model alongside a binary tree structure in two class classification problems using a linear classifier model. They discuss datasets containing degrees of variability in features such as expression, angles, lighting and poses that is highly common in real world datasets such as our celeb A dataset. This would especially apply to tasks A1 and A2 which are both binary classification problems and work on the Celeb A database which contains faces with a reasonable degree of variability.

In [2], the paper provides an implementation of using both a CNN and a support vector regression (SVR) model together to classify enriched emotion expressions rather than traditional methods which only classify basic emotions. The idea is that the CNN is used to extract features from the images and then the SVR is used as the classifier stage to predict more complicated emotions. This is rather different to normal CNN models that do both the feature extraction and classification in one. However, the idea of using separate models for feature extraction and classification may provide more optimal results even on basic emotion detection.

in [3], a multipurpose convolutional neural network was implemented that could simultaneously carry out facial detection, face alignment, gender recognition and smile detection

[1] The code is provided in GitHub project: https://github.com/zceeatv/AMLS_assignment20_21_SN17006378.git

to name a few examples. This was achieved from the principle of the CNN, where the lower layers learn the feature extraction and the higher layers can classify according to different problems. The development is in the formulation of a multi-task learning framework in which the lower layers are shared among different upper layers that learn for their respective tasks. This is a very interesting concept and perhaps could be applied to this project, where only a single CNN model needs to be trained for tasks A1 and A2 or for tasks B1 and B2.

In [4], the paper discusses the implementation of a cascading system of up to 12 convolutional neural networks that each isolate a separate resolution of the images, some focusing on creating the bounding box detection for the face and others specialised for finer details such as facial features. Through this cascade the model would be able to overcome difficult scenarios such as faces turned to an angle or distorted through a mirror which would provide a good solution for Task A1 and A2 which experience the same problems as a result of the celeb A dataset. Such a model would unfortunately be far too complicated and require a large amount of computational power to achieve, however perhaps reducing the number of cascaded CNNs may provide a good compromise between effectiveness and complexity.

In [5], the paper also talks about using SVM for automatic facial recognition and argues that the discrimination functions learned by SVMS can allow for higher accuracy than some popular multilayer perceptron models. This would be an interesting investigation to carry out, comparing SVM models with traditional MLP models and even CNNs to see how they perform on both binary and multiclass problems.

Kazami and Sullivan presents an algorithm [6] that performs face alignment with reportedly very fast speeds and high accuracy comparable with current state of the art methods. Though face alignment isn't a direct application that is needed for this project's problems, it offers a method that carries out feature extraction of faces through the use of a number of cascaded regressors. A python library called dlib is an implementation of such an algorithm which would come in helpful for tasks A2 and B2 where feature extraction would be required to find the location of the mouth or eyes.

3. DESCRIPTION OF MODELS

3.1. Convolutional Neural Networks

A convolutional neural network extends from a traditional multilayer perceptrons neural network by having convolutional layers comprising of filters and feature maps which allow the analysis of 2D data such as images. Traditionally, the image matrix would be flattened into a 1D vector which would cause the loss of spacial structure in the image. After the convolutional layers, the feature maps created will be passed to fully connected layers which will then be trained to classify the corresponding labels.

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))} \tag{1}$$

The output layer has an activation function of softmax for the use of categorical classification and often used when the labels are subjected to one hot encoding [7]. This is because softmax squashes the output of each neuron in the range of 0 and 1.

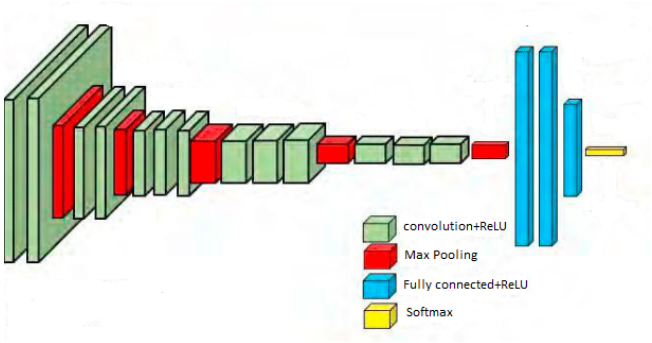


Fig. 1: Example of Convolutional Neural Network, adapted from [8]

Fig 1 shows an example of a CNN, however in my implementations less convolutional layers are used for simplicity. In the first layer of the CNN, all the pixels from the image are taken in and different filters are applied, forming representations of different parts of the image known as feature maps. A common filter size for CNNS are 3x3 [9] to cover both height and width of the image and is used during the implementation of this project.

$$Relu(x) = max(0, x) \tag{2}$$

The purpose of the activation function used in the layers is to take the values that represent the image after convolution and increases their non-linearity since the images were non-linear before the convolution process. Rectified Linear Unit (ReLU) is used to this extent to provide optimial performance [10].

Pooling is a key process after the data is activated where the information representing the image is compressed by removing information the network believes is not important. This allows the network to be less affected by noise and prevents overfitting. These additions to a regular Neural network allows for a specialised method for working with two dimensional image data such as our celebA and cartoon set datasets for classification problems.

3.2. Support Vector Machines

Support Vector Machines fundamentally work by plotting features from a dataset such as (x,y) coordinates onto a fea-

ture space and then determines the optimal hyperplane that separates the decision regions into separate categories as shown in Fig 2. SVM is responsible for choosing such a hyperplane that maximises the margins from both regions. This makes it particularly good for Task A1 and A2 due to their binary classification nature but can be extended to the multiclass problems by making use of multi dimensionality.

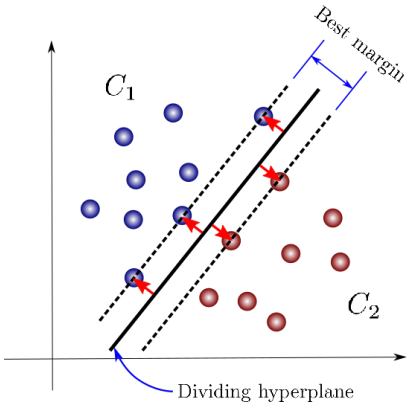


Fig. 2: Example of a feature plane with decision boundary

The use of a linear plane separating the decision regions such as the one shown in 2 is not the only possible decision boundary available since non-linear classifiers such as different degree polynomial and radial basis functions are also options.

3.3. Facial Landmark Detector

The facial landmark detector from the dlib library is used in a number of approaches for the tasks in this project. It is made using a classic Histogram of Oriented Gradients (HOG) feature combined with a linear classifier.

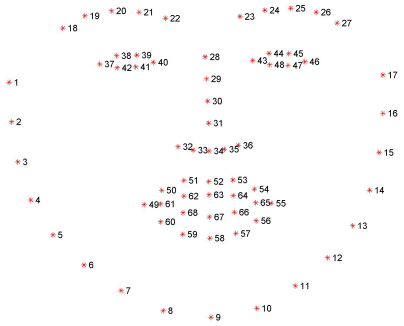


Fig. 3: 68 landmarks used in facial landmark detector

The structure of the model revolves around a cascade of regressors which make predictions on features such as pixel intensity, introducing some form of geometric invariance in each stage [6]. As it cascades through the regressors, the model can be more certain that a precise location on the face can be indexed with a landmark from Fig 3. the dlib library includes a pretrained model that is ready to be used. This landmark detector can be applied to all the tasks of the project in that it can extract x, y coordinates for landmark locations which can be applied directly to a SVM model, allowing the model to determine a suitable hyperplane to separate decision regions. Similarly it can be used in the preprocessing section for the CNN models as a way to abstract the images initially by removing none important information before it is passed to the CNN for classification learning.

3.4. Task A1: Gender Classification

Classifying gender involves analysis of the whole face, with all facial features having characteristics distinct for each gender. I decided to use two models of varying complexity with the aim of comparing their effectiveness. I chose to use a deep learning approach involving a convolutional neural network with basic preprocessing in the form of grayscaling the images and resizing them appropriately. The less complex method uses a linear SVM detector however more preprocessing was required, involving using facial landmarks to localise and label the main facial regions of each face.

3.5. Task A2: Smile Classification

In detecting whether a face is smiling or not, the analysis of the image can be localised to the mouth region, however I wanted to investigate to see whether a deep learning model could use other facial features such as whether the person was frowning to determine this classification. From the successes in my task A1 models, I decided to again implement a neural network that carried out basic preprocessing as well as an approach which used the facial landmark detector to identify the mouth area and crop to the surrounding area which was then passed to the neural network. Similarly, I used a SVM model with facial landmark detection carried out in the preprocessing of the images to compare the two approaches of varying complexity.

3.6. Task B1: Face Shape Classification

Working with cartoon faces instead of real life faces is expected to lead to more accurate performances as all the faces are uniform in that their size and orientation are the same. Again I decided to preprocess the data by grayscaling and resizing the images and passing it to a convolutional network. Alternatively I also used a facial landmark detector to preprocess the images and pass feature locations to a linear SVM model to investigate how a less complex machine learning model performs.

3.7. Task B2: Eye Colour Classification

Eye colour classification was more complicated than anticipated due to the presence of a number of faces having sunglasses on. These faces can be deemed as outliers that would only be problematic when passed to my machine learning models so were filtered out when compiling the training data set. I did investigate to see if a deep learning neural network could use other facial features such as hair colour to make a connection to the eye colour, despite the faces wearing sunglasses. However my final neural network model was given preprocessed faces that used the facial landmark detector to crop to the eye area, discarding any that were detected to be wearing sunglasses.

4. IMPLEMENTATION

4.1. Task A1: Gender Classification

4.1.1. Neural Network

The preprocessing for the images and labels were carried out using the pandas and openCV libraries. Each image was taken in and converted to a grayscale colour scheme and resized down from 178 x 218 pixels to 45 x 55 pixels using functions from the cv2 library. I chose to convert the image to grayscale as RGB colour doesn't make an impact on classifiying gender, but will reduce the size of the data per image by three times. Reducing the size of the image also reduces this size per image. If the values of the input data are too wide, it may have a negative effect on how the network trains so all the colour values, ranging from 0 to 255 are normalised to between 0 and 1. Pandas was used to open the csv file containing the labels and used to just extract the image number and the corresponding value for the gender classification. Using a numpy function also allowed me to convert these values to one hot encode so a 1 is represented by 10 and 0 as 01. Numpy arrays were also used to split the dataset into training, validation and testing sets using a 8:1:1 ratio.

Firstly a simple sequential Keras model was created with an input layer, two fully connected hidden layers and an output layer consisting of two neurons. Though the task is a binary classification task, since I chose to use one hot encoding there will be two neurons, each determining which gender the neural network will predict. Due to this I chose the softmax activation function, which will squash the output of each neuron in the range of 0 and 1. The loss function was chosen as "categorical crossentropy" and not "binary crossentropy" again since I use one hot encoding so there are actually two output neurons and not one. The Adam optimiser with a learning rate of 0.001 was initially used due to its history of good performance in image classification.

This model produced poor accuracy results, below 50%, so the neural network was adapted to include two convolutional layers, one with 100 filters and the next with 200 filters [2] where each has size 3x3. The input shape to the first convolutional layer is 55 x 45 x 1 for the size of the input image and the number of channels, which is 1 for grayscaled images. After each convolutional layer, pooling layers are used to down sample the output of the convolutional layers, with the aim of abstracting away unnecessary parts of the image. Batch normalisation is also used here to normalise the inputs heading into the next layer so the network can create activations with a desired distribution. The feature maps from these layers are then flattened into a long vector of sequentially ordered numbers that is then passed to the fully connected layer. To prevent overfitting, dropout layers are added between the connected layers to randomly eliminate 20% of existing connections as well as using kernel constraint set to 3. This constrains the magnitude of weights below a limit and has the effect of regularising data as the network learns.

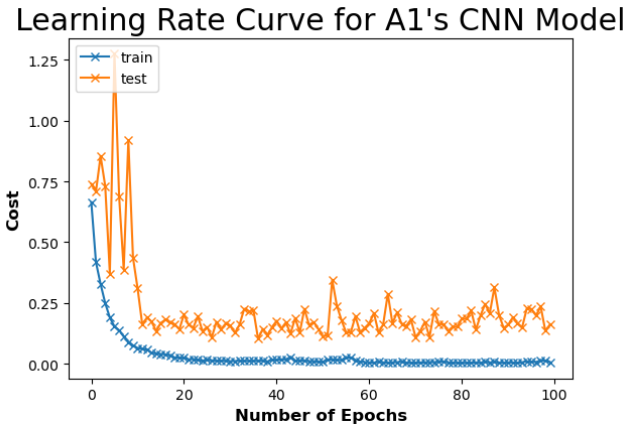


Fig. 4: Graph showing cost varies throughout training

This CNN performed very well, achieving accuracies above 90% when tested on the validation set. Learning curve based on the cost function created can be seen in Fig 4 showing that the loss reached an optimal level after about 20 epochs and leveled out. However there is a generalisation gap between the training and testing curves as well as noisy movement around the validation loss which lead to the validation loss not converging towards the training loss, meaning no matter how long the model is trained for, it will never come close to 100% accuracy. The learning rate of the Adam optimiser was reduced to 0.0001 from 0.001 to reduce the noisy movement of the loss.

The next step was to perform hyperparameter optimisation, which was carried out using grid search processes found in the scikit-learn libraries; results can be found in section 8.1. Some of the results for parameter optimisation were used such as running the training at 20 epochs with batch sizes of 64, using 128 and 64 neurons for the two connected layers and 30% dropout rate with kernel constraints of 3. Other hyperparameters such as the neuron activation functions, and

optimiser were kept the same as in the initial model as further testing showed these performed better. These include the activation functions being ReLU for the layers and loss function being "categorical_crossentropy." The grid search scripts took up significant amounts of computational power and often gave varying results so were not considered certain in identifying optimal parameters. With these optimisations the accuracy increased to 95%.

4.1.2. SVM

The preprocessing for using the SVM model requires the use of the dlib library as well as the "shape predictor 68 face landmarks.dat" data file to use a pre-trained facial landmark detector that estimates the location of 68 x, y coordinates that map the facial structures on the face. Some external helper functions found in preprocess_data.py are used in order to make the dlib detector compatible with our datasets. This includes `rect_to_bb()` which simply converts a rectangle bounding box from dlib to version that is compatible with OpenCV and `shape_to_np()` which takes in a shape object creating from dlib's detector and populates a numpy arraay.

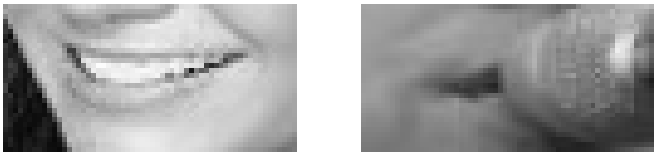
The faces are inputted into the landmark detector and which returns a numpy array of size (68, 2). This detector was able to detect faces for 4795 faces out of the 5000 samples. The cause for some being undetectable could be in the orientation of the face as a side profile may make the mouth or nose unidentifiable when compared to the landmark data file. The dataset has a shape of 4795 x 68 x 2 which is split into training, validation and testing sets with the ratio 8:1:1.

The Scikit-learn library is used to create an SVM classifier model. The training dataset is flattened to have an input of a 1D numpy array with 136 items and passed to the classifier for training. In terms of optimising hyperparameters, it is more straightforward than with neural networks and trying different kernel types for the algorithm resulted in the best type being linear giving validation accuracies around 91%. This shows that a considerably less complex approach can perform to satisfactory results, being only 3% less accurate.

4.2. Task A2: Smile Classification

4.2.1. Neural Network

Due to the success of my neural network model from task A1, I decided to reuse much of the model's structure, with two convolutional layers (100 and 200 filters[2]), sandwiched with max pooling layers and two connected layers (128 and 64 neurons) with batch normalisation and dropout (30%) between each layer. The Adam optimiser was also used with a 0.0001 learning rate. The activation functions for the layers were set to ReLU and loss function set to "categorical_crossentropy."



(a) Front profile smile (b) Difficult to identify

Fig. 5: Example of images from preprocessing faces

Initially the images were converted to gray scale and resized to 45 x 55 pixels and passed to the CNN, however applying the model to validation data resulted in around 70% accuracies. I decided to reuse the dlib library to carry out feature extraction on the images. Through this, the location of the mouth could be cropped out with 5 pixel cushioning on each side, resulting in a new 30 x 60 pixel image that is inputted into the CNN; examples shown in Fig 5. The colour values are all divided by 255 to normalise to between 0 and 1, while the labels are all converted to one hot encoding. Using this preprocessed dataset gave accuracies around 85%.

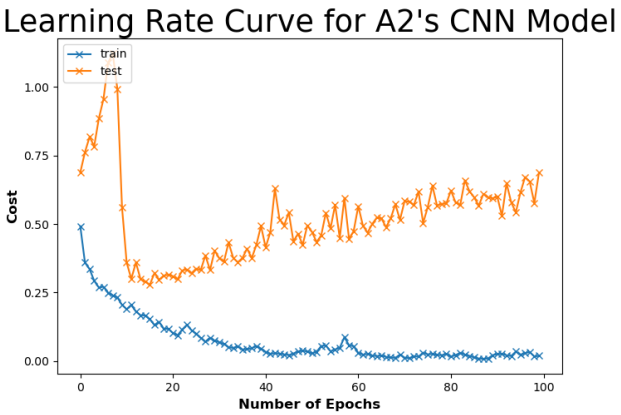


Fig. 6: Graph showing cost varies throughout training

Training this model for 100 epochs shows the learning rate curve found in Fig 6 which shows that over epochs, the validation loss actually begins to increase while the training loss decreases. This relationship indicates that the model is overfitting the training dataset and so training should be stopped at the inflection point around 20 epochs. A potential reason for this overfitting could be from the quality of the dataset, since some faces show a side profile of the face rather than a frontal profile. As seen in Fig 5 (b), some faces have obstructions such as microphones and fingers obscuring the mouth as well as faces sticking out their tongue which would be classified as outliers since it cannot be inferred whether they are smiling or not. This is an issue for this task in particular because it isolates to a region of the face as opposed to A1 where the entire face can help identify gender even if the mouth is obscurred, leading to a lower accuracy score.

4.2.2. SVM

The SVM implementation in Task A2 uses the same structure as in Task A1, with preprocessing being carried out by the dlib facial landmark detector which returns a (68,2) numpy array of x, y coordinates for each image. The labels like with all other models are converted to hot one encoding.

I investigated to see whether using all the facial landmarks affected how the model trained to identify a smile. For example, frowning of the face could be an indication of the face not smiling. Applying this straight to a SVM classifier with a linear kernel type gives a validation accuracy of 80%.

However, when I isolated the data to just the 19 x, y coordinates of the mouth area, changing the input shape of the classifier to a numpy array of size 38 (19x2) I received better validation results of around 87% which actually outperforms the convolutional neural network I implemented for the same task.

4.3. Task B1: Face Shape Classification

4.3.1. Neural Network

The preprocessing for the cartoon faces was essentially identical to that of Tast A1, where the images were converted to grey scale and resized to 50 x 50 pixels. The colour values are all divided by 255 to normalise to between 0 and 1, while the labels are all converted to one hot encoding, with a 5 bit width to account for the 5 different classes.

I chose to apply the same neural network structure that had been used in the binary classification tasks for tasks A1, with two convolutional layers (100 and 200 filters[2]), sandwiched with max pooling layers and two connected layers with batch normalisation and dropout (30%) between each layer. The Adam optimiser was also used with a 0.0001 learning rate. The activation functions for the layers were set to ReLU and loss function set to "categorical_crossentropy."

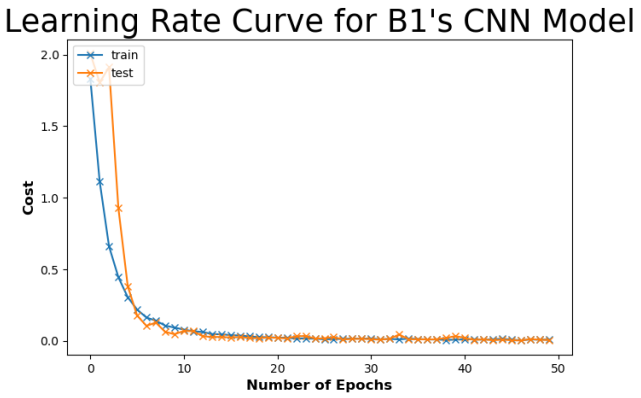


Fig. 7: Graph showing cost varies throughout training

This CNN performed very well, achieving accuracies above 95% when tested on the validation set. Learning curve based on the cost function created can be seen in Fig 4 showing that the loss reached an optimal level after about 20 epochs and leveled out. This curve is a good example of a good fitting learning curve, with a small gap between the training and validation loss. The learning rate of the Adam optimiser was set to 0.0001 as higher values caused the curve to have larger spikes during the learning process.

Applying hyperparameter optimising through grid search method (results in section 8.2, led to the following hyper parameters: training at 20 epochs with batch sizes of 64, using 512 and 256 neurons for the two connected layers and 20% dropout rate with kernel constraints of 4. With the following changes, the model was able to achieve validation accuracies of 99%.

4.3.2. SVM

The SVM implementation in Task A2 uses the same structure as in Task A1, with preprocessing being carried out by the dlib facial landmark detector which returns a (68,2) numpy array of x, y coordinates for each image. The labels like with all other models are converted to hot one encoding, with a 5 bit width to account for the 5 different classes.

For the face shape classification, I tried two approaches similarly to task A2 by using all 68 points and isolating to just the points associated with the face shape. Both gave the same validation accuracy of 88% when a linear kernel type was chosen. This showed the largest difference in performance when compared to the convolutional neural network which almost had an accuracy of 100%.

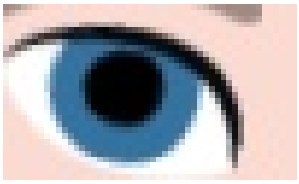
4.4. Task B2: Eye Colour Classification

Due to the success of my neural network model from task A1, I decided to reuse much of the model's structure, with two convolutional layers (100 and 200 filters[2]), sandwiched with max pooling layers and two connected layers (512 and 256 neurons) with batch normalisation and dropout (20%) between each layer. The Adam optimiser was also used with a 0.0001 learning rate. The activation functions for the layers were set to ReLU and loss function set to "categorical_crossentropy."

With the classification of the eye colour, the main concern is that there are faces in the dataset that contain sunglasses. For the preprocessing, initially the images were just resized to 50 x 50 pixel, the colour values all divided by 255 to normalise to between 0 and 1, while the labels are all converted to one hot encoding, with a 5 bit width to account for the 5 different classes. This was to test to see whether the network could make any connections between the hair colour of the faces with the eye colour. For example, faces with black hair are more likely to have black eyes. This relationship was unfortunately not seen by the model, resulting in accuracies around 80%. Therefore, the faces with sunglasses on can be

regarded as noise that should be eliminated from the training set.

(a) Cartoon Face with glasses



(b) Cartoon Face without glasses

Fig. 8: Example of images from preprocessing faces

I decided to reuse the dlib library to carry out feature extraction on the images, getting the location of the left eye and cropping to that section, resizing the eye to 50 x 30 pixels, as can be seen in Fig 8. To differentiate whether the face had sunglasses, the function `identify_sunglasses()` is used to find the colour of a section of the image and compared the other 1500 pixels. If over 2/3 of the image is the same colour, it can be safe to assume that sunglasses are present. Of the 10000 faces, 2400 were found to have sunglasses. This method however doesn't account for faces with tinted sunglasses. Splitting this data into training, validation with ratio 8:2 and passing it to the CNN gave validation accuracies of 98%. Creating a test set from the original dataset that included faces with sunglasses and making the trained model make predictions let to an accuracy of around 85

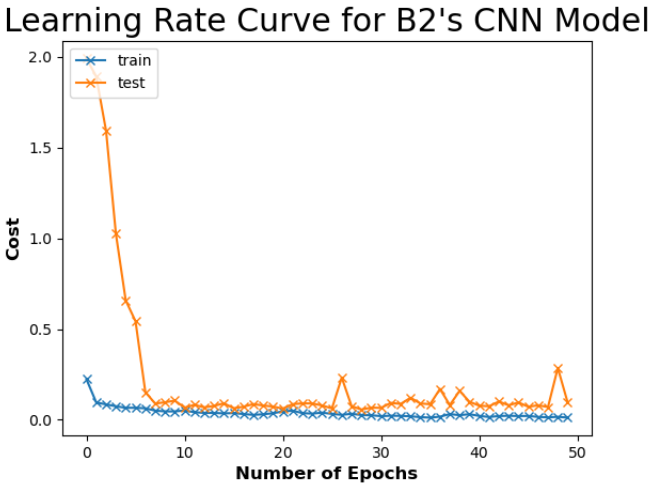


Fig. 9: Graph showing cost varies throughout training

When the CNN was provided with the none sunglasses eye images, it performed well and its learning curve can be seen in Fig 9. Similarly it shows a similar relationship to to the learning curve from Task B1 with the generalisation gap being minimal. It can be seen that the loss reaches a minimum around 10 epochs which is determined as the stopping criterion.

5. EXPERIMENTAL RESULTS AND ANALYSIS

Task	Model	Train Acc (%)	Val Acc (%)	Test Acc (%)
A1	CNN	99.77	95.40	94.00
A1	SVM	93.93	92.05	91.64
A2	CNN	94.28	86.85	85.62
A2	SVM	88.35	88.05	87.92
B1	CNN	99.74	99.80	99.87
B1	SVM	92.22	91.76	91.22
B2	CNN	99.53	98.63	87.98

Table 1: Table showing performances of all machine learning models used for each task

Table 1 shows the results for each model used to solve the following tasks, with the training accuracy being the accuracy of the model after the final epoch, the validation accuracy being the performance score on the trained model using the validation set and the test accuracy is the performance of the trained models on a separate data set that was not used during training. For task A1, the trained CNN performed better than its SVM equivalent and can be seen that the training accuracy approaches 100% while both the validation accuracy and testing accuracy are around 5% lower.

However, with A2 the SVM performed marginally better than the CNN model which shows that an algorithmic model of less complexity can perform equally as well as a more complicated model, provided it is chosen for a specific problem. Of all the tasks, A2 resulted in models with the least accurate performance scores. This is down to the problem of classifying smiles narrowing down the training space to just the mouth area. Working with real life photos of people brings significantly more diversity in the features as well as many photos having the faces shown at an angle. Had they all been showing the front profile like all the faces in the cartoon set, the models would have a better time learning the data.

For B1, the trained CNN performed significantly better than its SVM equivalent, which comes down to the content of the datasets. With cartoon faces there is less diversity in it's features and together with a larger training set available to train on, 10 000 vs 5000, allows the CNN to identify corresponding relationships for the face shape easily.

For B2, the issue of the sunglasses adding noise to the training dataset was resolved and the CNN performed well, achieving almost 100% training accuracy and 99% on the validation dataset, meaning for none sunglasses faces, it can be very confident in identifying eye colour. However when the testing data set was used which included sunglasses faces, the accuracy naturally went down. The model likely concludes

that the face has black eyes due to the vast majority of the sunglasses being black and since there are 5 different eye colours, the likelihood it is correct is 20% so if similar to the provided data set where around 20% of the faces have sunglasses, then a testing accuracy of 88% is considered a suitable result.

6. CONCLUSION

Five machine learning models were developed with all achieving respectable performances for tackling both binary and multiclass problems, working on real life celebrity images as well as cartoon images. It can be seen that classification tasks that focus on a particular area of the face such as the mouth or eyes incorporate extra difficulties for the models, leading to lower accuracies compared to tasks that involve the larger spectrum of the face such as gender and face shape. It is also seen that the models performed better overall on the tasks involving the cartoon faces due to the data having less noise, being more uniform allowing relationships to be more easily learnt. Furthermore, despite the CNN models performing better than SVM models, the SVM models may be better solutions due to almost performing equally as good but at a fraction of the training costs required for traing and hyperparameter optimisation.

An element of the training process that could be improved is the use of the dlib landmark detector, which took the majority of the time to carry out feature extraction due to a nested for loop that runs 68 times for each image. For Tasks A2 and B2 where the landmark detector is used to crop to a location of the face, the detector will still extract all the features of the face. By only extracting the mouth or eye error coordinates rather than all 68 points the processing time would be reduced considerably.

Working with convolutional neural networks showed that they are very complex models that require a large amount of computational power to optimise. I attempted to use grid search processes to determine certain hyperparameters however this was carried out on a large range of values. An example is the number of neurons in the hidden layers, where I tested 512, 256, 128, 64 neurons and found 512 and 256 gave the best results. With more time and computational power, I could narrow down the range to find an even better number of neurons between these two values to improve the model.

7. REFERENCES

[1] Guodong Guo, Stan Li, and Kapluk Chan, "Face recognition by support vector machines," *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, 02 1970.

[2] Y. Yang and Y. Sun, "Facial expression recognition based on arousal-valence emotion model and deep learning method," in *2017 International Conference on Computer Technology, Electronics and Communication (ICCTEC)*, 2017, pp. 59–62.

[3] R. Ranjan, S. Sankaranarayanan, C. D. Castillo, and R. Chellappa, "An all-in-one convolutional neural network for face analysis," in *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*, 2017, pp. 17–24.

[4] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua, "A convolutional neural network cascade for face detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[5] Omar Faruqe and Md. Al Hasan, "Face recognition using pca and svm," 09 2009, pp. 97 – 101.

[6] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1867–1874.

[7] Zhenyue Qin and Dongwoo Kim, "Rethinking softmax with cross-entropy: Neural network classifier as mutual information estimator," 11 2019.

[8] Amit Dhomne, Ranjit Kumar, and Vijay Bhan, "Gender recognition through face using deep learning," *Procedia Computer Science*, vol. 132, pp. 2 – 10, 2018, International Conference on Computational Intelligence and Data Science.

[9] W. S. Ahmed and A. a. A. Karim, "The impact of filter size and number of filters on classification accuracy in cnn," in *2020 International Conference on Computer Science and Software Engineering (CSASE)*, 2020, pp. 88–93.

[10] Hidenori Ide and Takio Kurita, "Improvement of learning for cnn with relu activation by sparse regularization," 05 2017, pp. 2684–2691.

8. APPENDIX

8.1. Task A1 Grid Search Results

```
Best: 0.888499 using {'batch_size': 60, 'epochs': 20}
0.842740 (0.061821) with: {'batch_size': 40, 'epochs': 10}
0.869505 (0.016017) with: {'batch_size': 40, 'epochs': 20}
0.863256 (0.022725) with: {'batch_size': 40, 'epochs': 30}
0.835265 (0.046097) with: {'batch_size': 40, 'epochs': 40}
0.671994 (0.165727) with: {'batch_size': 60, 'epochs': 10}
0.888499 (0.004333) with: {'batch_size': 60, 'epochs': 20}
0.846737 (0.063665) with: {'batch_size': 60, 'epochs': 30}
0.881506 (0.018216) with: {'batch_size': 60, 'epochs': 40}
0.624553 (0.191890) with: {'batch_size': 80, 'epochs': 10}
0.876253 (0.009466) with: {'batch_size': 80, 'epochs': 20}
0.869258 (0.022217) with: {'batch_size': 80, 'epochs': 30}
0.877498 (0.009939) with: {'batch_size': 80, 'epochs': 40}
```

(a) Task A1 grid search results for Epoch/batch size optimisation

```
Best: 0.899997 using {'optimizer': 'Nadam'}
0.889248 (0.020411) with: {'optimizer': 'SGD'}
0.873995 (0.013677) with: {'optimizer': 'RMSprop'}
0.863001 (0.015966) with: {'optimizer': 'Adagrad'}
0.724003 (0.008921) with: {'optimizer': 'Adadelta'}
0.889996 (0.011594) with: {'optimizer': 'Adam'}
0.894247 (0.022592) with: {'optimizer': 'Adamax'}
0.899997 (0.007737) with: {'optimizer': 'Nadam'}
```

(b) Task A1 grid search results for optimisation algorithm optimisation

```
Best: 0.898246 using {'init_mode': 'he_uniform'}
0.895003 (0.010377) with: {'init_mode': 'uniform'}
0.886256 (0.018667) with: {'init_mode': 'lecun_uniform'}
0.897244 (0.017695) with: {'init_mode': 'normal'}
0.537487 (0.069247) with: {'init_mode': 'zero'}
0.897746 (0.015595) with: {'init_mode': 'glorot_normal'}
0.893497 (0.018703) with: {'init_mode': 'glorot_uniform'}
0.875255 (0.017474) with: {'init_mode': 'he_normal'}
0.898246 (0.012406) with: {'init_mode': 'he_uniform'}
```

(c) Task A1 grid search results for network weight initialisation optimisation

```
Best: 0.887248 using {'activation': 'softsign'}
0.788751 (0.023586) with: {'activation': 'softmax'}
0.800244 (0.020090) with: {'activation': 'softplus'}
0.887248 (0.005853) with: {'activation': 'softsign'}
0.885748 (0.026611) with: {'activation': 'relu'}
0.852247 (0.019129) with: {'activation': 'tanh'}
0.758995 (0.013775) with: {'activation': 'sigmoid'}
0.784747 (0.010500) with: {'activation': 'hard_sigmoid'}
0.886500 (0.003152) with: {'activation': 'linear'}
```

(d) Task A1 grid search results for neuron activation function optimisation

```
Best: 0.903001 using {'dropout_rate': 0.3, 'weight_constraint': 3}
0.888249 (0.008025) with: {'dropout_rate': 0.1, 'weight_constraint': 2}
0.890257 (0.019072) with: {'dropout_rate': 0.1, 'weight_constraint': 3}
0.887494 (0.025950) with: {'dropout_rate': 0.1, 'weight_constraint': 4}
0.889748 (0.009883) with: {'dropout_rate': 0.2, 'weight_constraint': 2}
0.898249 (0.017518) with: {'dropout_rate': 0.2, 'weight_constraint': 3}
0.886244 (0.023567) with: {'dropout_rate': 0.2, 'weight_constraint': 4}
0.897249 (0.004322) with: {'dropout_rate': 0.3, 'weight_constraint': 2}
0.903001 (0.006111) with: {'dropout_rate': 0.3, 'weight_constraint': 3}
0.899497 (0.009657) with: {'dropout_rate': 0.3, 'weight_constraint': 4}
```

(e) Task A1 grid search results for dropout regularisation optimisation

```
Best: 0.905747 using {'neurons': 128}
0.884245 (0.015970) with: {'neurons': 256}
0.905747 (0.008021) with: {'neurons': 128}
0.901249 (0.003393) with: {'neurons': 64}
0.897497 (0.014569) with: {'neurons': 32}
0.896251 (0.007744) with: {'neurons': 16}
```

(f) Task A1 grid search results for neurons per layer optimisation

Fig. 10: Results for grid search carried out on Task A1's Neural Network

When optimising the neurons per layer, using a simple grid search I went through a number of options for the first layer and the second layer would have half the neurons in the first layer. So in this optimised case, the first layer had 128 neurons and the second has 64.

8.2. Task B1 Grid Search Results

```
Best: 0.988714 using {'optimizer': 'RMSprop'}
0.981571 (0.001952) with: {'optimizer': 'SGD'}
0.988714 (0.001416) with: {'optimizer': 'RMSprop'}
0.882429 (0.003234) with: {'optimizer': 'Adagrad'}
0.230289 (0.031753) with: {'optimizer': 'Adadelta'}
0.986142 (0.006146) with: {'optimizer': 'Adam'}
0.981857 (0.000884) with: {'optimizer': 'Adamax'}
0.959430 (0.022749) with: {'optimizer': 'Nadam'}
```

(a) Task B1 grid search results for optimisation algorithm optimisation

```
Best: 0.988715 using {'init_mode': 'he_uniform'}
0.979572 (0.007866) with: {'init_mode': 'uniform'}
0.981572 (0.005153) with: {'init_mode': 'lecun_uniform'}
0.975288 (0.012852) with: {'init_mode': 'normal'}
0.203857 (0.002867) with: {'init_mode': 'zero'}
0.907735 (0.103926) with: {'init_mode': 'glorot_normal'}
0.967857 (0.005764) with: {'init_mode': 'glorot_uniform'}
0.975861 (0.019186) with: {'init_mode': 'he_normal'}
0.988715 (0.002136) with: {'init_mode': 'he_uniform'}
```

(b) Task B1 grid search results for network weight initialisation optimisation

```
Best: 0.973143 using {'activation': 'linear'}
0.895574 (0.013733) with: {'activation': 'softmax'}
0.933001 (0.006288) with: {'activation': 'softplus'}
0.708866 (0.122443) with: {'activation': 'softsign'}
0.776685 (0.262031) with: {'activation': 'relu'}
0.378663 (0.254852) with: {'activation': 'tanh'}
0.374853 (0.027751) with: {'activation': 'sigmoid'}
0.209286 (0.012344) with: {'activation': 'hard_sigmoid'}
0.973143 (0.002018) with: {'activation': 'linear'}
```

(c) Task B1 grid search results for neuron activation function optimisation

```
Best: 0.989572 using {'dropout_rate': 0.2, 'weight_constraint': 4}
0.940994 (0.063449) with: {'dropout_rate': 0.1, 'weight_constraint': 2}
0.986571 (0.002277) with: {'dropout_rate': 0.1, 'weight_constraint': 3}
0.869698 (0.168855) with: {'dropout_rate': 0.1, 'weight_constraint': 4}
0.922019 (0.095443) with: {'dropout_rate': 0.2, 'weight_constraint': 2}
0.983285 (0.005157) with: {'dropout_rate': 0.2, 'weight_constraint': 3}
0.989572 (0.002380) with: {'dropout_rate': 0.2, 'weight_constraint': 4}
0.977713 (0.011184) with: {'dropout_rate': 0.3, 'weight_constraint': 2}
0.981142 (0.006484) with: {'dropout_rate': 0.3, 'weight_constraint': 3}
0.984428 (0.007876) with: {'dropout_rate': 0.3, 'weight_constraint': 4}
```

(d) Task B1 grid search results for dropout regularisation optimisation

```
Best: 0.990285 using {'neurons': 512}
0.990285 (0.003710) with: {'neurons': 512}
0.965284 (0.026403) with: {'neurons': 256}
0.628133 (0.322124) with: {'neurons': 128}
0.635848 (0.308642) with: {'neurons': 64}
```

(e) Task B1 grid search results for neurons per layer optimisation

Fig. 11: Results for grid search carried out on Task A1's Neural Network

When optimising the neurons per layer, using a simple grid search I went through a number of options for the first layer and the second layer would have half the neurons in the first layer. So in this optimised case, the first layer had 512 neurons and the second has 256.

8.3. Running main.py

```
Training model for A1:

Begin extracting facial landmarks
Finished extracting facial landmarks
Loaded Neural Network Model
 Accuracy: 95.00%
 Training model for A2:

 Begin processing faces
 Finished processing faces
 Loaded Neural Network Model
 Accuracy: 87.08%
 Training model for B1:

 Begin Preprocessing faces
 Finished preprocessing faces
 Loaded Neural Network Model
 Accuracy: 99.53%
 Training model for B2:

 Begin Preprocessing faces
 Finished preprocessing faces
 Loaded Neural Network Model
 Accuracy: 87.98%
 Model Test Acc
 TA1_NN:95.00;
 TA2_NN:87.08;
 TB1_NN:99.53;
 TB2_NN:87.98;
```

Fig. 12: Output log from running main.py with testing=True on provided datasets. Takes around 40 minutes to run

```
Training model for A1:

Begin extracting facial landmarks
Finished extracting facial landmarks
0.9977499842643738
Accuracy: 94.60%
Begin extracting facial landmarks
Finished extracting facial landmarks
Training Accuracy: 0.9393333333333334
Testing Accuracy: 0.9164345403899722
Training model for A2:

Begin processing faces
Finished processing faces
Accuracy: 86.25%
Begin processing faces
Finished processing faces
Training Accuracy: 0.883492252681764
Testing Accuracy: 0.8791666666666667
Training model for B1:

Begin Preprocessing faces
Finished preprocessing faces
Accuracy: 99.47%
Begin Preprocessing faces
Finished preprocessing faces
Training Accuracy: 0.9222454672245467
Testing Accuracy: 0.9152439024390244
Training model for B2:


Begin Preprocessing faces
Finished preprocessing faces
Accuracy: 97.66%
Model   Train Acc   Test Acc
TA1_NN:  99.77,     94.60;
TA1_SVM:   93.93,     91.64;
TA2_NN:    96.13,     86.25;
TA2_SVM:   88.35,     87.92;
TB1_NN:    99.11,     99.47;
TB1_SVM:   92.22,     91.52;
TB2_NN:    98.54,     97.66;
```

Fig. 13: Output log from running main.py with testing=False on provided datasets. Takes about 1 hour to run