**CV PROJECT: FACE MASK DETECTION**

Group Members:-

Manasi Amdekar (J003)

Sameeksha Kini (J029)

Nehal Nahta (J039)



**Abstract**

Ever since the beginning of covid, a mask has become as necessary as having a purse. Thus, we took upon the problem to detect whether or not a person is wearing a mask and whether they are wearing it properly or not. We’ve used models like CNN, resnet34, resnet50, resnet152 and mobilenet for detecting the presence of a mask on a face. The dataset that we used can be found on kaggle and the link for the same has been provided in a later section.

To start with solving the problem, we have to connect the files with annotations for the images(xml files) with the actual image files.

**Introduction**

We have taken up the face mask detection dataset and applied various classification methods to find out whether the person is wearing the mask, not wearing the mask or incorrectly wearing the mask.

**Dataset Details**

Masks play a crucial role in protecting the health of individuals against respiratory diseases.Especially during our current crisis of COVID-19, it is one of the few precautions which are available in the absence of immunisation. Using this dataset, it is possible to create a model to detect whether or not people are wearing masks, and if they are, whether they are wearing those masks improperly.

The link for the dataset: https://www.kaggle.com/andrewmvd/face-mask-detection

This dataset contains 853 images belonging to the 3 classes, as well as their bounding boxes in the PASCAL VOC format.

The 3 classes present in the dataset are:

* With mask
* Without mask
* Mask worn incorrectly

With Mask Without Mask Mask worn incorrectly

**Methodology**

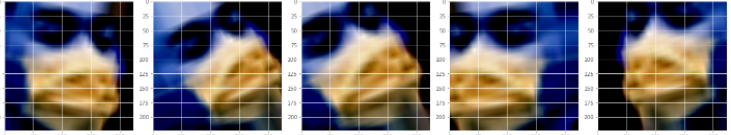
Image classification involves teaching an Artificial Intelligence (AI) how to detect objects in an image based on their unique properties.It analyses an image and identifies the 'class' the image falls under. A class is essentially a label, for instance, 'mask', 'no mask' and 'incorrectly worn'. The objective of image classification is to identify and portray, as a unique grey level (or colour), the features occurring in an image in terms of the object or type of land cover these features actually represent on the ground. There are certain types of classification like single label, multilabel classification etc.

We’ve implemented resnet50, resnet152, resnet34, CNN and mobilenet.

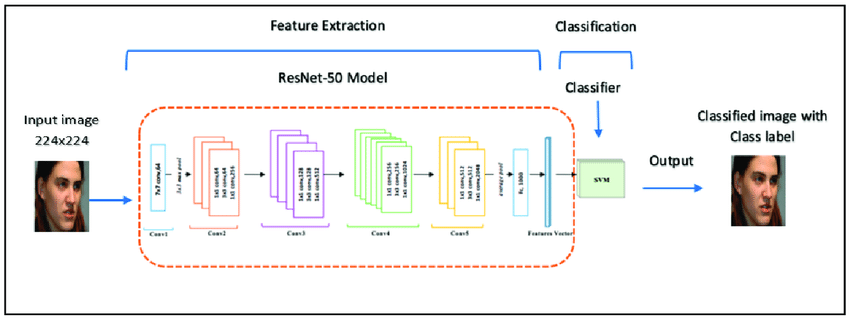
Augmented dataset:

We have used an image data generator for augmenting the dataset.

Here’s an example of how augmented images looks like:-



**ResNet**



* Importing Libraries: We start by importing all the necessary libraries
* Extraction of Images and Annotations: We used the OS module in Python for creating and removing a directory, fetching its contents, changing and identifying the current directory, etc. We use os.walk() along with os.path.join().

What os.walk() does is returns a generator which creates a 3-tuple ie:-

* + dirpath :- It is a string, which provides the path to the directory.
  + dirnames(in dirpath) : - It is a list of the names of the subdirectories in dirpath (excluding '.' and '..')
  + filenames(in dirpath):- It is a list of the names of the non-directory files in dirpath.

So everytime a generator is called, it goes through every directory one at a time i.e. recursively until no other sub-directories are available from the initial directory that walk was called upon.

What os.path.join(dirpath, name) does is get the full path to a directory or file in dirpath.

All the images are stored in img\_names list and annotations in xml\_names list.

* Visualisation and Analysis of Target Class: Target class belongs to with\_mask, mask\_weared\_incorrect and without\_mask. We plot graphs to see the percentage and the number of images which each target class has. We then get to know that the percentage of each target class is:-
  + with\_mask : 79.37 %
  + mask\_weared\_incorrect : 3.02 %
  + without\_mask : 17.61 %
* Images Identification with Target Class: Since there are 3 target classes, we can use 3 colours for cascading the face.
  + with\_mask : green
  + mask\_weared\_incorrect : yellow
  + without\_mask : red
* Image Preprocessing: For image Preprocessing we have used PyTorch sensors. We define the pipeline of basic data preprocessing using the **transforms** function of torchvision. A tensor is an algebraic object which describes a multilinear relationship between multiple sets of algebraic objects related to a vector space. The dataset created will be of tensor.
  + **xmltodict.parse()** is used to parse the given XML input and convert it into a dictionary. The input taken can either be a string or a file-like object. Here we are using read() method, which reads at most n bytes from file descriptor and returns a string containing the bytes read. If fd has reached the end of the file, an empty string is returned.
  + **transforms.functional.crop()** crops the given image at specified location and output size and it returns torch.Tensor.
  + **torchvision.transforms.Compose() :** torchvision.transforms is used for common image transformations. Compose is used to group several transforms together.
  + **torch.tensor()**: infers the dtype automatically. It always copies the data and torch.tensor(l) is equivalent to l.clone().detach().
  + **transforms.Resize()** : The default interpolation is InterpolationMode.BILINEAR. It resizes the input image depending on the height and width provided.
  + **transforms.ToTensor()** converts the input image to PyTorch tensor.
* Splitting Dataset into Training and Test Set: After we get our preprocessed data, we now split the dataset into training and testing dataset using **torch.utils.data.random\_split()**, which randomly splits the dataset into non-overlapping new datasets of given lengths.
* Samples in Training Set: **DataLoader()** combines a dataset and a sampler along with providing an iterable over the given dataset.
* Model Building: The Residual Network, or ResNet for short, is a model that makes use of the residual module. We are using Resnet34 model here for image recognition.
* Download the resnet34 layers pre-trained model: We run the command **torchvision.models.resnet34()**, and if pretrained is provided as True, it returns a model pre-trained on ImageNet.
* Feature Extraction: We are setting the attribute requires\_grad of the model parameters to false when feature extraction is done.
* Model Details: To view every layer, we run the command ‘model’. In the last line of the output, we see that the out\_features is 1000, so we’ll need to reinitialize the model to be a linear layer with 512 input features and 3 output classes. We do so by using the function **torch.nn.Linear()** that applies a linear transformation on the incoming data.
* Calculating Parameters of Convolutional Layer: We calculate the number of parameters for a 2D convolutional layer.
* Cross Entropy loss: The loss for the model is to be set, then run the training and validation function for the set number of epochs. The default learning rate is not optimal for all of the models, so to achieve maximum accuracy it would be necessary to tune for each model separately.
* Training Model: **param.requires\_grad()**, which is usually by default True for feature extraction, has been set to False. When it is True backpropagation will be there. In order for the model to avoid overfitting, the layers should be freezed and to do so, **requires\_grad** has to be set to False. Here, we are setting freezing layers from 1 to 6.

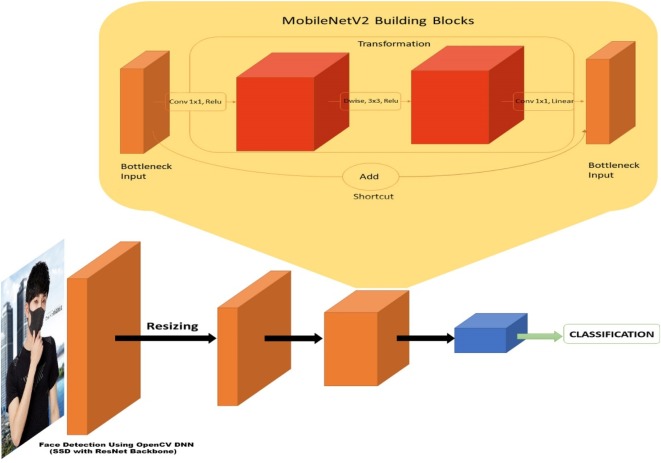
**CNN**

* Importing Libraries: We start by importing all the necessary libraries
* Extraction of Images and Annotations: We used the ElementTree.parse() module in Python for creating and removing a directory, fetching its contents, changing and identifying the current directory, etc. We use os.walk() along with os.path.join().
* Image Preprocessing: For image Preprocessing we have used
* Splitting Dataset into Training and Test Set: After we get our preprocessed data, we now split the dataset into training and testing dataset using **sklearn.model\_selection.train\_test\_split()** which randomly splits the dataset into non-overlapping new datasets of given lengths.
* Samples in Training Set: **DataLoader()** combines a dataset and a sampler along with providing an iterable over the given dataset.
* Model Building: The Convolutional Neural Network, or CNN is a type of neural network for working with images. This type of neural network takes input from an image and extracts features from an image and provides learnable parameters to efficiently do the classification, detection and a lot more tasks..
* Model Details: To view every layer, we run the command **model.summary()**.This function gives us all the details about our model and its specifications for each layer.
* Cross Entropy loss: The loss for the model is to be set, then run the training and validation function for the set number of epochs. The default learning rate is not optimal for all of the models, so to achieve maximum accuracy it would be necessary to tune for each model separately.

**MobileNet**

MobileNet model is a network model using depthwise separable convolution as its basic unit.

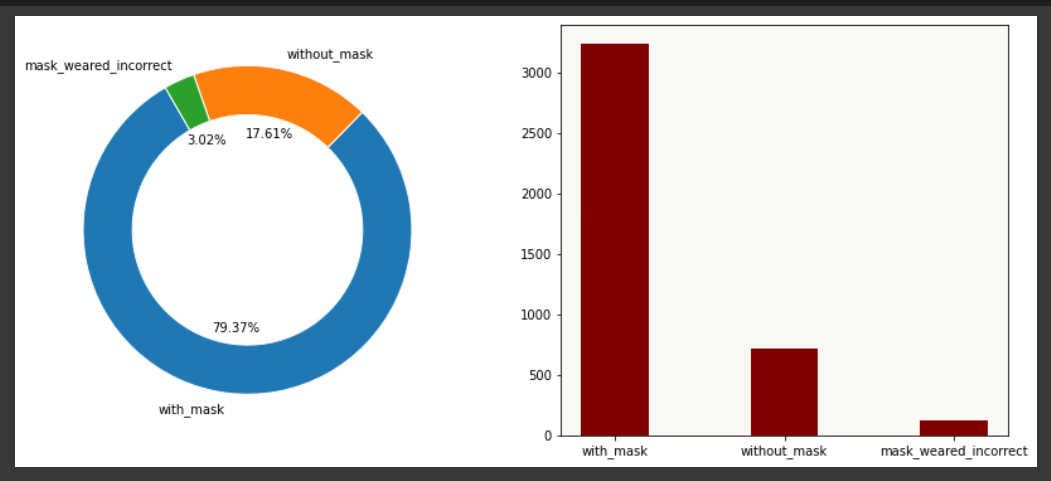
* Image Preprocessing: storied images and annotations into dataframes. Label encoding the labels as 0,1 and 2.
* Model Building: We use 2 Dense layers along with the baseline model.



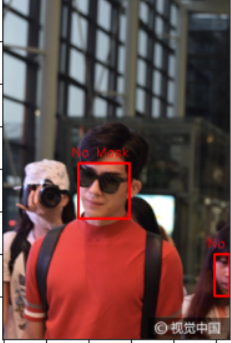
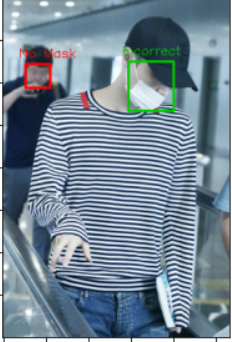
* Loss: Cross Entropy loss. The loss for the model is to be set, then run the training and validation function for the set number of epochs.

**Analysis**

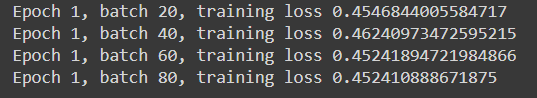
Given below is the percentage as well as the number of the target classes i.e. with\_mask, mask\_weared\_incorrect and without\_mask.



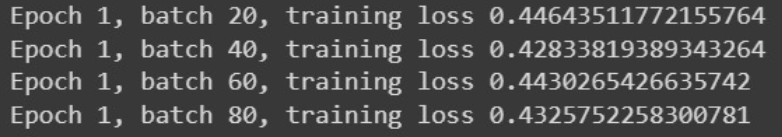
**Results**

****

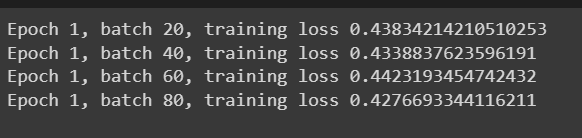
Result for Resnet34:



Result for Resnet50:



Result for Resnet152:

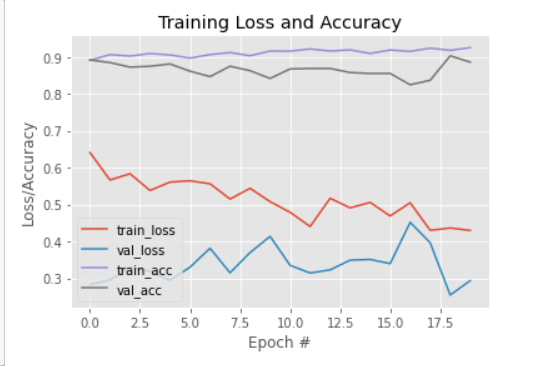


Result for CNN:



Result for MobileNet:





Based on the outputs of the various models, we have made a summary in the form of a table:-

| **Model** | **Loss** |
| --- | --- |
| **Resnet 34** | 0.4524 |
| **Resnet 50** | 0.4325 |
| **Resnet 152** | 0.4276 |
| **CNN** | 0.2966 |
| **MobileNet** | 0.4510 |

**References**

* <https://www.ijeat.org/wp-content/uploads/papers/v10i6/F30500810621.pdf>
* <https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8>
* <https://docs.python.org/3/library/os.html#os.walk>
* <https://pytorch.org/docs/stable/index.html>