COMP6721 Applied Artificial Intelligence (Fall 2021)
Project Assignment, Part II

AI Face Mask Detector

Team Information

Team Name:
NS_08

Team Members:

- Rahul Patil(**40166394**) - Training Specialist
- Yash Shah(**40163452**) - Data Specialist
- Rucha Prajapati(**40170922**) - Evaluation Specialist

Resources

Link to Repository : https://github.com/iRahulP/FaceMask__AI

Link to Dataset :

- Full Dataset: https://drive.google.com/file/d/1MNZVh5MOExrypXtIql6QqHkvlLwuB7X1/view?usp=sharing
- Capped Dataset : https://drive.google.com/file/d/1E91_9H335kU4xWXNZ193rXPAA4baom2H/view?usp=sharing

Link to Colab Notebook :
https://colab.research.google.com/drive/17bVpguQP_XBjfkmHTqp-JhmslAXXnTgh?usp=sharing

DataSet

Dataset here is used for training and testing the deep learning model for analysis of bias introduced through training data. Our dataset consists of 8605 belonging to four classes which we gathered from below-mentioned sources. Following table represents the number of images of a particular category.

Categories	**Number of Images**
without_mask	2160
with_surgical_mask	2103
with_cloth_mask	2160
with_ffp2_mask	2180

Here we have analyzed our AI for the following categories: **age, race** (*black*, *white*) and **gender** (*male*, *female*) for our dataset out of 3 given categories: age, gender and race.

Initial *Face mask Image* Dataset:
(source: https://susanqq.github.io/UTKFace/)

UTKFace dataset is a large-scale face dataset with a long age span (range from 0 to 116 years old). The dataset consists of over 20,000 face images with annotations of age, gender, and ethnicity. The images cover large variation in pose, facial expression, illumination, occlusion, resolution, etc. This dataset could be used on a variety of tasks, e.g., face detection, age estimation, age progression/regression, landmark localization, etc. Some sample images are shown as following:

Highlights

- consists of 20k+ face images in the wild (only single face in one image)
- provides the correspondingly aligned and cropped faces
- provides the corresponding landmarks (68 points)
- images are labelled by age, gender, and ethnicity

Labels

The labels of each face image is embedded in the file name, formated like [age]_[gender]_[race]_[date&time].jpg

- [age] is an integer from 0 to 116, indicating the age
- [gender] is either 0 (male) or 1 (female)
- [race] is an integer from 0 to 4, denoting White, Black, Asian, Indian, and Others (like Hispanic, Latino, Middle Eastern).
- [date&time] is in the format of yyyymmddHHMMSSFFF, showing the date and time an image was collected to UTKFace

Sample Face Data:

License Claim

- The UTKFace dataset is available for non-commercial research purposes only.
- The aligned and cropped images, as well as landmarks, are obtained by Dlib.
- Please note that all the images are collected from the Internet which are not property of AICIP. AICIP is not responsible for the content nor the meaning of these images.
- The copyright belongs to the original owners. If any of the images belongs to you, please let us know and we will remove it from our dataset immediately.
- The ground truth of age, gender and race are estimated through the DEX algorithm and double checked by a human annotator. If you find anything inaccurate, please let us know.

Script used to segregate above images to the following categories as folders: ***femaleBlack , femaleWhite, maleBlack*** and ***maleWhite***.

```
#!/bin/bash
for file in part*/*_1_1*;
do
        cp "$file" femaleBlack
done
```

(facesData source:
https://drive.google.com/drive/u/0/folders/1hY4gC6zdwiRpwnhtZxzfK1ijrUF2fZT2)

- *Mask the Face*:

Then mask the images of all the faces within an image and apply the user-selected masks to them taking into account various limitations such as face angle, mask fit, lighting conditions, etc. using computer-vision script.
(source:https://towardsdatascience.com/masktheface-cv-based-tool-to-mask-face-dataset-1a71d5b68703)

Source Data :




Now we have images of different types of masks (cloth,surgical,FFP2 and no mask) in both the above mentioned categories.After the data preprocessing, we split the dataset into train and test parts and use 70% of the data for the training phase and the remaining 30% for the testing phase. Then, we apply the 10-fold cross-validation on the train set.

Dataset can be found at :
https://drive.google.com/drive/folders/1c8OLNvSYqCMJ7QtihsSHEbObjXAknsiQ

Data References:

- https://www.kaggle.com/omkargurav/face-mask-dataset just provided all the majority of the images for our dataset.it has people with and without mask and with different categories of mask.
- https://www.google.com/search?q=images+of+people+with+ffp2+masks&tbm=isch&ved=2ahUKEwjg9J7ur4z0AhVugXIEHVNDDOQQ2-cCegQIABAA&oq=images+of+people+with+ffp2+masks&gs_lcp=CgNpbWcQAzoFCAAQgARQ3qqUQ1i6vJRDYM6_lENoAXAAeACAAbIBiAHtB5IBAzIuNpgBAKABAaoBC2d3cy13aXotaW1n

wAEB&sclient=img&ei=IvyKYeDbJ-6CytMP04axoA4&bih=692&biw=1600&rlz=1C1CHZN_enIN940IN940

- https://makeml.app/datasets/mask

- https://towardsdatascience.com/masktheface-cv-based-tool-to-mask-face-dataset-1a71d5b68703 - Images were manually created using the tool provided for all masks dataset using open source face images from.

CNN ARCHITECTURE

1) Data Extraction:

We download the face mask dataset from the drive and unzip it using the drive REST Api. Then we load the images and label them to a pandas dataframe. Afterwards, we save the dataframe as a pickle file, which can be subsequently used for training the model.

```
os.makedirs('data/dataset')
datasetPath = Path('data/dataset/dataset.zip')
gdd.download_file_from_google_drive(file_id='1UDvUL7nuzt3JStnN1KOl_onC5h7q6H9O',
                                    dest_path=str(datasetPath),
                                    unzip=True)

# https://drive.google.com/file/d/1UDvUL7nuzt3JStnN1KOl_onC5h7q6H9O/view?usp=sharing
# delete zip file
datasetPath.unlink()

datasetPath = Path('data/dataset')
# maskPath = datasetPath/'with_mask'
clothMaskPath = datasetPath/'with_cloth_mask'
surgicalMaskPath = datasetPath/'with_surgical_mask'
ffp2MaskPath = datasetPath/'with_ffp2_mask'
nonMaskPath = datasetPath/'without_mask'

maskDF = pd.DataFrame()

for imgPath in tqdm(list(nonMaskPath.iterdir()), desc='without_mask'):
    maskDF = maskDF.append({
        'image': str(imgPath),
        'mask': 0
    }, ignore_index=True)

for imgPath in tqdm(list(surgicalMaskPath.iterdir()), desc='with_surgical_mask'):
    maskDF = maskDF.append({
        'image': str(imgPath),
        'mask': 1
    }, ignore_index=True)

for imgPath in tqdm(list(clothMaskPath.iterdir()), desc='with_cloth_mask'):
    maskDF = maskDF.append({
        'image': str(imgPath),
        'mask': 2
    }, ignore_index=True)

for imgPath in tqdm(list(ffp2MaskPath.iterdir()), desc='with_ffp2_mask'):
    maskDF = maskDF.append({
        'image': str(imgPath),
        'mask': 3
    }, ignore_index=True)
```

Output DataFrame:

```
Downloading 1E91_9H335kU4xWXNZ193rXPAA4baom2H into data/dataset/Capped.zip... Done.
Unzipping...Done.
without_mask: 100%|██████████| 2160/2160 [00:05<00:00, 372.49it/s]
with_surgical_mask: 100%|██████████| 2103/2103 [00:05<00:00, 369.10it/s]
with_cloth_mask: 100%|██████████| 2162/2162 [00:05<00:00, 367.90it/s]
with_ffp2_mask: 100%|██████████| 2180/2180 [00:05<00:00, 373.61it/s]
                                                  image  mask
0     data/dataset/without_mask/25_0_1_2017011313414...   0.0
1     data/dataset/without_mask/25_0_1_2017011314252...   0.0
2     data/dataset/without_mask/25_0_1_2017011314514...   0.0
3     data/dataset/without_mask/25_0_1_2017011314521...   0.0
4     data/dataset/without_mask/25_0_1_2017011314542...   0.0
...                                                 ...   ...
8600  data/dataset/with_ffp2_mask/65_0_1_20170120223...   3.0
8601  data/dataset/with_ffp2_mask/65_0_1_20170120223...   3.0
8602  data/dataset/with_ffp2_mask/65_0_1_20170120224...   3.0
8603  data/dataset/with_ffp2_mask/65_1_0_20170110183...   3.0
8604  data/dataset/with_ffp2_mask/65_1_1_20170110124...   3.0

[8605 rows x 2 columns]
Saving Dataframe to: data/dataset/dataset.pickle
```

Dataset class: Having a panda dataframe, MaskDetectionDataset class is used to query images by pytorch. All images are transformed using torchvision.transforms resizing them to 32*32, converting to torch Tensor and normalizing them with standard mean and std values which then forces the network to be between 0-1 values.

```python
class MaskDetectionDataset(Dataset):
    def __init__(self, dataFrame):
        self.dataFrame = dataFrame

        self.transformations = Compose([
            Resize((32, 32)),
            ToTensor(),
            Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.225, 0.225, 0.225]
                )
        ])

    def __getitem__(self, key):
        if isinstance(key, slice):
            raise NotImplementedError('slicing is not supported')

        row = self.dataFrame.iloc[key]
        image = Image.open(row['image']).convert('RGB')
        return {
          'image': self.transformations(image),
          'mask': tensor([row['mask']], dtype=long),
          'path': row['image']
        }

    def __len__(self):
        return len(self.dataFrame.index)
```

Defining CNN model:

In the model, we use the nn.module that helps to build neural network models. In part 1 of our project we have only 2 convolution layers but in part 2 we increase it to 4

In __init__() we define the layers and subsequent dimensions within them for the CNN as follows :

1) 4 convolution layers (for channels):
    - **Conv1** applies to Conv2d with input channels as 3(for RGB image) and output channels as 32.
    - With a kernel size of 3 and padding of 1: the dimensions for the 32*32 images remain the same. # 32 - 3 + 1 + 2*1
    - Same goes for **conv2** for input channels of 32 and output channels as 32.following with **conv3** with input channel of 32 and out of 128 and **conv4** with input of 128 and out of 64.
2) 2 pooling layers:
    - This layer pools up the data from the convolution layer using functions such as Max,Average etc.
    - We use 2 Max pool layers with stride=2 and kernel size=2
    - Resultant matrix of Max pool layer 1 and Max pool layer 2 is 16*16 and 8*8 respectively.
3) 3 fully connected layers (for features):
    - fc1 applies linear with input of (8*8*64) to output features as 1000.
    - With fc2 and fc3 with reduction to 512 and 4 output features respectively.
4) We used the LeakReLU activation function that defines when a node will be fired. And it is based on ReLU.

Further we carry **forward pass** in the forward() method, which does actual transformation of the network by passing the data into the computation graph (i.e. neural network) which maps an input tensor to a prediction output tensor as follows in current CNN:

- Initial out calls a convolution layer and flatten it and reshape it into a one dimensional object.And it is fed to a fully connected layer and it returns the output.

```
super(FaceMaskDetectorCNN, self).__init__()
self.conv_layer = nn.Sequential(
  nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1), # input layer RGB    32
  nn.BatchNorm2d(32),
  nn.LeakyReLU(inplace=True),
  nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1),# feature specific   32
  nn.BatchNorm2d(32),
  nn.LeakyReLU(inplace=True),
  nn.MaxPool2d(kernel_size=2, stride=2),
  nn.Conv2d(in_channels=32, out_channels=128, kernel_size=3, padding=1), # 16 * 16
  nn.BatchNorm2d(128),
  nn.LeakyReLU(inplace=True),
  nn.Conv2d(in_channels=128, out_channels=64, kernel_size=3, padding=1), #16
  nn.BatchNorm2d(64),
  nn.LeakyReLU(inplace=True),
  nn.MaxPool2d(kernel_size=2, stride=2),  # 8 * 8
)
self.fc_layer = nn.Sequential(
  nn.Dropout(p=0.1),
  nn.Linear(8 * 8 * 64, 1000),  # Fully connected (input (l*b*channels) , output features) How
  nn.ReLU(inplace=True),
  nn.Linear(1000, 512),
  nn.ReLU(inplace=True),
  nn.Dropout(p=0.1),
  nn.Linear(512, 4)
)
```

Here is the detailed summary of the layers:

```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 32, 32]             896
       BatchNorm2d-2          [-1, 32, 32, 32]              64
         LeakyReLU-3          [-1, 32, 32, 32]               0
            Conv2d-4          [-1, 32, 32, 32]           9,248
       BatchNorm2d-5          [-1, 32, 32, 32]              64
         LeakyReLU-6          [-1, 32, 32, 32]               0
         MaxPool2d-7          [-1, 32, 16, 16]               0
            Conv2d-8         [-1, 128, 16, 16]          36,992
       BatchNorm2d-9         [-1, 128, 16, 16]             256
        LeakyReLU-10         [-1, 128, 16, 16]               0
           Conv2d-11          [-1, 64, 16, 16]          73,792
      BatchNorm2d-12          [-1, 64, 16, 16]             128
        LeakyReLU-13          [-1, 64, 16, 16]               0
        MaxPool2d-14            [-1, 64, 8, 8]               0
          Dropout-15                [-1, 4096]               0
           Linear-16                [-1, 1000]       4,097,000
             ReLU-17                [-1, 1000]               0
           Linear-18                 [-1, 512]         512,512
             ReLU-19                 [-1, 512]               0
          Dropout-20                 [-1, 512]               0
           Linear-21                   [-1, 4]           2,052
================================================================
Total params: 4,733,004
Trainable params: 4,733,004
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.78
Params size (MB): 18.05
Estimated Total Size (MB): 20.84
----------------------------------------------------------------
None
```

Preparation of Dataset by splitting:.Dataset is splitted in 70:30 for training and testing.For this purpose we use the train_test_split() function and pass a dataset label to its parameter. Loading of data is done using Dataloader and uses it as a batch of 32 at the time for training the model.To reduce the time of loading the batch samples, the number of workers is set to 4.

```python
from sklearn.model_selection import train_test_split
from torch import Tensor
from torch.nn import (Conv2d, CrossEntropyLoss, Linear, MaxPool2d, ReLU, Sequential)
from torch.optim import Adam
from torch.optim.optimizer import Optimizer
from torch.utils.data import DataLoader
import itertools
import matplotlib.pyplot as plt

def plot_cm(cm, classes, normalize=False, title='Visualization of the confusion matrix', cmap=plt.cm.Reds):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Actual True label')
    plt.xlabel('Predicted label')

def prepare_data(mask_df_path) -> None:
        mask_df = pd.read_pickle(mask_df_path)
        # print the distribution
        print(mask_df['mask'].value_counts())
        train, validate = train_test_split(mask_df, test_size=0.3, random_state=0,
                                           stratify=mask_df['mask'])
        return [
            MaskDetectionDataset(train),
            MaskDetectionDataset(validate),
            CrossEntropyLoss()
            ]

def train_dataloader(train_df) -> DataLoader:
    return DataLoader(train_df, batch_size=32, shuffle=True, num_workers=4)

def val_dataloader(validate_df) -> DataLoader:
    return DataLoader(validate_df, batch_size=32, num_workers=4)
```

We have output which shows the number of images of each type.

```
3.0    2180
2.0    2162
0.0    2160
1.0    2103
Name: mask, dtype: int64
```

Training the Model: We use Adam to optimize its parameters and the learning rate is set to 0.001. Then, we iterate through the training dataset and retrieve batches to get images and labels, pass it to the model and compute the loss of that batch. The Number of epochs is 20 for this project which can be changed.

```
epochs = 20
learning_rate = 0.001
retrain = False
"""
Training Step
"""

import warnings
warnings.filterwarnings('ignore')

def train_model():
    optimizer = Adam(face_mask_detector_cnn.parameters(), lr=learning_rate)
    for epoch in range(epochs):
        loss_train = 0.0
        for i, data in enumerate(train_dataloader(train_df), 0):
            inputs, labels = data['image'], data['mask']
            labels = labels.flatten()
            outputs = face_mask_detector_cnn(inputs)
            loss = cross_entropy_loss(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            loss_train += loss
        print(f' LOSS :: Training Loss (after epoch {epoch}):', loss_train)

train_model()
print('Model training has finished')
```

Evaluate the model: We use matplotlib which predicts labels on X-axis and actual labels on Y-axis.First create empty tensors each for prediction and actual data, iterate through validate dataset and retrieve batches to get images and labels to evaluate the accuracy on unseen images.Then, we concatenate the sequence of tensors in the given dimension.

Afterwards we define four classes (without_mask, with_mask_cloth, with_mask_surgical and with_mask_ffp2) and plot confusion matrix having actuals and predictions on its axis.

```
from numpy import vstack
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix

def evaluate_model():
    predictions, actuals = torch.tensor([]), torch.tensor([])
    for i, data in enumerate(val_dataloader(validate_df)):
        inputs, targets = data['image'], data['mask']
        targets = targets.flatten()
        output = face_mask_detector_cnn(inputs)
        output = torch.argmax(output,axis=1)
        predictions = torch.cat((predictions, output.flatten()), dim=0)
        actuals = torch.cat((actuals, targets), dim=0)

    #print metrics
    classes = ['without_mask','with_mask_surgical', 'with_mask_cloth', 'with_mask_ffp2']
    print(classification_report(actuals, predictions, digits = 4, target_names=classes))
    confusion_mat = confusion_matrix(actuals.numpy(), predictions.numpy())
    plot_cm(confusion_mat, classes)
evaluate_model()
```

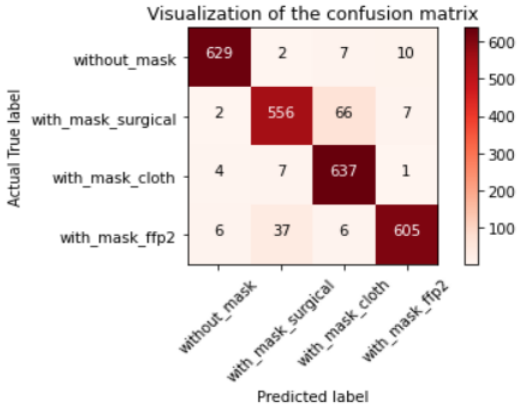
Output of Confusion Matrix:

```
                    precision    recall  f1-score   support

      without_mask     0.9813    0.9707    0.9760       648
with_mask_surgical     0.9236    0.8811    0.9019       631
   with_mask_cloth     0.8897    0.9815    0.9333       649
    with_mask_ffp2     0.9711    0.9251    0.9475       654

          accuracy                         0.9400      2582
         macro avg     0.9414    0.9396    0.9397      2582
      weighted avg     0.9416    0.9400    0.9399      2582

Confusion matrix, without normalization
[[629   2   7  10]
 [  2 556  66   7]
 [  4   7 637   1]
 [  6  37   6 605]]
```



Prediction of random images using trained Model:

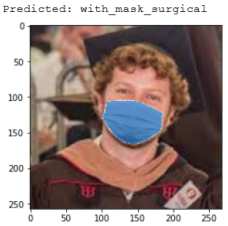
Iteration over one batch is done of 32 images of the test data frame(validate_df) and after loading and running one image is selected at random from it using Random.choice() and it is plotted in a graph and its actual and predicted values are obtained.

```
class_mapping = {
    0: "without_mask",
    1: "with_mask_surgical",
    2: "with_mask_cloth",
    3: "with_mask_ffp2",
}

def predict():
  rand_sampler = torch.utils.data.RandomSampler(validate_df, num_samples=32, replacement=True)
  data = iter(DataLoader(validate_df, batch_size=32, num_workers=1, sampler=rand_sampler)).next()
  inputs,targets = data['image'], data['mask']
  output = face_mask_detector_cnn(inputs)
  output = torch.argmax(output,axis=1)
  rand_ind = random.choice(list(range(0,32)))
  print(data['path'][rand_ind])
  img = Image.open(data['path'][rand_ind])
  plt.imshow(np.asarray(img))
  print("Actual:", class_mapping[targets[rand_ind].tolist()[0]])
  print("Predicted:",class_mapping[output[rand_ind].tolist()])

predict()
```

```
data/dataset/with_surgical_mask/30_0_1_20170117130551448_surgical.jpg
Actual: with_mask_surgical
Predicted: with_mask_surgical
```



Bias : In our AI model we have analyzed the following categories for bias in our training data :

- *Gender*
- *Race*

Our initial dataset was capped to the sum of images provided above and being trained with k fold cross validation. The same trained model is used further to analyze the possible bias throughout the training data.
200 image sets were considered for each label within each bias term across the dataset which accounts to a total of 800 images for each bias.

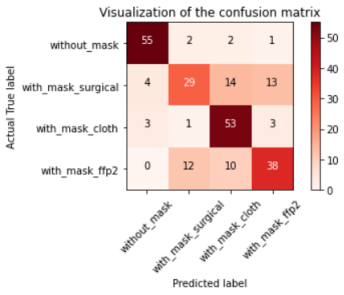
- Consistency across **Male and Female**

Dataset for Male was sourced from the maleWhite and maleBlack folders and Female was sourced from femaleWhite and femaleBlack folders from the original mask the face dataset.
Dataset was rebalanced and capped to 200 images across each labels for Male (https://drive.google.com/file/d/1ftwVEFn8XZlx5qJAeRp5ZEOXbXLAL32Q/view?usp=sharing) as well as Female (https://drive.google.com/file/d/1F-3Y8UoIgvd_IE7znsTu1Lq0w5PWcnqZ/view?usp=sharing).
The same was tested using the model which provided accuracies of 82% and 73% respectively.

[240 rows x 2 columns]

	precision	recall	f1-score	support
without_mask	0.8871	0.9167	0.9016	60
with_mask_surgical	0.6591	0.4833	0.5577	60
with_mask_cloth	0.6709	0.8833	0.7626	60
with_mask_ffp2	0.6909	0.6333	0.6609	60
accuracy			0.7292	240
macro avg	0.7270	0.7292	0.7207	240
weighted avg	0.7270	0.7292	0.7207	240

Confusion matrix, without normalization
[[55 2 2 1]
[4 29 14 13]
[3 1 53 3]
[0 12 10 38]]

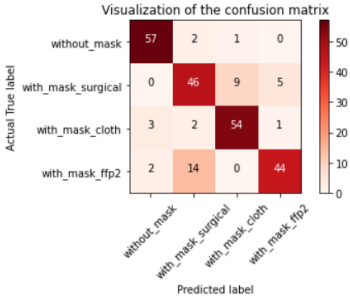


Female

[240 rows x 2 columns]

	precision	recall	f1-score	support
without_mask	0.9194	0.9500	0.9344	60
with_mask_surgical	0.7188	0.7667	0.7419	60
with_mask_cloth	0.8438	0.9000	0.8710	60
with_mask_ffp2	0.8800	0.7333	0.8000	60
accuracy			0.8375	240
macro avg	0.8405	0.8375	0.8368	240
weighted avg	0.8405	0.8375	0.8368	240

Confusion matrix, without normalization
[[57 2 1 0]
[0 46 9 5]
[3 2 54 1]
[2 14 0 44]]



Male

- *Consistency across **Black and White***

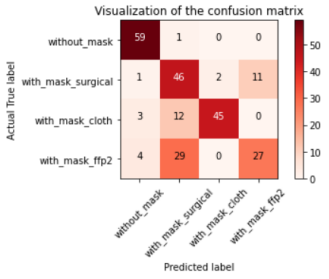
Dataset for White was sourced from the maleWhite and femaleWhite folders and Black was sourced from maleBlack and femaleBlack folders from the original mask of the face dataset.
Dataset was rebalanced and capped to 200 images across each labels for White (https://drive.google.com/file/d/1AxLKi-zhyeQfxL_ARYExgTp0cal2Wkum/view?usp=sharing) as well as Black (https://drive.google.com/file/d/1H7AVjsqI0r9dVDn-J_BMWEWIJdW_viEe/view?usp=sharing).
The same was tested using the model which provided accuracies of 77% and 74% respectively.

[240 rows x 2 columns]

	precision	recall	f1-score	support
without_mask	0.8806	0.9833	0.9291	60
with_mask_surgical	0.5227	0.7667	0.6216	60
with_mask_cloth	0.9574	0.7500	0.8411	60
with_mask_ffp2	0.7105	0.4500	0.5510	60
accuracy			0.7375	240
macro avg	0.7678	0.7375	0.7357	240
weighted avg	0.7678	0.7375	0.7357	240

Confusion matrix, without normalization

```
[[59  1  0  0]
 [ 1 46  2 11]
 [ 3 12 45  0]
 [ 4 29  0 27]]
```

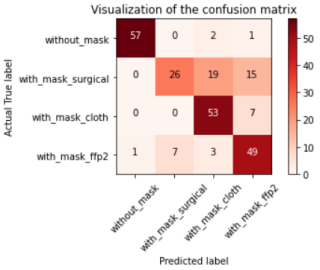


[240 rows x 2 columns]

	precision	recall	f1-score	support
without_mask	0.9828	0.9500	0.9661	60
with_mask_surgical	0.7879	0.4333	0.5591	60
with_mask_cloth	0.6883	0.8833	0.7737	60
with_mask_ffp2	0.6806	0.8167	0.7424	60
accuracy			0.7708	240
macro avg	0.7849	0.7708	0.7603	240
weighted avg	0.7849	0.7708	0.7603	240

Confusion matrix, without normalization

```
[[57  0  2  1]
 [ 0 26 19 15]
 [ 0  0 53  7]
 [ 1  7  3 49]]
```



Black White

K-fold cross-validation: Once a model is trained we need to validate it. To evaluate the performance of any machine learning model we need to test it on some unseen data. Depending on the performance of data one can say whether the model is Underfitted, Overfitted or Organised. Therefore, to improve the evaluation across the different classes we use k-fold cross-validation. We perform 10-fold cross-validation evaluation (with random shuffling) on our data. In 10-Fold Cross Validation a given dataset is split into a 10 number of folds, where each fold is the 10% of the dataset and used as a validation set at some point.

In the first iteration, the first fold (10% of the data) is used to validate the model and the rest are used to train the model. In the second iteration, second fold is used as the validation set while the rest serve as the training set. This process is repeated until each fold of the 10 folds have been used as the validation set.

The outcome of 10 fold Cross Validation is given below:

```
8605
Dataset ImageFolder
    Number of datapoints: 8605
    Root location: ./data/dataset/
    StandardTransform
Transform: Compose(
               Resize(size=(32, 32), interpolation=bilinear, max_size=None, antialias=None)
               ToTensor()
               Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
           )
>> TRAINING DATA SIZE :  6024
>> TESTING DATA SIZE :  2581
<class 'torchvision.datasets.folder.ImageFolder'>
<class 'torch.utils.data.dataset.Subset'>
<class 'torch.utils.data.dataloader.DataLoader'>
fold
0
Epoch [1/4], Step [100/189], Loss: 0.1109, Accuracy: 93.75%
Epoch [1/4], Step [200/189], Loss: 0.0231, Accuracy: 100.00%
Epoch [2/4], Step [100/189], Loss: 0.0383, Accuracy: 96.88%
Epoch [2/4], Step [200/189], Loss: 0.2527, Accuracy: 90.62%
Epoch [3/4], Step [100/189], Loss: 0.1198, Accuracy: 96.88%
Epoch [3/4], Step [200/189], Loss: 0.0425, Accuracy: 96.88%
Epoch [4/4], Step [100/189], Loss: 0.0325, Accuracy: 100.00%
Epoch [4/4], Step [200/189], Loss: 0.0043, Accuracy: 100.00%
fold
```

```
fold
1
Epoch [1/4], Step [100/189], Loss: 0.2665, Accuracy: 84.38%
Epoch [1/4], Step [200/189], Loss: 0.0222, Accuracy: 100.00%
Epoch [2/4], Step [100/189], Loss: 0.2342, Accuracy: 93.75%
Epoch [2/4], Step [200/189], Loss: 0.3409, Accuracy: 84.38%
Epoch [3/4], Step [100/189], Loss: 0.0430, Accuracy: 96.88%
Epoch [3/4], Step [200/189], Loss: 0.0730, Accuracy: 96.88%
Epoch [4/4], Step [100/189], Loss: 0.1152, Accuracy: 96.88%
Epoch [4/4], Step [200/189], Loss: 0.0118, Accuracy: 100.00%
fold
2
Epoch [1/4], Step [100/189], Loss: 0.0910, Accuracy: 93.75%
Epoch [1/4], Step [200/189], Loss: 0.0366, Accuracy: 96.88%
Epoch [2/4], Step [100/189], Loss: 0.0989, Accuracy: 96.88%
Epoch [2/4], Step [200/189], Loss: 0.0132, Accuracy: 100.00%
Epoch [3/4], Step [100/189], Loss: 0.0804, Accuracy: 96.88%
Epoch [3/4], Step [200/189], Loss: 0.0039, Accuracy: 100.00%
Epoch [4/4], Step [100/189], Loss: 0.0089, Accuracy: 100.00%
Epoch [4/4], Step [200/189], Loss: 0.0041, Accuracy: 100.00%
```

```
fold
3
Epoch [1/4], Step [100/189], Loss: 0.0103, Accuracy: 100.00%
Epoch [1/4], Step [200/189], Loss: 0.0475, Accuracy: 96.88%
Epoch [2/4], Step [100/189], Loss: 0.0382, Accuracy: 96.88%
Epoch [2/4], Step [200/189], Loss: 0.0098, Accuracy: 100.00%
Epoch [3/4], Step [100/189], Loss: 0.0037, Accuracy: 100.00%
Epoch [3/4], Step [200/189], Loss: 0.0026, Accuracy: 100.00%
Epoch [4/4], Step [100/189], Loss: 0.0617, Accuracy: 96.88%
Epoch [4/4], Step [200/189], Loss: 0.0093, Accuracy: 100.00%
fold
4
Epoch [1/4], Step [100/189], Loss: 0.0024, Accuracy: 100.00%
Epoch [1/4], Step [200/189], Loss: 0.0420, Accuracy: 96.88%
Epoch [2/4], Step [100/189], Loss: 0.0113, Accuracy: 100.00%
Epoch [2/4], Step [200/189], Loss: 0.0003, Accuracy: 100.00%
Epoch [3/4], Step [100/189], Loss: 0.0002, Accuracy: 100.00%
Epoch [3/4], Step [200/189], Loss: 0.0101, Accuracy: 100.00%
Epoch [4/4], Step [100/189], Loss: 0.0023, Accuracy: 100.00%
Epoch [4/4], Step [200/189], Loss: 0.0004, Accuracy: 100.00%
fold
5
Epoch [1/4], Step [100/189], Loss: 0.0159, Accuracy: 100.00%
Epoch [1/4], Step [200/189], Loss: 0.0085, Accuracy: 100.00%
Epoch [2/4], Step [100/189], Loss: 0.0333, Accuracy: 96.88%
Epoch [2/4], Step [200/189], Loss: 0.1990, Accuracy: 93.75%
Epoch [3/4], Step [100/189], Loss: 0.0034, Accuracy: 100.00%
Epoch [3/4], Step [200/189], Loss: 0.0035, Accuracy: 100.00%
Epoch [4/4], Step [100/189], Loss: 0.0116, Accuracy: 100.00%
Epoch [4/4], Step [200/189], Loss: 0.0385, Accuracy: 100.00%
```

```
fold
6
Epoch [1/4], Step [100/189], Loss: 0.0078, Accuracy: 100.00%
Epoch [1/4], Step [200/189], Loss: 0.0255, Accuracy: 96.88%
Epoch [2/4], Step [100/189], Loss: 0.0134, Accuracy: 100.00%
Epoch [2/4], Step [200/189], Loss: 0.0411, Accuracy: 96.88%
Epoch [3/4], Step [100/189], Loss: 0.0004, Accuracy: 100.00%
Epoch [3/4], Step [200/189], Loss: 0.0008, Accuracy: 100.00%
Epoch [4/4], Step [100/189], Loss: 0.0221, Accuracy: 100.00%
Epoch [4/4], Step [200/189], Loss: 0.0177, Accuracy: 100.00%
fold
7
Epoch [1/4], Step [100/189], Loss: 0.1289, Accuracy: 93.75%
Epoch [1/4], Step [200/189], Loss: 0.0653, Accuracy: 96.88%
Epoch [2/4], Step [100/189], Loss: 0.0035, Accuracy: 100.00%
Epoch [2/4], Step [200/189], Loss: 0.0004, Accuracy: 100.00%
Epoch [3/4], Step [100/189], Loss: 0.0083, Accuracy: 100.00%
Epoch [3/4], Step [200/189], Loss: 0.0019, Accuracy: 100.00%
Epoch [4/4], Step [100/189], Loss: 0.0183, Accuracy: 100.00%
Epoch [4/4], Step [200/189], Loss: 0.0009, Accuracy: 100.00%
fold
8
Epoch [1/4], Step [100/189], Loss: 0.0006, Accuracy: 100.00%
Epoch [1/4], Step [200/189], Loss: 0.0023, Accuracy: 100.00%
Epoch [2/4], Step [100/189], Loss: 0.0006, Accuracy: 100.00%
Epoch [2/4], Step [200/189], Loss: 0.0039, Accuracy: 100.00%
Epoch [3/4], Step [100/189], Loss: 0.0159, Accuracy: 100.00%
Epoch [3/4], Step [200/189], Loss: 0.0013, Accuracy: 100.00%
Epoch [4/4], Step [100/189], Loss: 0.1041, Accuracy: 96.88%
Epoch [4/4], Step [200/189], Loss: 0.0286, Accuracy: 96.88%
```

```
fold
9
Epoch [1/4], Step [100/189], Loss: 0.0229, Accuracy: 100.00%
Epoch [1/4], Step [200/189], Loss: 0.0005, Accuracy: 100.00%
Epoch [2/4], Step [100/189], Loss: 0.0032, Accuracy: 100.00%
Epoch [2/4], Step [200/189], Loss: 0.0026, Accuracy: 100.00%
Epoch [3/4], Step [100/189], Loss: 0.0067, Accuracy: 100.00%
Epoch [3/4], Step [200/189], Loss: 0.0098, Accuracy: 100.00%
Epoch [4/4], Step [100/189], Loss: 0.0009, Accuracy: 100.00%
Epoch [4/4], Step [200/189], Loss: 0.0000, Accuracy: 100.00%
```

<u>Cloud Model vs Local Model</u> : The colab version of the cnn model is specific to computation power of cpu, whereas the local model had ambiguities for the same as to usage of cpu and gpu devices resulting in runtime error of : Input type (torch.cuda.FloatTensor) and weight type (torch.FloatTensor) should be the same. Thus the local python notebook has been

configured to use the cudo:0 device for all computations and num_workers were also kept 0 in case of running the instance on windows pc.

References:

Following are all the resources being referred and used for implementation as well as error resolution for the project:

- https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
- BrokenPipeError: [Errno 32] Broken pipe: https://stackoverflow.com/questions/50480689/pytorch-torchvision-brokenpipeerror-errno-32-broken-pipe
- RuntimeError: Input type (torch.cuda.FloatTensor) and weight type (torch.FloatTensor) should be the same:https://discuss.pytorch.org/t/runtimeerror-input-type-torch-cuda-floattensor-and-weight-type-torch-floattensor-should-be-the-same/21782
- https://towardsdatascience.com/how-i-built-a-face-mask-detector-for-covid-19-using-pytorch-lightning-67eb3752fd61
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
- Convolution Layers :https://pytorch.org/docs/stable/nn.html#convolution-layers
- Layer Dimensions walkthrough : https://towardsdatascience.com/pytorch-layer-dimensions-what-sizes-should-they-be-and-why-4265a41e01fd
- Kernel size : https://stats.stackexchange.com/questions/296679/what-does-kernel-size-mean/339265
- Image Classification :https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/
- Architecture: https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/
- K fold cross validation in pytorch: https://medium.com/dataseries/k-fold-cross-validation-with-pytorch-and-sklearn-d094aa00105f
- K fold hands on: https://www.machinecurve.com/index.php/2021/02/03/how-to-use-k-fold-cross-validation-with-pytorch/