Detecting Incorrect Face Mask Usage

Team Styracosaurus, CPEN 291, May 2021

Final Report

—

Brief Description of Project

Use of face masks has become an important part of our life due to the ongoing COVID-19 pandemic. There are some guidelines to wear a mask in an appropriate way that are outlined by WHO for reducing the transmission of the virus.

The idea of this project is to correctly detect masked faces, covering nose, mouth and chin from images using Resnet18 Deep Neural Network architecture to help identify the incorrectly worn face masks and the violators.

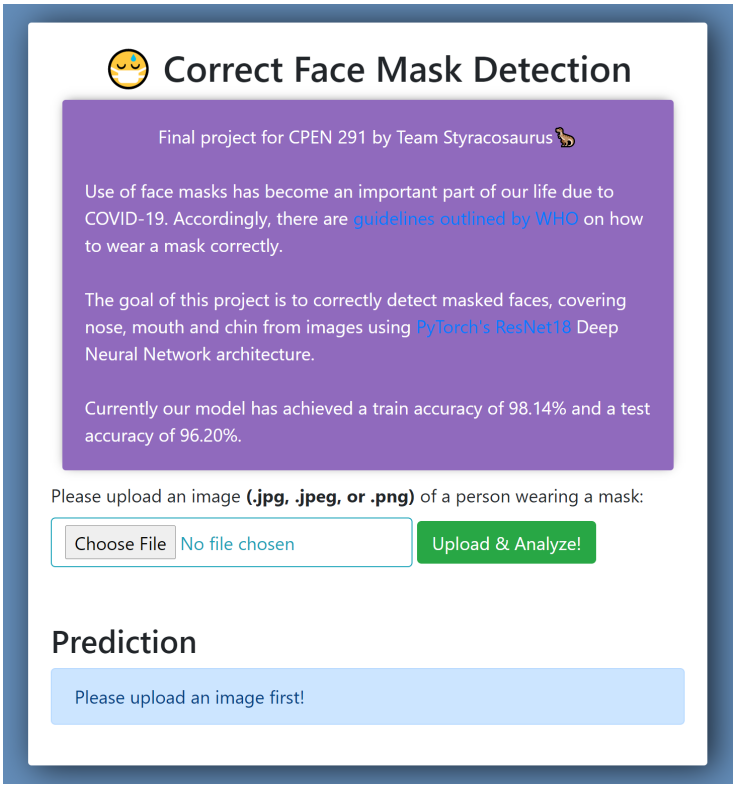
Walkthrough Examples

Our webpage allows the user to choose an image of a person wearing a mask using the “Choose File” button.

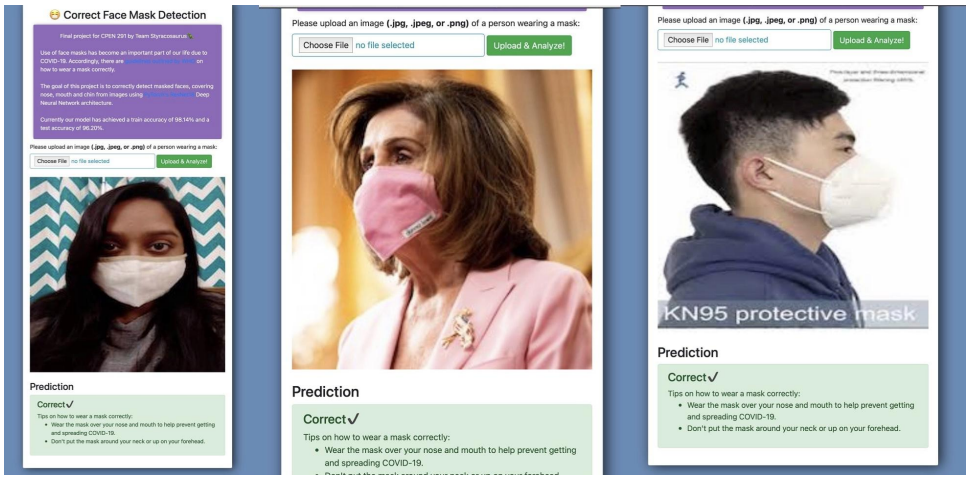
After an image has been chosen, clicking the “Upload & Analyze” button will upload the image.

The image is then evaluated by our ML model in the backend, and the prediction of whether it was correct or incorrect will be outputted on the bottom of the page.

Below are examples of images of correctly worn and incorrectly worn masks.

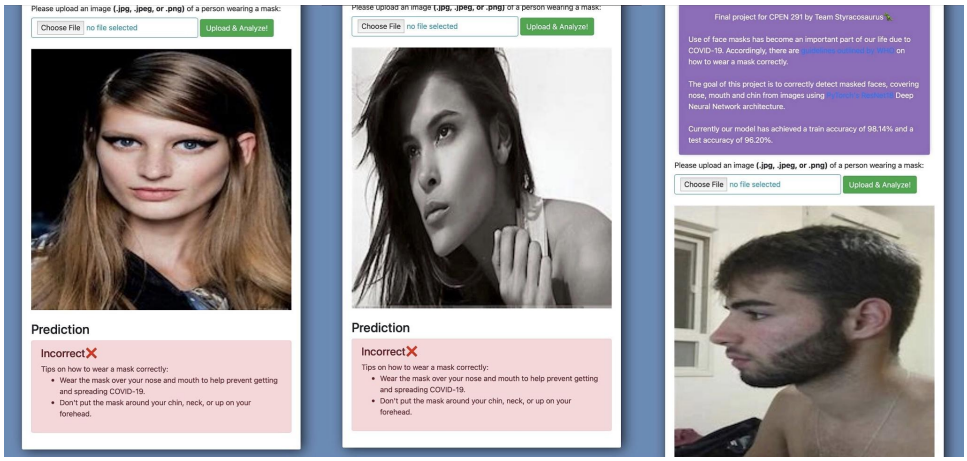


Correctly worn:

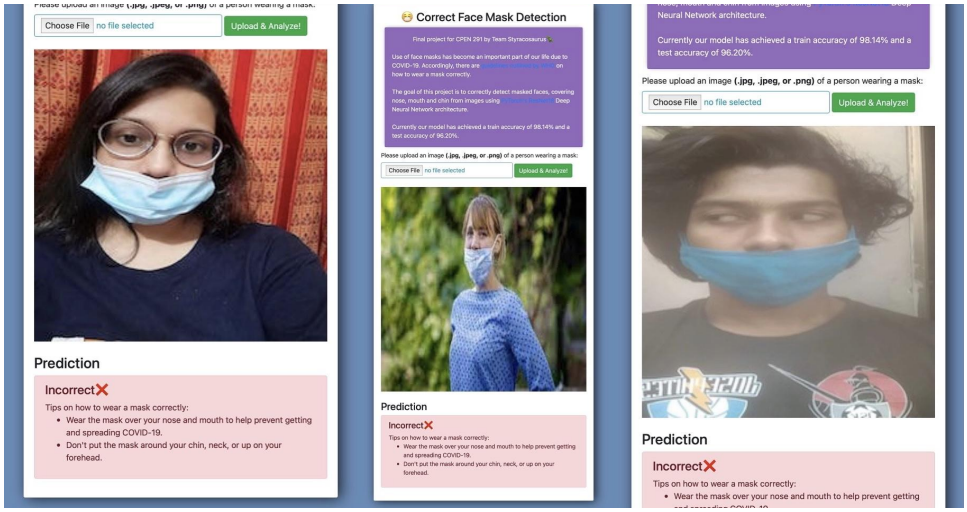


Incorrectly worn:

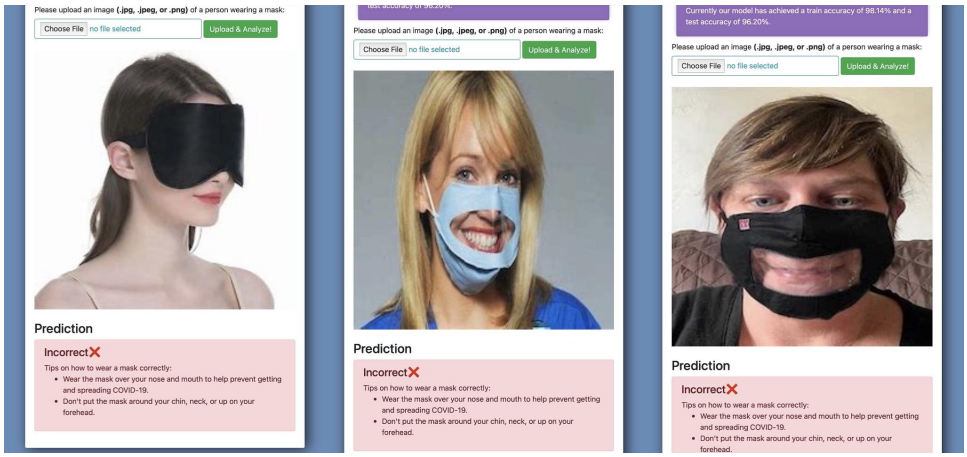
➢ No mask:



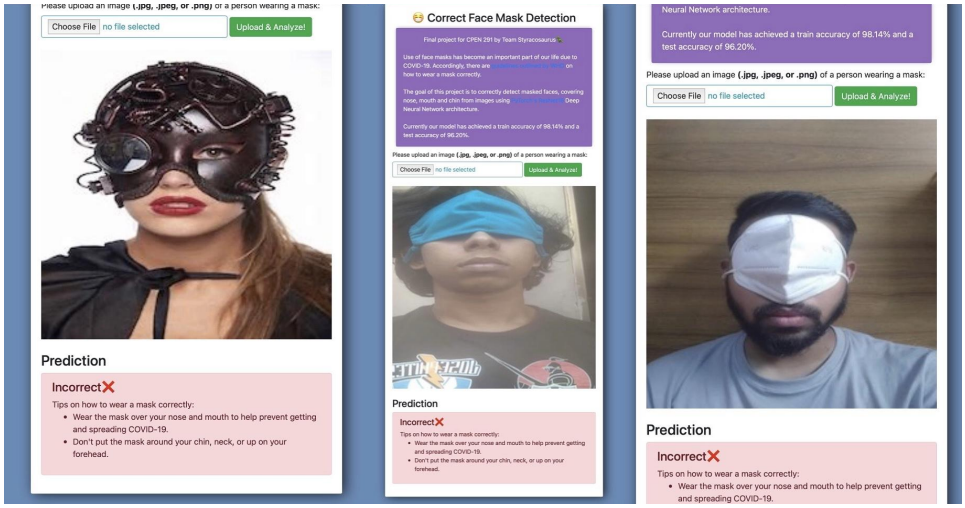
➢ Uncovered nose



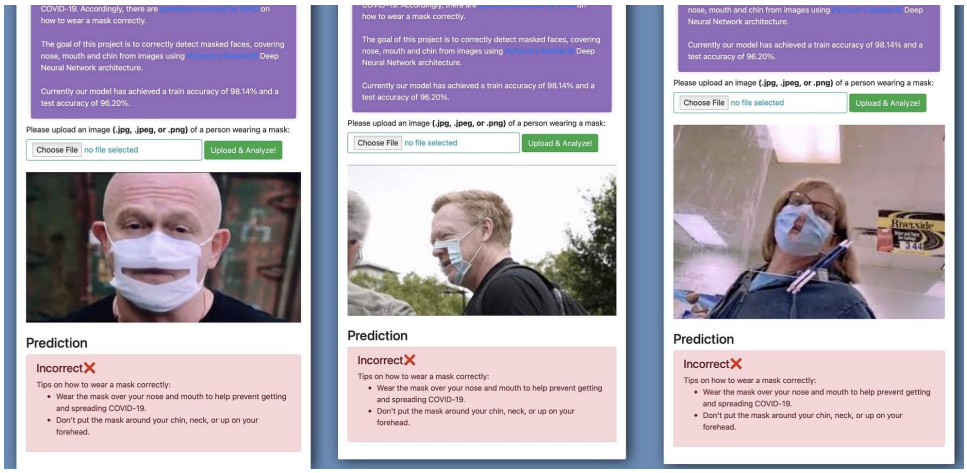
➢ Uncovered mouse:



➢ Uncovered chin:



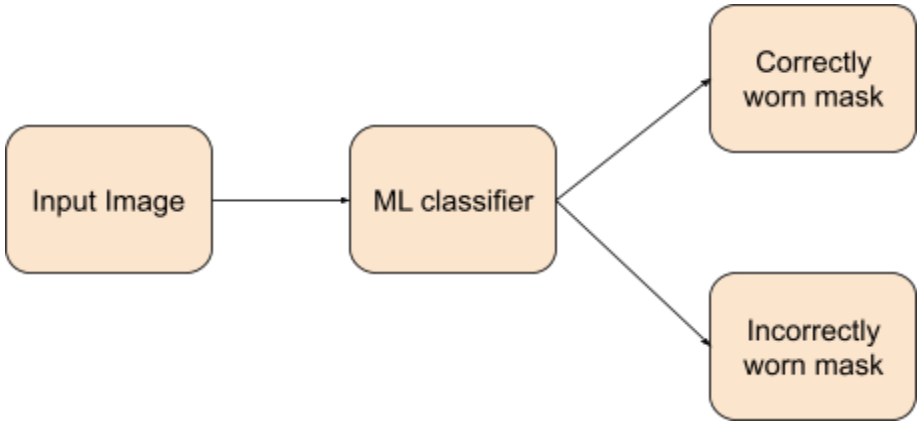
➢ Mask with hole cut out:



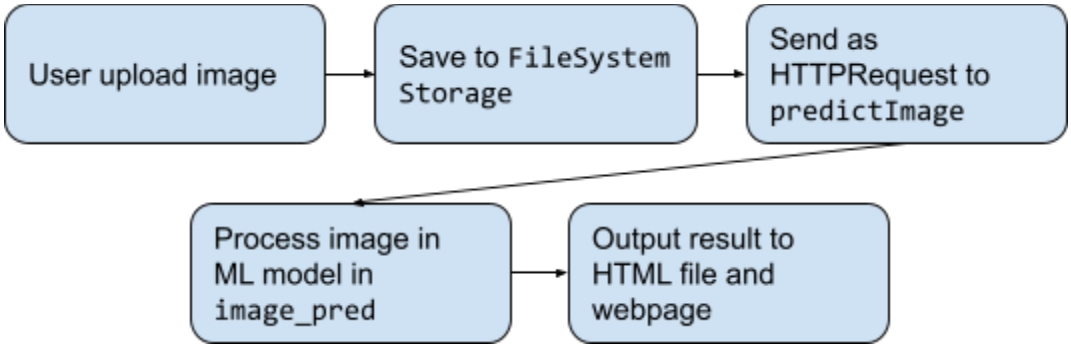
System Diagrams

Significant Components and Data Flow

➢ ML Component



➢ Non-ML Component



Descriptions of Components

ML Component

The ML classifier is developed using a CNN called ResNet18, which is loaded from the ResNets family in torchvision.models. For our project, we replaced the last layer of the model, which is the fc layer, with the same kind of layer but has only 2 outputs, incorrectly worn masks and correctly worn masks. The layer is created using:

```
Model.fc = nn.Linear(model.fc.in_features, 2)
torch.nn.init.xavier_uniform_(model.fc.weight)
```

➢ Why ResNet18?

ResNet18 is a powerful model that can classify over a million images from the ImageNet database into 1000 object categories. Since our task is also on image classification, it would be appropriate to start with a pretrained version of this model. We also tried using ResNet50, which is a very similar model to ResNet18 but with more layers of neural network. However, we realized that ResNet50 is too strong for our task and its test accuracy is also not as good as ResNet18.

➢ Transfer learning and training of the Resnet18 model

After the preparation of the model, we started developing the training process. Transfer learning is applied here, which fine-tunes the model to use things it already knows for the purpose of detecting the incorrect mask usage. The methodology here is using cross entropy for calculating the loss of our prediction, passing the loss to the SGD optimizer for a better set of weights in our model, and changing the learning rate of the above processes using the StepLR scheduler.

```
criterion = nn.CrossEntropyLoss()

def run_train(model, opt, sched):
    nsamples_train = len(dataset_train)
    loss_sofar, correct_sofar = 0, 0
    model.train()
    with torch.enable_grad():
        for samples, labels in loader_train:
            samples = samples.to(device)
            labels = labels.to(device)
            opt.zero_grad()
            outs = model(samples)
            _, preds = torch.max(outs.detach(), 1)
            loss = criterion(outs, labels)
            loss.backward()
            opt.step()
            loss_sofar += loss.item() * samples.size(0)
            correct_sofar += torch.sum(preds == labels.detach())
    sched.step()
    return loss_sofar / nsamples_train, correct_sofar / nsamples_train

optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=4, gamma=0.1)
```

The training for this model uses GPU for speed-up, and the model ends up with the capability of classifying images taken from realistic angles, containing people across

all ages, ethnicity and physique, as well as covering special cases such as masks with holes cut out.

➢ Data Augmentation and Optimization

```
xform = transforms.Compose([
        transforms.Resize((224,224)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomAffine(degrees=0,translate=(0.2,0.2),scale=
        (0.8,1.2)),
        transforms.RandomGrayscale(),
        transforms.ToTensor()])
```

To increase variability of our dataset and improve the robustness of our model, we applied several forms of data augmentation as seen above. While there are many different types of transformations, we only picked the ones that are relevant for our context. For example, `transforms.RandomHorizontalFlip()` flips the image horizontally whereas `transforms.RandomAffine()` scales and moves the image by a little. These augmentations are meant to deal with the fact that the person could be anywhere in the image, not just directly in the middle.
`transforms.RandomGrayscale()` grayscales an image and this can be helpful in dealing with images which are taken in poorly lit areas, resulting in a not so colorful photo. Some examples of transforms we did not include would be `RandomVerticalFlip()` and `transforms.RandomCrop()`. It would be very uncommon to see a person hanging upside down from the ceiling so vertically flipping the image might not be a good idea. On the other hand, cropping the image at random might result in the masked face to be cropped out which would be disastrous.

The parameters used for the optimizer and scheduler are not at all random. It was chosen after testing many different combinations as seen in the image below.

Different parameters:

learning rate	momentum	step size	gamma	minibatch	test loss	test acc	best epoch
0.01	0.9	5	0.1	4	0.1606	0.6479	16
0.0001	0.9	5	0.1	4	0.0385	0.9504	11
0.001	0.85	5	0.1	4	0.0294	0.9636	16
0.001	0.95	5	0.1	4	0.0312	0.9587	15
0.001	0.9	4	0.1	4	0.0259	0.9785	15
0.001	0.9	6	0.1	4	0.0234	0.9653	7
0.001	0.9	3	0.1	4	0.0283	0.9603	15
0.001	0.9	4	0.05	4	0.0242	0.9702	19
0.001	0.9	4	0.15	4	0.0263	0.9686	4
0.001	0.9	4	0.1	16	0.0070	0.9736	12
0.001	0.9	4	0.05	8	0.0283	0.9636	13
0.001	0.9	4	0.05	32	0.0287	0.5405	13

Best combination:

learning rate	momentum	step size	gamma	minibatch	test loss	test acc	best epoch
0.001	0.9	4	0.1	4	0.0259	0.9785	15

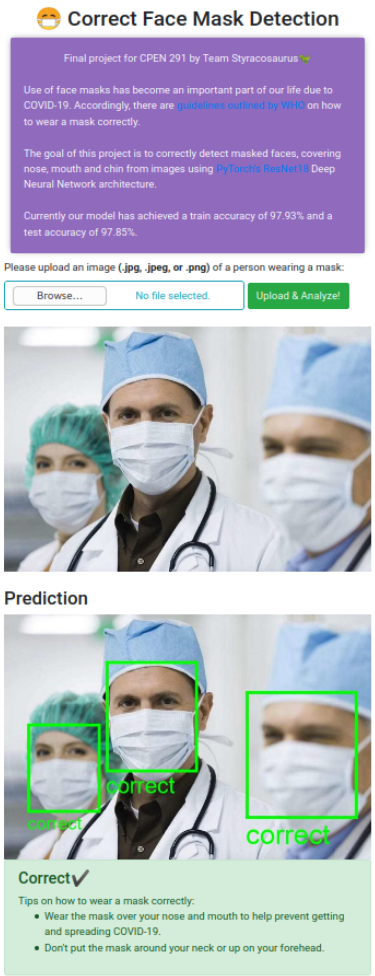
The two parameters highlighted in yellow give the best results. Although we deemed the combination that gives the highest accuracy as the best combination, the second highlighted combination also performs very well and gives the lowest test loss.

Because our dataset was later updated again to include more images, we had to retrain the model. This time however we only tried with those two best combinations. The final results we attained is a train accuracy of 0.9719% and a test accuracy of 96.20%.

➢ Limitations

One key limitation of this model is that it is not meant to classify images which have more than one person in it. In cases where there is one correctly masked face and one incorrectly masked face, for example, the results may vary. It is likely that the result will lean towards the bigger and clearer masked face on the image. Nonetheless, we have come up with a solution to counteract this problem.

One way to ensure we can deal with multiple faces is to use facial recognition. In this version of the mask detector, we first use a facial recognition system (facenet) to identify faces in an image. Next, we crop these images out and feed them into our classification model. Finally, we return a new image which shows the result of the mask detector on every face. Here are some examples:



Non-ML Component

Developing the Django application was a very humbling and educational journey as someone with zero-to-none Python and web development experiences. Below are the details on the application's step-by-step development process.

➢ Choosing Between Django vs. Flask and Creating the Application

Although our proposal initially suggested the usage of Django, we had the opportunity to choose the usage of Flask instead. Since both frameworks each had their own strengths, some simple research was done in order to choose between the two. Although much of the points mentioned in the article ended up beyond the scope of our project, what stood out to us about Django was the ability to explicitly pass an HTTP request around, the built-in support for forms, as well as the Django REST Framework. We finally decided to choose Django due to its high flexibility and practicality.

We used Django 3.2 and Python 3.9.5 to develop the Django application. The application is simply named `detection`:

```
python manage.py startapp detection
```

This provided a basic directory of files required for the Django application to start running.

➢ Handling Uploaded Files

The first step was to find a way to accept and handle files uploaded by users since our project involves processing photos of people wearing masks.

In the HTML file, a form was created along with two buttons, one for choosing the file and another for uploading:

```
<form method="POST" enctype="multipart/form-data" action="predictImage">
    {% csrf_token %}
    <input ... type="file" name="filePath">
    <input ... type="submit" value="Upload & Analyze!">
</form>
```

The request would be passed to the `predictImage` function in views.py, with the uploaded file named as `filePath`. The CSRF token is added to prevent malicious

attacks. In `predictImage`, the uploaded file is stored in a `FileSystemStorage` and then to the /media/ folder.

The file can then be easily accessed via its added `urlpatterns` with the `MEDIA_URL` in /detection/urls.py.

➢ Loading the ML Model and Processing the Image

After saving our ML model as a .pth file in the Google Colab notebook, the model was loaded in /py_templates/my_model.py:

```
model = models.resnet18(pretrained=False)
model.fc = nn.Linear(model.fc.in_features, 2)
model.load_state_dict(torch.load(model_path, map_location=torch.device('cpu')))
model.eval()
```

Then `predictImage` of views.py calls and passes the image url to `image_pred` of /py_templates/my_model.py, which does the actual prediction process:

```
def image_pred(url):
    raw_url = urllib.parse.unquote(image_path + url)
    img = Image.open(raw_url)
    img = img.convert(mode='RGB')
    image = image.unsqueeze(0)

    out = model(image)
    out = out.detach().numpy()
    out = np.argmax(out)
    return out
```

The return value `out` of `image_pred` will be either `0` for correct mask usage or `1` for incorrect mask usage.

➢ Outputting the Prediction Result to the User

The return value `out` is then passed back to `predictImage`, which then is rendered with the context into an HTTPResponse object:

```
out = image_pred(file_name)
out = {0:'Correct', 1:'Incorrect'}[int(out)]

context = {'filePathName':filePathName, 'pred':out}
return render(request, 'detection/interface.html', context)
```

This HTTPResponse object is then passed to the HTML file, interface.html. Using an if conditional, `pred` (the value of `out`) is checked and results are displayed on the web page accordingly depending on the value of `pred`:

```
{% if pred == 'Correct' %}
    // output result: correct
{% elif pred == 'Incorrect' %}
    // output result: incorrect
{% else %}
    // output result: no image uploaded
{% endif %}
```

(Examples of the outputted results can be found in the Walkthrough Examples section above.)

➢ Formatting the Website and Deploying to Heroku

Our web pages utilizes a stylesheet from Bootstrap 4.3.1, most notably the button styling and the output result styling. (The “Choose File” button was notoriously difficult to hack around. Out of time consideration, it was left as an outlined Bootstrap style button instead of a solid Bootstrap style button.)

Since our project was intended to be a web page, we decided it made no sense if we hadn’t deployed it for public use. Heroku was a very suitable choice for us as it abstracts away lots of the lower-level administration and it is easy to deploy and update. (Most importantly, it is free for a small project like ours!) Our slug size was slightly over Heroku’s soft cap (309MB out of 300MB), but the deployment was successful overall: http://correct-mask-detection.herokuapp.com/.

➢ Mobile Application Development

As an addition to the non-ML part of the project, we developed a mobile application for our webpage. The application is developed using Flutter with the help of Dart. It currently runs on both android and iOS mobile phones but it has been tested using the android device. The user of the application can upload images from mobile phones in our model through the webpage and can get their masked images detected.

The app is developed in debug mode so it has the red ribbon on top right. It is a material app with 3 screens. The home screen of the app has two buttons that leads to two pages - mask recognition page and camera page. The mask recognition

page is the webview of the webpage that we had developed for the non-ML part. This page allows the user to upload images from the gallery or from the camera. The flutter plugins called webview_flutter, flutter_webview_plugin and permission_handler were used to interact with the webpage. The camera page uses the plugin image_picker and path_provider for taking and saving the images in the local storage.

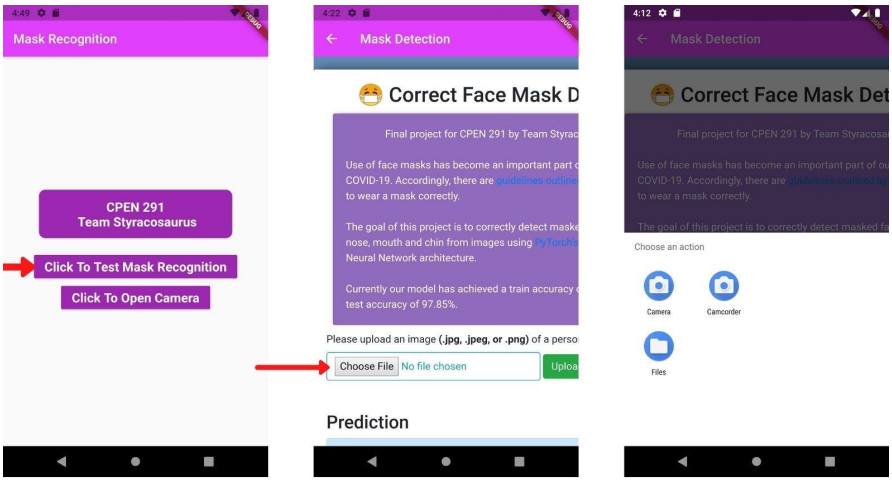


Figure-1: Mask Detection page

The homepage of the application has two buttons. Clicking the first button leads to Figure-1 and the second button leads to Figure-3. Figure-2 depicts the webview of our webpage. The 'Choose File' option allows a user to upload and select images from the gallery as well as the camera. Clicking the 'Files' option from Figure-1 leads to the page illustrated by Figure-2.

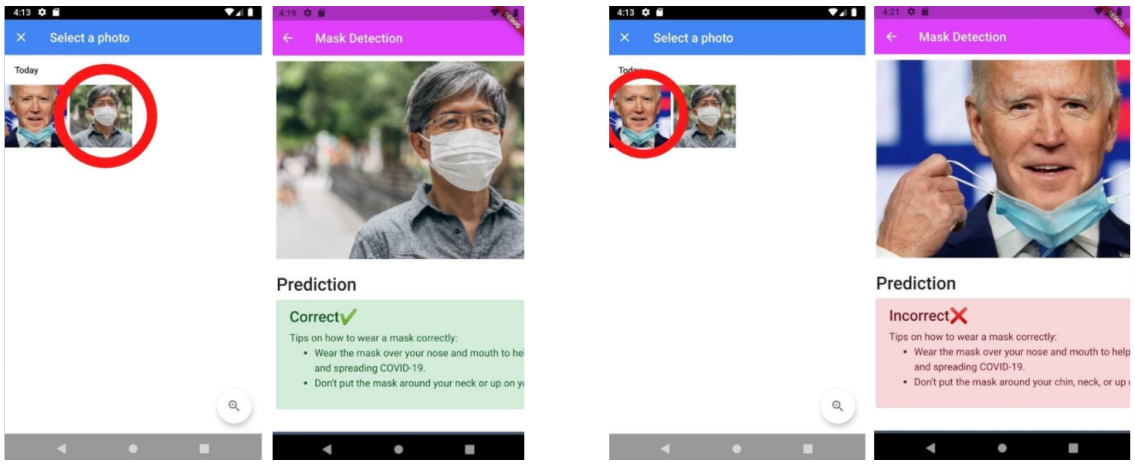


Figure-2: Mask Detection page with prediction results

Any image from the gallery can be uploaded and analyzed in our webpage. Figure-2 also shows the output of the image (whether the person is wearing the mask in a correct or incorrect way) along with the tips on wearing a mask correctly.
The 'Choose File' button also allows the user to take a photo in real time using the 'Camera' option. It directly uploads the clicked photo to the webpage after saving it.

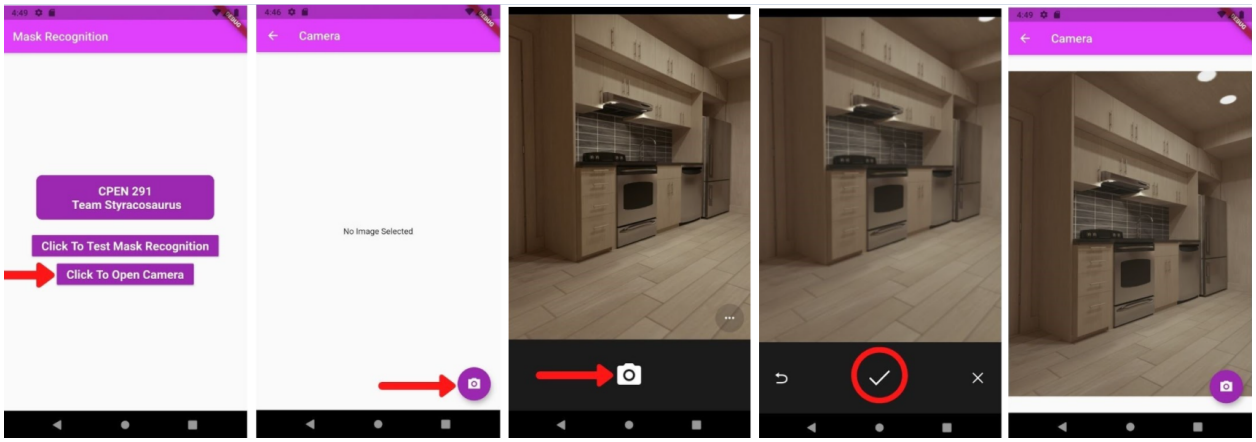


Figure-3: Camera page

Clicking the second button in the homepage leads to the camera page of the application. It allows a user to take a photo and view it in fullscreen. This gives the user a chance to see the image before uploading it to the webpage. This dedicated page for camera adds the camera functionality to our application.

Overall Contributions

Azwad

Data collection, ML model development and Mobile Application Development.

Chunhao

Data collection and ML model development.

Jessica

Data collection and Django web application development and deployment.

Kevin

Data collection and ML model development.