Utilising Convolutional Neural Networks for Facial Feature Classification

15000514
University College London
https://github.com/hasifuzir/AMLSassignment_15000514
https://www.dropbox.com/sh/sjl3mcfn1lndedl/AAC0qVq02C_p90A_fUAbVRDEa?dl=0

Abstract

Facial features are complex structures which give importance in the identification of identity, emotion, age, and other social information. As such, various work has been done to introduce the ability to computers, with neural networks achieving exceptional success.

In this paper, we compare a Convolutional Neural Network, Multilayer Perceptron and Inception v3 in 5 image classification tasks; Binary (emotion, age, glasses, human) and Multiclass (hair color). Several experiments were also done with the CNN to identify the best hyperparameter and other parameter methods.

Based on tests, Inception v3 performed the best, although its improvements were marginal compared to the CNN, which required less computational power and training time. All models performed better on binary classification compared to multiclass, mainly due to the lack of training data per class. In the CNN, experiments showed that automatically calculating class weights based on training data, using a 3X3 filter size, increasing FC layer size, only using image augmentation as a regulariser, a cross entropy loss and a Sigmoid activation for the FC layer lead to the best performance for image classification tasks.

1. Introduction

In this paper, 5 image classification tasks based on facial features are performed using several machine learning models. The classification tasks consist of 4 binary problems and 1 multiclass problem. The binary tasks include; Emotion recognition (smiling or non-smiling), Age identification (young or old), Glasses detection (with or without glasses) and Human detection (real human subjects (CelebA) or cartoon subjects (Cartoon set)). The multiclass task involves classification of images between 6 different hair classes; Blond, Ginger, Grey, Brown, Black or Bald (no hair). The machine learning models will then be trained to predict the correct class of testing images.

1.1. Dataset

The dataset consists of 5000 labelled images which contain subsets of two other datasets; CelebFaces Attributes Dataset (CelebA) and Cartoon Set. CelebA contains images of faces of various celebrities. Cartoon Set contains 2D human cartoon avatars. In addition, the dataset also contains noise images without any faces e.g. images of backgrounds of nature or a single colour. For images with faces, in addition to the variety in basic facial features, may also include accessories, specifically glasses. The images are formatted in PNG and each has a resolution of 256 X 256 pixels. Images with faces have a bit depth of 24 while noise images have a bit depth ranging from 8 to 32.

1.2. Classes

The images are labelled via a CSV file, attribute_list.csv, containing each image's class labels according to several attributes. The dataset is heavily imbalanced for smiling, age, glasses and hair classification, as the classes are not of equal proportion. Only for the human attribute is the dataset near balance with a 12.38% percentage difference.

1.3. Preprocessing

In all preprocess methods, Python libraries were used.

1.3.1 Noise removal

Although noise can be real-world outliers for which there are methods for data integration, in this dataset, noise is unwanted [1]. Noise can affect the performance, as neural networks are susceptible to false-positives [2].

For noise removal, the noise images in the dataset are discarded from the use of training, validation, and testing. Using the Pandas library, the attribute list CSV file was read as a dataframe. Then the dataframe was filtered from all noise images by removing rows where every column (class) had a label of -1. Noise images had a value of -1 across all columns. The new dataframe is then saved and used as a reference for training, validation, and testing models, resulting in a dataframe containing 4565 rows.

1.3.2 Training-Validation-Testing split

The dataset is split into training, validation, and testing sets. In this paper, the testing set is used as an unbiased evaluation of model performance after training while the validation set is used for tuning hyperparameters [3].

In the Implementation section, the Keras method flow_from_dataframe is used to input a Pandas dataframe and image directory to generate data for training. This requires the dataset to be split into two folders, i.e. training and testing. A function creates the folders and randomly copies images from the dataset into the new folders according to a training-testing split. For copying images, the OpenCV library methods cv2.imread and cv2.imwrite were used. Scitkit-learn's train_test_split method was used to split the dataframe into random train and test subsets.

A training-testing split of 80/20 was used, producing 3652 training images and 913 testing images. This split produced the best performance, producing the lowest

average loss of 0.0740. Further details are presented in the Determining Training-Testing Ratio section of the Supplementary. For training-validation, a 75/25 split was used instead to maximise performance as shown in the Determining Training-Validation section of the Supplementary and to ensure that the training, validation, and testing split is 60/20/20 for an equal validation and test set. For the validation set, the Keras ImageDataGenerator built-in validation_split parameter was used, which allows a validation set to be created during image augmentation.

1.3.3 Image augmentation

Image augmentation is a useful method to inflate the size of a dataset and improve image classification without needing to source additional images. Research has proved the effectiveness of traditional and advanced data augmentation techniques [4]. The Keras Image Preprocessing ImageDataGenerator method was used to generate batches of augmented images in real-time. For all sets, images were converted from an RGB value between 0-255 to a 0-1 range through a 1/255 scaling factor. This avoids the need for massive processing power.

Figure 1. Image augmentations of image 1006.png. From left: Original, shear, zoom, horizontal flip.

For the training and validation set images, three different image transformations were used; Shearing (skews the image), zooms (magnifies features) and a horizontal flip, shown in Figure 1. These transformations were applied randomly to images. These augmentations produced a more varied dataset to increase model performance.

2. Proposed Algorithms

2.1. Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of Deep Neural Network (DNN), inspired by the biological process of the human visual cortex [5]. It is useful for image classification as the architecture recognises that image features are dispersed and leverages convolutions way to extract features from multiple locations without treating each pixel as a separate input, as demonstrated by the pioneering LeNet5 [6]. This makes CNN excellent for the image classification problem in this paper.

A CNN consists of Convolutional, Non-Linear, Pooling, and Fully Connected (FC) layers [7]. The architecture of the CNN implemented is summarised in Figure 2.

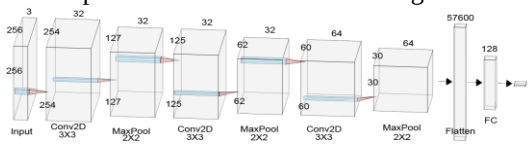

Figure 2. The CNN architecture with 256X256 RGB images input to 3 convolutional layers, 3 max pooling layers, and 2 FC layers.

First, an RGB image with dimensions 256X256X3 is used as input. The image has a depth of 3 representing the 3 colour channels. The image is essentially three 256X256 2D arrays. The convolutional layer applies convolutional filters (kernels) on inputs. A 3X3 filter window is moved across the image. At every position, the convolution operation is applied - the 3X3 window's values are multiplied by the values of the image covered by the filter. The convolutional layer filters the image and "picks up" details of features from the images, resulting in effective pattern checking. Each filter has an associated weight which changes during training and outputs a high value when the same pattern is recognised. Mathematically, the equation for the convolution is as per Equation 1.

$$(f * h) = \sum_{k} \sum_{l} f(k,l)h(i-k, j-l) \qquad [1]$$

Where f is the image, h is the filter.

Three convolutional layers were used, with the first 2 layers using 32 kernels and the last layer using 64 kernels instead. The stride, or distance the window moves is set at 1 to avoid losing any detail. No padding was used in convolutions. As neural networks model human neural activity, activation functions are used to model the firing of neurons. In all convolutional layers, the Rectified Linear Unit (ReLU) activation is used.

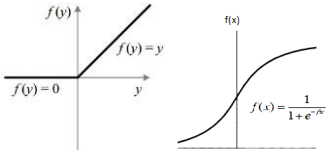

Figure 3. Graphs of the ReLU and Sigmoid activation functions.

ReLU is used because it is computationally inexpensive, require less training time (faster convergence), and performs better than other activations such as logistic sigmoid [8] [9]. ReLU removes negative numbers from the output and passes positive values as shown in Equation 2.

$$f(x) = \{0 \; for \; x < 0, \; x \; for \; x \geq 0\} \qquad [2]$$

Another major benefit to the ReLU function is that it avoids gradient vanishing since the gradient is constant at values of x above 0, which is 1. During backpropagation, the gradients of error loss are calculated with respect to the weights and every layer reduces the gradient exponentially when other functions are used [10]. ReLU avoids this problem as its derivative is either 0 or 1. Consequently, ReLU also avoids an exploding gradient. However, a dying ReLU problem may arise. If too many activations are 0, no gradients are backpropagated and the neurons may become stuck and die off, as weights cannot be altered [7].

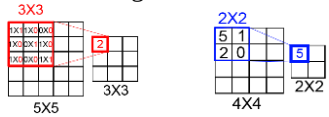

Figure 4. Visualisation of the convolution operation using a 3X3 filter and a pooling operation using a 2X2 pool with a stride of 2.

The pooling layer applies a pooling operation over an input. Max pooling is the pooling function used. The kernel window is moved over the entire input and only takes the

largest value from the image covered by the kernel window. Figure 4 depicts an example of the max pooling operation. Pooling reduces feature map (image) size and introduces invariance by obscuring the location of features extracted by the convolution layer. This results in a network that is computationally inexpensive and insensitive to a feature's exact location.

2X2 pools with a stride of 2 are used. A greater stride will increase invariance. The first two max pooling layers use 32 kernels and the last uses 64. The output of the third pooling layer is then flattened into a feature vector to be used by the dense FC layers to perform classification [7].

The dense layers' neurons are connected to each other and other neurons in the previous layer. The first FC layer consists of 128 neurons which are activated using ReLU and the output is passed to another FC layer which either uses 1 neuron with a Sigmoid activation for binary classification or multiple neurons with a Softmax activation for multiclass classifications.

In addition to introducing nonlinearity (which benefits learning), the Sigmoid tends to bring activations to either side of its curve as shown in Figure 3, as values near the middle of the curve relatively steep. Equations 3 and 4 show the Sigmoid function equation and its derivative.

$$f(x) = \frac{1}{1 + e^{-x}} \quad [3]$$

$$f'(x) = f(x)(1 - f(x)) \quad [4]$$

The output of this function will be in the range of 0-1, preventing a blow-up. Its output is suitable for binary predictions for clear distinctions in predictions. Its derivative is computationally inexpensive. However, it can lead to vanishing gradients. As the curve tapers out at each end, the gradients are small, which massively reduces loss gradients during backpropagation [11].

For multiclass classification, a Softmax function is used instead. The function acts like a Sigmoid by ensuring outputs are bound between a value of 0 and 1 but it divides each output so that the total sum of all outputs is equal to 1, i.e., probabilities. This is shown in Equation 5. The output is equivalent to the categorical probability distribution, suitable for multiclass predictions [7].

$$f(x) = \frac{e^{x_k}}{\sum_k e^{x_k}} \quad [5]$$

Where k is the number of classes.

For training, the CNN performs backpropagation to calculate partial derivatives, allowing it to associate features with its classes through supervised learning. For the loss function, which measures performance, cross-entropy was used. Cross-entropy loss is given in Equation 6 and simplifies to Equation 7 for binary classification.

$$loss = -\sum_{c=1}^{N} y \times log(p) \quad [6]$$

Where y is the class ground truth, p is predicted probability that observation is of class c and N is the total classes.

$$loss = -(y \times log(p)) + (1 - y)log(1 - p)) \quad [7]$$

Where y is the class ground truth, p is the predicted probability.

Cross entropy heavily penalises predictions of high confidence and value, which are incorrect. Cross-entropy is more preferred for classification tasks as Mean Squared Error can be badly defined for a distinct set of classes [7].

For the optimiser, which allows the model to update weights and reduce loss, Adaptive Moment Estimation (Adam) is used. It uses fractions of previous gradients, allowing faster convergence, reduced oscillation, and computational efficiency [12]. It uses moments (gradient of past steps) of first and second order. The learning rates are adapted on the first moment (mean) and average of the second moments of the gradients.

2.2. Multilayer Perceptron

A Multilayer Perceptron (MLP) is a type of Artificial Neural Network consisting of a network of neurons called perceptrons. A perceptron is as a linear binary classifier [13]. Based on several inputs, its weights, and bias, a single output is calculated as shown in Equation 8.

$$y = \alpha(\sum_{i=1}^{n} w_i x_i + b) \quad [8]$$

Where y is the output, α is the non-linear activation function, x is weight, b is bias, and x is input.

The activation function is acts as a threshold for activation and output strength. Non-linearity allows MLPs to model nearly any arbitrary complex function. Without it, a neural network would act as a linear regressor [13].

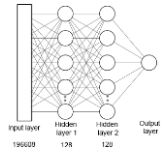

Figure 5. The architecture of the MLP with 2 hidden layers.

Figure 5 depicts the architecture of the MLP used. The first layer is an input layer with a size of 196608 (image flattened into a vector). This is input to the two hidden layers, both containing 128 neurons with a ReLU activation. Although the Universal Approximation Theorem states a single hidden layer is enough, adding another allows better training and performance [14] [15].

Based on the task, the output layer contains 1 neuron with a Sigmoid activation or multiple neurons with a Softmax activation. The network was trained using cross entropy loss and Adam optimizer. Benefits and justification to these choices have been discussed earlier.

2.3. Inception v3

Inception v3 is the third iteration of the Inception deep convolutional architecture. GoogLeNet (Inception v1) introduced Inception modules; convolutions on inputs using different filter sizes and max pooling which are concatenated. In image classification, salient parts in the image can vary. Different kernel sizes allow different distribution of features to be accommodated. Inception modules allow NNs to get wider, as deeper networks are

prone to overfit and are computationally expensive [16]. Inception v3 contains 11 inception modules in 42 layers.

Inception v3 implemented factorising convolutions, grid size reduction, and aggressive regularisation. Convolutions were factorised into smaller sizes and asymmetric convolutions. Factorisation reduces the number of parameters without a decrease in performance. Feature map downsizing is typically done via max pooling. The model proposed a new method by concatenating a convolutional and max pooling layer, proving to be less expensive. Label smoothing regularisation was used to prevent one logit from becoming much larger and acted as a dropout [17].

The Inception architecture rethought how CNNs were built, going wider instead of deeper. Inception v3 contained fewer parameters than other popular models such as AlexNet and VGGNet and achieved better performance. It managed a lower Top-5 error (4.2%) compared to VGGNet, PreLU-Net and Inception v2 in the ImageNet Large Scale Visual Recognition Competition (ILSVRC).

3. Implementation

All models were implemented in Python and use the Keras library running on top of the TensorFlow library.

3.1. Convolutional Neural Network

A Keras Sequential object was created which will allow stacks of layers for the CNN [18]. A Conv2D layer was then for a convolutional layer. The input shape dimensions were set at the dataset image dimensions of 256X256, with a depth of 3 to indicate the image RGB colour channels. A total of 32 3X3 convolutional filters were used.

A small 3X3 kernel size was chosen with the assumption that features are highly local, allowing accurate detection of subtle features. A lower number of weights due to the smaller receptive field requires less computationally power. A larger output dimension due to a smaller filter ensures more information is available for later layers and improved combinatorics due to increased non-linearity [7] [19]. 32 filters were used to balance hardware limitations and number of weights for better training. The stride value was 1, to ensure filters would cover every part of the image for better accuracy. No padding was used as features were most likely to be centered [7]. A MaxPooling2D layer with a pool size of 2X2 and stride of 2 was added. A pool size of 2X2 will reduce the feature map by half to reduce the number of parameters and computation expense as well as introduce invariance. Another 2 Conv2D and MaxPooling2D layers were added with similar parameters. However, the final Conv2D layer uses 64 filters instead. As this was the final convolutional layer before the FC layers, the number of filters was doubled for better performance.

Then a Flatten layer was added to convert 3D feature maps into a 1D feature vector for classification. This is fed into a FC layer, a Dense layer with 128 neurons, to ensure good performance whilst avoiding hardware limitations.

For the output layer, a single unit Dense layer or a 7-unit Dense layer was used for the binary and multiclass classification respectively. Both used cross entropy loss, but the binary model used the binary_crossentropy parameter and the multiclass used categorical_crossentropy. Both models used the Adam optimizer with a learning rate of 0.00001. This allows convergence to occur within 60 epochs. A range of values was tested with higher rates unable to converge and lower rates taking more epochs to converge. The model was then compiled via the compile method and trained using the fit_generator method. fit_generator trains the model on generated images via batches and the batch size was set at 32, a value most often used. Epochs, the number of iterations of training to run, was set at 100. Steps taken was the size of the batches divided by its total. An early_stopping method was used for early stopping of training to prevent overfitting. For class_weights, which are the weights assigned to each class during training, several options were designed. As the dataset was imbalanced, it was important to use class weights to heavily notice classes with lower representation to avoid detrimental performance [20]. Although this may also be solved via oversampling, class weights were also an effective method. The chosen method was using a Scikit-learn method, class_weight to calculate the weights according to the distribution of images. Keras' predict_generator method performs predictions on the test dataset and the results saved as a Pandas dataframe. Using the method to_csv allows the results and its accuracy score obtained from Keras to be exported as a CSV file.

3.1.1 Training convergence and overfitting possibilities

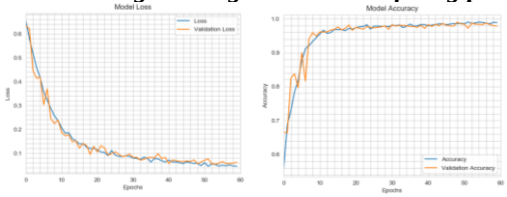

Figure 6. Training loss and accuracy against validation loss and accuracy using the CNN model for eyeglasses prediction.

Figure 6 depicts the learning curves for the model using the eyeglasses classification task. To prevent overfitting, early stopping was used. Early stopping prevents further training for the model and wasting power. First, the model was run without early stopping to overfit. In the Eyeglasses task, the model overfits after epoch 60 on average as the training loss begins to stagnate even though the training loss keeps decreasing. After implementing early stopping, the model converges on average at epoch 43, achieving accuracy of 98.98%. In Keras, using the early_stopping callback method, the patience parameter was set to 5, for further training of 5 additional epochs after the validation loss does not decrease.

Unlike binary classification, which took 60 epochs to converge on average, the multiclass task required 15 epochs to converge, and further training lead to overfitting.

3.2. Multilayer Perceptron

Using a Keras Sequential object, a Flatten layer was added to reduce the dimensions of the input image into a vector. Then, two Dense layers with 128 units and ReLU activation was added. Based on the task, the output Dense layer's units and activation were either 1 and Sigmoid or 7 and Softmax. Justification is the same as the CNN model. Like the CNN, the MLP was trained with a cross entropy loss optimised with a 0.00001-learning rate Adam. Compile, training, class weight and predict methods are as the CNN, with the only difference being that the maximum epoch was set at 200 as MLPs took longer to train.

3.3. Inception v3

Inception v3 is built into Keras and requires importing. For this model, Keras' functional Model API was used, as it allows more complex models to be created [21]. An InceptionV3 method was called with parameters imagenet for weights, an average pooling (to reduce parameters) and include_top being False. The top layers from literature are not needed as the model will be trained with a new output layer. A Dense layer was used as the output layer with its units and activations determined by the type of problem as described in the CNN model. A Model was created with the InceptionV3 model as the input and the Dense layer as output. The compile, training and prediction methods and parameters are as in CNN as both are Neural Networks. The only difference is that this model's train epoch was set at 50 because it achieved convergence faster than the CNN.

4. Experiments

A comparison of the models was done to assess performance differences. Several experiments were also conducted to assess the impact of hyperparameters and other factors on the performance of the CNN. The CNN was evaluated on the same task; Eyeglasses. This was chosen because, in testing, Eyeglasses was not the easiest (human) nor the hardest (hair color) task. All tests were run 5 times and relevant metrics averaged out. Additional tests are also presented in the Supplementary.

4.1. Comparison between models

The models compared are the CNN, MLP and Inception v3. Average loss, accuracy, F1, epoch duration, and convergence epoch was recorded and compared.

4.2. Filters

Filters function as feature extractors. Larger filters are preferable for larger features and smaller filters for more localised features. Several filter sizes were compared; 1X1, 3X3, 5X5, and 9X9. 1X1 filters are used for dimensionality reduction and add non-linearity. According to Yann LeCun, 1X1 convolutions and FC layers are similar [22].

4.3. Fully Connected Layer Size

The FC layer acts as the classifier with the size being the number of neurons in the layer. The model performance was compared with different FC layer sizes. There is no standard for this parameter and as such, is explored.

5. Results

5.1. Convolutional Neural Network

Task	Loss	Acc	F1	Top-2	Top-3
Emotion	0.242	0.887	0.942	-	-
Age	0.301	0.821	0.906	-	-
Eyeglasses	0.0453	0.989	0.997	-	-
Human	0.00012	0.998	0.999	-	-
Hair Colour	0.914	0.652	0.755	0.843	0.938

Table 1. CNN performance on 5 different classification tasks.

Table 1 depicts the performance of the CNN on the classification tasks. Training time was 9 seconds per epoch. For binary classification, the model performed best on the Human classification task, achieving accuracy and F1 scores of nearly 100%. An explanation for this is that the non-human images (Cartoon Set) are uniform in shape, have flat colours and are facing forwards. More complex features can be observed from the CelebA images. It is possible these distinct differences made classification easy.

For binary classification, the model performed worst on the Age task, with accuracy of 86.1%. Across both sets, differences between young and old subjects were very subtle, especially in the Cartoon Dataset, as cartoons lacked age-related features such as wrinkles. The model made more errors in old subjects, with an 18.2% error rate.

In the multiclass task, the model performed poorly with accuracy of only 65.2% and F1 score of 75.5%. However, the accuracy increases significantly in the Top-2 and Top-3 scores. Upon inspection of the prediction CSV and answer files, it was discovered that the noise labels heavily influenced the predictions. Many of the incorrect predictions had the noise class as a prediction, with the correct class being the second highest probability. As there were multiple classes, there was less training data per class, which led to relatively poorer generalisation in the model.

5.2. Comparison between models

Model	Loss	Acc	F1	Epoch duration	Conv
CNN	0.0453	0.989	0.997	9	43
MLP	0.328	0.853	0.877	13	105
Inception v3	0.0159	0.997	0.998	49	14

Table 2. Performance comparisons between CNN, MLP and Inception v3.

Inception v3 had the best performance in both accuracy and F1 scores. Although it can be argued that the performance improvement over CNN is marginal, with only a 0.8% improvement in accuracy and very marginal F1 improvement. Inception v3 required more computational power, taking more time to train. Although it achieved faster convergence, it took 49 seconds per epoch on average, making its overall training time longer than the CNN model (9s). The excellent performance of CNN and Inception v3 over MLP proves the superiority of newer Neural Network architectures over classical Machine Learning algorithms such as MLP in image

classification tasks. MLP took the longest to converge and yet had the worst performance.

5.3. Filters

Filter Size	Loss	Accuracy	F1
1X1	0.415	0.798	0.843
3X3	0.0453	0.989	0.997
5X5	0.275	0.9024	0.954
9X9	0.402	0.812	0.875

Table 3. Comparison of different kernel sizes.

Based on Table 3, the 3X3 filter produced the best results. The 1X1 filter only reduces the feature map and thus performs the worst as this would mean no proper feature extraction was performed. Although eyeglasses as a feature is not as a subtle or small as age or smiles, the smaller filter window had the best performance. A reason could be because the smaller filter was able to better pick up on what makes up the shape and colour of glasses, instead of a general shape, leading to better predictions.

5.4. Fully Connected Layer Size

Neurons	Loss	Accuracy	F1
32	0.375	0.842	0.727
64	0.426	0.805	0.785
128	0.0453	0.989	0.997
256	0.0475	0.991	0.998

Table 4. Comparison between FC layer size performance.

Utilising more neurons produced better results with 256 neurons having the best performance. However, the increase in accuracy and F1 score is only marginal compared to 128 neurons. Utilising more neurons uses more computational power and may not be more cost-effective. An interesting observation is that 32 neurons produced better accuracy than 64 neurons, but it had a worse F1 score. The imbalanced dataset may have affected the accuracy calculations and is further reason why F1 as a metric is important in unbalanced dataset problems.

6. Related Work

With the advent of Big Data, image datasets used for classification have grown significantly. This lead to a problem with noisy labels, especially for images scraped from the internet. Wu et al. proposed a CNN framework to embed a universal face representation in a light CNN containing a smaller number of parameters and semantic bootstrapping to relabel or remove noisy labels [24]. This resulted in a faster and smaller CNN. However, this model performed only marginally better or even worse to models such as GoogLeNet. Also, Semantic bootstrapping requires training on a "clean" dataset before being able to relabel and requires tuning to avoid incorrect labels. This model has potential for real-time recognition systems.

Humans are highly capable of one-shot learning, being able to distinguish objects given only one example, unlike DNNs, which require high amounts of labelled data. As such, Siamese Neural Networks have been developed to perform one-shot image recognition. Siamese networks contain two branches of CNNs to obtain two feature vectors. The two vectors are merged to obtain the distance between each point. A smaller distance indicates a highly similar image. Koch et al.'s approach to implementing one-shot learning was to train the model to predict whether the images in each branch were of the same class [25].

An advantage of this model is that very little training data was required. In a verification task, this model performed better than other one-shot learning models such as Affine and Hierarchical Networks but worse than Hierarchical Bayesian Program Learning. However, the model was trained on the Omniglot dataset, which consists of characters, and requires further development to one-shot learn images of humans. Another flaw is that it requires examples from every class to be used for comparison, which can quickly grow if more classes are added.

Although deeper neural networks can lead to better performance, they become more difficult to train due to the massive number of weights. Residual Networks such as the model presented by He et al. proposes building layers with residual mappings to learn residual functions from the input layers [26]. Residual mappings allow a deeper network to be created with equal or fewer parameter than its shallower counterparts. In the paper, a Residual network of 152 layers was 8 times deeper than a VGG network but used fewer parameters. The model is easier to optimise due to each residual block being optimised by the previous and achieved superior performance in image recognition, winning 1st place in the ImageNet 2015. Several variations have been presented such as ResNeXt and DenseNet [27] [28]. However, the model has 152 layers, which still required massive computational power, although it has relatively low convergence times. As such, many new networks only utilise the residual mapping idea introduced to reduce convergent times and increase network depth.

7. Conclusion

In conclusion, CNN and Inception v3 performed much better than MLP in the classification tasks with Inception v3 performing marginally better. However, Inception v3 is computationally more expensive and takes longer to train than CNN. The models performed better on binary classification tasks compared to the multiclass task, mainly due to the lack of training data per. For the CNN model, binary classification was best for the Human task and multiclass classification was best when equal class weights were used. Experiments showed that automatically calculating class weights, 3X3 filter sizes, increasing FC layer size, only using image augmentation as a regulariser, a cross entropy loss and a Sigmoid activation for the FC layer lead to the best performance.

For better statistical significance and accuracy, cross-validation and stratified k-fold methods could be implemented due to the small dataset size. Other networks could also be implemented to compare performance such as Inception-ResNet, ResNet, Squeeze-and-Excitation networks and Neural Architecture Search (NAS).

Supplementary

1. Dataset

1.1. Dataset Class Distribution

The images are labelled via a CSV file, attribute_list.csv, containing each image's class labels according to several attributes. The distributions of images in each class are illustrated according to Tables 1, 2, 3, 4 and 5.

There are 435 noise images, which is 8.7% of the entire dataset. Removal of noise would leave a total of 4565 usable images for the binary classification tasks. From the 4565 images, 663 contain label noise for the multiclass classification task. The label noise is represented as a N/A class in the hair attribute. Thus, there are technically only 4337 labelled images for multiclass classification.

Class	Total	Percentage
smiling	3634	79.61%
not smiling	931	20.39%

Table 1. Counts and percentages of the smiling attribute

Class	Total	Percentage
young	3614	79.17%
not young (old)	951	20.83%

Table 2. Counts and percentages of the age attribute.

Class	Total	Percentage
eyeglasses	1328	26.56%
no eyeglasses	3237	70.91%

Table 3. Counts and percentages of the glasses attribute.

Class	Total	Percentage
human	2000	43.81%
not human	2565	56.19%

Table 4. Counts and percentages of the human attribute.

Class	Total	Percentage
N/A	663	15.29%
Bald	88	2.03%
Blond	995	22.94%
Ginger	547	12.61%
Brown	943	21.74%
Black	788	18.17%
Grey	541	12.47%

Table 5. Counts and percentages of the hair attribute.

A more detailed presentation of class distributions is given in Tables 1, 2, 3, 4, and 5. Heavily imbalanced data may lead the model to only predict a single class as it will lead to the highest accuracy [29].

2. Preprocessing

2.1. Determining Training-Testing Ratio

Ratio	Avg loss
0.5	0.1152
0.4	0.0803
0.3	0.0791
0.2	0.0740
0.1	0.0779

Table 6. Training-testing ratios against average test loss

The goal in determining a suitable training-test ratio is to ensure that there is enough testing data to prevent high variance in model performance but enough training data to prevent high variance in parameter estimates.

Determination of the training-testing ratio is important to maximise the bias-variance tradeoff. This helps in preventing overfitting due to high variance and underfitting due to high bias [30]. B. Neal et al. discovered that the variance caused by the loss decreases with more hidden units and increases with layers [31]. However, variance remained the same due to sampling.

Using the CNN performed on a simple binary classification task described in the main paper with all parameters the equal, the training-test ratio was modified, and the test loss was recorded. Each ratio was used to run 5 separate training sessions and the average loss was calculated.

It was found that the model on average, performed the best with a 0.2 ratio, at an average loss of 0.074. However, the differences in loss between 0.1, 0.2 and 0.3 and even 0.4 may be too small for a definitive answer.

2.2. Determining Training-Validation Ratio

Ratio	Avg loss
0.5	0.0954
0.4	0.0784
0.3	0.0778
0.2	0.0576
0.1	0.0670

Table 7. Training-validation ratios against average test loss

The goal in determining a suitable training-test ratio is to ensure that there is enough testing data to prevent high variance in model performance but enough training data to prevent high variance in parameter estimates.

Using the CNN described in the main paper, the training-validation ratio was modified, and the test loss was recorded. Each ratio was used to run 5 separate training sessions and the average loss was calculated.

The 0.2 ratio provided the best performance with an average loss of 0.0576.

3. Proposed Algorithms

3.1. CNN Tensor Output Sizes

The size of the output tensor of a convolution layer is depicted in Equation 1.

$$O = \frac{I - K + 2P}{S} + 1 \quad [1]$$

Where O is the size of the output, I is the size of the input, K is the size of the filter, P is padding, S is stride.

The output size of every pooling layer is given in Equation 2.

$$O = \frac{I - P}{S} + 1 \quad [2]$$

Where O is the size of the output, I is the size of the input, P is the pool (kernel) size, S is stride.

3.2. Backpropagation

Backpropagation is the supervised learning training process in neural networks to calculate the gradient to adjust the value of weights or filters to minimise a given loss function. The result desired would be a model that has

a low loss and high accuracy in solving a certain task. This involves 4 distinct subprocesses; Forward pass, loss function, backward pass and weight update [32].

First, in the forward pass, an input is passed through the whole network. All weights or filter values initially have random values. No conclusion can be drawn from this. Then, the loss function is applied. A loss function is a method to evaluate the performance of an algorithm. A higher value would indicate poor performance and conversely, good performance [33]. A CNN can be summarised as a function in Equation 3.

$$y = f(x, w) \quad [3]$$

Where y is the output, x is the input, w is the network weights.

To reduce the loss, a minimisation problem must be solved. The derivative (gradient) of the loss function with respect to the weights and inputs must be calculated and taken. Thus, backward pass is used. It is assumed that the gradient of the lass with respect to the output is obtained from previous layers. Through the usage of the chain rule of derivates, the gradient of the loss with respect to the weights and inputs (outputs from the previous layer) is calculated. Any change in the weight associated with a filter will affect the output. These changes contribute to the final loss. This is shown in Equations 4 and 5.

$$\frac{\partial E}{\partial y_{n-1}} = \frac{\partial E}{\partial y_n} \times \frac{\partial y(w, y_{n-1})}{\partial y_{n-1}} \quad [4]$$

$$\frac{\partial E}{\partial w_n} = \frac{\partial E}{\partial y_n} \times \frac{\partial y(w, y_{n-1})}{\partial w_n} \quad [5]$$

Where E is the loss, w is weight, y is output, n is current layer index

This determines the weights contributing most to the loss. Then, weight update occurs. Weights are updated to reduce the loss function. The entire process is then iterated for a predetermined number of steps.

3.3. Momentum

In Momentum, instead of just using the current step's gradient, momentum accumulates gradients of past steps.

3.4. RMSProp

RMSProp is an adaptive learning rate method [34]. It uses a moving average of squared gradients to normalise itself to balance the step size. The step will be smaller for large gradients to avoid explosions, or larger for small gradients to avoid vanishing of the gradient.

3.5. Inception Modules

Inception modules perform convolutions on an input using 3 different filter sizes; 1X1, 3X3 and 5X5. Max pooling is also performed on the input and the results are concatenated as a single output. Figure 1 depicts an implementation of the module in GoogLeNet using dimension reductions [16].

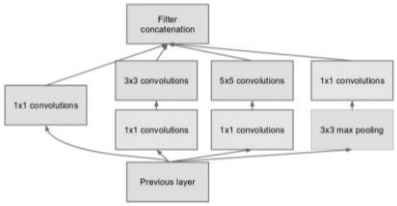

Figure 1. Inception module with dimension reductions.

Dimension reductions are done using 1X1 convolutions before the filters and after the max pooling to further reduce the input feature size and reduce computational expense.

4. Results

4.1. Accuracy and F1 as Metrics

Accuracy is an intuitive metric for performance, however as demonstrated earlier in the paper, the dataset given is heavily imbalanced for most of the tasks and may skew results. As such, F1 is also introduced as a metric.

F1 is a metric that is the harmonic mean of precision and recall. This takes all classes into account as a weighted average. The formula for F1 is shown in Equation 6.

$$F1 = 2X\frac{Precision\ X\ Recall}{Precision\ +\ Recall} \quad [6]$$

Where Precision is ratio of True Positives and Recall is ratio of True Negatives.

F1 scores were calculated based on the confusion matrix generated after model training.

4.2. Convolutional Neural Network

The learning graphs for the CNN model from the other classification tasks are presented in Figure 2, 3,4 and 5.

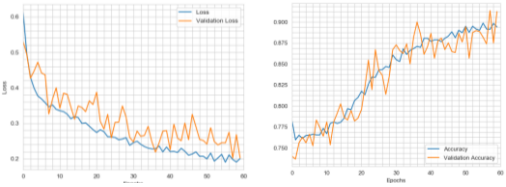

Figure 2. Emotion classification task learning graphs.

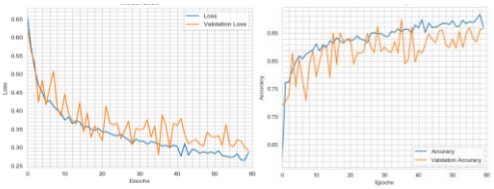

Figure 3. Age classification task learning graphs.

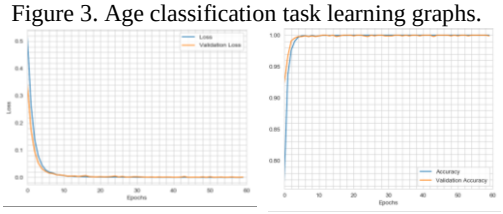

Figure 4. Human classification task learning graphs.

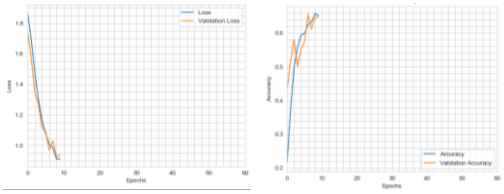

Figure 5. Hair color classification task learning graphs.

5. Additional Tests and Clarifications

5.1. Class Weights

As presented in the Dataset section, the dataset is heavily imbalanced for most of the tasks. An unbalanced dataset may lead to the model bias towards a single class to achieve better accuracy. As such, several strategies to assign class weights were compared. The multiclass classification task was also included in the experiment because the multiclass problem had noise labels which required different strategies to resolve.

For binary classifications, Auto Binary uses Scikit-learn's class_weights method to automatically balance the classes using calculated class weights based on the training dataset distribution. Equal Binary means that the same class weight, 1, is used across all classes. Dataset Binary is the class weights manually calculated based on the entire dataset distribution, not just the training dataset.

Auto, Equal and Dataset class weight methods are the same for the multiclass classification task. However, the hair color problem contains noise labels which are addressed in two additional methods; Dataset NaN suppress uses the dataset distribution but assigns 0 to the noise class for the model to ignore the class in training. NaN Suppress is based on the auto Scikit-learn method with the noise class also set to 0.

Class Weight	Loss	Accuracy	F1	Conv.
Auto Binary	0.0453	0.989	0.997	43
Equal Binary	0.0591	0.978	0.989	45
Dataset Binary	0.361	0.850	0.851	35
Auto Multi	0.914	0.652	0.755	16
Equal Multi	0.660	0.748	0.812	87
Dataset Multi	0.915	0.680	0.740	-
Dataset NaN Suppress	1.224	0.711	0.699	-
NaN Suppress	0.335	0.747	0.836	-

Table 8. Class weight methods comparison in performance.

In the binary classification task, the auto-balanced weights had the best performance and balancing the weights based on the entire dataset produced the worst results. As the dataset is unbalanced for eyeglasses, the result was expected. Since the training dataset is a subset of the entire dataset, it makes little sense to balance the weights based on the entire dataset as the randomised preprocessing methods may randomly select an even more unbalanced range of images.

For the multiclass classification task, using equal weights across classes produced the best result at the cost of convergence times. A possible explanation would be that balancing the weights based on class counts made it harder for the model to learn as there is very little data per class to learn on. Removal of the noise class via class weights, although increased performance, did not achieve convergence. It could be because although noise labels were not considered in training, the model was validating the outputs against validation sets that still had noise labels as answers to compare against. This means that accuracy could be underrepresented.

5.2. Image Augmentations

Based on the Image Augmentation section in the paper, several augmentation strategies were compared. None means no augmentation was performed and Normal is the augmentation strategy outlined earlier. Heavy is a modification of the normal image augmentation strategy to include rotation and shifting the widths and heights of the image.

Augmentation	Loss	Accuracy	F1
None	0.0575	0.983	0.985
Normal	0.0453	0.989	0.997
High	0.154	0.897	0.889

Table 9. Comparison between image augmentation methods.

The results of the experiment are shown in Table 9. The normal image augmentation strategy from the main paper had the best performance, although marginal compared to the high augmentation method. However, the F1 score in the high augmentation method was more than marginal in difference. This could mean that the high augmentation method had the worst performance. An explanation is that the additional augmentations to the image was not meaningful and prevented the model from properly learning the features from images. It could be the additional augmentations were too extreme and did not reflect the unaugmented test dataset.

5.3. Loss function

Different loss functions were used and compared to identify the differences in performance. The loss functions compared were Mean Squared Error (MSE), Mean Squared Logarithmic Error (MSLE), Hinge, Squared hinge and Cross entropy (logarithmic).

Loss	Loss	Accuracy	F1	Conv.
Cross Entropy	0.0453	0.989	0.997	43
MSE	0.0118	0.988	0.992	61
Hinge	0.682	0.318	0	-
Poisson	0.520	0.837	0.873	-
KL Divergence	0.001	0.315	0	-

Table 10. Comparison between loss function performance.

Cross entropy had the best accuracy and F1 score, at 0.989 and 0.997 respectively. However, the accuracy improvement was only marginal compared to MSE. Cross entropy also managed to converge the quickest, taking on average 43 epochs.

The model was unable to converge using Hinge loss and had terrible performance. Both Hinge and KL Divergence losses did not manage to converge and

produced poor accuracy. The F1 score of 0 indicates that the losses swung predictions into a single class. Hinge loss is typically used for SVMs. The learning rate could be lowered for both losses. However, for this experiment, all other parameters are kept the same.

5.4. Activation function

Different activations in the output layer were compared to assess performance differences. Sigmoid, Softmax, Exponential Linear Unit (ELU), Hyperbolic Tangent (Tanh) and ReLU were used.

Activation	Loss	Accuracy	F1	Conv.
Sigmoid	0.0453	0.989	0.997	43
Softmax	0.0591	0.978	0.989	45
ELU	0.00521	0.785	0.842	30
Tanh	5.397	0.178	0	-
ReLU	0.0494	0.787	0.836	32

Table 11. Comparison between final layer activation performance.

Based on the results in Table 11, Sigmoid managed to perform the best with the highest accuracy and F1 scores, even though it did not converge the fastest. ELU converged the fastest in 30 epochs, however, it did not perform the best. This behaviour is expected as ELU manages faster learning as shown by Clevert et al [35]. Since there are no negative inputs in features, it is not surprising that RELU produced a similar performance as ELU. Tanh did not manage to converge and had the worst performance. It could be due to a vanishing gradient as the Tanh curve is steeper compared to a Sigmoid.

5.5. Regularisation

Regularisation reduces overfitting. 3 different methods are compared; Dropout, L1 and L2. Dropout randomly drops out neurons during each weight update. For this experiment, a light dropout where a 20% dropout after the FC layer and heavy dropout where 20% dropout was added after every layer is compared. L1 and L2 regularisation add a regularisation term to the cost function which modifies the weights in the network [23].

Regulariser	Loss	Accuracy	F1	Conv.
None	0.0575	0.983	0.985	43
Light Dropout	0.291	0.889	0.887	198
Heavy Dropout	0.584	0.712	0.811	89
L1	0.674	0.714	0.719	81
L2	0.499	0.825	0.774	83

Table 12. Comparison between regulariser methods.

Based on the results in Table 12, using any regulariser worsened performance and increased convergence time. Even a single dropout reduced performance by 10%. Since the model did not suffer from overfitting in the first place, additional regularisation is unnecessary as image augmentation already acted as a regulariser. The small training dataset size did not allow the possibility of overfitting so that the regularisation methods can be taken advantage of.

6. Related Work

6.1. Light CNN and Semantic Bootstrapping

A Max-Feature Map (MFM) operation was introduced, an operation to suppress neurons so that CNN models are relatively lighter. The operation is based on the neural science of Lateral Inhibition. Unlike ReLU, this will avoid loss of information and instead separate noisy and informative signals. Semantic bootstrapping, based on child linguistic development theory, is an approach to model training samples during the forward process by resampling. This allows the trained model to sample training data from noisy datasets and relabel the noise. A new dataset can then be reconstructed.

6.2. Siamese Neural Networks

Typical deep neural networks contain a massive number of parameters to gradient descent upon, which is problematic when it needs to generalise based only a single example. Siamese networks utilise transfer learning by training each branch as a classifier and merged into a fully connected layer calculating the L1 Siamese distance. L1 distance is given in Equation 7.

$$distance = \sum |x_1 - x_2| \qquad [7]$$

Where x is the feature vector

Siamese networks are designed to be symmetric and as such, the hyperparameter tuning was the same for each branch. The "head" of the network is a simple linear classifier which outputs a prediction between 0 to 1 on whether the two images are similar.

6.3. Residual Networks

Deeper networks are exposed to a degradation problem where accuracy saturates and then rapidly degrades. He et al. posits that reframing layer mappings can lead to easier optimisation [26].

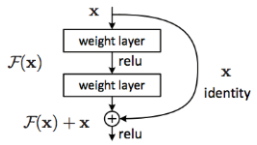

Figure 6. A fundamental block of a residual network.

Figure 6 depicts a residual mapping using 2 layers. A typical neural network would only have an *F(x)* mapping, where x is the input. By skipping layers, a residual *F(x)* + *X* mapping was created.

For optimisation, it is easier to optimise the residual F(x) function rather than optimise a whole stack of non-linear CNN layers with a function *F(x) = y*. Subsequent bocks in the network can fine-tune the output of a previous block.

References

[1] J. Han, M. Kamber and J. Pei, *Data mining*, 3rd ed. Amsterdam: Elsevier/Morgan Kaufmann, 2012.

[2] E. Kalapanidas, N. Avouris, M. Craciun and D. Neagu, "*Machine Learning algorithms: a study on noise sensitivity*", 2003. Available: http://delab.csd.auth.gr/bci1/Balkan/356kalapanidas.pdf.

[3] J. Brownlee, "What is the Difference Between Test and Validation Datasets?", *Machine Learning Mastery*, 2017. [Online]. Available: https://machinelearningmastery.com/difference-test-validation-datasets/.

[4] J. and L. Perez, "*The Effectiveness of Data Augmentation in Image Classification using Deep Learning*", 2017. Available: https://arxiv.org/pdf/1712.04621.pdf.

[5] D. Hubel and T. Wiesel, "Receptive fields of single neurones in the cat's striate cortex", *The Journal of Physiology*, vol. 148, no. 3, pp. 574-591, 1959. Available: 10.1113/jphysiol.1959.sp006308.

[6] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998. Available: 10.1109/5.726791.

[7] A. Karpathy and F. Li, "CS231n: Convolutional Neural Networks for Visual Recognition", *Cs231n.github.io*, 2017. [Online]. Available: http://cs231n.github.io/.

[8] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks", *Communications of the ACM*, vol. 60, no. 6, pp. 84-90, 2017. Available: 10.1145/3065386.

[9] X. Glorot, A. Bordes and Y. Bengio, "Deep Sparse Rectifier Neural Networks", *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15, 2011. Available: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

[10] A. Sharma, "Understanding Activation Functions in Neural Networks", *Medium*, 2017. [Online]. Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[11] P. Koprinkova-Hristova, V. Mladenov and N. Kasabov, Artificial Neural Networks: Methods and Applications in Bio-/Neuroinformatics. Cham: *Springer International Publishing*, 2015.

[12] D. Kingma and J. Lei Ba, "Adam: A Method for Stochastic Optimization", in *International Conference on Learning Representations*, San Diego, 2015. Available: https://arxiv.org/pdf/1412.6980.pdf.

[13] A. Honkela, "Multilayer perceptrons", *Nonlinear Switching State-Space Models*, 2001. [Online]. Available: http://users.ics.aalto.fi/ahonkela/dippa/node41.html.

[14] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303-314, 1989. Available: 10.1007/bf02551274.

[15] G. Panchal, A. Ganatra, Y. Kosta and D. Panchal, "Behaviour Analysis of Multilayer Perceptrons with Multiple Hidden Neurons and Hidden Layers", *International Journal of Computer Theory and Engineering*, pp. 332-337, 2011. Available: 10.7763/ijcte.2011.v3.328.

[16] C. Szegedy et al., "Going deeper with convolutions", *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. Available: 10.1109/cvpr.2015.7298594.

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. Available: 10.1109/cvpr.2016.308.

[18] Keras, "Guide to the Sequential model - Keras Documentation", *Keras.io*, 2018. [Online]. Available: https://keras.io/getting-started/sequential-model-guide/.

[19] IceCream Labs, "3×3 convolution filters - A popular choice - IceCream Labs", *IceCream Labs*, 2018. [Online]. Available: https://icecreamlabs.com/2018/08/19/3x3-convolution-filters%E2%80%8A-%E2%80%8Aa-popular-choice/.

[20] M. Buda, A. Maki and M. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks", *Neural Networks*, vol. 106, pp. 249-259, 2018. Available: 10.1016/j.neunet.2018.07.011

[21] "Model (functional API) - Keras Documentation", *Keras.io*, 2019. [Online]. Available: https://keras.io/models/model/.

[22] A. Prakash, "One by One [1 x 1] Convolution - counter-intuitively useful", *Iamaaditya.github.io*, 2016. [Online]. Available: https://iamaaditya.github.io/2016/03/one-by-one-convolution/.

[23] S. Jain, "An Overview of Regularization Techniques in Deep Learnin", *Analytics Vidhya*, 2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/.

[24] X. Wu, R. He, Z. Sun and T. Tan, "A Light CNN for Deep Face Representation With Noisy Labels", *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2884-2896, 2018. Available: 10.1109/tifs.2018.2833032.

[25] G. Koch, R. Zemel and R. Salakhutdinov, "Siamese Neural Networks for One-shot Image Recognition", *ICML Deep Learning Workshop*, vol. 2, 2015. Available: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf.

[26] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. Available: 10.1109/cvpr.2016.90

[27] S. Xie, R. Girshick, P. Dollar, Z. Tu and K. He. *Aggregated Residual Transformations for Deep Neural Networks*. arXiv preprint arXiv:1611.05431v1,2016.

[28] G. Huang, Z. Liu, K. Q. Weinberger and L. Maaten. *Densely Connected Convolutional Networks*. arXiv:1608.06993v3,2016.

[29] J. Brownlee, "8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset", *Machine Learning Mastery*, 2018. [Online]. Available: https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/.

[30] T. Hastie, J. Friedman and R. Tibshirani, *The elements of statistical learning*, 2nd ed. New York: Springer Science & Business Media, 2009.

[31] B. Neal et al., *"A Modern Take on the Bias-Variance Tradeoff in Neural Networks"*, 2018. Available: https://arxiv.org/pdf/1810.08591.pdf.

[32] M. Nielsen, *Neural Networks and Deep Learning*, 1st ed. Mountain View: Determination Press, 2015.

[33] Algorithmia, "Introduction to Loss Functions | Algorithmia", Algorithmia, 2018. [Online]. Available: https://blog.algorithmia.com/introduction-to-loss-functions/.
[34] G. Hinton, *"Overview of mini-batch gradient descent"*, University of Toronto, 2014.
[35] D. Clevert, T. Unterthiner and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)", *International Conference on Learning Representations (ICLR) 2016*, 2015. Available: https://arxiv.org/pdf/1511.07289.pdf.