Facemask Detection Machine Learning Model Development

Joshua Bainbridge - 250869629

Chandan Swaraj Chandel – 250914472

ECE 9063: Data Analytics

Dr. Katarina Grolinger

Due: December 03, 2021

1. Introduction

1.1 Project Problem

A massive amount of labour in many sectors of society (public and private) is used to monitor mask usage as individuals enter sensitive areas (i.e. public bus, shopping center, schools). This process could be automated to reallocate labour more effectively. This paper will explore an attempt to use image data to classify whether an individual is wearing a facemask or not.

1.2 Available Dataset

The dataset carried over 12000 images between the testing, training, and validation data subsets, because the dataset was organised and delivered in the folder structure shown in *Figure 1.*, the labels for each image were already given (folder name is the label). The data size breakdown is delivered in, *Table 1.*, with the training set being an order of magnitude larger than the test and validation sets. The images were RGB, and none were grayscale, additionally the size of the images varied considerably.

Table 1. Breakdown of data set

	WITH MASK	WITHOUT MASK
TEST	438	509
TRAIN	5000	5000
VALIDATION	400	400

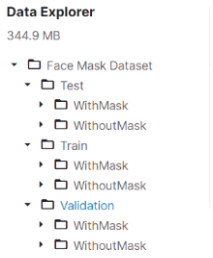

Figure 1. Data set figure structure

The training set is a set of examples used for learning, that is to fit the parameters of the classifier. The validation set is a set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network. The test set is a set of examples used only to assess the performance of a fully specified classifier[1]. The data came organized into the test, train, and validation categories, meaning no data splitting solution was implemented by the authors of this paper (manual or automated through program).

2. Background and Methodology

2.1 Data Preprocessing

The available data underwent preprocessing, prior to being used to train or test any deep learning solution. The preprocessing was conducted in three steps: surveying of the data to remove outliers, removal of background information and scaling-normalization. Other data preprocessing techniques such as data balancing and vectorization were not required because both categories of data were equally represented in the date set and all the data were in the form of images.

2.1.1 Surveying Data to Remove Outliers

Most of the images in the available data can be categorized as portraits (images of the head and shoulders). Additionally, the majority of these portraits are front facing and close ups, meaning the subject takes up the majority of the image space.

The available data also included in small number of outlier images that were not representative of the data set. These outliers were removed form the full data set in order to ensure the machine learning algorithms were being trained and tested on the same type of data. The second justification for removing outliers was the vast majority of outlying images were part of the *With Mask* dataset and may have biased the algorithms towards categorizing outlying data points a such. Some examples of the images categorized and removed as outliers include images of multiple faces, full body images, see through masks and corrupted images as shown in *Figure 2*.

Figure 2. Removed outlying images

Filtering of outliers was done manually with the assistance of an image displaying program developed in python, *Figure 3*. The program looped through and displayed all the images in each of the test, train and validation directories and allowed a group member to easily remove the image through a console input. After manual outlier filters roughly 563 images were removed (327 from training, 96 from testing and 113 from validation).

Figure 3. Manual data surveying program

2.1.2 Removal of Background Information

The second stage of data preprocessing was to remove the background, if one existed, in the any of the images. The primary reason behind removing the backgrounds from images was to attempt to prevent the deep learning algorithm form developing a bias. For example since it is more likely an individual will be wearing a mask indoors then outdoors, it is possible that more images classified as *With Mask* will have white or tope backgrounds and images classified as *Without Mask* may be more likely to have green or blue backgrounds. Since any data related to the environment the picture was taken in is independent to whether or not an individual is wearing a mask, the background could be considered

noise in the image. By removing the noise the deep learning algorithm should be less influence by irrelevant data and give a greater weight to the face and mask data.

To remove the background the image data with pipelined through a pretrained convolutional neural network (CNN) called ModNet [2][3]. ModNet has been trained to separate the foreground image of a person from the background and replace the background with a solid black colour. The pure black background was selected as the final background colour because it of the high contrast and against an individual's face, *Figure 4*.

Figure 4. Raw test image data (left), ModNet output (right)

2.1.3 Scaling and Normalization

The third and final stage of data preprocess, was scaling and normalization of the data. The available data is given as image data ranging from 39 by 39 pixels to 255 pixels by 255 pixels. To train a CNN the input data was required to be a constant size. Each image was scaled to 150 pixels by 150 pixels prior to being used into train and test the machine learning algorithm.

Additionally, all the image data was normalized. Image data is given as pixel values with integers ranging from 0 to 255. In contrast the learning rates and initial biases of a neural network are set to small values (≈0.001). Due to the large difference in magnitude between the input information and the hidden layer weights, tuning a deep learning algorithm with raw image data would increase convergence time and computation power to train the network. To reduce the computational costs of training a neutral network, all the images were normalized to have values between 0 to 1[4], *Figure 5*. It should be noted all of these images are RGB, which further stretches computational resources – in a future iteration where performance is less valuable than computational resources there may be a valid case for grey scaling the dataset.

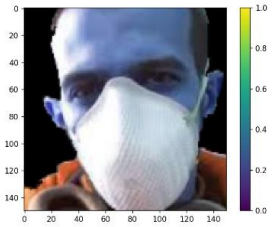

Figure 5. Scaled image

2.2 Learning Paradigm Selection

A learning paradigm is used to solve a specific challenge, related to developing a deep learning solution to a problem. In the case of this project, when single convolutional neural networks were tested, during

the early exploration phase, the models only had an accuracy of 60% to 68%. To address the low performance of the individual neural networks, multiple avenues were considered such as; reassessing parameters & hyperparameters, increasing training set size, and the implementation of an ensemble learning paradigm.

2.2.1 Ensemble Learning

Ensemble learning is learning paradigm strategy where multiple constituent models are trained, and their outputs are combined to provide a single output. This consensus-based strategy should provide better performance than any one constituent model might be able to achieve. This strategy was originally proposed by researchers for classification problems but can be applied to other problem types [5].

2.2.2 Implementing Ensemble Learning

Implementation of ensemble learning was done by constructing a pipeline, *Figure 6*. The raw data was preprocessed to remove the background, as well as scaling and normalizing. After which the data was fed to ten constituent CNN models within the ensemble learning paradigm. The CNN's had multiple variations associated with them and each was trained on the full dataset. This improved accuracy but did not improve the computational expense. Each CNN had a combination of hyperparameters including different number of layers, neurons, learning rates and alpha, beta values along with different activation functions. A minimum F1 score of 0.65 was required before a CNN model could be added to the overall ensemble model. A grid search method was used to test combinations of parameters for the CNN models. To ensure the grid search did not simply convene on the same solution for each constituent neural network, the modifiable hyperparameters for each model's grid search were very different and included exclusive parameters to search through. Finally, a voting mechanism was required by an ensemble learning model to decide what the classification would be.

Commented [CSC1]: Is this true?

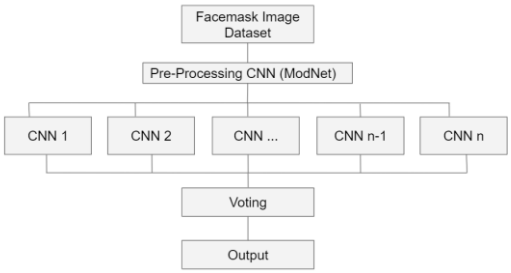

Figure 6. Ensemble learning architecture

2.2.3 Voting

Hard vs Soft Voting

There are two voting methods used in classification systems in an ensemble learning paradigm. The first method is to sum up the classification output of each constituent model and produce a final classification based on the most popular output, this is known as hard voting. An alternative second method is called soft voting [6]. It uses the average of the probability assigned by each constituent

model to classify the output. This method requires that the confidence interval data regarding classifier output is available. Soft voting typically yields improved accuracy by assessing edge cases in more detail than a hard voting system may allow.

Selected Voting Method: Soft Voting

A soft voting system was used as the voting method in the ensemble learning model. To implement a soft voting system into the final python program the classification outputs of *With Mask* and *Without Mask* must be vectorized. This was done by assigning a value of 1 to all classifications of *With Mask* and a value of 0 to all classifications of *Without Mask*. After the outputs were vectorized the confidences interval was made to be negative if the output was 0 or remained positive if the output was 1, *Equation 1*. This process was repeated for each CCN in the ensemble until an array of c_i was constructed.

$$c_i = \begin{matrix} cv & if\ output = 1 \\ -cv & if\ output = 0 \end{matrix} \qquad (Equation\ 1)$$

Following the soft voting methodology the weighted average was calculated. Each CNN in the ensemble was given the same weight, which was normalized to 0.1, *Equation 2*. This resulted in a possible range for the sv values of -1 to 1.

$$sv = \sum_{i=0}^{N-1} c_i * 0.1 \qquad (Equation\ 2)$$

Finally, the output is calculated using the piecewise function, *Equation 3*. The final output is assigned *With Mask* or *Without Mask* classifications before being recorded.

$$Output = \begin{matrix} 1 & if\ sv > 0 \\ 0 & if\ sv < 0 \\ undefined & if\ sv = 0 \end{matrix} \qquad (Equation\ 3)$$

2.3 Convolutional Neural Network

2.3.1 Architecture

A Convolutional Neural Network has no loops and information flows in one direction (input to output). There are two different types of layers between the input and the output layer, these are the convolutional and pooling layers, as seen in *Figure 7* [7]. Convolutional layers systematically convolve the input through a learned filter and pass the result onto the next layer, which can be a pooling layer. A pooling layer also applies an operation to group of pixels however this is not a learned filter, it is pre-defined. One of the major failings of a feedforward network (FNN) is that it does not scale well with images. Due to the size of input data of each image the process of training a FNN is very computationally expensive. For example, an input image that is 32 pixels x 32 pixels in size would require 3072 input neurons. Using a CNN greatly reduces the computational cost. Convolutional neural networks were designed to be incredibly adept at dealing with image inputs by rapidly applying convolutions to scale down the size of the computational problem[8].

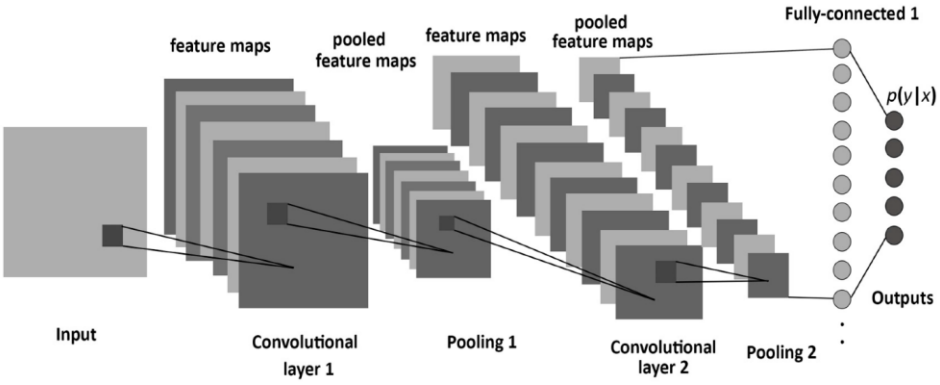

Figure 7. Convolutional neural network [9]

2.3.2 Training and Validation

In a convolutional neural network, each connection is assigned a weight or bias that must be calibrated through training of the network. A CNN can be trained using a technique called back-propagation. At every iteration through a training cycle, henceforth called epoch, the output value of the model is compared to the expected output and an error function is calculated. The error is sent back through the network and the weights of connections are adjusted to reduce the error, typically using the gradient descent method.

The weights can be optimized using multiple different strategies; however, the most fundamental strategy is gradient descent. The strategy is to calculate the derivative of the error function with respect to the weight of the connection of the neurons and then "roll" further down the hill at each epoch. There are many gradient descent variations such as 'adam' and stochastic gradient descent. These strategies can include manipulating factors such as momentum, learning rate changes and more. The learning rate in particular can affect the outcome of gradient descent strongly and is the second most important factor in determining model accuracy after network architecture as noted in Lectures.

Commented [CSC2]: Is this weird?

In the process of training a network the model can sometimes be prone to overfitting. Overfitting is when the model fits the training data (nearly) exactly, when the model is faced with new data it will probably fail to make an accurate prediction because it hasn't truly captured the underlying relationship between the independent variable (input) and the dependant variable (output). Overfitting can be dealt with by using a smaller network size / training data set size ratio, regularization or early stopping.

Early Stopping determines when to stop training a model by essentially checking to see whether the error function has not reduced for a certain number of epochs. Early stopping was implemented by the authors of this paper, however, even before early stopping was implemented, there was no overfitting because the ratio of network size to training data set size was already appropriate.

2.4 Tuning Convolutional Neural Networks

To tune the constituent CNN models a grid search tuning method was employed.

2.4.1 Overview

Grid search is a tuning method that exhaustively (brute force) considers all possible combinations of hyperparameters to find the best performing neural network. Each of the F1 scores were then recorded and used to select the final ten models that were then used in the final ensemble algorithm. The hyperparameters tested included number of layers, perceptrons per layers, activation function, learning rate, filter size and batch size.

2.4.2 Grid Search Strategy

The number of exhaustive permutations possible required an infeasible amount of computational power and time, this challenge can be briefly described as $O(n^{\#\text{ of total attributes}})$. One of the strategies that was discussed to improve the computational and time challenge was to first assess major hyperparameters and then attempt to tune with a second set of more minor hyperparameters[10]. Thus a more constructive way to break down this problem would be to split the grid search into two parts $O(n^{\#\text{ of major attributes}}) + O(n^{\#\text{ of minor attributes}})$. The assumption is made that the results from the major attributes have a much greater influence on the performance of the neural network then the other hyperparameters. By first tuning the models on the major parameters, lower performing permutations could be excluded from further tuning. By doing that the computational cost would be greatly reduce. There was a risk of possibly missing a high performing model. However this risk was mitigated by the implementation of the ensemble learning paradigm, because it was not required that the highest performing model be found. Instead computation power focused on the models that had the greatest likely hood of high performing models.

Commented [JB3]:

Commented [CSC4]: May need a better way to cite that

Major hyperparameters were selected based on a preliminary grid search with fewer hyperparameter permutations conducted on a base model built with randomly selected hyperparameters, *Table 2* – to provide a quantitative basis for classifying the relative importance of network features. Based on the results, hyperparameters have been divided as shown in *Table 3.*

Table 3. Base model

Number of Layers	**5**
Perceptrons per Layer	16,32,64,96,128
Activation Function	'relu'
Learning Rate	0.001
Batch Size	16
Filter Size	[3,3]

Table 2. Hyperparameters

Major Hyperparameters	Minor Hyperparameters
Number of Layers	Learning rate
Perceptron per Layer	Batch size
Activation Function	Filter size

2.4.2.1 Major Hyperparameter Tuning

Number of Layers

The hyperparameter called "number of layers", refers to the number of convolutional layers used in the CNN. The number of layers possible tested in the grid search algorithm were [1,3,5,7].

Perceptron per Layer

The permutations of perceptrons per layer were [16,32,64,96,128]. The grid search algorithm searched for constant perceptron's on each layer (i.e. [16,16,16]), ascending perceptron's on each layer (i.e.

[16,32,64]), descending perceptron's on each layer (64,32,16) and ascending-descending perceptrons on each layer (i.e. [16,32,16]).

Exhaustive list of perceptrons per layer is outline in *Table 4.*

Table 4.

LAYERS	PERCEPTRON'S PER LAYER IN ORDER (INPUT TO OUTPUT)
3	[16, 16, 16], [32, 32, 32], [64, 64, 64], [96, 96, 96], [128, 128, 128], [16, 32, 64], [16, 64, 96], [16, 64, 128], [32, 64, 96], [32, 64, 128], [64, 96, 128], [16, 32, 16], [16, 64, 32], [32, 64, 32]
5	[16, 16, 16, 16, 16], [32, 32, 32, 32, 32], [64, 64, 64, 64, 64], [96, 96, 96, 96, 96], [128, 128, 128, 128, 128], [16, 32, 64, 96, 128], [128, 96, 64, 32, 16], [16, 64, 128, 96, 32]

Activation Function

The permutations of activations function were set to be rectified linear function ('relu'), sigmoid activation function ('sigmoid'), and probability distributions ('softmax')[11].

Results

When the number of layers was 1, the model never converged and therefore all models with a single convolutional layer were ejected. Additionally, none of the models with 7 convolutional layers performed better than the other "number of layers" permutations. As a result, these models were also rejected because the additional layers greatly increased the computational expense required to train and test the model. The final constituent CNN models had either 3 or 5 convolutional layers.

A second trend was observed during the major hyperparameter grid search. The activation function has a huge impact on the performance of the model. The models with 'relu' outperformed the other activation function by a large margin, *Figure 8.*

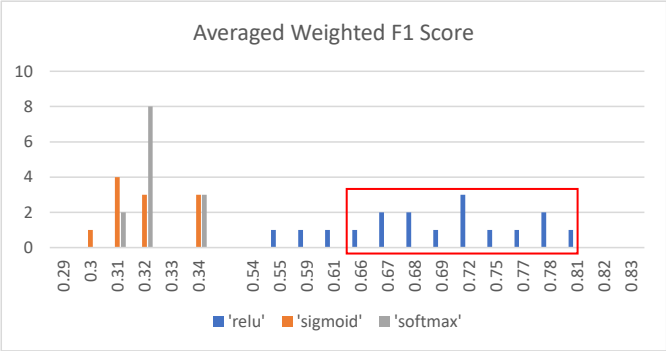

Figure 8. Major hyperparameter tuning results

The models that have an F1 score over 0.65 were selected from this stage in the grid search algorithm and tuned again with the minor hyperparameters, 14 models were selected to be retuned with the minor hyperparameters.

2.4.3 Minor Hyperparameter Tuning

Learning Rate

The learning rate is the amount each bias is altered by during the training of the neural network. Learning rates of 0.0005 to 0.005 with an interval of 0.0001 were used the learning rate hyperparameters. An adaptive learning rate was not tested, as methods of learning schedule was not a selected hyperparameter for the grid search[12]. However, in future work learning schedules should be included.

Batch Size

The batch size refers to the number of images used to train at once. The batch size determines the number of steps the model trains per epoch. Batch sizes of [1,8,16,32,64] were set as possible values for this hyperparameter.

Filter Size

The filter size refers to the size of the convolutional filter used in in each layer. The sizes of the filters were set to [[3,3],[5,5],[10,10]].

2.4.4 Final CNN Parameters

Model Number	Number of Layers	Perceptrons per Layer	Activation Function	Learning Rate	Batch Size	Filter Size	F1 Score
1	3	[64,64,64]	'relu'	0.001	16	[3,3]	0.78
2	3	[32,64,96]	'relu'	0.0009	16	[3,3]	0.72
3	3	[32,64,128]	'relu'	0.0013	16	[3,3]	0.75
4	3	[64,96,128]	'relu'	0.001	16	[3,3]	0.81
5	5	[32,32,32,32,32]	'relu'	0.001	16	[3,3]	0.72
6	5	[64,64,64,64,64]	'relu'	0.0012	16	[3,3]	0.78
7	5	[96,96,96,96.96]	'relu'	0.001	16	[3,3]	0.72
8	5	[16,32,64,96,128]	'relu'	0.0008	16	[3,3]	0.7
9	5	[128,96,64,32,16]	'relu'	0.0011	16	[3,3]	0.69
10	5	[16,64,128,96,32]	'relu'	0.001	16	[3,3]	0.72

2.5 Other Applicable Technologies

2.5.1 CUDA

Throughout this project a lot of models were trained on a fairly large dataset, this is the kind of situation where processing power becomes an issue. To speed up processing CUDA can be used, this is a tool which was made by NVIDIA to unlock the GPU for running computationally expensive calculations [13]. It has been used in a wide variety of fields including computational biology, cryptography, crypto mining and for Machine Learning Model Development.

Commented [CSC5]: CUDA - Wikipedia

2.5.2 Subset training

Training on subsets of the data simply reduces the scale of the computational problem by reducing the amount of training, there is typically a trade-off with performance when this strategy is implemented. This strategy is reflected in Transfer Learning and can be reflected (optionally) in Ensemble Learning.

2.5.3 Model Saving and Loading

Another tactic which could save time is simply saving and reloading the model, this allows a runtime to be completed and the device to be shut off without the need to retrain the model on reboot as all of the relevant training information is still saved.

3. Final Results

The following section details the classification evaluations metrices used to measure the viability of the ensemble mask classification solution and the final results achieved by the deep learning algorithm.

3.1 Classification Evaluation Metrics

For classification problems a confusion matrix is a useful tool for evaluating the performance of a model. There are two possible classifications for this model including With Mask & Without Mask. Thus, the confusion matrix output is 2x2 (X axis is predicted, Y axis is actual and there is 2 valid classes). From this confusion matrix there are three metrics which can be derived; Precision, Recall & F1 Score[14]. Precision, Recall & F1 Score answer the following questions.

"Precision: Given all the predicted labels (for a given class X), how many instances were correctly predicted?

"Recall: For all instances that should have a label X, how many of these were correctly captured?"

F1 Score combines precision and recall giving a more inclusive picture of the model accuracy. Since the F1 score evaluates the average performance of the whole system it is the best metric to focus on.

$$\text{F1 Score} = \frac{2 * Precision * Recall}{Precision + Recall} \qquad \text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive} \qquad \text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Commented [CSC6]: Make sure table is actually below this block of text.

Commented [CSC7]: [Text Mining, Analytics & More: Computing Precision and Recall for Multi-Class Classification Problems (rxnlp.com)]

3.2 Results

3.2.1 Confusion Matrix

The final ensemble learning algorithm was tested using the full test images dataset. A subset of 30 test images were plotted and outputted with the predicted classification and confidence level, *Figure 9*. The results of testing the complete test data set were compile into a confusion matrix.

Commented [CSC8]: This number is wrong, should be 947

	PREDICTION: WITH MASK	PREDICTION: WITHOUT MASK
IMAGE: WITH MASK	392	46
IMAGE: WITHOUT MASK	91	418

Recall: $\frac{TP}{TP+FN} = \frac{392}{392+46} = \mathbf{0.895}$

Precision: $\frac{TP}{TP+FP} = \frac{392}{392+91} = \mathbf{0.812}$

F1: $\frac{2(Recall)(Precision)}{Recall+Precision} = \frac{392}{392+91} = \mathbf{0.851}$

Figure 9. Example Results

4. Conclusion

4.1 Review of Solution

Good results were captured from the constituent models with F1 scores ranging between 0.69 and 0.81, however the overall ensemble learning model achieved an F1 score of 0.85. This is inline with the expectation that ensemble learning could improve performance more than any single constituent model. Please note, in cases where the model was incorrect, the model confidence was extremely low. To better reflect the model's analytical abilities a third category could be implemented which describes suspect or uncertain cases where confidence ranges between -0.2 to 0.2. By changing the model into a three-way classifier, the F1 score would further improve remarkably and provide a clearer snapshot of what the system is capable of identifying without actually looking at confidence interval data.

4.2 Ethics

There are ethical concerns surrounding the deployment of this technology. This device was designed after identifying a need but without looking at the ethical implications. Information privacy is the largest area of concern because; facial ID data could be captured, location data could be captured, mask compliance data could be captured. Other notable areas of concern include labour automation & inaccurate results. In short, the ethical concerns may outweigh the usefulness of the technology.

Bibliography

[1] "What is the Difference Between Test and Validation Datasets?" https://machinelearningmastery.com/difference-test-validation-datasets/ (accessed Dec. 02, 2021).

[2] "Remove the background from images using AI and Python." https://livecodestream.dev/post/remove-the-background-from-images-using-ai-and-python/ (accessed Dec. 02, 2021).

[3] "Basic classification: Classify images of clothing | TensorFlow Core." https://www.tensorflow.org/tutorials/keras/classification (accessed Dec. 02, 2021).

[4] "How to use Data Scaling Improve Deep Learning Model Stability and Performance." https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/ (accessed Dec. 02, 2021).

[5] Y. Yang, "Ensemble Learning," *Temporal Data Mining Via Unsupervised Ensemble Learning*, pp. 35–56, 2017, doi: 10.1016/B978-0-12-811654-8.00004-X.

[6] "Hard vs Soft Voting Classifier Python Example - Data Analytics." https://vitalflux.com/hard-vs-soft-voting-classifier-python-example/ (accessed Dec. 02, 2021).

[7] "A Gentle Introduction to Pooling Layers for Convolutional Neural Networks." https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/ (accessed Dec. 02, 2021).

[8] Q. Zhang, Y. Liu, G. Liu, G. Zhao, Z. Qu, and W. Yang, "An automatic diagnostic system based on deep learning, to diagnose hyperlipidemia," *Diabetes, Metabolic Syndrome and Obesity: Targets and Therapy*, vol. 12, p. 637, 2019, doi: 10.2147/DMSO.S198547.

[9] Y. Pan, Y. Xia, Y. Song, and W. Cai, "Locality constrained encoding of frequency and spatial information for image classification," *Multimedia Tools and Applications*, vol. 77, no. 19, pp. 24891–24907, Oct. 2018, doi: 10.1007/S11042-018-5712-3.

[10] "Narrowing the Search: Which Hyperparameters Really Matter?" https://blog.dataiku.com/narrowing-the-search-which-hyperparameters-really-matter (accessed Dec. 02, 2021).

[11] "Optimizers." https://keras.io/api/optimizers/ (accessed Dec. 02, 2021).

[12] "Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning | by Suki Lau | Towards Data Science." https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1 (accessed Dec. 02, 2021).

[13] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High Performance Network Intrusion Detection Using Graphics Processors," *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID)*, vol. 5230, pp. 116–134, 2008, doi: 10.1007/978-3-540-87403-4_7.

[14] “Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures - Exsilio Blog.” https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/ (accessed Dec. 02, 2021).

Appendix

Code

Due to the length of the programs built for this project all the code has been upload to Google Colab. Each program can be view by their respective links.

Preprocessing:

https://colab.research.google.com/drive/1ZtCrhR0X8I853rDKi3gEVy5ixLpIbHce?usp=sharing

ModNet:

https://colab.research.google.com/drive/1PaZJpk_xlYsJgsbNqNIwzfmolpa8r1tZ?usp=sharing

Grid Search :

https://colab.research.google.com/drive/1HmrWar-8_UUY8oYCdyd9SKzMReXXlp_t?usp=sharing

Ensemble Learning:
https://colab.research.google.com/drive/13MmO5s9Xtf7yQOIpJDhzNh6_IuB8JSOS?usp=sharing