# CS 7641 Project - Facemask Detection

*Project Website: https://github.gatech.edu/pages/nbadami3/facemask.github.io/*

## Introduction:

The rise of the COVID-19 pandemic has brought about unprecedented changes all around the world. There is currently no treatment or vaccine for coronavirus and our best defense against it is properly wearing a face mask. Unfortunately, it's been observed that a lot of people do not wear their masks properly thus putting their and other's life at risk of catching this deadly virus.

## Problem:

As pandemic fatigue sets in around the country, more and more people are eschewing face mask use. Being able to identify whether a person is properly wearing a mask or not is crucial to identify and prevent a potential outbreak. We propose a face mask detector that can identify whether an individual is wearing a facemask or not.

## Data Collection:

We are using FaceMask 12K dataset on kaggle. The dataset is split into 10,000 in training, 992 in test and 800 in validation and has equal representation of both classes.

(***https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset*. )**

We are not doing any explicit feature selection on the data. We rely on CNNs and VAE to automatically learn these features while training.

The dataset that we obtained is balanced between the two classes so we are not performing any explicit data cleaning.

## Methods:

We propose the use of Convolutional Neural Networks to tackle the problem. We model the problem as a binary classification task, where individuals wearing a mask pertain to positive class and individuals not wearing a mask pertain to negative class. We choose **Pytorch** as our preferred framework to develop the project.

Our methodology is broadly divided into three subtasks:

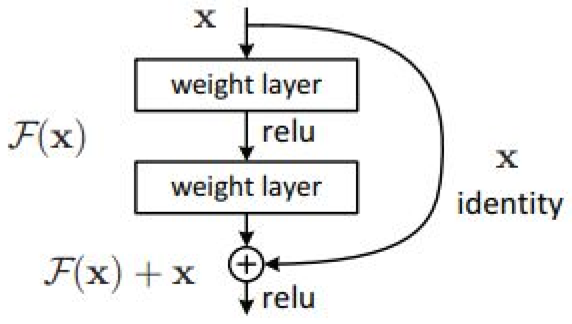
## 1. Supervised Learning:

For the supervised learning task of the assignment, we propose building a deep neural network to perform the classification. Following are various parameters used for building the CNN:

a) Architectures:

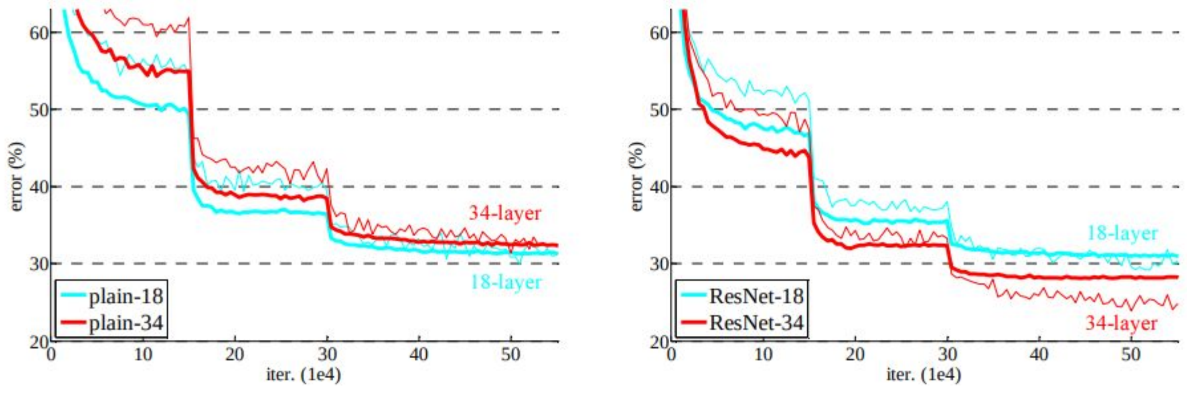
- **ResNet :**

  ResNet is a Convolutional Neural Network model. The major problem that ResNet addresses is how it is not just important to have multiple layers stacked to get better results. In fact, having deep layers gives rise to the vanishing gradient problem and also the accuracy gets saturated and later would degrade. ResNet introduces a residual learning block where an identity mapping is added at the end of the block to solve the degradation problem as follows:


x
weight layer
relu
weight layer
relu
$\mathcal{F}(\mathbf{x})$
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
x
identity

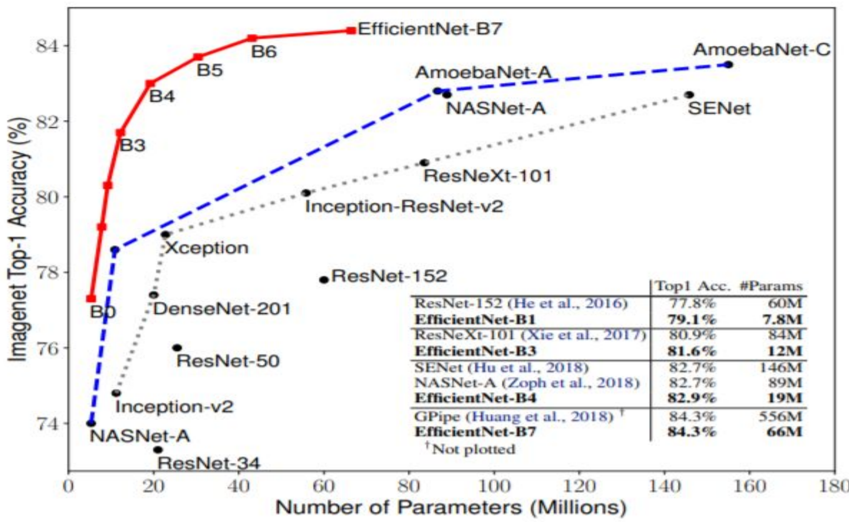
[6]

The hypothesis is that instead of expecting the stacked layer to fit the desired mapping function, an identity mapping is used to let it fit the residual mapping which makes it easier to reduce the error towards zero. This is achieved by introducing shortcut connections between the stacked layers to add the identity mapping. The assumption is that the identity mappings do not introduce additional parameters. It is shown that with this approach, error is reduced with deeper layers:



[6]

- **<u>Efficient Net</u>**

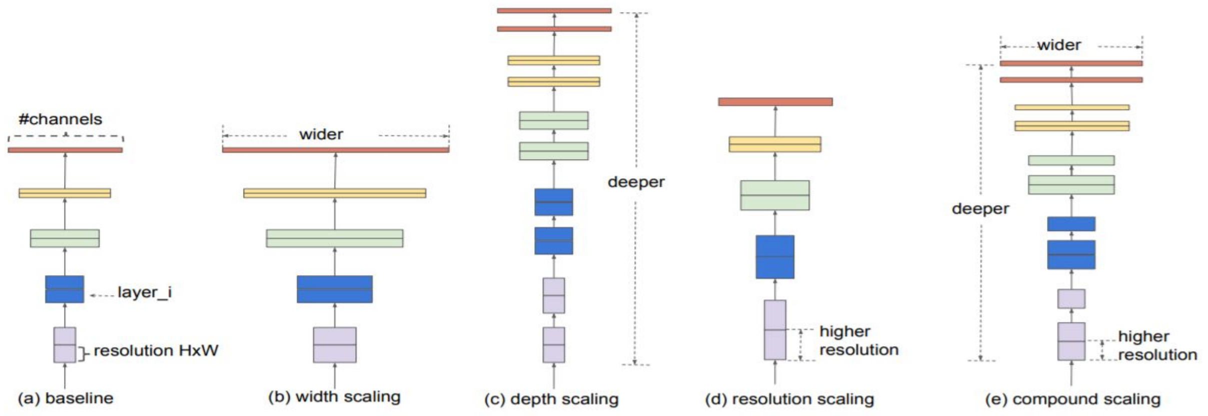


| | Top1 Acc. | #Params |
|---|---|---|
| ResNet-152 (He et al., 2016) | 77.8% | 60M |
| **EfficientNet-B1** | **79.1%** | **7.8M** |
| ResNeXt-101 (Xie et al., 2017) | 80.9% | 84M |
| **EfficientNet-B3** | **81.6%** | **12M** |
| SENet (Hu et al., 2018) | 82.7% | 146M |
| NASNet-A (Zoph et al., 2018) | 82.7% | 89M |
| **EfficientNet-B4** | **82.9%** | **19M** |
| GPipe (Huang et al., 2018) † | 84.3% | 556M |
| **EfficientNet-B7** | **84.3%** | **66M** |

†Not plotted

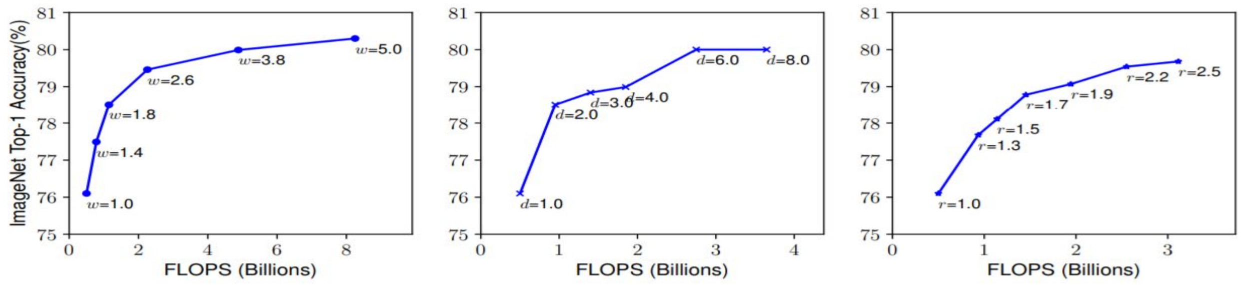
[10]

EfficientNet is a new technique achieving state-of-the-art image classification accuracy from Google AI research. This technique rethinks the way we scale convolutional neural networks. Traditionally, one way to scale up CNN is to add more layers, for example, ResNet18 to ResNet 200. In EfficenetNet, this is done systematically which looks at three properties, width, depth and_resolution.



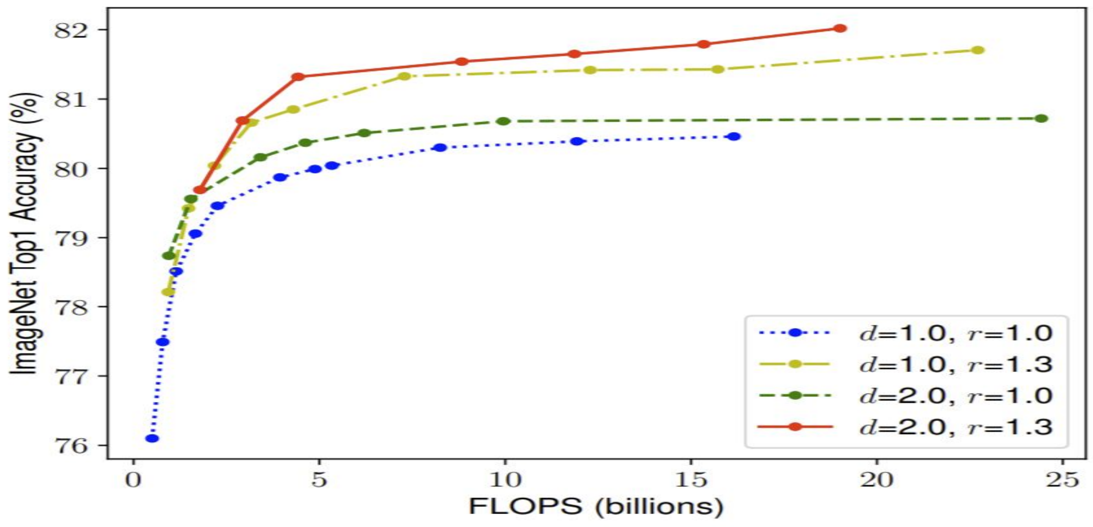
[10]

Width scaling captures fine-grained features by adding more feature maps; depth scaling adds more layers to the CNN and finally resolution scaling is changing the size of the input image. Scaling any of these will improve performance but it quickly plateaus as the images below shows.



[10]

Left is width scaling, middle is depth scaling, and then far right is image resolution scaling, hence performance is gained on each but it quickly saturates. The authors then discovered that there is a synergy in scaling multiple dimensions together. Meaning if you increase depth you will also see a performance boost when you increase image resolution as well. So there is a dependency amongst scaling each of these dimensions.



[10]

After extensive grid search, they derived the theoretically optimal formula for compound scaling using these coefficients. Compound scaling method is the key idea in EfficenetNet which balances the up sampling with Width, Depth and Resolution by scaling with a constant ratio.

b) **<u>Optimizers:</u>**

We experimented with the following optimizers to train our network:

- Adam
- Stochastic Gradient Descent
- ADADELTA
- Average Stochastic Gradient Descent

We have demonstrated the training accuracy curve for the following optimizers in the results section.

c) Activation Functions:

- ReLU
- TanH
- Leaky ReLU

**2. <u>Unsupervised Learning:</u>**

For unsupervised learning, we are experimenting with different architectures of Autoencoder. Specifically, we aim to use Autoencoders for Dimensionality Reduction and map the input observations from a high-dimensional feature space to a low-dimensional feature space without significant information loss. Since autoencoders can capture the non-linear relations between the features, they are a powerful tool to detect subtleties in the input and therefore helps to augment the dataset. This way we can increase the overall training accuracy of our classification task

a) Denoising Autoencoder

In a Denoising Autoencoder (DAE), noise is added stochastically to the input dataset, and then the autoencoder is trained to recover the original, unperturbed signal. The main motivation to use DAE is to improve the robustness of the latent space representation by preventing the autoencoder from simply learning an identity function that can map the output to the input. For our project we aimed to use DAE to filter out the unnecessary features from the input images (like blurring out the background) and further increase the accuracy of the classification task. However, the results were not satisfactory and from visual inspection of the output showed that it will not help to improve the overall classification accuracy.

*Architecture Summary.*

- The encoder has 4, 2d Convolution layers with kernel size 3, and padding 1.
- The decoder has 4, 2d Transpose Convolution layers with kernel size 3, stride 2 and padding 1.
- ReLu is used as the activation function after every hidden layer along with a maxpooling layer.
- MSE loss with Adam optimizer is used to train the model

b) Variational Autoencoder

Due to the poor results from Denoising Autoencoder, we tried using the Variational Autoencoder (VAE) to augment the dataset. VAEs are powerful generative models that minimize reconstruction loss of an encoded image. We believe that augmenting the data with VAEs will help us to capture relevant features of an image. In VAE, the final loss is calculated as the sum of KL divergence and reconstruction loss. The optimization involves minimising the KL divergence and maximizing the expectation of the reconstruction of data points from the latent vector. We re-train our classification model with the augmented data and compare the results with normal data.

Architecture Summary.

Following is the architecture specifications of the VAE we are using now:

- The encoder has 5, 2d Convolution layers with kernel size 3, stride 2 and padding 1.
- The decoder has 5, 2d Convolution layers with kernel size 3, stride 2 and padding 1.
- LeakyReLu is used as the activation function after every hidden layer along with a batch normalization layer.
- BCE loss is used with Adam optimizer to train the model.

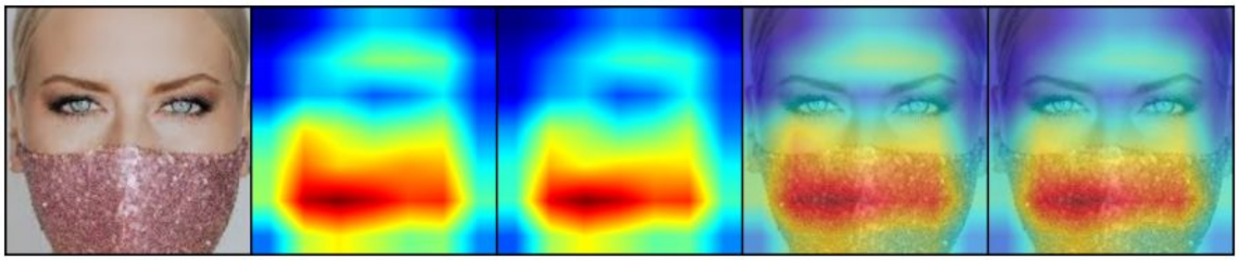
## 3. Visualization:

### Grad-CAM

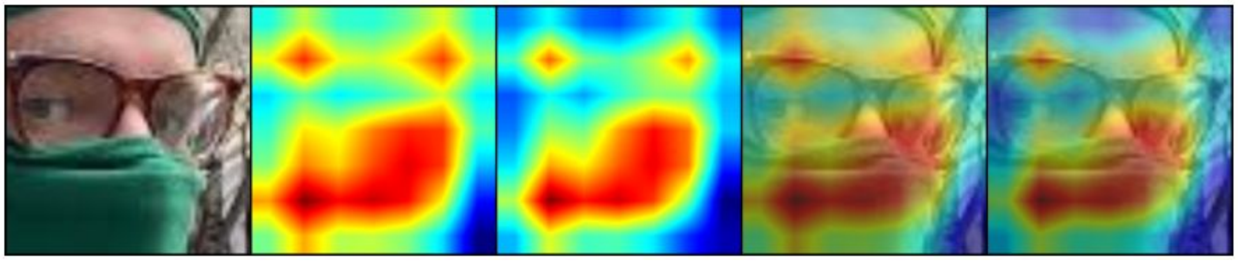
GradCAM[2] is a visualization technique which does not need any re-training for the Convolutional Neural Network based models. It creates a heatmap of features critical to image classification. In the images below, we use GradCAM and GradCAM++, an improvement on the original GradCAM that provides better visual explanation through better object localization. For simplicity, when we refer to GradCAM we are referring to both GradCAM and GradCAM++ unless specified otherwise.

We use a pytorch implementation[4] of GradCAM, maintained by Vicky Liin, for this project. Here, we pass the last convolutional layer of our network to the GradCAM API which then processes the output of that layer to generate the required heatmaps. Below is an example of how GradCAM helps us with the visualization. The following images were obtained from running ResNet with GradCAM.

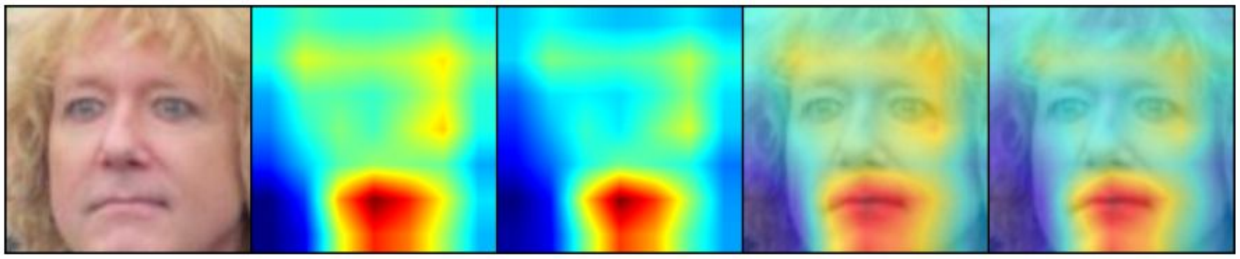
*From left to right the images are as follows: Original Input, GradCAM heatmap, GradCAM++ heatmap, GradCAM heatmap overlaid on the input image, GradCAM++ heatmap overlaid on the input image*



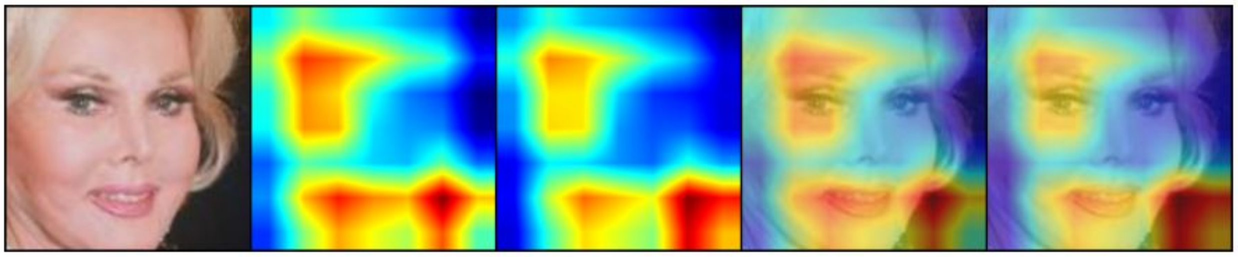
*GradCam with Facemask*



*Gradcam with Facemask with face in an angle*



*GradCam without Facemask*

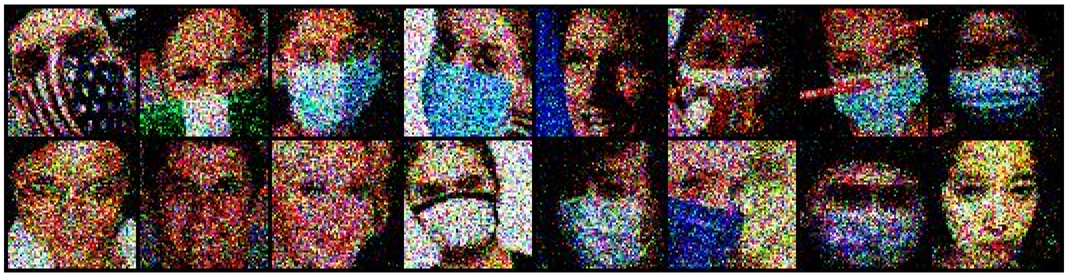


*GradCam with face at an angle*

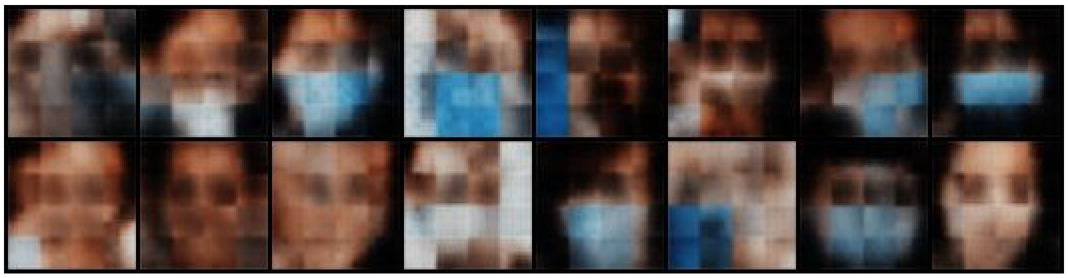
## Results:

- **Denoising Autoencoder:**

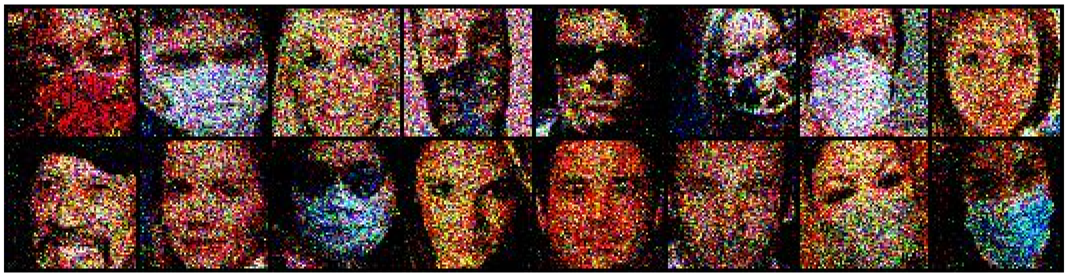
  We trained the Denoising Autoencoder on our dataset and the following are the reconstruction results from the noisy image.



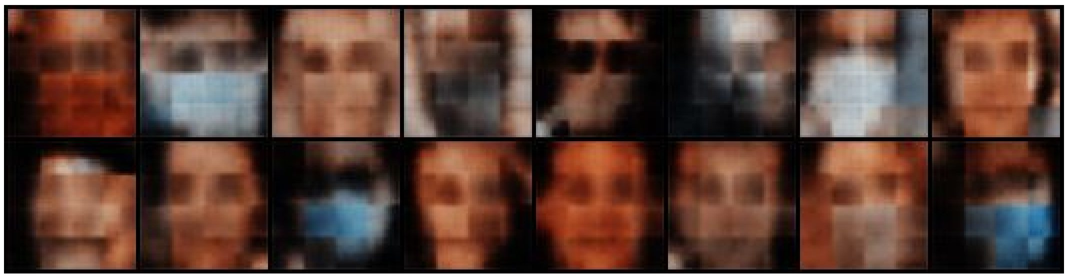
*Noisy Image*



*Reconstructed Image after epoch 3*



*Noisy Image*

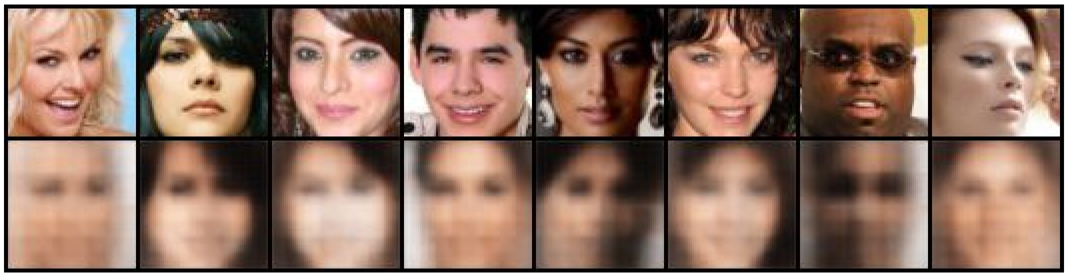


*Reconstructed Image after epoch 10*

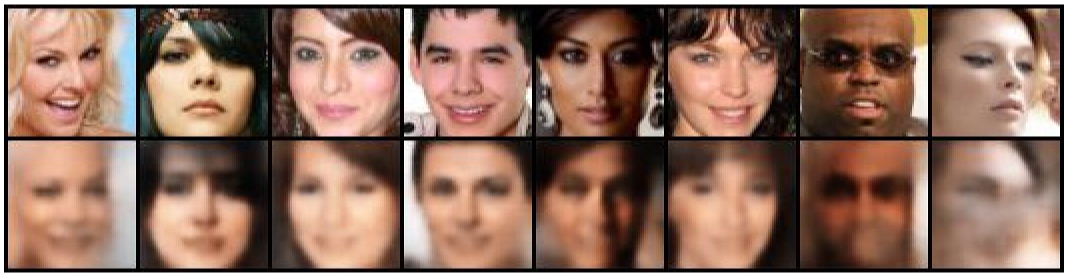
- From the above images, we can see that there has been no considerable improvement in the output from the noisy image. In some of the images, though the mask area has been captured in the reconstructed image. For example, masks with blue color have been captured in both the figures.

- **Variational Autoencoder:**

  We trained VAE models on our dataset and the following are the reconstruction results and training graphs that we obtained.

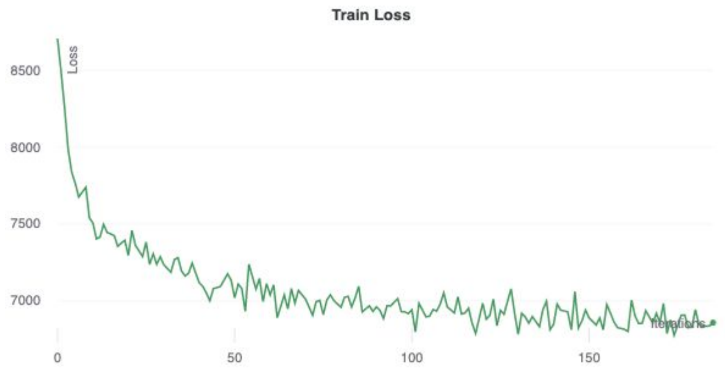
  - 

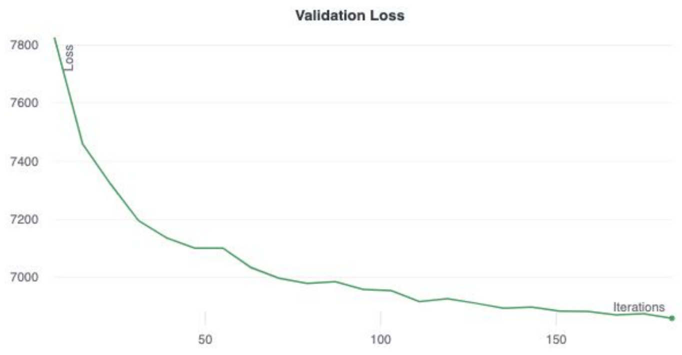
*Reconstruction Results after 3 epochs*



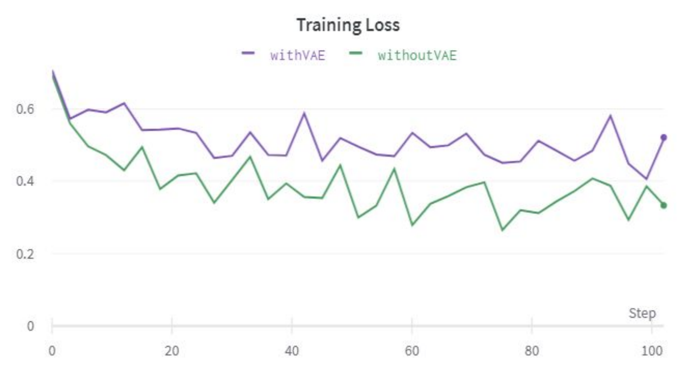
*Reconstruction Results after 30 epochs*

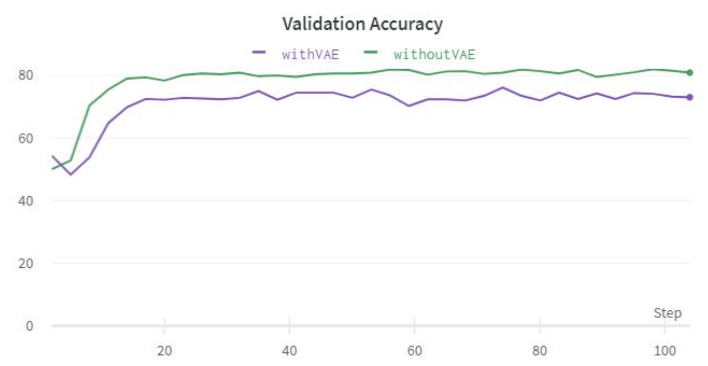
- As we can see in the above two images, we are performing better at epoch 30, but still we believe that we can make our model better and get better results.
- Also following are the train and validation loss graphs that we obtained while training the VAE model.





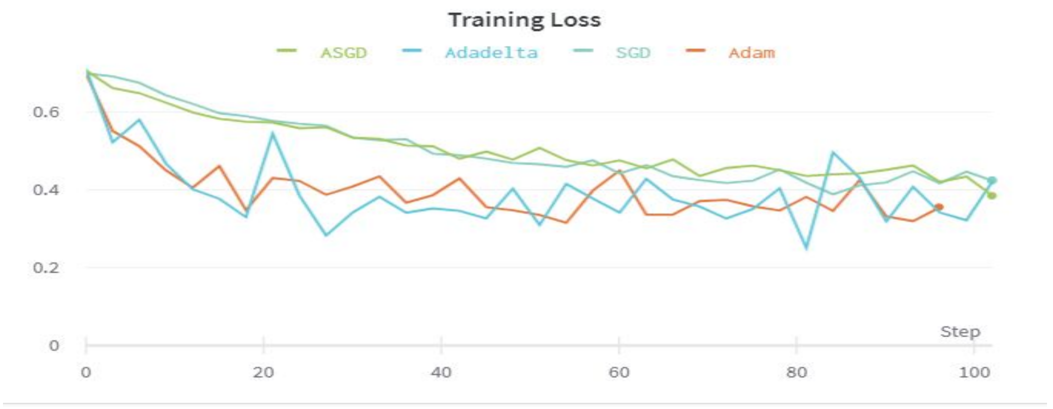
- Comparing the performance of the classification task with and without VAE, we observed that we are getting worse validation accuracy with VAE data. This could be attributed to our current vanilla VAE model. We plan to experiment with different VAE models and find an optimal one that can outperform the validation accuracy on normal data for our final report.

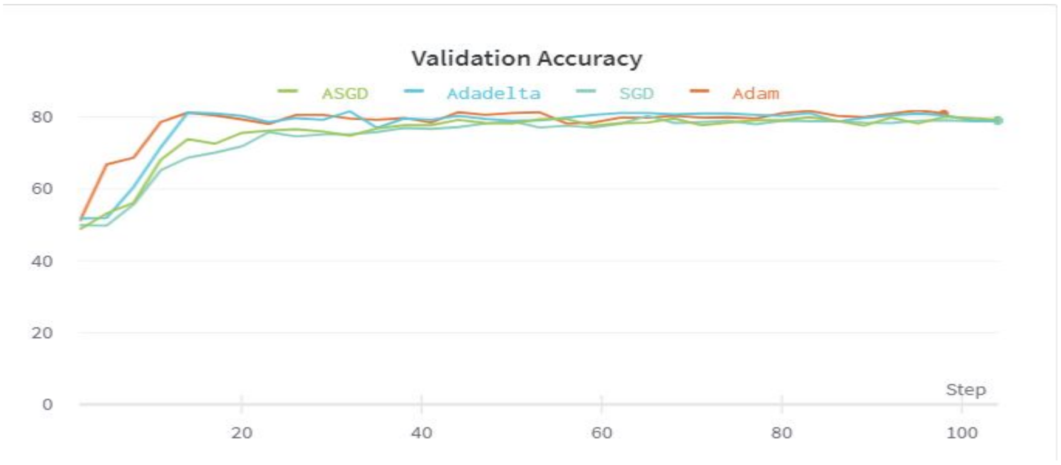




**Optimizer:**

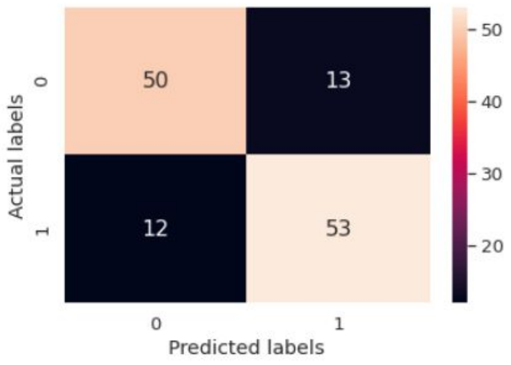
Below are the results for various optimizers used on the classification task using EfficientNet:





**Confusion Matrix (For a model trained with efficient_net):**

A table of confusion (sometimes also called a confusion matrix) is a table with two rows and two columns that reports the number of false positives, false negatives, true positives, and true negatives.



**F-1 Score (For a model trained with efficeint_net)**

The F-measure or balanced F-score (F1 score) is the harmonic mean of precision and recall:.

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

$\mathrm{TP}$ = number of true positives

$\mathrm{FP}$ = number of false positives

$\mathrm{FN}$ = number of false negatives

Using the above mentioned formula we get the following F-1 score for our efficientNet architecture:

***Precision:0.8448***

***Recall: 0.7538***

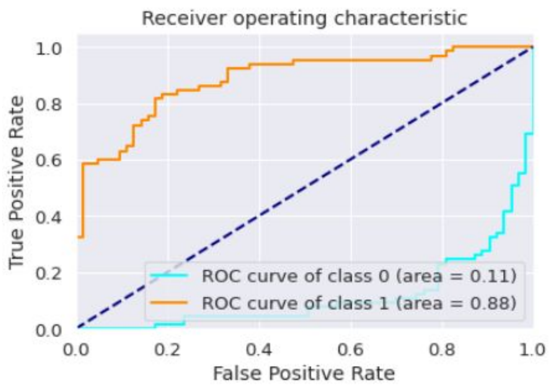
***F1 = 0.7967***

**Receiver operating characteristic (For a model trained with efficeint_net)**

The Receiver operating characteristic (ROC) curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. [7]

AUC (Area Under the Curve) provides an aggregate measure of performance across all possible classification thresholds.

AUC - ROC curve is a performance measurement for classification problems at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes.[8]



## Discussion:

In our midterm checkpoint we have been able to cover the following:

1. We trained our models using efficientNet and ResNet and were able to classify the images with good accuracy. Although we plan to include more architectures for our final report like AlexNet and googlenet.
2. We had trained a Denoising Autoencoder and demonstrated the reconstruction results after training the model for 10 epochs. The model was able to capture the mask areas in some of the images but the results showed a lot of features were blurred out. One reason would be to train the model for more epochs and see if the results get better over further epochs.
3. We trained a VanillaVAE and demonstrated the reconstruction results. Thereafter we ran the classification task on the VAE generated dataset and compared the performance of models with and without VAE. Currently, we are getting worse performance when we classify with VAE, we attribute to our current vanilla VAE model. We plan to experiment with different VAE models and find an optimal one that can outperform the validation accuracy on normal data for our final report.
4. We included GradCam in our pipeline. It helps better visualize the regions of images that the architecture is focusing on while making the predictions. We have shown appropriate heatmaps demonstrating GradCam results and they indeed show that the model focuses mainly on areas near the chin and nose to make its predictions.

5. We have computed various performance metrics like F-1 score, confusion matrix to demonstrate how well our model is performing. The results have been included in the results section. Currently we have computed these metrics on efficientNet but we plan to do a comprehensive comparison of different architectures based on these metrics for our final report.

**References**

1. Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014.
2. Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization”, IJCV 2019.
3. Pu, Yunchen, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. "Variational autoencoder for deep learning of images, labels and captions." In Advances in neural information processing systems, pp. 2352-2360. 2016..
4. https://pypi.org/project/pytorch-gradcam/
5. K Subramanian. Pytorch-vae. https://github.com/AntixK/PyTorch-VAE, 2020
6. 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), Las Vegas, NV, 2016.
7. https://en.wikipedia.org/wiki/Receiver_operating_characteristic
8. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.
9. https://en.wikipedia.org/wiki/Confusion_matrix
10. Tan, Mingxing, and Quoc V. Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." arXiv preprint arXiv:1905.11946 (2019).