# Face Mask and Face Shield Detection using Artificial Intelligence

Guan Zihang, Lho Chen Yen, Toh De Xun, Zeng Shijia

Academic Year 2020/2021, Semester I

**Abstract**

With the recent COVID-19 pandemic, it is mandatory for individuals to practice social responsibility and wear face masks in public spaces to curb the spread of the coronavirus. The detection of masked faces is thus extremely important, as such algorithm can be integrated in various surveillance methods to complement current enforcement methods, which is done manually. In this project, our group applied Neural Networks, Support Vector Machines, Histogram of Gradients, Decision Trees, Random Forests, and Transfer Learning for classification. We found that ResNet yielded the best performance, with 100.0% accuracy.

## Contents

# 1 Dataset

Dataset is at: https://www.dropbox.com/s/q3674olgj4z7i4m/face_mask.zip?dl=0

Google Collab Code: https://colab.research.google.com/drive/1DIwFcD1mNDJQ2VPtsc9r-nvjK2jIKKAj?usp=sharing

# 2 Literature Review

The outbreak of COVID-19, the novel coronavirus SARS-CoV-2 infection, was first reported on December 31, 2019, in Wuhan, China. Since then, COVID-19 has been spreading rampantly throughout the world, with the World Health Organization (WHO) declaring COVID-19 a pandemic due to alarming levels of spread and severity on March 11, 2020 [1].

COVID-19 is the third coronavirus to have threatened global public health in the past 20 years, following Severe Acute Respiratory Syndrome Coronavirus (SARS-CoV) in 2002, and Middle East Respiratory Syndrome Coronavirus (MERS-CoV) in 2012 [2].

The usage of masks to curb the spread of these coronaviruses have been well established [3] [4], with more than one hundred countries issuing nationwide mask mandates [5] as a source control to reduce community transmission.

While facial detection has been well studied, face mask detection is a difficult problem as facial features are covered up by masks. Furthermore, masks come in different shapes, sizes, and colours. This results in current face recognition algorithms to struggle with masked faces, with even the best algorithms having failure rates of between 5% to 50% [6].

Prior to the advent of deep learning, in particular, the models LeNet-5 and AlexNet, Decision trees, Deep Neural Networks (DNN), k-nearest neighbours (kNN) and Support Vector Machines (SVM) were commonly used to solve image classification problems [7]. With recent research developments that allow for effective training of deeper networks, e.g., the introduction of rectified linear units and dropout layers, Convolutional Neural Networks and Transfer Learning methods have been increasing in popularity, and are now commonly used to solve image classification problems [11].

In this paper, we explore different machine learning models to identify if an individual is wearing a face mask or a face shield or neither, which is different from identifying a specific person wearing a mask, or face shield. This can be implemented in surveillance cameras, allowing real-time monitoring of people wearing masks, replacing the cumbersome process of manual monitoring, as well as complementing current facial recognition technology.

# 3 Data Description

The data used consists of 3,000 images in total, with 1,000 images of face shields, faces and masked faces each. All images are preprocessed in Python, with face, masked faces and face shield images having labels of 0,1 and 2 respectively. Depending on the model applied, the size of the images were then rescaled accordingly.

The face masks images and face images were obtained from the Real World Masked Face Dataset (RWMFD), which is currently one of the world's largest dataset for facial recognition. For the project, we randomly selected $1,000$ masked images out of $90,000$ images, and $1,000$ face images out of $5,000$ face available. We were unable to find datasets containing face shields, possibly due to the lack of people wearing face shields prior to the pandemic.

Hence, our group manually sourced for pictures from various e-commerce platforms, such as Shopee, Lazada, and Taobao, and applied image augmentation methods to artificially increase our sample size.

(a) Face Image

(b) Masked Face Image


(c) Face Shield Image

Figure 1: Images In Our Dataset

# 4 Data Augmentation

Data augmentation is a strategy where, by introducing small variations to the original data in the form of rotation, shearing, translation, zooming, etc., we artificially increase the size of our data set. While this is not as good as new data, it helps to prevent overfitting.

# 5 Feature Extraction

In this project, we are interested in the set of coloured pictures that will be able to identify if a person is wearing a mask, a face shield, or neither. To do so, we will extract the pixels from the images, which will be the features of our models. For example, for our Transfer Learning model VGG16, all of our images will be rescaled to $224 \times 224 \times 3$ due to the underlying architecture. We will flatten this into a $150,528$ column vector, and each of our images will have $150,528$ features.

# 6 Models

## 6.1 Dense Neural Network

The first model that we used is a simple and fully-connected (i.e. neurons in two adjacent layers are fully-connected), dense layer model.

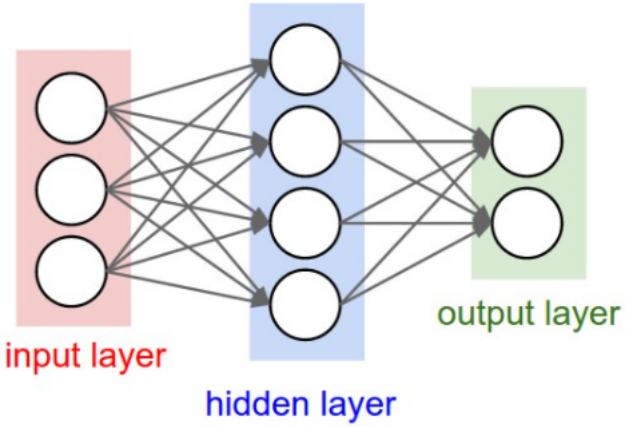



Figure 2: Dense Neural Network

The first layer of our DNN model contains of the input layer, which is flattened by $50 \times 50$. The second layer is a hidden layer defined with dense nodes of 128 and ReLu activation. The last layer consists of 3 dense nodes with a softmax activation function. Softmax function is used as we want our model to solve a multinomial classification problem. For our subsequent CNN models, we will also be using the softmax activation with 3 nodes in our final layer.

We first set the number of dense nodes in the second layer to be 128. As part of the hyperparameter tuning process, we will vary the number of dense nodes in this layer to be between 32 and 512, and choose the best number of dense nodes that will best maximise the accuracy, hence increasing the performance of our Dense ANN.

Through trial and error, we will choose 128 nodes to be in our dense layer. This is our baseline model, and it achieved an accuracy of 89.76%.

## 6.2 Convolutional Neural Network

### 6.2.1 Background

The second model that we will apply to our project is Convolutional Neural Networks (CNNs). CNNs have been applied to visual tasks since the late 1980s, and have recently increased in popularity with developments in com-

puting power and advent of large amounts of labelled data. This has brought them to the forefront of a neural network renaissance that has seen rapid progression since 2012 [22], and they are now considered the most popular neural network model for Image Classification problems[23]. CNN excels at image classification problems due to the following reasons:

1. CNN architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network [20].

2. Unlike a Dense Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth.

3. CNN compares the image pixel by pixel. Thus, it is able to see greater similarities among images.

4. Dense Neural Networks does not scale well to full images, especially image of large sizes. With the large amount of parameters in the large images, the full connectivity of a DNN will result in overfitting.

### 6.2.2 CNN Architecture

CNN is a neural network used to process data with grid-like topology. Specifically, it excels in processing data which has spatial correlation between neighbourhood data points, and is defined as neural networks that use convolutions in place of general matrix multiplication in at least one of their layers.

The figure below shows a possible architecture of a CNN model, which contains convolutional layers, pooling layer, and fully-connected layers [25].

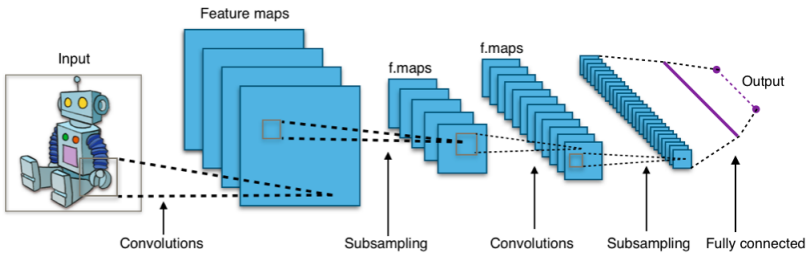



Figure 3: Convolutional Neural Network Architecture

The architecture of CNN is as follows[21]:

- Convolution Layer: The main building block of CNN. The network employs convolution to merge two sets of information. In CNN, the convolution is applied on the input data using a convolution filter to produce a feature map.

  The formula below shows how the convolution operation in CNN works, with the values of entries in the feature map calculated according to the formula. Note that the input image is denoted by $f$, while our kernel is denoted by $h$. The indices of rows and columns of the result matrix are denoted by $m$ and $n$ respectively [26].

  $$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k]$$

  The following demonstrates how the convolution operation is performed in the convolutional layer of a CNN. First, we place our filter over a selected pixel, then we take each value from kernel and perform pairwise multiplication with corresponding values from the input image. Finally, we sum up everything and the sum will be placed in the right place of the output feature map.

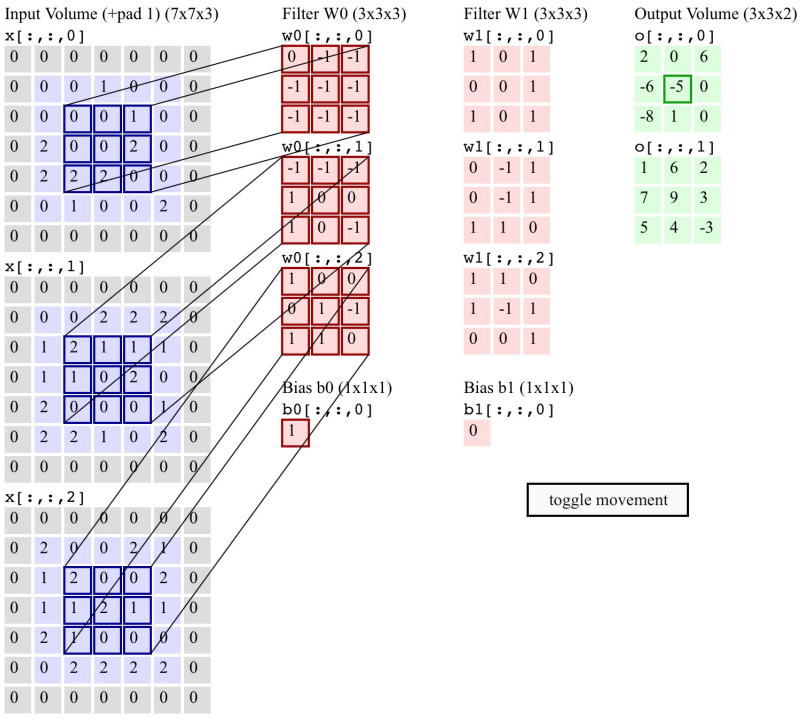



Figure 4: Convolutional Operation Demo

The more interesting thing is that different kernels can have different functionality when performing on a full image. We randomly selected an image from our training set to visualize the functionalities of some possible kernels [24].

This is an example of an identity kernel which leaves our input image unchanged.


Figure 5: Identity Kernel Example

This is an example of an outline kernel, which is also called an "edge" kernel. It is used to highlight large differences in pixel values. If a pixel next to neighbor pixels with close to the same intensity, then it will appear black in the new output image while the pixel next to neighbor pixels that differ strongly will appear white. This kernel can be used to detect edges.

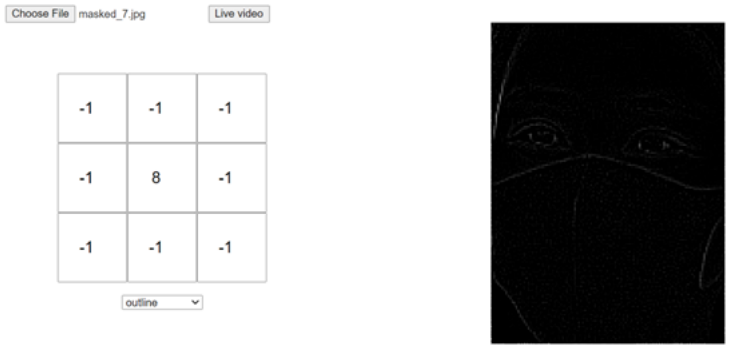



Figure 6: Edge Kernel Example

This is an example of a sharpen kernel which emphasizes differences in adjacent pixel values.


Figure 7: Sharpen Kernel Example

- MaxPooling Layer: The pooling function replaces the output of the net at a certain location with summary statistics of its nearby outputs. Max Pooling reports the maximum output within a rectangular neighborhood. The hyperparameter stride specifies how much we move the convolution filter at each step and then allows pooling with downsampling. Pooling layer is invariant to translation, which is meaningful for our project since we care more about whether some feature is present, rather than its' specific location.
- Dropout Layer: A computationally inexpensive but powerful method of regularizing a broad family of models.
- Flattening Layer: Used to reshape images into a single dimension array
- 2 well-connected Dense Layers

These features result in CNN model having the following advantages:

- Sparse interactions (or sparse connectivity, sparse weights): In regular fully connected layers, every output unit interact with every input unit. In CNN layers, sparse interactions is accomplished by having a small convolution kernel size.
- Parameter sharing: Instead of learning a separate set of parameters for every location, CNN learns only one set. This is because the same

edges appear everywhere in an image, and it is practical for the model to share parameters across the entire image.

- Equivariant representations: Equivariance means that if the input changes, the output changes accordingly. In CNN, parameter sharing leads to CNN being translation invariant. We note that this is not equivalent to equivariance under rotation. Rotating the images will result in the CNN to learn different feature maps, which may not be what we want to achieve.

### 6.2.3 Building our CNN model

We first build CNN with 2 Convolutional Layers. Throughout the training process, we implemented dropout as a simple and effective regularisation method to control the capacity of our CNN model to prevent overfitting.

We first used a Grid Search to investigate all possible combinations of the following hyperparameters:

- filter size of 16 and 32 (for both convolutional layers)
- kernel_size 2 and 4 (for both convolutional layers)
- pool_size of 2 and 4 (for MaxPooling layer)
- strides of 2 and 3 (for MaxPooling layer)
- dropout_rate of 0.2 and 0.4 (for both Dropout Layers)
- 16 and 32 dense nodes (for well-connected Layers)

Due to lack of computing resources, we limited the tuning process to contain only 2 values per hyperparameter.

Overall 64 models will be compiled in this step. Among the 64 models, we selected two models with the same best test accuracy being 97.6%. We found that the best model is:

- filters: 16, kernel_size: 4, pool_size: 4, strides: 2, dropout_rate: 0.4, dense nodes: 32

## 6.3 Transfer Learning

Conventional machine learning and deep learning algorithms, so far, have been traditionally designed to work in isolation. These algorithms are trained to solve specific tasks, and models have to be rebuilt from scratch once the feature space distribution changes. Transfer learning is the idea of overcoming the isolated learning paradigm and utilising knowledge acquired for one task to solve related ones.

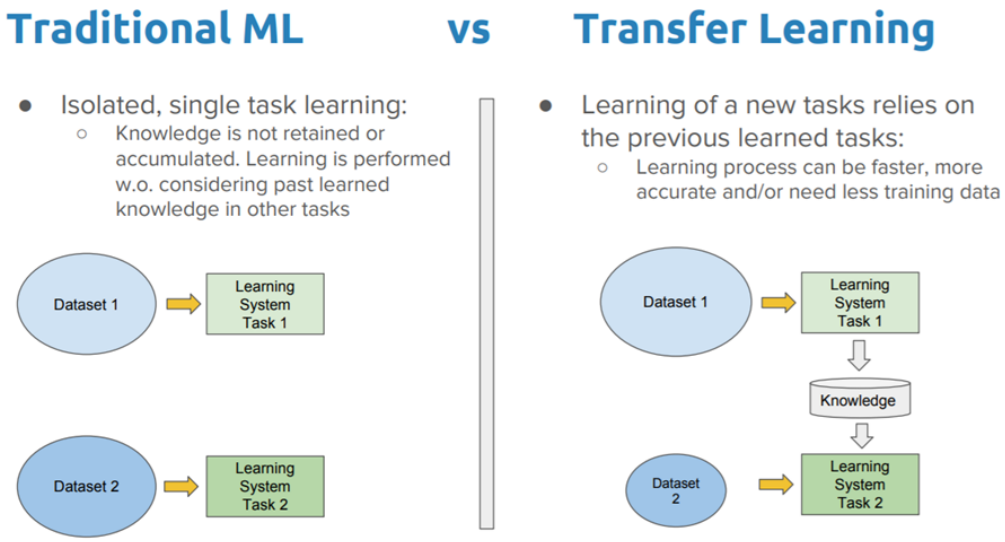



Figure 8: Traditional ML vs Transfer Learning

This is useful as the learning process of transfer learning is known to be faster and easier than training a model from scratch with the usage of pre-trained models[8].

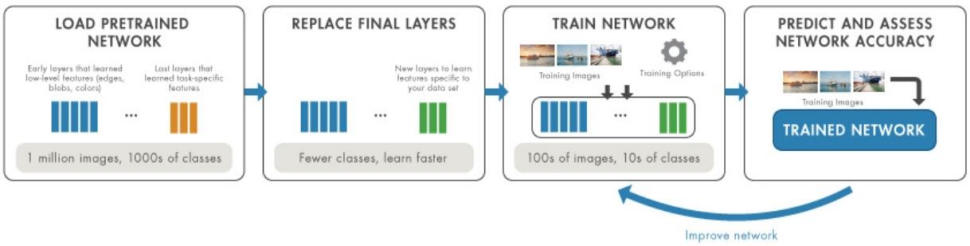



Figure 9: Transfer Learning Workflow

### 6.3.1 Definition

While the problem in our project is a multinomial image classification problem, we will follow the definition mentioned in an article by Pan and Yang[18].

A domain $D$ consists of a feature space $\chi$ and a marginal probability distribution $P(X)$ over the feature space, where $X = x_1, \ldots, x_n \in \chi$. For our image classification problem, $\chi$ is the space of all image representations, $x_i$ is the $i^{th}$ column vector of pixels corresponding to some image, and $X$ is the set of images used for training.

Given a domain, $D = \{\chi, P(X)\}$, a task $\tau$ consists of a label space $Y$ and a conditional probability distribution $P(Y \mid X)$ that is typically learned from the training data consisting of pairs $x_i \in X$ and $y_i \in Y$. In our image classification example, $\gamma$ is the set of all labels i.e. face, masked, face shields, and $y_i$ is either face, masked or face shields.

Given a source domain $D_S$, a corresponding learning task $\tau_S$, a target domain $D_T$ and learning task $\tau_T$, the objective of transfer learning now is to enable us to learn the target conditional probability distribution $P(Y_T \mid X_T)$ in $D_T$ with the information gained from $D_S$ and $\tau_S$ where $D_S \neq D_\tau$ or $\tau_S \neq \tau_T$.

As both the domain $D$ and the task $T$ are defined as tuples, these inequalities give rise to four transfer learning scenarios, which we will discuss below.

### 6.3.2 Transfer Learning Scenarios

1. $\chi_S \neq \chi_T$. The feature spaces of the source and target domain are different. For example, this occurs in translated learning, where we

want to learn a mapping function from images to text.

2. $P(X_S) \neq P(X_T)$. The marginal probability distribution distributions of source and target domain are different. This scenario is known as domain adaptation. In computer vision problems, even if the training and the test data looks the same, the training data may still contain a bias that is imperceptible to humans, but which the model will exploit to overfit on the training data. For example, some images may be professional photographs, while some may be amateur snapshots from the Internet[19].

3. $\chi_S \neq \chi_T$. The label spaces between the two tasks are different, e.g. images need to be assigned different labels in the target tasks. In practice, this scenario usually occurs with scenario 4, as it is extremely rare for two different tasks to have different label spaces, but exactly the same conditional probability distributions.

4. $P(Y_S \mid X_S) \neq P(Y_T \mid X_T)$. The conditional probability distributions of the source and target tasks are different, e.g. source and target images are unbalanced with regards to their classes.

### 6.3.3 VGG16

In our project, we explore the use of VGG16, a CNN model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" [9]. The VGG16 architecture consists of 12 convolutional layers, some of which are followed by maximum pooling layers and then 3 fully connected layers, and finally a $1,000$-way softmax classifier. A summary of the specific features can be seen in Fig 6.3.3. As we only have 3 classes in our problem, we changed the last layer to a softmax classifier with 3 classes.

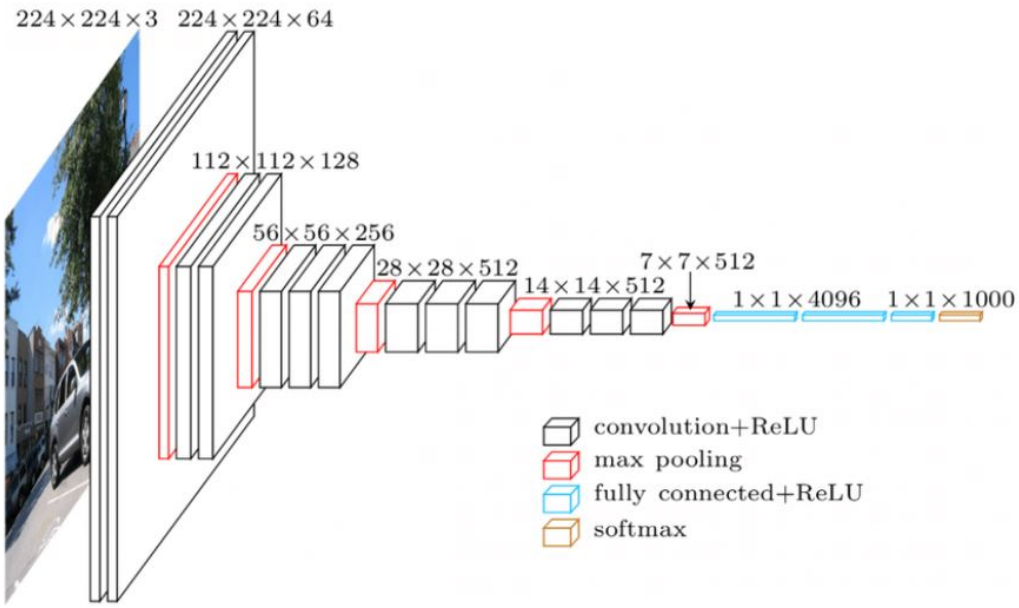



Figure 10: VGG16 Network

We have following hyperparameters in our model:

- epochs: One run of the gradient descent algorithm through the entire training set. [10, 20] (we only consider small epochs in order to reduce training time)
- batch_size: Instead of summing the loss function over the entire training set, we sum it over a small subset called a batch. [32, 64, 128]
- optimizer: [SGD, Adam, RMSprop]
- learning_rate: [0.0001, 0.001, 0.01]
- dropout_rate: Dropout rate can be understood as a rate/probability that the link of a node from the previous layer and the node from the next layer will be terminated. In other words, this will reduce some parameters to be estimated.[0.2, 0.4, 0.6]
- kernel_initializer: Initializers define the way to set the initial random weights of layers. [uniform, zero, normal]

- kernel_constraint(maxnorm): It constrains the weights incident to each hidden unit to have a norm less than or equal to a desired value. [50, 100, 500]

| Layer | | Feature Map | Size | Kernel Size | Stride | Activation |
|---|---|---|---|---|---|---|
| Input | Image | 1 | 224 x 224 x 3 | - | - | - |
| 1 | 2 X Convolution | 64 | 224 x 224 x 64 | 3x3 | 1 | relu |
| | Max Pooling | 64 | 112 x 112 x 64 | 3x3 | 2 | relu |
| 3 | 2 X Convolution | 128 | 112 x 112 x 128 | 3x3 | 1 | relu |
| | Max Pooling | 128 | 56 x 56 x 128 | 3x3 | 2 | relu |
| 5 | 2 X Convolution | 256 | 56 x 56 x 256 | 3x3 | 1 | relu |
| | Max Pooling | 256 | 28 x 28 x 256 | 3x3 | 2 | relu |
| 7 | 3 X Convolution | 512 | 28 x 28 x 512 | 3x3 | 1 | relu |
| | Max Pooling | 512 | 14 x 14 x 512 | 3x3 | 2 | relu |
| 10 | 3 X Convolution | 512 | 14 x 14 x 512 | 3x3 | 1 | relu |
| | Max Pooling | 512 | 7 x 7 x 512 | 3x3 | 2 | relu |
| 13 | FC | - | 25088 | - | - | relu |
| 14 | FC | - | 4096 | - | - | relu |
| 15 | FC | - | 4096 | - | - | relu |
| Output | FC | - | 1000 | - | - | Softmax |

Figure 11: Summary of VGG16 Architecture

To prevent our model from overfitting, we added a dropout layer in the second last layer with rate 0.4 and we also used maxnorm weight constraints to regularize our model. This method seems to work especially well in combination with a dropout layer [12]. As we were unable to run a proper Grid Search due to extremely long processing times, we did a trial and error to determine the optimal values of our hyperparameters and found that the accuracy of our model is 99%.

### 6.3.4 Residual Networks

While shallow networks may be suitable for small datasets [14], very deep convolutional networks is better for classification accuracy for larger datasets.

For large datasets, the models will be able to generalise better to a wide range of tasks and datasets with deeper networks, matching or outperforming more complex recognition pipelines built around less deep image representations [9] [13]. However, very deep neural networks are difficult to train due to problems relating to vanishing gradients. While this problem have been largely solved via normalization and stochastic gradient descent optimization methods, it was observed that accuracy gets saturated and degrades rapidly as the network depth increases [15]. Theoretically, this should not happen, as the layers can simply learn the identity function to make the deeper layers useless.

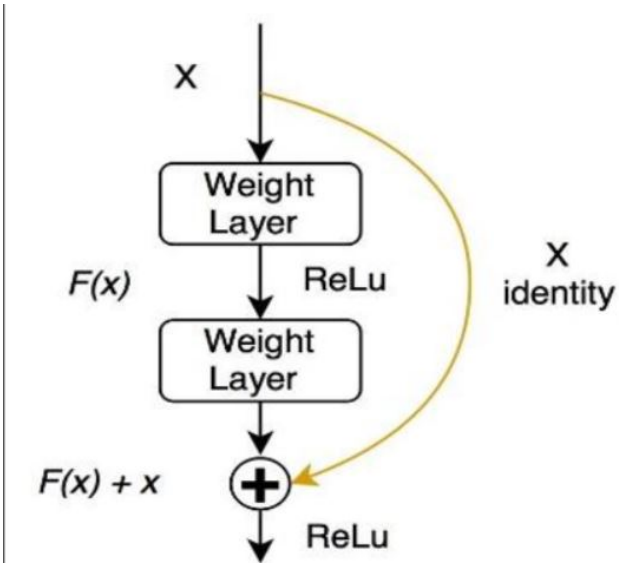



Figure 12: Single Residual Block

Residual networks circumvents this by taking the activation from one layer and feed it to another layer much deeper in the network. Let us consider a neural network block with input $x$, and we will like to learn the true distribution $F(x)$. Let us denote the difference (or residuals) between the true distribution and $x$ as :

$$F(x) = H(x) - x$$

Rearranging, we get,

$$H(x) = F(x) + x$$

While the layers in a traditional neural network learns the output $H(x)$, the layers in a residual network learns the residual $F(x)$, and can now learn the identity function by simply setting residuals to 0. By relying on skip connections, our network will now be able to propagate larger gradients to initial layers, which will learn as fast as the deeper layers, giving us the ability to train deeper networks.

During training, Residual Networks makes the network dynamic by allowing it to either train the layers in the residual blocks, or skip the training for those layers using skip connections. Layers that are not useful and do not add value in overall accuracy may be skipped by the network. Depending on the training data points, different parts of the networks will be trained at different rates based on how the error flows backwards in the network, which can be thought of training an ensemble of different models on the dataset and getting the best possible accuracy[17].

For our project, we used the ResNet-50 model, and applied a dropout layer along with various data augmentation such as translations and rotations as a form of regularization to prevent overfitting. As ResNet-50 is essentially a CNN with skip layers, they have the same hyperparameters, with the inclusion of skip layers. Through trial and error, we found that the accuracy of our final model is 100%.

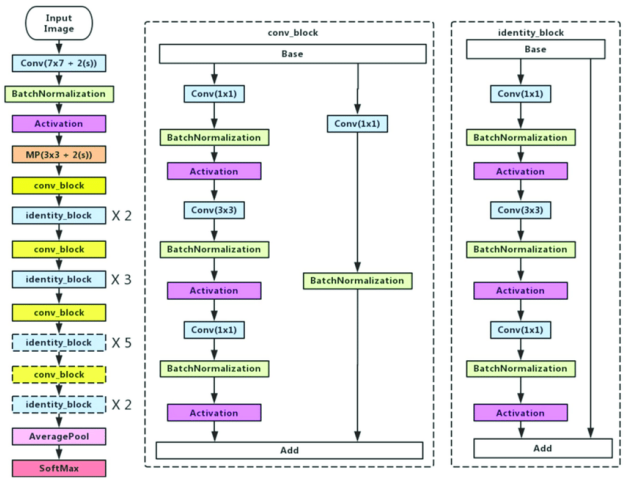



Figure 13: ResNet-50 Architecture

## 6.4 Support Vector Machine (SVM)

SVM a is widely used for classification tasks as they tend to perform well in a variety of problem domains. It performs classification by constructing hyperplanes that divide the cases of different class labels. As we have three classes, namely face images, masked face images and face shield images. We would have to construct hyperplanes in best divides our dataset into the three respective cases. Hence, we will use a multi-class SVM for this task.

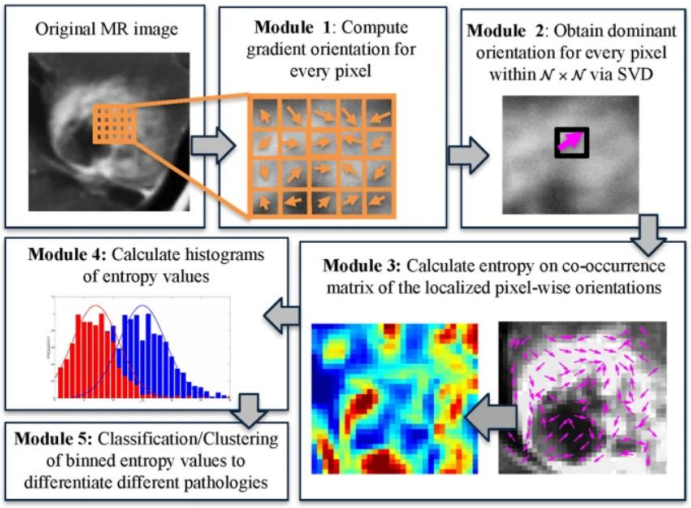



Figure 14: HOG Workflow

We will also attempt to use HOG(histogram of oriented gradients)-based SVM to identify if there can be evidence of mask or face shield to be found in the observations. HOG utilizes the edges, or outline of an object to describe the content of an image. Edges can be described in terms of the gradient and direction of each pixel in an image. [10].

HOG first calculates the gradient magnitude of every pixel in the image. Gradients of a pixel are the change in the x and y directions. The diagrams below shows a patch taken from an image and the pixel matrix of this patch.

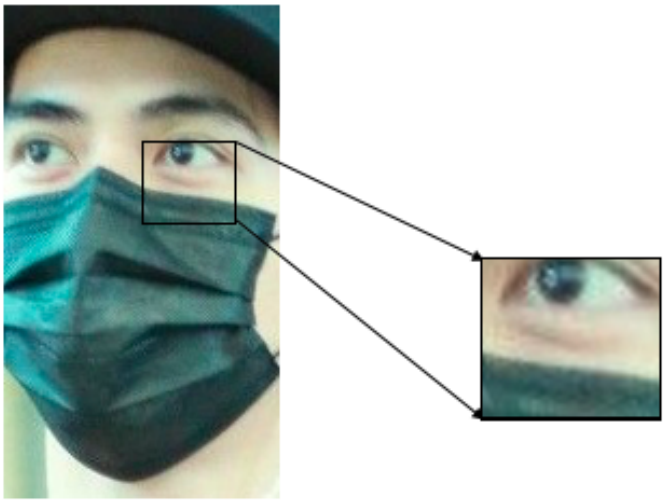


Figure 15: A patch from an image

| | | | | |
|---|---|---|---|---|
| 130 | 20 | 80 | 101 | 133 |
| 71 | 80 | 84 | 90 | 95 |
| 190 | 214 | 56 | 200 | 211 |
| 166 | 157 | 71 | 122 | 167 |
| 201 | 75 | 88 | 104 | 40 |

Figure 16: Pixel matrix

For the highlighted pixel, the respective change in direction are:

$$G_x = 200–214 = 14$$

$$G_y = 84–71 = 13$$

The magnitude of the highlighted pixel G will then be:

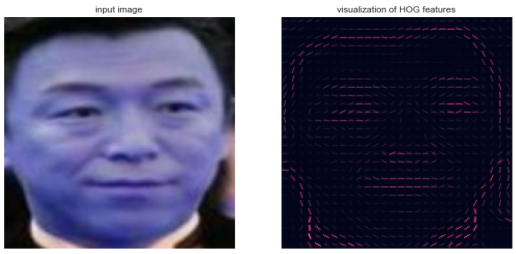
$$G = \sqrt{G_x^2 + G_y^2} = 19.105$$

This magnitude would be higher when there is a sharp change in intensity, such as around the edges. Now, to find the direction $\theta$ of the pixel, we have:

$$\theta = \arctan(G_y/G_x) = 42.88^\circ$$

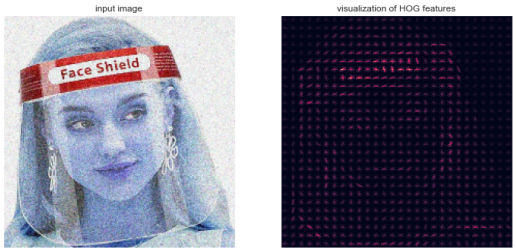
Based on the gradient magnitude and direction of each pixel, we can then generate and visualize a HOG (as seen in the figure 10), which gives us a clear outline of the image. For our SVM model, we would like to to find masks and face shields in an image. We then picture the outline ("edges") of a person's face compared, the outline of a mask or a face shield.

Before applying HOG, we used a Gaussian Naive Bayes as our baseline model to see how much HOG improved our results. The GaussianNB model gave an accuracy of 91.9%, while HOG gives us an accuracy of 96.83%.




(a) HOG Face Image


(b) HOG Masked Face Image




(c) HOG Face Shield Image

Figure 17: HOG Images

## 6.5 Random Forest

Random Forest is an ensemble learning method for both classification and regression tasks, constructed by training several trees in parallel and uses the majority decision of the trees as the final decision of the random forest model. Each decision tree is trained on random subsamples of dataset with various subsamples of the available features.

The hyperparameters in a Random Forest are:

- n_estimators: number of trees in the forest in the model
- max_depth: specifies maximum depth of each tree
- min_samples_split: specifies the minimum number of samples required to split an internal leaf node
- min_samples_leaf: specifies the minimum number of samples required to be at a leaf node.

Without hyperparameter tuning, we obtained a baseline accuracy of 92.5%.

Using grid search to tune our hyperparameters, we found out that the optimal values of hyperparameters were as follows:

Table 1: Best Hyperparameters for Random Forest

| **Hyperparameters** | **Value** |
|---|---|
| n_estimators | 73 |
| min_samples_split | 10 |
| min_samples_leaf | 2 |
| max_depth | 76 |

After tuning the hyperparameters via cross validation, we found out that the accuracy increased to 95.9%.

# 7 Summary of Results

Table 2: Summary of Results

| **Model** | **Accuracy** |
|---|---|
| DNN | 94.0% |
| CNN | 97.6% |
| VGG-16 | 99.0% |
| ResNet | 100% |
| Gaussian Naive Bayes | 91.9% |
| Histogram of Gradients | 96.8% |
| Random Forest | 95.9% |

We observe that our ResNet model vastly outperforms the other models, having an accuracy of 100% on our test set.

# 8 Conclusion

While our project mainly focused on classifying images with people wearing masks, face shields, or neither, this can be extended to images with people wearing both face shields and face masks, and images without people, or even identifying specific individuals wearing face masks or face shields. This will be extremely useful in the security industry, particularly since facial recognition systems embedded in many security systems around the world have been affected with the increased usage of masks.

Due to a lack of computing resources, the structure of our models are relatively simple, and hyperparameters for most models were mainly tuned via trial and error. More analysis and experimentation can be done via Grid Search cross validation with a GPU computing environment.

For future projects, different regularization techniques and models, such as cutout regularization, capsule networks, and Generative Adversarial Networks can be explored and compared with existing, common image classification models such as CNN.

# References


[1] World Health Organization (2020). Retrieved from CLICK ME. [Online; accessed 25-Oct-2020]

[2] Zhixing Zhu, Xihua Lian, Xiaoshan Su, Weijing Wu, Giuseppe A. Marraro, Yiming Zeng. (2020). *From SARS and MERS to COVID-19: a brief summary and comparison of severe acute respiratory infections caused by three highly pathogenic human coronaviruses*, Respiratory Research 21, 224.

[3] Eikenberry et al. (2020). *To mask or not to mask: Modeling the potential for face mask use by the general public to curtail the COVID-19 pandemic*, Infectious Disease Modelling, 5, pp. 293–308

[4] C. Raina MacIntyre, Abrar Ahmad Chughtai. (2020). *A rapid systematic review of the efficacy of face masks and respirators against coronaviruses and other respiratory transmissible viruses for the community, healthcare workers and sick patients*, International Journal of Nursing Studies, Volume 108, 103629.

[5] Claire Felter, Nathalie Bussemaker. (2020). *Which Countries Are Requiring Face Masks?* Retrieved from https://www.cfr.org/in-brief/which-countries-are-requiring-face-masks

[6] Mei Ngan, Patrick Grother, Kayee Hanaoka. (2020). *Ongoing Face Recognition Vendor Test (FRVT) Part 6A: Face recognition accuracy with masks using pre-COVID-19 algorithms*, National Institute of Standards and Technology.

[7] D.Lu, Q.Weng. (2007). *A survey of image classification methods and techniques for improving classification performance*, International Journal of Remote Sensing, Vol. 28, No. 5, 10 March 2007, pp. 823–870

[8] Anand Borad. (2019). *How Does Transfer learning Speeds Up Deep Learning Projects?*. Retrieved from https://www.einfochips.com/blog/how-does-transfer-learning-speeds-up-deep-learning-projects/

[9] Karen Simonyan, Andrew Zimmerman. (2015). *Very Deep Convolutional Networks For Large-Scale Image Recognition*. Visual Geometry Group, Department of Engineering Science, University of Oxford. Retrieved from https://arxiv.org/pdf/1409.1556.pdf

[10] Navneet Dalal, Bill Triggs (2005). *Histograms of oriented gradients for human detection*. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) Retrieved from https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

[11] Daniël M. Pelt, James A. Sethian. (2017). *A mixed-scale dense convolutional neural network for image analysis*. Proceedings of the National Academy of Sciences. 115 (2) pp 254-259. Retrieved from https://www.pnas.org/content/115/2/254

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research 15 (2014) 1929-1958. Retrieved from http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

[13] Hrushikesh N. Mhaskar, Tomaso Poggio. (2016). *Deep vs. Shallow Networks: an Approximation Theory Perspective*. Retrieved from https://arxiv.org/pdf/1608.03287.pdf

[14] Alexander Schindler, Thomas Lidy, Andreas Rauber. (2016). *Comparing Shallow versus Deep Neural Network Architectures for Automatic Music Genre Classification*. Retrieved from http://ceur-ws.org/Vol-1734/fmt-proceedings-2016-paper2.pdf

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. (2015). *Deep Residual Learning for Image Recognition*. Retrieved from https://arxiv.org/pdf/1512.03385.pdf

[16] Kaiming He, Jian Sun. (2014). *Convolutional Neural Networks at Constrained Time Cost*. Retrieved from https://arxiv.org/abs/1412.1710

[17] Sabyasachi Sahoo. (2018). *Residual blocks — Building blocks of ResNet*. Retrieved from https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec

[18] Sinno Jialin Pan, Qiang Yang. (2009). *A Survey on Transfer Learning*. Retrieved from https://www.cse.ust.hk/~qyang/Docs/2009/tkde_transfer_learning.pdf

[19] Antonio Torralba, Alexei A. Efros. (2011). *Unbiased Look at Dataset Bias*. Retrieved from https://people.csail.mit.edu/torralba/publications/datasets_cvpr11.pdf

[20] Fei-Fei Li, Ranjay Krishna, Danfei Xu. (2020) *CS231n Convolutional Neural Networks for Visual Recognition*. Retrieved from https://cs231n.github.io/convolutional-networks/

[21] Ian Goodfellow, Yoshua Bengio, Aaron Courville. (2016). *Deep Learning*, MIT Press, pp. 326–339. Retrieved from http://www.deeplearningbook.org

[22] Waseem Rawat, Zenghui Wang. (2017). *Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review*. Neural Computation, Vol 29, Issue 9, pp.2352-2449. Retrieved from http://www.deeplearningbook.org

[23] James Le. (2018). *The 4 Convolutional Neural Network Models That Can Classify Your Fashion Images*. Retrieved from CLICK ME.

[24] Victor Powell. (n.d.) *Image Kernels Explained Visually*. Retrieved from https://setosa.io/ev/image-kernels/

[25] Wikipedia. (n.d.). *Convolutional neural network*. Retrieved from https://en.wikipedia.org/wiki/Convolutional_neural_network

[26] Piotr Skalski. (2019). *Gentle Dive into Math Behind Convolutional Neural Networks*. Retrieved from https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9