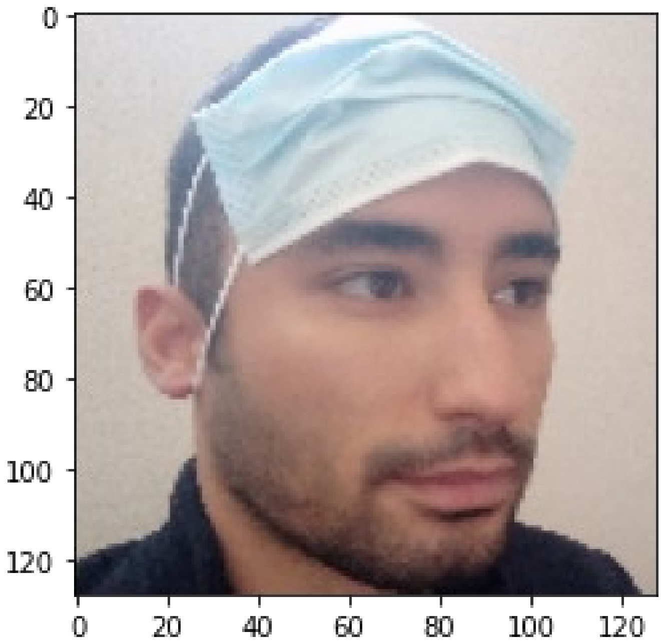
Notebook for fine tuning pretrained torchvision models

The 'create_dataset' notebook should have already been run to create the tensors for training and testing the models.

```
import sys

import os, pickle
import numpy as np
import torch
from torch import nn, optim
import torch.nn.functional as F
from torch.optim.lr_scheduler import ReduceLROnPlateau

import torchvision.models as models

from sklearn.metrics import classification_report, confusion_matrix

import seaborn as sns
import matplotlib.pyplot as plt

from config import Config
from process_images import ImageUtils

%matplotlib inline
%config IPCompleter.greedy=True
%config Completer.use_jedi = False

# init config which holds various constants
conf = Config()
conf.im_size = 128

# init image utils
im_utils = ImageUtils(conf)
```

Load the combined dataset (created from images in WWMR, FMD, MFN, and GAN sets). The data is balanced with approximately 1000 images for each of the three classes. The data is already shuffled.

```
x = torch.load('%s/x_%d.pt' % (conf.combined_data_path, conf.im_size))
y = torch.load('%s/y.pt' %
(conf.combined_data_path)).type(torch.LongTensor)
with open('%s/im_names.txt' % (conf.combined_data_path), 'r') as f:
    im_names = f.read().split('\n')

x.shape #torch.Size([2973, 3, 128, 128])

torch.Size([2973, 3, 128, 128])

#show an image
im_utils.show_image(x[2])
```



```python
# load 3 class label to index map
with open('%s/lab2idx.pkl' % conf.combined_data_path, 'rb') as f:
    lab2idx = pickle.load(f)
lab2idx  #{'without_mask': 0, 'with_mask': 1, 'mask_weared_incorrect':
2}
```

```
{'without_mask': 0, 'with_mask': 1, 'mask_weared_incorrect': 2}
```

```python
# divide data into train/crossval/test

x_test = x[:500]
y_test = y[:500]
test_im_names = im_names[:500]

x_cv = x[500:750]
y_cv = y[500:750]
cv_im_names = im_names[500:750]

x_train = x[750:]
y_train = y[750:]
train_im_names = im_names[750:]

#train: (torch.Size([2223, 3, 128, 128]),
#cv: torch.Size([250, 3, 128, 128]),
#test: torch.Size([500, 3, 128, 128]))
x_train.shape, x_cv.shape, x_test.shape
```

```
(torch.Size([2223, 3, 128, 128]),
 torch.Size([250, 3, 128, 128]),
 torch.Size([500, 3, 128, 128]))
```

Train GoogLeNet on 3 class combined dataset

```
def finetune_model(model, optimizer, model_path, lab2idx,
                   x_train, y_train, x_cv, y_cv, x_test, y_test,
                   device, batch_size=32, n_epochs=20):
    '''Fine tunes a pretrained model.'''
    model.train()

    #use cross entropy loss objective function
    criterion = nn.CrossEntropyLoss()

    #use a learning rate scheduler to reduce the LR upon failure to
improve
    scheduler = ReduceLROnPlateau(optimizer, 'min', patience=10)

    #store losses for plotting
    train_losses = []
    cv_losses = []

    #create the model checkpoint folder
    if not os.path.exists(model_path):
        os.mkdir(model_path)

    #track the best CV accuracy for checkpointing
    best_acc = 0

    #optional early stopping if no CV improvement is observed for a
given number of epochs
    no_improvement = 0

    for epoch in range(n_epochs):
        print('Epoch: %d' % epoch)

        #shuffle the training data each epoch to combat overfitting of
specific batches
        idx = torch.randperm(x_train.size(0))
        x_train = x_train[idx]
        y_train = y_train[idx]

        #the batch number
        ep_ttl = 0

        for j in range(0, x_train.size(0), batch_size):
            ep_ttl += 1

            #select a batch
            x_batch = x_train[j:j+batch_size].to(device)
            y_batch = y_train[j:j+batch_size].to(device)

            #reset gradient for this iteration
            optimizer.zero_grad()
```

```
            #run the data through the model
            output = model(x_batch)

            #get the cross entropy loss
            loss = criterion(output, y_batch)

            #calculate the gradients
            loss.backward()

            #update the parameters from the gradients
            optimizer.step()

            #print status every 50 batches
            if ep_ttl%50==0:
                print('Epoch: %d, Batch: %d, Loss: %.6f' % (epoch, j,
loss.item()))
                train_losses.append(loss.item())

        #get the accuracy and loss of the cross validation data to see
whether to store a checkpoint
        print('Testing model...')
        acc, cv_loss = test(model, criterion, x_cv, y_cv, lab2idx)

        #pass CV loss to LR scheduler to decide whether the LR should
be lowered
        scheduler.step(cv_loss)

        #store CV loss for plotting
        cv_losses.append(cv_loss)
        print('CV accuracy %.6f, prev best acc: %.6f %s\n' % (acc,
best_acc, '!! IMPROVED !!' if acc>best_acc else ''))

        #store model checkpoint if the best CV accuracy has been
surpassed
        if acc>best_acc:
            best_acc = acc
            no_improvement = 0
            print('Saving model...')
            torch.save(model.state_dict(), '%s/model.pt' % model_path)
            torch.save(optimizer.state_dict(), '%s/optimizer.pt' %
model_path)
        else:
            no_improvement += 1

        #no improvements for a while, break early
        if no_improvement >= 10:
            print('no improvement in several epochs, breaking')
            break
```

```
    #load the stored best checkpoint
    model.load_state_dict(torch.load('%s/model.pt' % model_path))

    #run the best model on the test data
    test_acc, _ = test(model, criterion, x_test, y_test, lab2idx,
True)
    print('final test accuracy: %.6f' % test_acc)

    #return the model in eval mode
    model.eval()

    return model, train_losses, cv_losses


def test(model, criterion, x_test, y_test, lab2idx,
print_report=False):
    '''Test the model'''
    model.eval()

    correct = 0
    loss = 0
    with torch.no_grad():
        #run the test data through the model
        output = model(x_test)

        #get the test loss
        loss = criterion(output, y_test)

        #select the indices of the maximum output values/prediction
        _, y_pred = torch.max(output, 1)

        #compare them with the target digits and sum correct
predictions
        correct = y_pred.eq(y_test).sum()

    #calculate the accuracy
    acc = correct / y_test.size()[0]

    print('Test accuracy %.6f, %d of %d' % (acc, correct,
y_test.size(0)))

    #print the classification report and confusion matrix
    if print_report:
        idx2lab = {v:k for k,v in lab2idx.items()}
        class_labels = [idx2lab[i] for i in range(len(idx2lab))]

        print('\n\n')
        print(classification_report(y_test.tolist(), y_pred.tolist(),
target_names=class_labels, digits=4))
```

```
        print('\n\n')

        cm = confusion_matrix(y_test.tolist(), y_pred.tolist())
        fig, ax = plt.subplots(figsize=(12,10))
        f = sns.heatmap(cm, annot=True, fmt='d',
xticklabels=class_labels, yticklabels=class_labels, ax=ax)

    model.train()

    return acc, loss.item()

# retrieve the publicly available GoogLeNet pretrained model

model = models.googlenet(pretrained=True)
model
```

```
GoogLeNet(
  (conv1): BasicConv2d(
    (conv): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2),
padding=(3, 3), bias=False)
    (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (maxpool1): MaxPool2d(kernel_size=3, stride=2, padding=0,
dilation=1, ceil_mode=True)
  (conv2): BasicConv2d(
    (conv): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (conv3): BasicConv2d(
    (conv): Conv2d(64, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (maxpool2): MaxPool2d(kernel_size=3, stride=2, padding=0,
dilation=1, ceil_mode=True)
  (inception3a): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(192, 96, kernel_size=(1, 1), stride=(1, 1),
bias=False)
```

```
        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(96, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(192, 16, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(16, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception3b): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
```

```
        (conv): Conv2d(128, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(256, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(32, 96, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (maxpool3): MaxPool2d(kernel_size=3, stride=2, padding=0,
dilation=1, ceil_mode=True)
  (inception4a): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(480, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(480, 96, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(96, 208, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
```

```
        (bn): BatchNorm2d(208, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(480, 16, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(16, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(16, 48, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(480, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4b): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(512, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(112, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(112, 224, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(224, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
```

```
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 24, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(24, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4c): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 24, kernel_size=(1, 1), stride=(1, 1),
bias=False)
```

```
        (bn): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(24, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4d): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(512, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(112, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 144, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(144, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(144, 288, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(288, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
```

```
        (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4e): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(528, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(528, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(160, 320, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(528, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(528, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (maxpool4): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=True)
  (inception5a): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(832, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(832, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(160, 320, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(832, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
```

```
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception5b): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(832, 384, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(832, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(832, 48, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(48, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
```

```
        (conv): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (aux1): None
  (aux2): None
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (dropout): Dropout(p=0.2, inplace=False)
  (fc): Linear(in_features=1024, out_features=1000, bias=True)
)
```

```
#change the output layer from 1000 nodes to the 3 nodes for our
classes
model.fc = nn.Linear(1024, len(lab2idx))

#initialize the Adam optimizer with a low learning rate to prevent
'catastrophic forgetting'
optimizer = optim.AdamW(model.parameters(), lr=1e-4)

# train model and store when crossval score increases

device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')

model_path = 'googlenet_3class_model'
if not os.path.exists(model_path):
    os.mkdir(model_path)

model, train_losses, cv_losses = finetune_model(model, optimizer,
model_path, lab2idx,
                                                x_train, y_train,
x_cv, y_cv, x_test, y_test,
                                                device, batch_size=8,
n_epochs=30)
```

```
Epoch: 0
Epoch: 0, Batch: 392, Loss: 0.442851
Epoch: 0, Batch: 792, Loss: 0.705809
Epoch: 0, Batch: 1192, Loss: 0.403428
Epoch: 0, Batch: 1592, Loss: 1.555436
Epoch: 0, Batch: 1992, Loss: 0.350390
Testing model...
Test accuracy 0.924000, 231 of 250
CV accuracy 0.924000, prev best acc: 0.000000 !! IMPROVED !!
Saving model...
Epoch: 1
Epoch: 1, Batch: 392, Loss: 0.170749
Epoch: 1, Batch: 792, Loss: 0.376148
```

```
Epoch: 1, Batch: 1192, Loss: 0.066875
Epoch: 1, Batch: 1592, Loss: 0.511688
Epoch: 1, Batch: 1992, Loss: 0.188354
Testing model...
Test accuracy 0.944000, 236 of 250
CV accuracy 0.944000, prev best acc: 0.924000 !! IMPROVED !!
Saving model...
Epoch: 2
Epoch: 2, Batch: 392, Loss: 1.756253
Epoch: 2, Batch: 792, Loss: 0.075338
Epoch: 2, Batch: 1192, Loss: 0.312643
Epoch: 2, Batch: 1592, Loss: 0.014179
Epoch: 2, Batch: 1992, Loss: 0.019474
Testing model...
Test accuracy 0.924000, 231 of 250
CV accuracy 0.924000, prev best acc: 0.944000
Epoch: 3
Epoch: 3, Batch: 392, Loss: 0.245336
Epoch: 3, Batch: 792, Loss: 0.073457
Epoch: 3, Batch: 1192, Loss: 0.045530
Epoch: 3, Batch: 1592, Loss: 0.011326
Epoch: 3, Batch: 1992, Loss: 0.012779
Testing model...
Test accuracy 0.952000, 238 of 250
CV accuracy 0.952000, prev best acc: 0.944000 !! IMPROVED !!
Saving model...
Epoch: 4
Epoch: 4, Batch: 392, Loss: 0.032724
Epoch: 4, Batch: 792, Loss: 0.005281
Epoch: 4, Batch: 1192, Loss: 0.015984
Epoch: 4, Batch: 1592, Loss: 0.006722
Epoch: 4, Batch: 1992, Loss: 0.023144
Testing model...
Test accuracy 0.976000, 244 of 250
CV accuracy 0.976000, prev best acc: 0.952000 !! IMPROVED !!
Saving model...
Epoch: 5
Epoch: 5, Batch: 392, Loss: 0.100661
Epoch: 5, Batch: 792, Loss: 0.086896
Epoch: 5, Batch: 1192, Loss: 0.321226
Epoch: 5, Batch: 1592, Loss: 0.143771
Epoch: 5, Batch: 1992, Loss: 0.111167
Testing model...
Test accuracy 0.964000, 241 of 250
CV accuracy 0.964000, prev best acc: 0.976000
Epoch: 6
Epoch: 6, Batch: 392, Loss: 0.002216
Epoch: 6, Batch: 792, Loss: 0.002270
Epoch: 6, Batch: 1192, Loss: 0.036031
Epoch: 6, Batch: 1592, Loss: 0.004360
```

```
Epoch: 6, Batch: 1992, Loss: 0.010659
Testing model...
Test accuracy 0.964000, 241 of 250
CV accuracy 0.964000, prev best acc: 0.976000
Epoch: 7
Epoch: 7, Batch: 392, Loss: 0.001537
Epoch: 7, Batch: 792, Loss: 0.002823
Epoch: 7, Batch: 1192, Loss: 0.040850
Epoch: 7, Batch: 1592, Loss: 0.004131
Epoch: 7, Batch: 1992, Loss: 0.175634
Testing model...
Test accuracy 0.952000, 238 of 250
CV accuracy 0.952000, prev best acc: 0.976000
Epoch: 8
Epoch: 8, Batch: 392, Loss: 0.024831
Epoch: 8, Batch: 792, Loss: 0.003983
Epoch: 8, Batch: 1192, Loss: 0.049688
Epoch: 8, Batch: 1592, Loss: 0.070681
Epoch: 8, Batch: 1992, Loss: 0.004775
Testing model...
Test accuracy 0.948000, 237 of 250
CV accuracy 0.948000, prev best acc: 0.976000
Epoch: 9
Epoch: 9, Batch: 392, Loss: 0.003844
Epoch: 9, Batch: 792, Loss: 0.242795
Epoch: 9, Batch: 1192, Loss: 0.003337
Epoch: 9, Batch: 1592, Loss: 0.022558
Epoch: 9, Batch: 1992, Loss: 0.002255
Testing model...
Test accuracy 0.956000, 239 of 250
CV accuracy 0.956000, prev best acc: 0.976000
Epoch: 10
Epoch: 10, Batch: 392, Loss: 0.738080
Epoch: 10, Batch: 792, Loss: 0.006091
Epoch: 10, Batch: 1192, Loss: 0.009036
Epoch: 10, Batch: 1592, Loss: 0.004141
Epoch: 10, Batch: 1992, Loss: 0.044993
Testing model...
Test accuracy 0.952000, 238 of 250
CV accuracy 0.952000, prev best acc: 0.976000
Epoch: 11
Epoch: 11, Batch: 392, Loss: 0.024131
Epoch: 11, Batch: 792, Loss: 0.000455
Epoch: 11, Batch: 1192, Loss: 0.159794
Epoch: 11, Batch: 1592, Loss: 0.016084
Epoch: 11, Batch: 1992, Loss: 0.033072
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 12
```

```
Epoch: 12, Batch: 392, Loss: 0.001677
Epoch: 12, Batch: 792, Loss: 0.000900
Epoch: 12, Batch: 1192, Loss: 0.002060
Epoch: 12, Batch: 1592, Loss: 0.600186
Epoch: 12, Batch: 1992, Loss: 0.002127
Testing model...
Test accuracy 0.976000, 244 of 250
CV accuracy 0.976000, prev best acc: 0.976000
Epoch: 13
Epoch: 13, Batch: 392, Loss: 0.012122
Epoch: 13, Batch: 792, Loss: 0.004420
Epoch: 13, Batch: 1192, Loss: 0.002689
Epoch: 13, Batch: 1592, Loss: 0.004589
Epoch: 13, Batch: 1992, Loss: 0.000978
Testing model...
Test accuracy 0.972000, 243 of 250
CV accuracy 0.972000, prev best acc: 0.976000
Epoch: 14
Epoch: 14, Batch: 392, Loss: 0.002583
Epoch: 14, Batch: 792, Loss: 0.000918
Epoch: 14, Batch: 1192, Loss: 0.000493
Epoch: 14, Batch: 1592, Loss: 0.004814
Epoch: 14, Batch: 1992, Loss: 0.001706
Testing model...
Test accuracy 0.964000, 241 of 250
CV accuracy 0.964000, prev best acc: 0.976000
Epoch: 15
Epoch: 15, Batch: 392, Loss: 0.011343
Epoch: 15, Batch: 792, Loss: 0.317342
Epoch: 15, Batch: 1192, Loss: 0.008835
Epoch: 15, Batch: 1592, Loss: 0.005481
Epoch: 15, Batch: 1992, Loss: 0.177189
Testing model...
Test accuracy 0.960000, 240 of 250
CV accuracy 0.960000, prev best acc: 0.976000
Epoch: 16
Epoch: 16, Batch: 392, Loss: 0.066142
Epoch: 16, Batch: 792, Loss: 0.019954
Epoch: 16, Batch: 1192, Loss: 0.003688
Epoch: 16, Batch: 1592, Loss: 0.007314
Epoch: 16, Batch: 1992, Loss: 0.001208
Testing model...
Test accuracy 0.964000, 241 of 250
CV accuracy 0.964000, prev best acc: 0.976000
Epoch: 17
Epoch: 17, Batch: 392, Loss: 0.002066
Epoch: 17, Batch: 792, Loss: 0.007471
Epoch: 17, Batch: 1192, Loss: 0.003849
Epoch: 17, Batch: 1592, Loss: 0.003116
Epoch: 17, Batch: 1992, Loss: 0.010420
```

```
Testing model...
Test accuracy 0.960000, 240 of 250
CV accuracy 0.960000, prev best acc: 0.976000
Epoch: 18
Epoch: 18, Batch: 392, Loss: 0.286319
Epoch: 18, Batch: 792, Loss: 0.001501
Epoch: 18, Batch: 1192, Loss: 0.003803
Epoch: 18, Batch: 1592, Loss: 0.002907
Epoch: 18, Batch: 1992, Loss: 0.000429
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 19
Epoch: 19, Batch: 392, Loss: 0.000618
Epoch: 19, Batch: 792, Loss: 0.000732
Epoch: 19, Batch: 1192, Loss: 0.006972
Epoch: 19, Batch: 1592, Loss: 0.008424
Epoch: 19, Batch: 1992, Loss: 0.002818
Testing model...
Test accuracy 0.964000, 241 of 250
CV accuracy 0.964000, prev best acc: 0.976000
Epoch: 20
Epoch: 20, Batch: 392, Loss: 0.000389
Epoch: 20, Batch: 792, Loss: 0.012294
Epoch: 20, Batch: 1192, Loss: 0.001993
Epoch: 20, Batch: 1592, Loss: 0.028788
Epoch: 20, Batch: 1992, Loss: 0.006319
Testing model...
Test accuracy 0.964000, 241 of 250
CV accuracy 0.964000, prev best acc: 0.976000
Epoch: 21
Epoch: 21, Batch: 392, Loss: 0.003896
Epoch: 21, Batch: 792, Loss: 0.031396
Epoch: 21, Batch: 1192, Loss: 0.002383
Epoch: 21, Batch: 1592, Loss: 0.001240
Epoch: 21, Batch: 1992, Loss: 0.000424
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 22
Epoch: 22, Batch: 392, Loss: 0.004411
Epoch: 22, Batch: 792, Loss: 0.000356
Epoch: 22, Batch: 1192, Loss: 0.001221
Epoch: 22, Batch: 1592, Loss: 0.003484
Epoch: 22, Batch: 1992, Loss: 0.000387
Testing model...
Test accuracy 0.964000, 241 of 250
CV accuracy 0.964000, prev best acc: 0.976000
Epoch: 23
Epoch: 23, Batch: 392, Loss: 0.001915
```

```
Epoch: 23, Batch: 792, Loss: 0.000542
Epoch: 23, Batch: 1192, Loss: 0.000561
Epoch: 23, Batch: 1592, Loss: 0.000307
Epoch: 23, Batch: 1992, Loss: 0.002889
Testing model...
Test accuracy 0.972000, 243 of 250
CV accuracy 0.972000, prev best acc: 0.976000
Epoch: 24
Epoch: 24, Batch: 392, Loss: 0.005200
Epoch: 24, Batch: 792, Loss: 0.009985
Epoch: 24, Batch: 1192, Loss: 0.000968
Epoch: 24, Batch: 1592, Loss: 0.000171
Epoch: 24, Batch: 1992, Loss: 0.000498
Testing model...
Test accuracy 0.972000, 243 of 250
CV accuracy 0.972000, prev best acc: 0.976000
Epoch: 25
Epoch: 25, Batch: 392, Loss: 0.000954
Epoch: 25, Batch: 792, Loss: 0.000454
Epoch: 25, Batch: 1192, Loss: 0.000134
Epoch: 25, Batch: 1592, Loss: 0.001581
Epoch: 25, Batch: 1992, Loss: 0.000497
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 26
Epoch: 26, Batch: 392, Loss: 0.000535
Epoch: 26, Batch: 792, Loss: 0.000456
Epoch: 26, Batch: 1192, Loss: 0.000134
Epoch: 26, Batch: 1592, Loss: 0.000526
Epoch: 26, Batch: 1992, Loss: 0.000223
Testing model...
Test accuracy 0.964000, 241 of 250
CV accuracy 0.964000, prev best acc: 0.976000
Epoch: 27

Epoch: 27, Batch: 392, Loss: 0.001420
Epoch: 27, Batch: 792, Loss: 0.000417
Epoch: 27, Batch: 1192, Loss: 0.000324
Epoch: 27, Batch: 1592, Loss: 0.001162
Epoch: 27, Batch: 1992, Loss: 0.000918
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 28
Epoch: 28, Batch: 392, Loss: 0.001898
Epoch: 28, Batch: 792, Loss: 0.000139
Epoch: 28, Batch: 1192, Loss: 0.004928
Epoch: 28, Batch: 1592, Loss: 0.000850
Epoch: 28, Batch: 1992, Loss: 0.000729
```

```
Testing model...
Test accuracy 0.960000, 240 of 250
CV accuracy 0.960000, prev best acc: 0.976000
Epoch: 29
Epoch: 29, Batch: 392, Loss: 0.000673
Epoch: 29, Batch: 792, Loss: 0.000346
Epoch: 29, Batch: 1192, Loss: 0.007288
Epoch: 29, Batch: 1592, Loss: 0.000248
Epoch: 29, Batch: 1992, Loss: 0.000655
Testing model...
Test accuracy 0.964000, 241 of 250
CV accuracy 0.964000, prev best acc: 0.976000
Epoch: 30
Epoch: 30, Batch: 392, Loss: 0.000175
Epoch: 30, Batch: 792, Loss: 0.000553
Epoch: 30, Batch: 1192, Loss: 0.005110
Epoch: 30, Batch: 1592, Loss: 0.001050
Epoch: 30, Batch: 1992, Loss: 0.000171
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 31
Epoch: 31, Batch: 392, Loss: 0.000625
Epoch: 31, Batch: 792, Loss: 0.000497
Epoch: 31, Batch: 1192, Loss: 0.000444
Epoch: 31, Batch: 1592, Loss: 0.000445
Epoch: 31, Batch: 1992, Loss: 0.001782
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 32
Epoch: 32, Batch: 392, Loss: 0.000477
Epoch: 32, Batch: 792, Loss: 0.010259
Epoch: 32, Batch: 1192, Loss: 0.000163
Epoch: 32, Batch: 1592, Loss: 0.001076
Epoch: 32, Batch: 1992, Loss: 0.000654
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 33
Epoch: 33, Batch: 392, Loss: 0.004162
Epoch: 33, Batch: 792, Loss: 0.001038
Epoch: 33, Batch: 1192, Loss: 0.002234
Epoch: 33, Batch: 1592, Loss: 0.000135
Epoch: 33, Batch: 1992, Loss: 0.001317
Testing model...
Test accuracy 0.972000, 243 of 250
CV accuracy 0.972000, prev best acc: 0.976000
Epoch: 34
Epoch: 34, Batch: 392, Loss: 0.000609
```

```
Epoch: 34, Batch: 792, Loss: 0.000415
Epoch: 34, Batch: 1192, Loss: 0.038002
Epoch: 34, Batch: 1592, Loss: 0.000318
Epoch: 34, Batch: 1992, Loss: 0.009826
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 35
Epoch: 35, Batch: 392, Loss: 0.000385
Epoch: 35, Batch: 792, Loss: 0.000757
Epoch: 35, Batch: 1192, Loss: 0.000381
Epoch: 35, Batch: 1592, Loss: 0.000227
Epoch: 35, Batch: 1992, Loss: 0.001362
Testing model...
Test accuracy 0.972000, 243 of 250
CV accuracy 0.972000, prev best acc: 0.976000
Epoch: 36
Epoch: 36, Batch: 392, Loss: 0.000358
Epoch: 36, Batch: 792, Loss: 0.010055
Epoch: 36, Batch: 1192, Loss: 0.000596
Epoch: 36, Batch: 1592, Loss: 0.001279
Epoch: 36, Batch: 1992, Loss: 0.001465
Testing model...
Test accuracy 0.972000, 243 of 250
CV accuracy 0.972000, prev best acc: 0.976000
Epoch: 37
Epoch: 37, Batch: 392, Loss: 0.001163
Epoch: 37, Batch: 792, Loss: 0.001693
Epoch: 37, Batch: 1192, Loss: 0.012746
Epoch: 37, Batch: 1592, Loss: 0.001677
Epoch: 37, Batch: 1992, Loss: 0.007605
Testing model...
Test accuracy 0.964000, 241 of 250
CV accuracy 0.964000, prev best acc: 0.976000
Epoch: 38
Epoch: 38, Batch: 392, Loss: 0.000651
Epoch: 38, Batch: 792, Loss: 0.000226
Epoch: 38, Batch: 1192, Loss: 0.000452
Epoch: 38, Batch: 1592, Loss: 2.764525
Epoch: 38, Batch: 1992, Loss: 0.000281
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 39
Epoch: 39, Batch: 392, Loss: 0.005674
Epoch: 39, Batch: 792, Loss: 0.000102
Epoch: 39, Batch: 1192, Loss: 0.000328
Epoch: 39, Batch: 1592, Loss: 0.001578
Epoch: 39, Batch: 1992, Loss: 0.004853
Testing model...
```

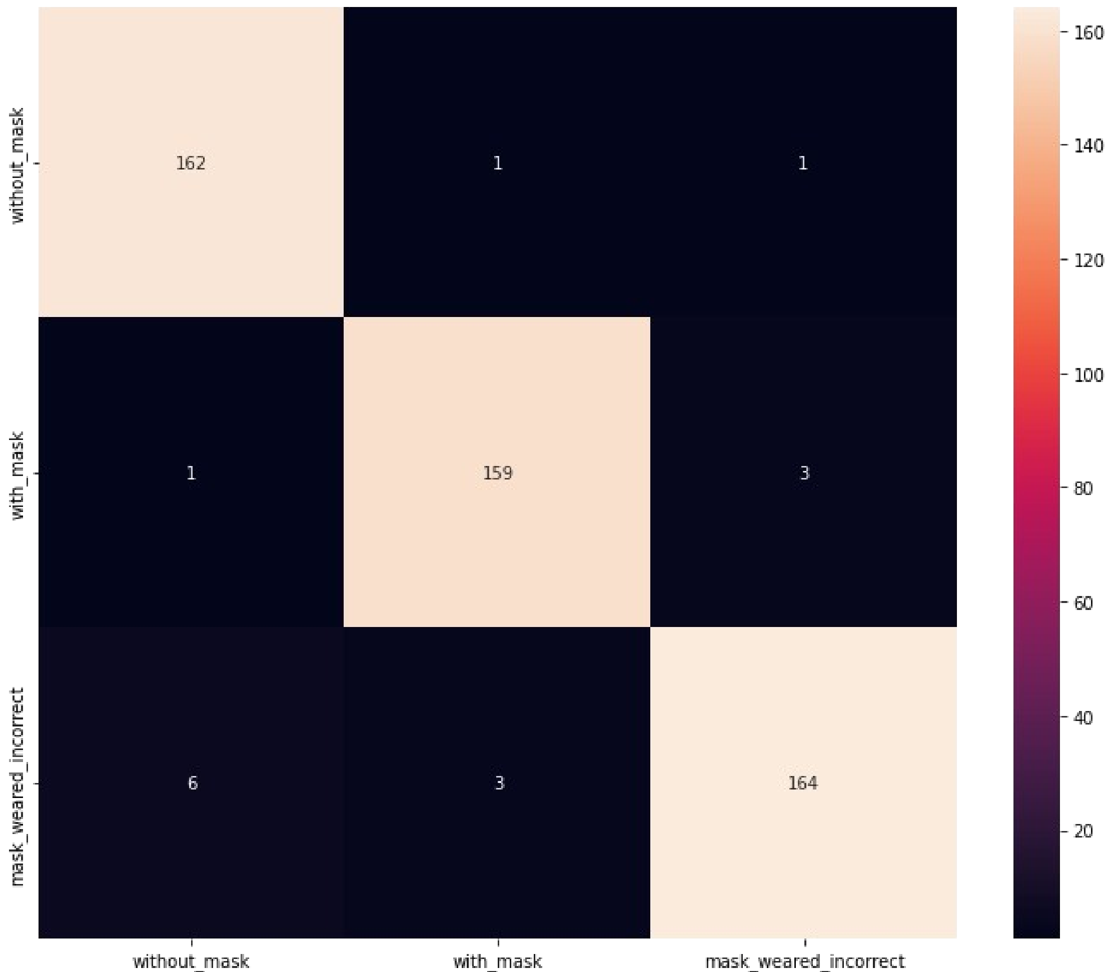
```
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 40
Epoch: 40, Batch: 392, Loss: 0.000496
Epoch: 40, Batch: 792, Loss: 0.000315
Epoch: 40, Batch: 1192, Loss: 0.001703
Epoch: 40, Batch: 1592, Loss: 0.000236
Epoch: 40, Batch: 1992, Loss: 0.000339
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 41
Epoch: 41, Batch: 392, Loss: 0.004552
Epoch: 41, Batch: 792, Loss: 0.000201
Epoch: 41, Batch: 1192, Loss: 0.001440
Epoch: 41, Batch: 1592, Loss: 0.001253
Epoch: 41, Batch: 1992, Loss: 0.000771
Testing model...
Test accuracy 0.972000, 243 of 250
CV accuracy 0.972000, prev best acc: 0.976000
Epoch: 42
Epoch: 42, Batch: 392, Loss: 0.000348
Epoch: 42, Batch: 792, Loss: 0.001171
Epoch: 42, Batch: 1192, Loss: 0.002031
Epoch: 42, Batch: 1592, Loss: 0.000656
Epoch: 42, Batch: 1992, Loss: 0.000596
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 43
Epoch: 43, Batch: 392, Loss: 0.002203
Epoch: 43, Batch: 792, Loss: 0.001087
Epoch: 43, Batch: 1192, Loss: 0.000298
Epoch: 43, Batch: 1592, Loss: 0.000403
Epoch: 43, Batch: 1992, Loss: 0.000505
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 44
Epoch: 44, Batch: 392, Loss: 0.001828
Epoch: 44, Batch: 792, Loss: 0.000341
Epoch: 44, Batch: 1192, Loss: 0.003247
Epoch: 44, Batch: 1592, Loss: 0.001596
Epoch: 44, Batch: 1992, Loss: 0.000216
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 45
Epoch: 45, Batch: 392, Loss: 0.000460
Epoch: 45, Batch: 792, Loss: 0.005060
```

```
Epoch: 45, Batch: 1192, Loss: 0.002148
Epoch: 45, Batch: 1592, Loss: 0.001824
Epoch: 45, Batch: 1992, Loss: 0.000360
Testing model...
Test accuracy 0.972000, 243 of 250
CV accuracy 0.972000, prev best acc: 0.976000
Epoch: 46
Epoch: 46, Batch: 392, Loss: 0.002654
Epoch: 46, Batch: 792, Loss: 0.008595
Epoch: 46, Batch: 1192, Loss: 0.000155
Epoch: 46, Batch: 1592, Loss: 0.000435
Epoch: 46, Batch: 1992, Loss: 0.002592
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 47
Epoch: 47, Batch: 392, Loss: 0.001186
Epoch: 47, Batch: 792, Loss: 0.000692
Epoch: 47, Batch: 1192, Loss: 0.000352
Epoch: 47, Batch: 1592, Loss: 0.000758
Epoch: 47, Batch: 1992, Loss: 0.000818
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 48
Epoch: 48, Batch: 392, Loss: 0.003147
Epoch: 48, Batch: 792, Loss: 0.000638
Epoch: 48, Batch: 1192, Loss: 0.013208
Epoch: 48, Batch: 1592, Loss: 0.000355
Epoch: 48, Batch: 1992, Loss: 0.003210
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Epoch: 49
Epoch: 49, Batch: 392, Loss: 0.001278
Epoch: 49, Batch: 792, Loss: 0.000709
Epoch: 49, Batch: 1192, Loss: 0.000313
Epoch: 49, Batch: 1592, Loss: 0.000919
Epoch: 49, Batch: 1992, Loss: 0.000755
Testing model...
Test accuracy 0.968000, 242 of 250
CV accuracy 0.968000, prev best acc: 0.976000
Test accuracy 0.970000, 485 of 500
```

	precision	recall	f1-score	support
without_mask	0.9586	0.9878	0.9730	164
with_mask	0.9755	0.9755	0.9755	163

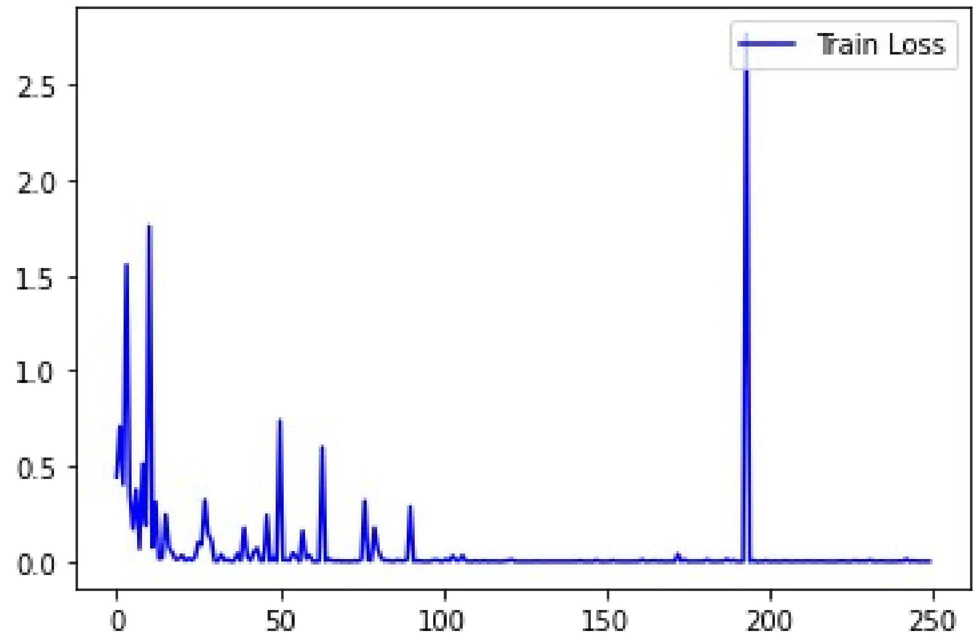
```
mask_weared_incorrect     0.9762    0.9480    0.9619       173

             accuracy                         0.9700       500
            macro avg     0.9701    0.9704    0.9701       500
         weighted avg     0.9702    0.9700    0.9699       500
```

```
final test accuracy: 0.970000
```



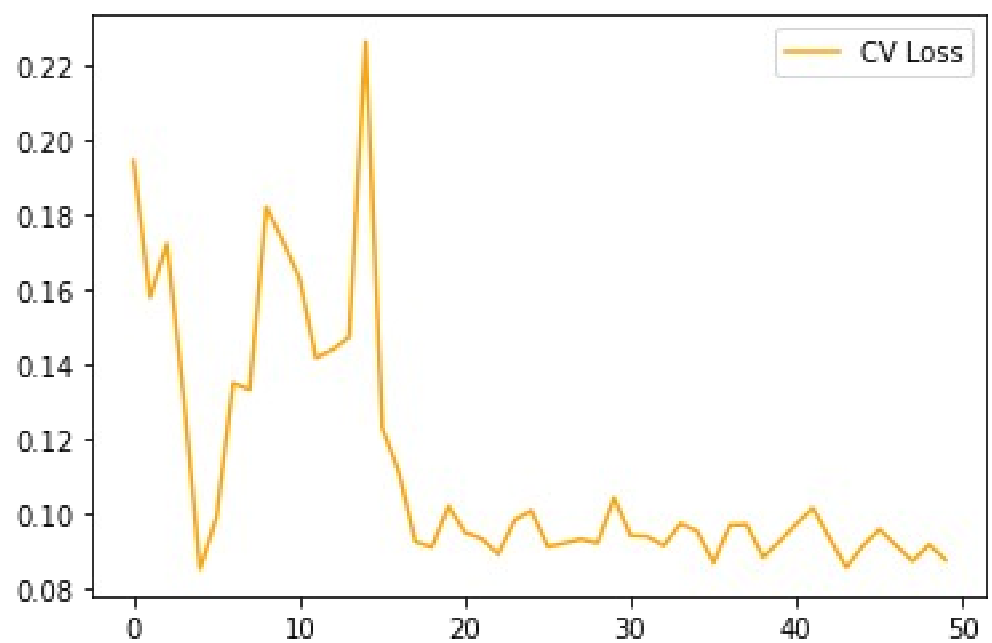

```python
#plot the train losses

fig = plt.figure()
plt.plot(train_losses, color='blue')
plt.legend(['Train Loss'], loc='upper right')
```

```
<matplotlib.legend.Legend at 0x226f64302e0>
```




```
#plot the CV losses

fig = plt.figure()
plt.plot(cv_losses, color='orange')
plt.legend(['CV Loss'], loc='upper right')
plt.show()
```

```
<matplotlib.legend.Legend at 0x226fded40a0>
```



Visualize incorrect predictions

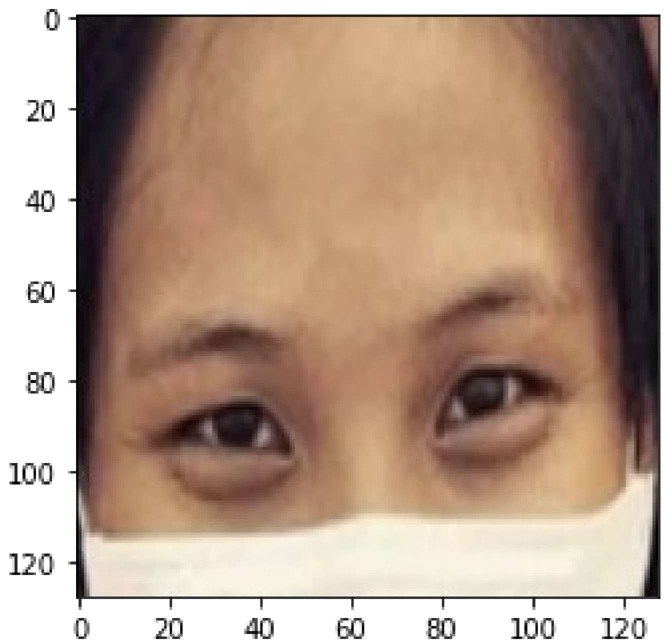
```
#get the indices of the incorrectly predicted test images
model.eval()
with torch.no_grad():
    output = model(x_test)

    #select the indices of the maximum output values/prediction
    _, y_pred = torch.max(output, 1)

    #compare them with the target digits and sum correct predictions
    correct = y_pred.eq(y_test)

wrong_idx = (correct==False).nonzero(as_tuple=True)[0]

#[0, 30, 110, 113, 155, 162, 290, 317, 320, 341, 351, 414, 420, 441,
494]
wrong_idx
```

```
c:\ml\env\lib\site-packages\torch\nn\functional.py:780: UserWarning:
Note that order of the arguments: ceil_mode and return_indices will
changeto match the args list in nn.MaxPool2d in a future release.
  warnings.warn("Note that order of the arguments: ceil_mode and
return_indices will change"
```

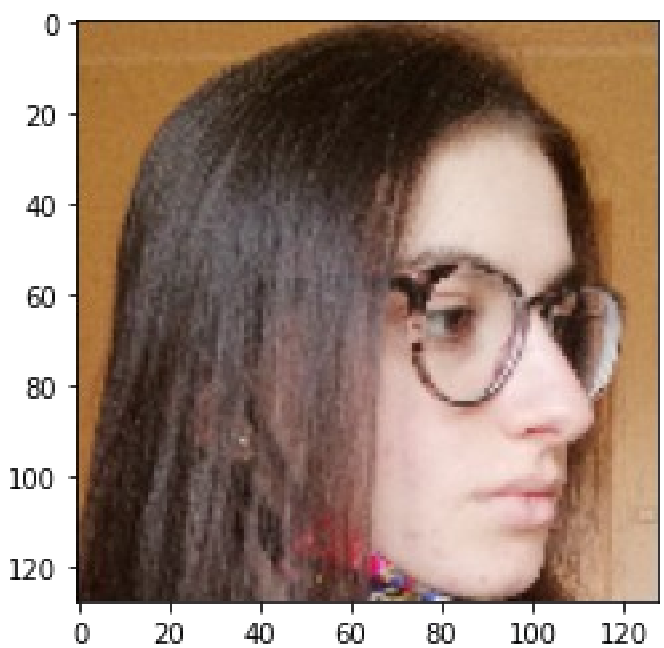
```
tensor([  0,  30, 110, 113, 155, 162, 290, 317, 320, 341, 351, 414,
420, 441,
        494])
```

```
idx2lab = {v:k for k,v in lab2idx.items()}

for idx in wrong_idx.tolist():
    #get the image
    im = x_test[idx]

    #get the ground truth and predicted class
    corr = y_test[idx].item()
    pred = y_pred[idx].item()

    #get the image file name
    nm = test_im_names[idx]

    #show the image
    print('\n\n%s - predicted: %s, ground truth: %s' % (nm,
idx2lab[pred], idx2lab[corr]))
    im_utils.show_image(im)
```

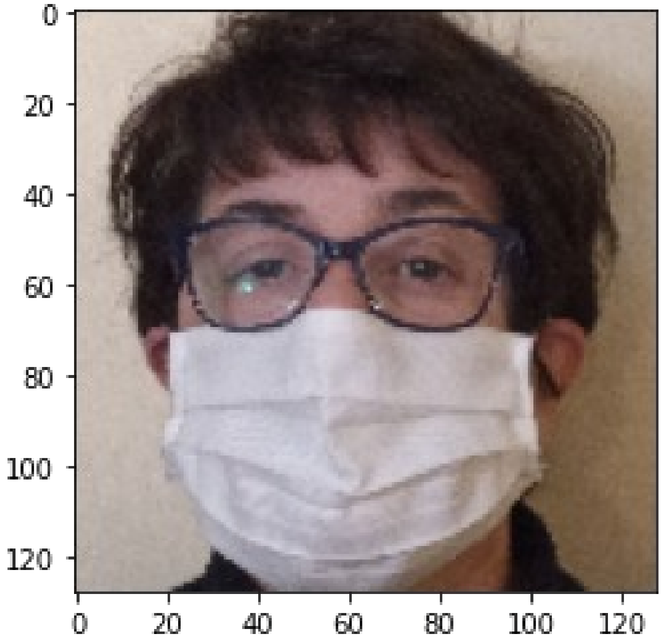
```
maksssksksss264.png - predicted: without_mask, ground truth: with_mask
```



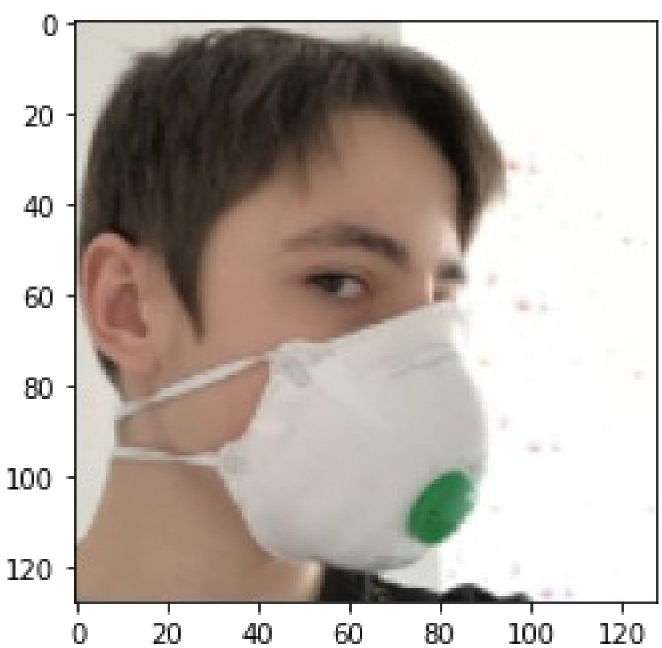
0042_MRTN_DRNV_0045 - predicted: without_mask, ground truth: mask_weared_incorrect



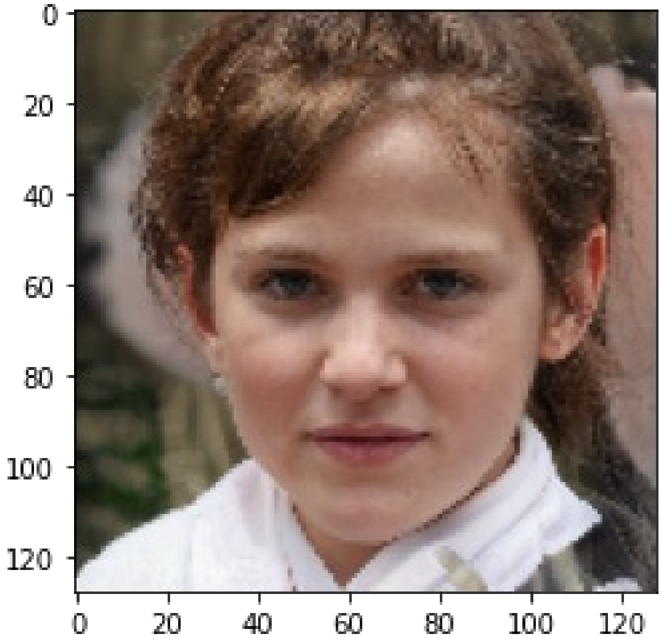
0004_MRNC_SRGM_0000 - predicted: with_mask, ground truth: mask_weared_incorrect



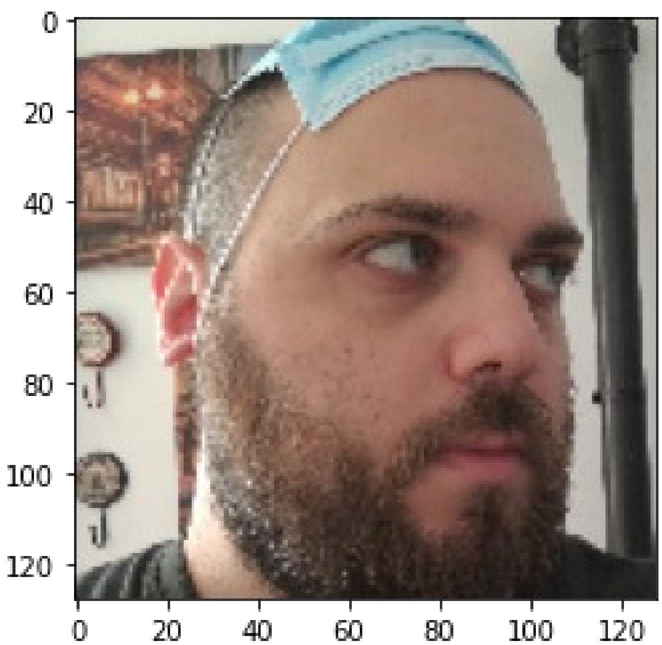
maksssksksss791.png - predicted: mask_weared_incorrect, ground truth: with_mask



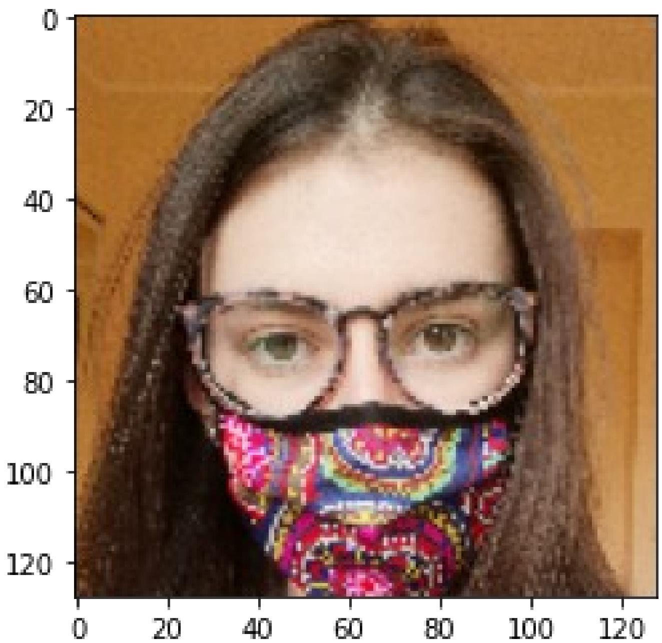
seed0782.png - predicted: mask_weared_incorrect, ground truth: without_mask



0006_MRNN_SRGM_0000 - predicted: without_mask, ground truth: mask_weared_incorrect

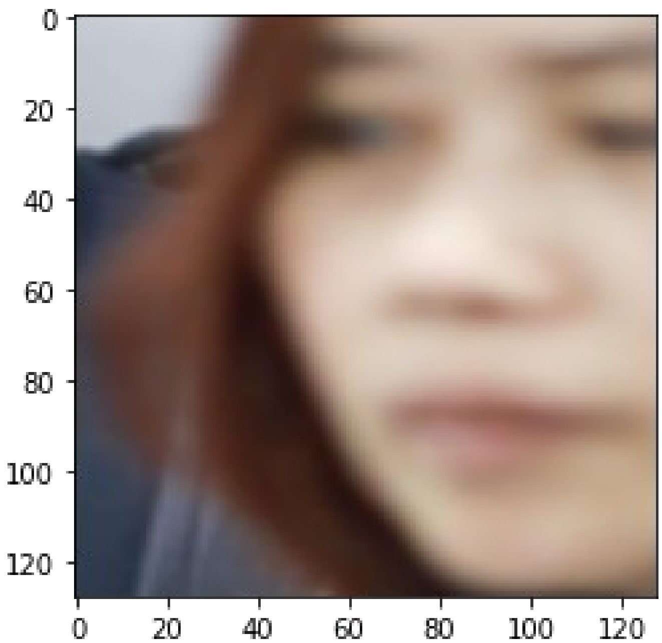


0015_MRNC_DRNV_0000 - predicted: mask_weared_incorrect, ground truth: with_mask

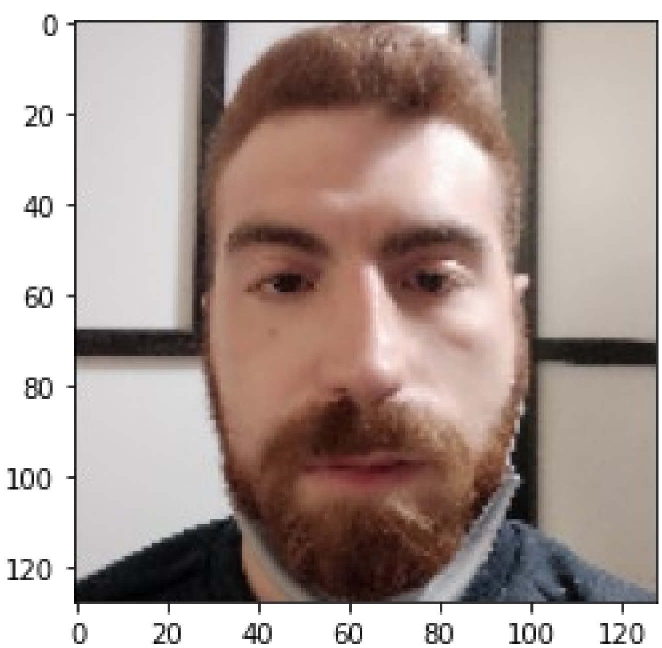


maksssksksss743.png - predicted: mask_weared_incorrect, ground truth: with_mask

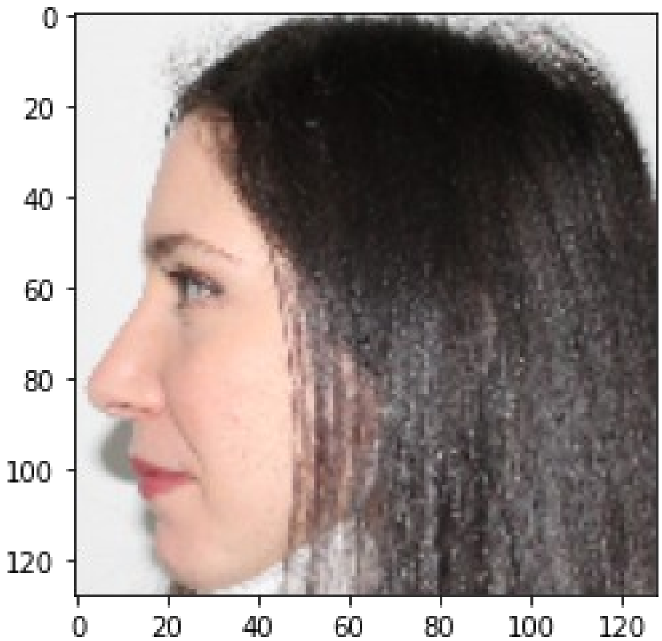
maksssksksss347.png - predicted: with_mask, ground truth: without_mask



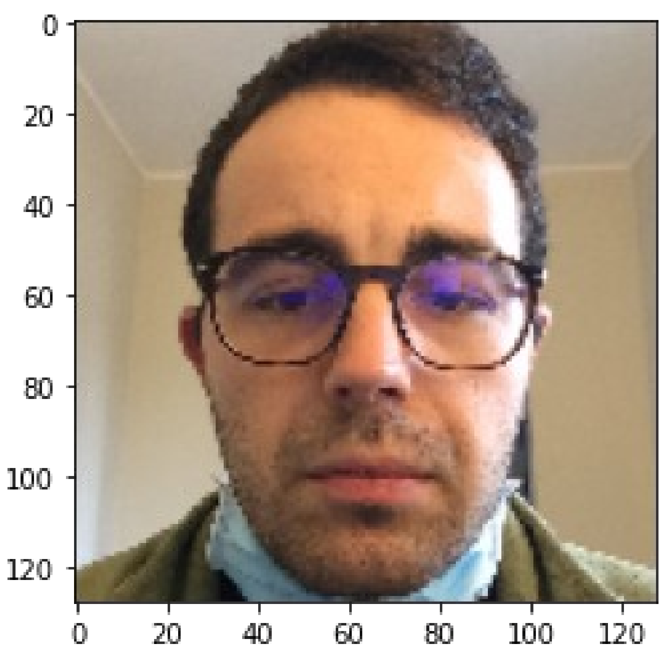
0015_MRNW_0000 - predicted: without_mask, ground truth: mask_weared_incorrect



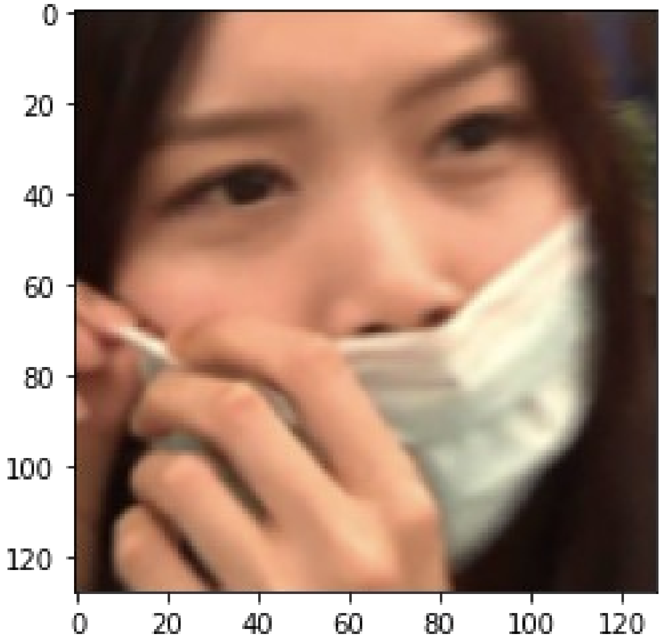
0002_MRFH_SRGM_0000 - predicted: without_mask, ground truth: mask_weared_incorrect



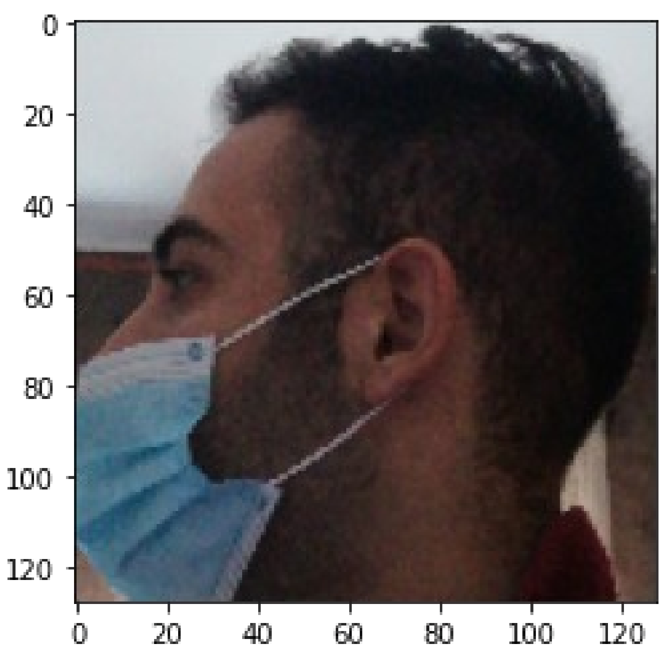
0016_MRNC_NMDM_0045 - predicted: without_mask, ground truth: mask_weared_incorrect



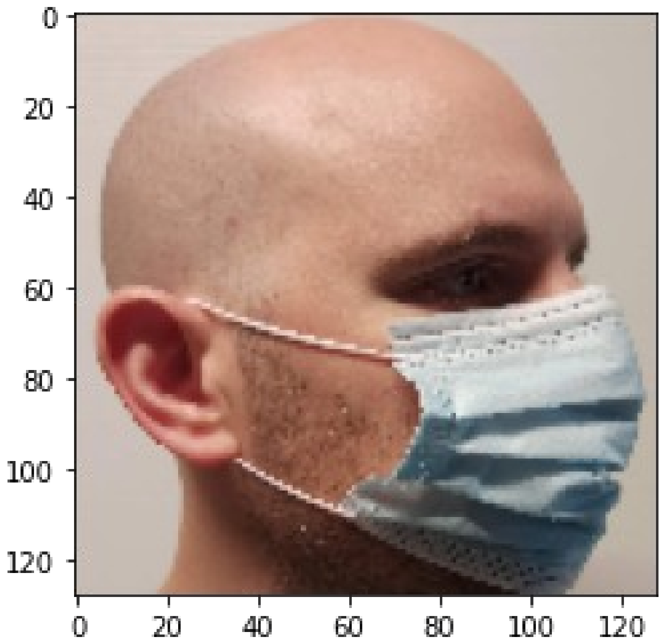
makssskskssss748.png - predicted: without_mask, ground truth: mask_weared_incorrect



0003_MRFH_SRGM_0045 - predicted: with_mask, ground truth: mask_weared_incorrect



0004_MSFC_NMDM_0000 - predicted: with_mask, ground truth: mask_weared_incorrect



Train Inceptionv3

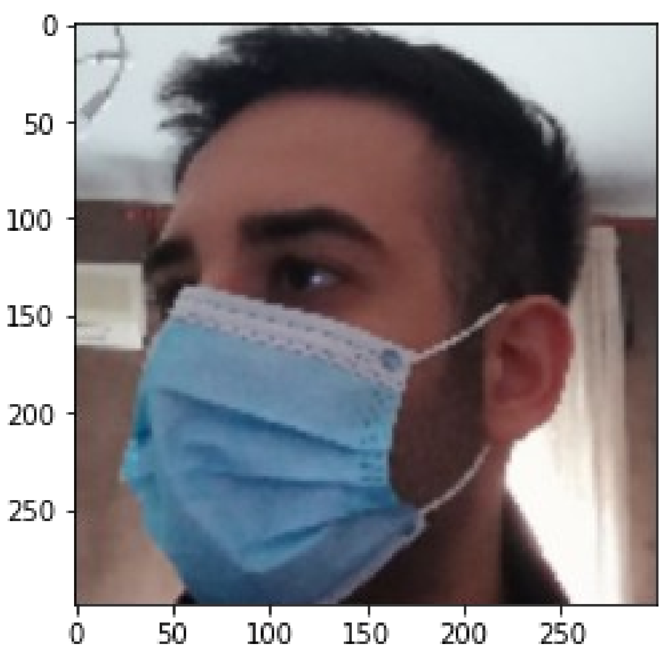
```
#the images must be 299 pixels square for InceptionV3. Resize the
images if not already done.

if os.path.exists('%s/x_train_299.pt' % (conf.combined_data_path)):
    print('loading...')
    x_train_resized = torch.load('%s/x_train_299.pt' %
(conf.combined_data_path))
    x_cv_resized = torch.load('%s/x_cv_299.pt' %
(conf.combined_data_path))
    x_test_resized = torch.load('%s/x_test_299.pt' %
(conf.combined_data_path))
else:
    #resize the images and store them
    x_train_resized = im_utils.resize_image_tensors(x_train,
new_size=299)
    x_cv_resized = im_utils.resize_image_tensors(x_cv, new_size=299)
    x_test_resized = im_utils.resize_image_tensors(x_test,
new_size=299)

    torch.save(x_train_resized, '%s/x_train_299.pt' %
(conf.combined_data_path))
    torch.save(x_cv_resized, '%s/x_cv_299.pt' %
(conf.combined_data_path))
    torch.save(x_test_resized, '%s/x_test_299.pt' %
(conf.combined_data_path))


#load y and image names
```

```
y = torch.load('%s/y.pt' %
(conf.combined_data_path)).type(torch.LongTensor)
with open('%s/im_names.txt' % (conf.combined_data_path), 'r') as f:
    im_names = f.read().split('\n')


#split targets and image names like the input images
y_test = y[:500]
test_im_names = im_names[:500]

y_cv = y[500:750]
cv_im_names = im_names[500:750]

y_train = y[750:]
train_im_names = im_names[750:]

#(torch.Size([2223, 3, 299, 299]),
# torch.Size([250, 3, 299, 299]),
# torch.Size([500, 3, 299, 299]))
x_train_resized.shape, x_cv_resized.shape, x_test_resized.shape
```

```
loading...
```

```
(torch.Size([2223, 3, 299, 299]),
 torch.Size([250, 3, 299, 299]),
 torch.Size([500, 3, 299, 299]))
```

```
#show a resized image
im_utils.show_image(x_train_resized[0])
```



```
# load 3 class label to index map
with open('%s/lab2idx.pkl' % conf.combined_data_path, 'rb') as f:
    lab2idx = pickle.load(f)
lab2idx
```

```
{'without_mask': 0, 'with_mask': 1, 'mask_weared_incorrect': 2}
```

```
# retrieve pretrained InceptionV3 model, turn off auxiliary outputs
model_incep = models.inception_v3(pretrained=True, aux_logits=False)

# replace the 1000 class output with 3 class output layer
model_incep.fc = nn.Linear(2048, 3)

#init the Adam optimizer
optimizer = optim.AdamW(model_incep.parameters(), lr=1e-4)

# train model and store when crossval score increases

device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')

model_path = 'inceptionv3_3class_model'
if not os.path.exists(model_path):
    os.mkdir(model_path)

model_incep, train_losses, cv_losses = finetune_model(model_incep,
optimizer, model_path, lab2idx,
                                                x_train_resized,
y_train, x_cv_resized, y_cv, x_test_resized, y_test,
                                                device, batch_size=8,
n_epochs=15)
```

```
Epoch: 0
Epoch: 0, Batch: 392, Loss: 0.175050
Epoch: 0, Batch: 792, Loss: 0.155708
Epoch: 0, Batch: 1192, Loss: 0.056521
Epoch: 0, Batch: 1592, Loss: 0.119818
Epoch: 0, Batch: 1992, Loss: 0.355796
Testing model...
Test accuracy 0.948000, 237 of 250
CV accuracy 0.948000, prev best acc: 0.000000 !! IMPROVED !!

Saving model...
Epoch: 1
Epoch: 1, Batch: 392, Loss: 0.071014
Epoch: 1, Batch: 792, Loss: 0.197997
Epoch: 1, Batch: 1192, Loss: 0.131895
Epoch: 1, Batch: 1592, Loss: 0.027407
Epoch: 1, Batch: 1992, Loss: 0.061337
Testing model...
Test accuracy 0.964000, 241 of 250
```

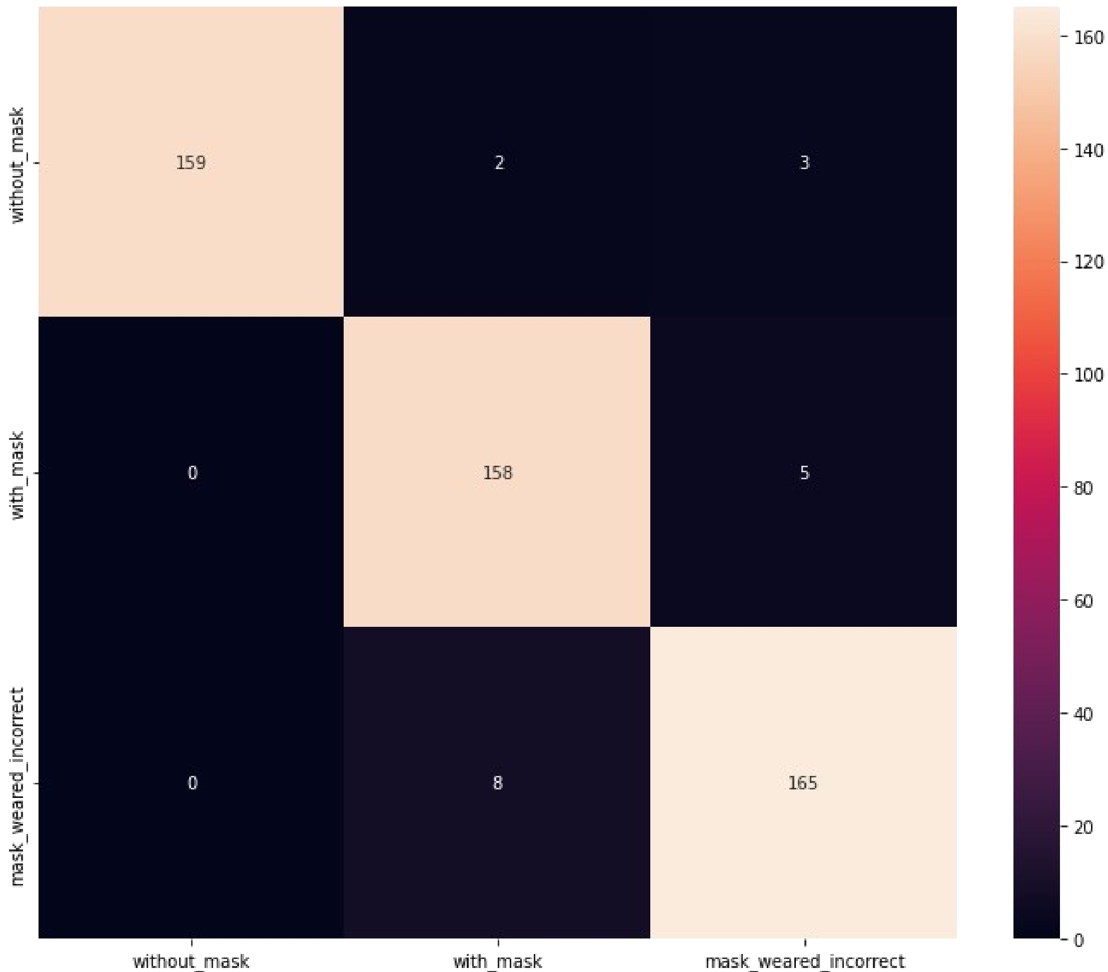
```
CV accuracy 0.964000, prev best acc: 0.948000 !! IMPROVED !!

Saving model...
Epoch: 2
Epoch: 2, Batch: 392, Loss: 0.010945
Epoch: 2, Batch: 792, Loss: 0.011674
Epoch: 2, Batch: 1192, Loss: 0.011239
Epoch: 2, Batch: 1592, Loss: 0.178480
Epoch: 2, Batch: 1992, Loss: 0.032663
Testing model...
Test accuracy 0.912000, 228 of 250
CV accuracy 0.912000, prev best acc: 0.964000

Epoch: 3
Epoch: 3, Batch: 392, Loss: 0.010882
Epoch: 3, Batch: 792, Loss: 0.062649
Epoch: 3, Batch: 1192, Loss: 0.021430
Epoch: 3, Batch: 1592, Loss: 0.060916
Epoch: 3, Batch: 1992, Loss: 0.027304
Testing model...
Test accuracy 0.976000, 244 of 250
CV accuracy 0.976000, prev best acc: 0.964000 !! IMPROVED !!

Saving model...
Epoch: 4
Epoch: 4, Batch: 392, Loss: 0.304726
Epoch: 4, Batch: 792, Loss: 0.003349
Epoch: 4, Batch: 1192, Loss: 0.125047
Epoch: 4, Batch: 1592, Loss: 0.023439
Epoch: 4, Batch: 1992, Loss: 0.147887
Testing model...
Test accuracy 0.992000, 248 of 250
CV accuracy 0.992000, prev best acc: 0.976000 !! IMPROVED !!

Saving model...
Epoch: 5
Epoch: 5, Batch: 392, Loss: 0.006274
Epoch: 5, Batch: 792, Loss: 0.031536
Epoch: 5, Batch: 1192, Loss: 0.047898
Epoch: 5, Batch: 1592, Loss: 0.010016
Epoch: 5, Batch: 1992, Loss: 0.019657
Testing model...
Test accuracy 0.984000, 246 of 250
CV accuracy 0.984000, prev best acc: 0.992000

Epoch: 6
Epoch: 6, Batch: 392, Loss: 0.002119
Epoch: 6, Batch: 792, Loss: 0.000903
Epoch: 6, Batch: 1192, Loss: 0.071264
Epoch: 6, Batch: 1592, Loss: 0.029547
```

```
Epoch: 6, Batch: 1992, Loss: 0.003784
Testing model...
Test accuracy 0.980000, 245 of 250
CV accuracy 0.980000, prev best acc: 0.992000

Epoch: 7
Epoch: 7, Batch: 392, Loss: 0.019927
Epoch: 7, Batch: 792, Loss: 0.009850
Epoch: 7, Batch: 1192, Loss: 0.019637
Epoch: 7, Batch: 1592, Loss: 0.023853
Epoch: 7, Batch: 1992, Loss: 0.029142
Testing model...
Test accuracy 0.984000, 246 of 250
CV accuracy 0.984000, prev best acc: 0.992000

Epoch: 8
Epoch: 8, Batch: 392, Loss: 0.001317
Epoch: 8, Batch: 792, Loss: 0.027588
Epoch: 8, Batch: 1192, Loss: 0.053312
Epoch: 8, Batch: 1592, Loss: 0.001636
Epoch: 8, Batch: 1992, Loss: 0.009119
Testing model...
Test accuracy 0.976000, 244 of 250
CV accuracy 0.976000, prev best acc: 0.992000

Epoch: 9
Epoch: 9, Batch: 392, Loss: 0.001397
Epoch: 9, Batch: 792, Loss: 0.009823
Epoch: 9, Batch: 1192, Loss: 0.007046
Epoch: 9, Batch: 1592, Loss: 0.003347
Epoch: 9, Batch: 1992, Loss: 0.010028
Testing model...
Test accuracy 0.976000, 244 of 250
CV accuracy 0.976000, prev best acc: 0.992000

Epoch: 10
Epoch: 10, Batch: 392, Loss: 0.008081
Epoch: 10, Batch: 792, Loss: 0.012927
Epoch: 10, Batch: 1192, Loss: 0.001235
Epoch: 10, Batch: 1592, Loss: 0.002286
Epoch: 10, Batch: 1992, Loss: 0.006534
Testing model...
Test accuracy 0.976000, 244 of 250
CV accuracy 0.976000, prev best acc: 0.992000

Epoch: 11
Epoch: 11, Batch: 392, Loss: 0.002155
Epoch: 11, Batch: 792, Loss: 0.001788
Epoch: 11, Batch: 1192, Loss: 0.015114
Epoch: 11, Batch: 1592, Loss: 0.000420
```

```
Epoch: 11, Batch: 1992, Loss: 0.000700
Testing model...
Test accuracy 0.980000, 245 of 250
CV accuracy 0.980000, prev best acc: 0.992000

Epoch: 12
Epoch: 12, Batch: 392, Loss: 0.000611
Epoch: 12, Batch: 792, Loss: 0.007657
Epoch: 12, Batch: 1192, Loss: 0.003410
Epoch: 12, Batch: 1592, Loss: 0.005497
Epoch: 12, Batch: 1992, Loss: 0.000934
Testing model...
Test accuracy 0.972000, 243 of 250
CV accuracy 0.972000, prev best acc: 0.992000

Epoch: 13
Epoch: 13, Batch: 392, Loss: 0.009519
Epoch: 13, Batch: 792, Loss: 0.037833
Epoch: 13, Batch: 1192, Loss: 0.031553
Epoch: 13, Batch: 1592, Loss: 0.012478
Epoch: 13, Batch: 1992, Loss: 0.000741
Testing model...
Test accuracy 0.972000, 243 of 250
CV accuracy 0.972000, prev best acc: 0.992000

Epoch: 14
Epoch: 14, Batch: 392, Loss: 0.003957
Epoch: 14, Batch: 792, Loss: 0.002748
Epoch: 14, Batch: 1192, Loss: 0.001999
Epoch: 14, Batch: 1592, Loss: 0.000221
Epoch: 14, Batch: 1992, Loss: 0.006582
Testing model...
Test accuracy 0.988000, 247 of 250
CV accuracy 0.988000, prev best acc: 0.992000

Test accuracy 0.964000, 482 of 500
```

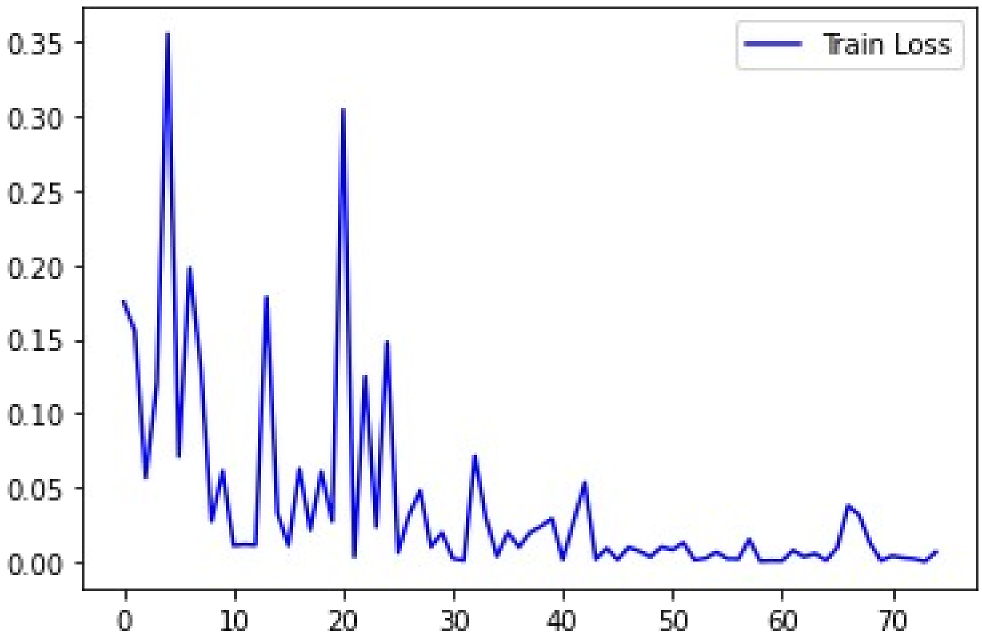
	precision	recall	f1-score	support
without_mask	1.0000	0.9695	0.9845	164
with_mask	0.9405	0.9693	0.9547	163
mask_weared_incorrect	0.9538	0.9538	0.9538	173
accuracy			0.9640	500
macro avg	0.9647	0.9642	0.9643	500
weighted avg	0.9646	0.9640	0.9641	500

```
final test accuracy: 0.964000
```

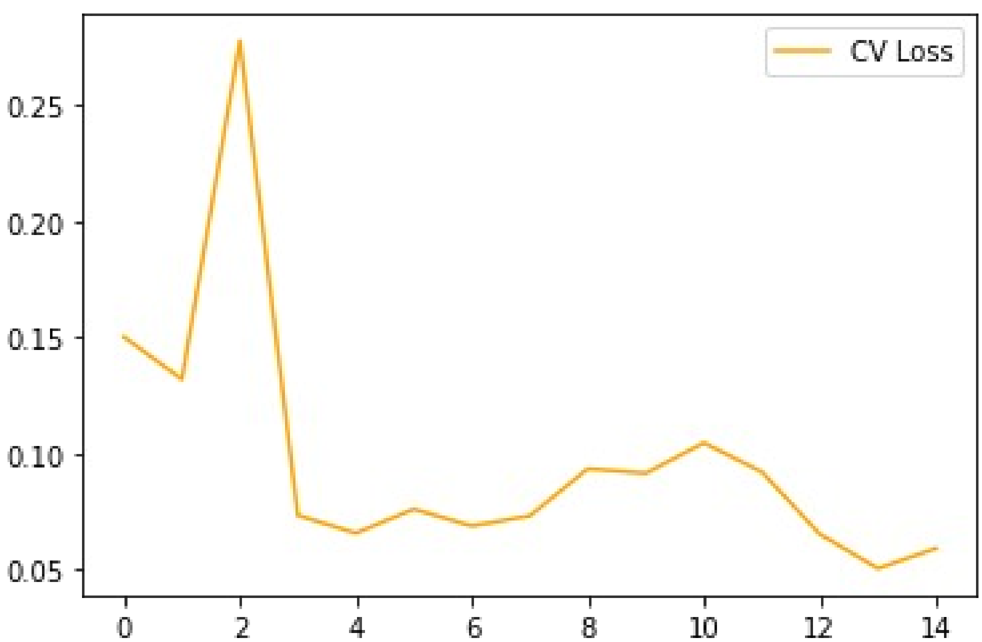



```
#plot train loss

fig = plt.figure()
plt.plot(train_losses, color='blue')
plt.legend(['Train Loss'], loc='upper right')
```

```
<matplotlib.legend.Legend at 0x1dc76aea880>
```




```
#plot cv loss

fig = plt.figure()
plt.plot(cv_losses, color='orange')
plt.legend(['CV Loss'], loc='upper right')
plt.show()
```



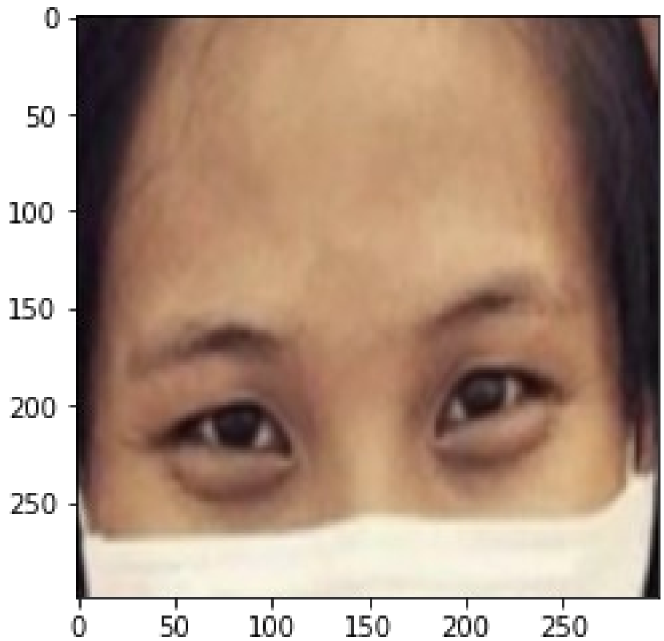

```
# get the indices of incorrectly predicted images
model_incep.eval()
with torch.no_grad():
```

```
    output = model_incep(x_test_resized)
    _, y_pred = torch.max(output, 1)
    correct = y_pred.eq(y_test)

#[  0, 110, 141, 163, 273, 290, 297, 320, 331, 340, 362, 367, 377,
420, 441, 474, 494, 498]
wrong_idx = (correct==False).nonzero(as_tuple=True)[0]
wrong_idx
```

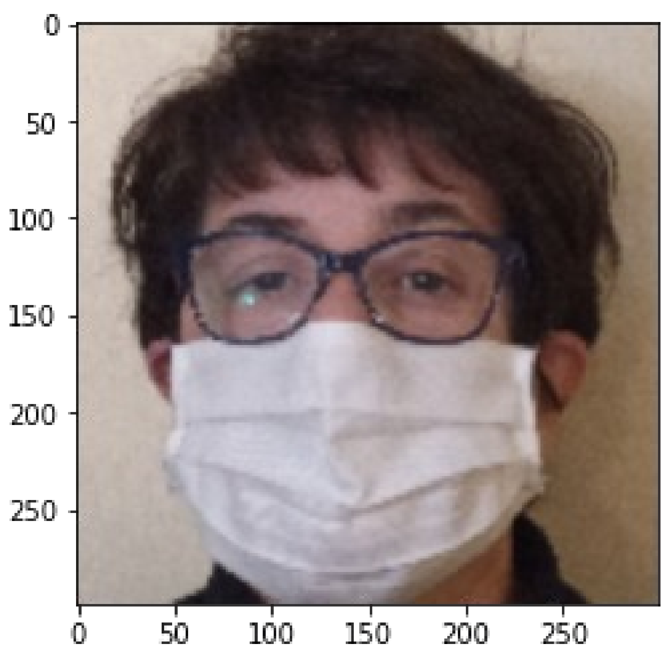
```
tensor([  0, 110, 141, 163, 273, 290, 297, 320, 331, 340, 362, 367,
377, 420,
        441, 474, 494, 498])
```

```
#display the incorrectly predicted images

idx2lab = {v:k for k,v in lab2idx.items()}

for idx in wrong_idx.tolist():
    im = x_test_resized[idx]

    corr = y_test[idx].item()
    pred = y_pred[idx].item()

    nm = test_im_names[idx]
    print('\n\n%s - predicted: %s, ground truth: %s' % (nm,
idx2lab[pred], idx2lab[corr]))

    im_utils.show_image(im)
```

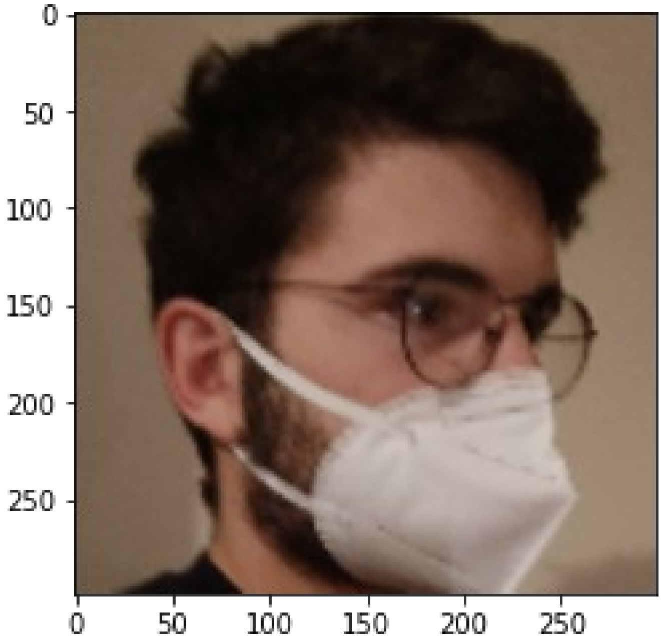
```
maksssksksss264.png - predicted: mask_weared_incorrect, ground truth:
with_mask
```



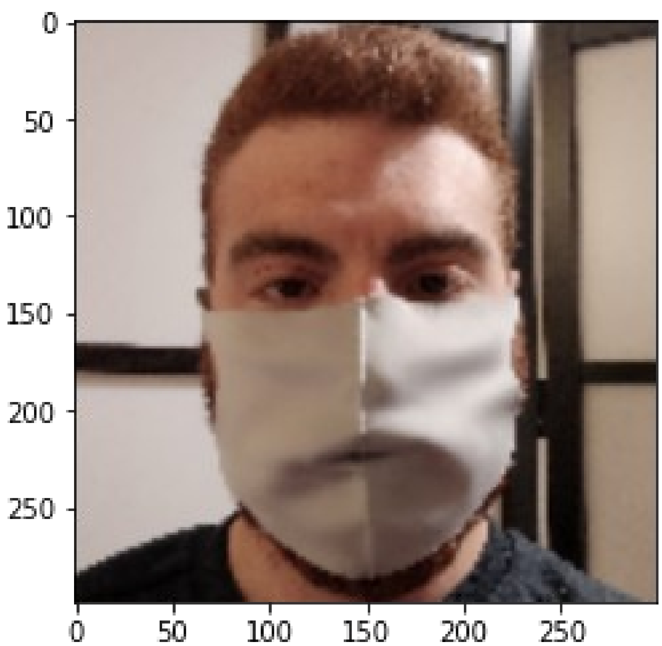
0004_MRNC_SRGM_0000 - predicted: with_mask, ground truth: mask_weared_incorrect



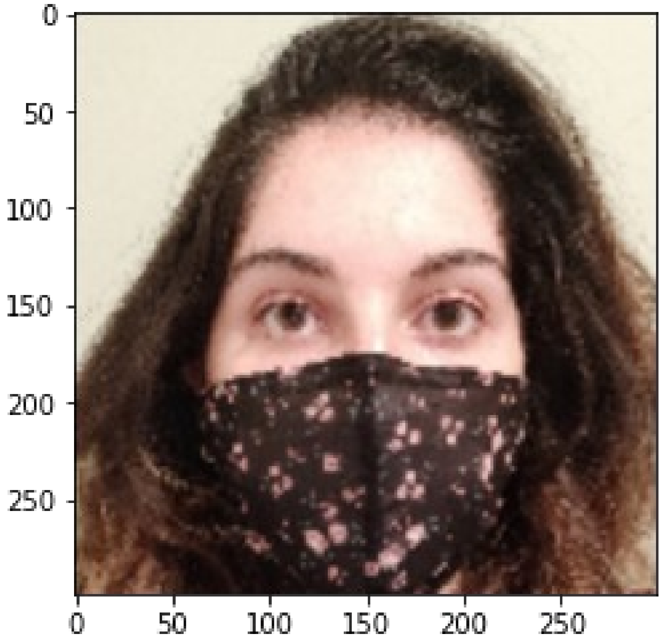
0009_MRNC_SRGM_0045 - predicted: with_mask, ground truth: mask_weared_incorrect



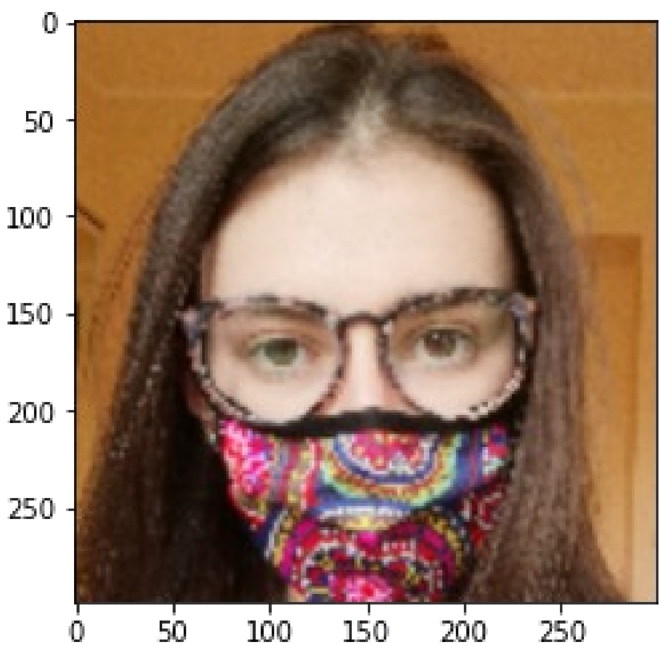
0030_MRNC_DRWV_0090 - predicted: with_mask, ground truth: mask_weared_incorrect



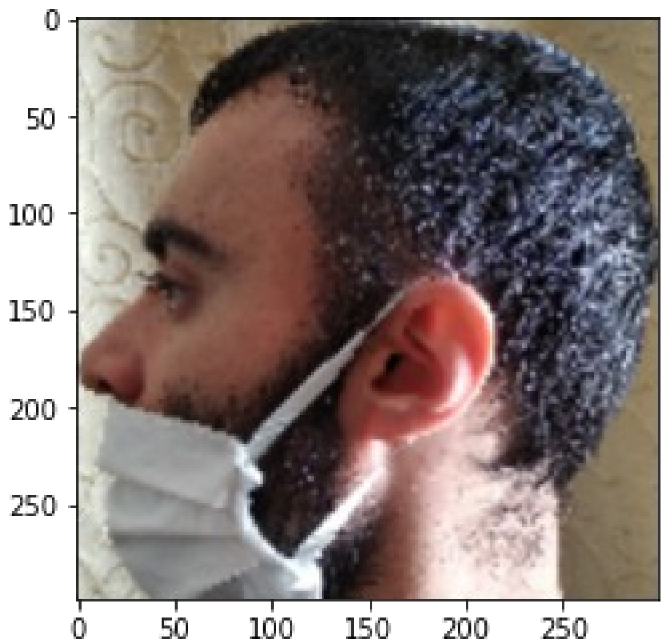
0032_MRCW_NMDM_0000 - predicted: mask_weared_incorrect, ground truth: with_mask



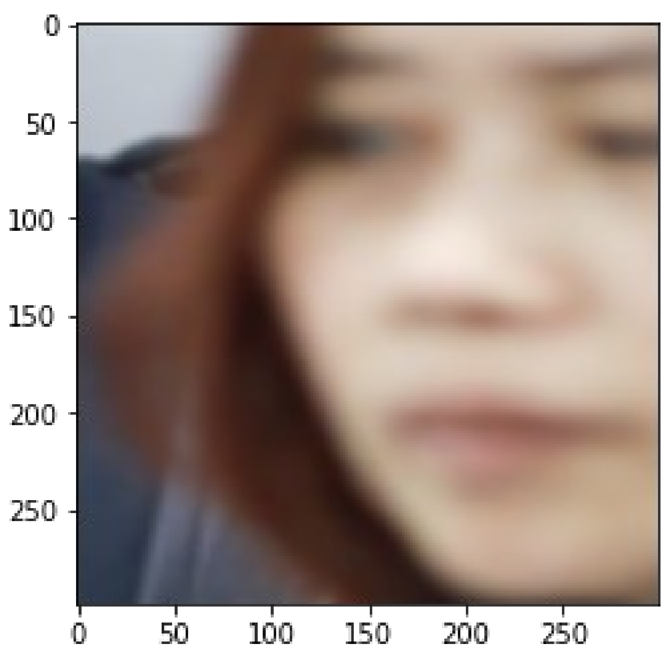
0015_MRNC_DRNV_0000 - predicted: mask_weared_incorrect, ground truth: with_mask



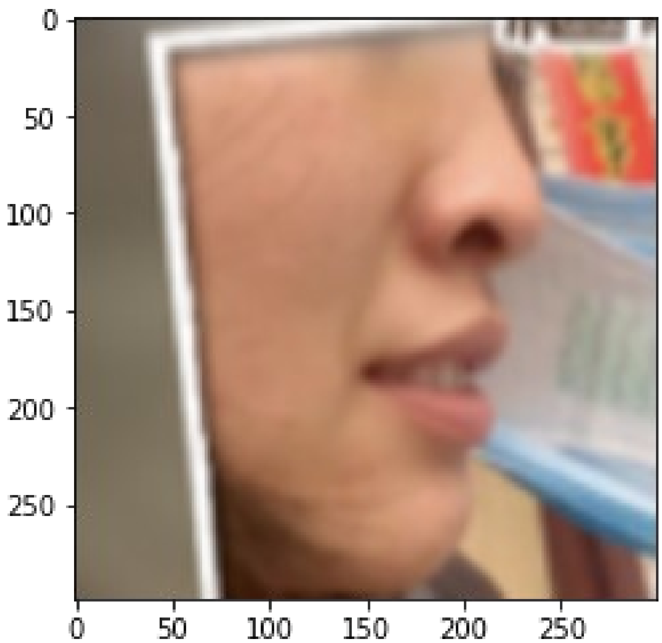
0009_MRFH_SRGM_0000 - predicted: with_mask, ground truth: mask_weared_incorrect



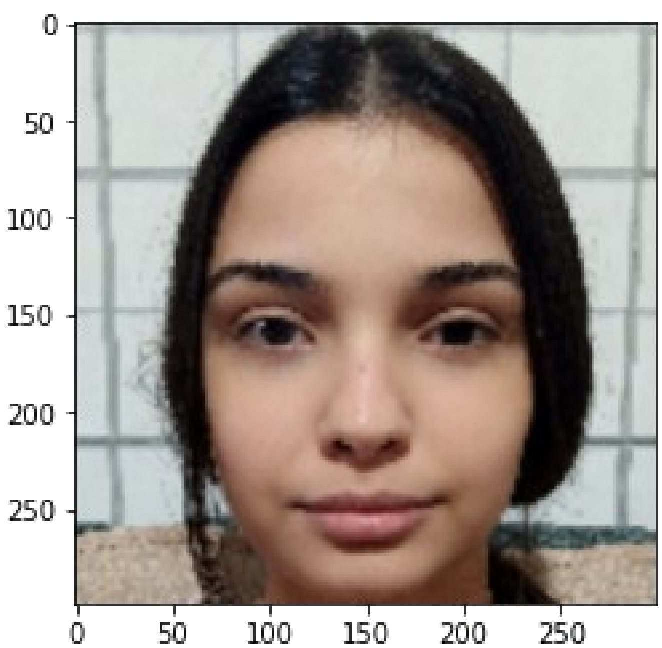
maksssksksss347.png - predicted: with_mask, ground truth: without_mask



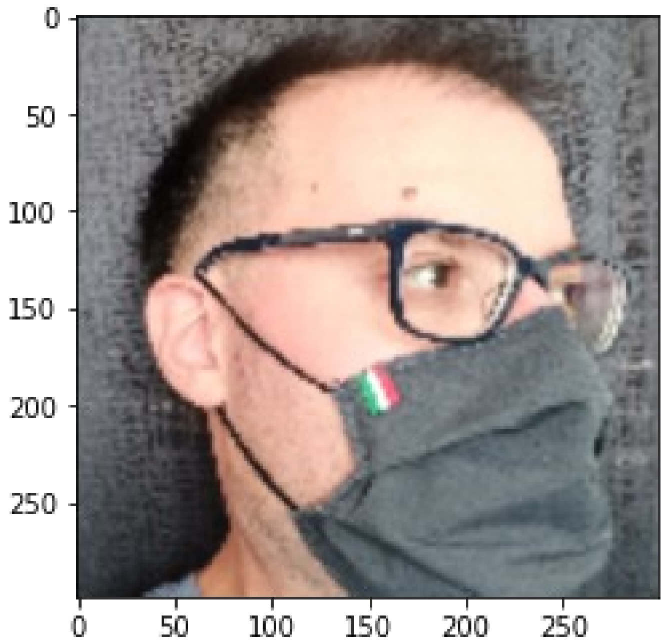
maksssksksss621.png - predicted: with_mask, ground truth: without_mask



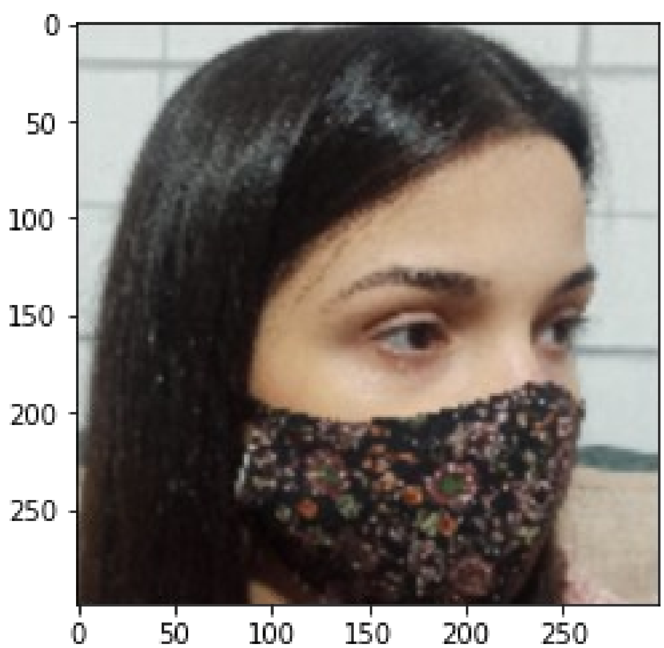
0014_MRNN_SRGM_0045 - predicted: mask_weared_incorrect, ground truth: without_mask



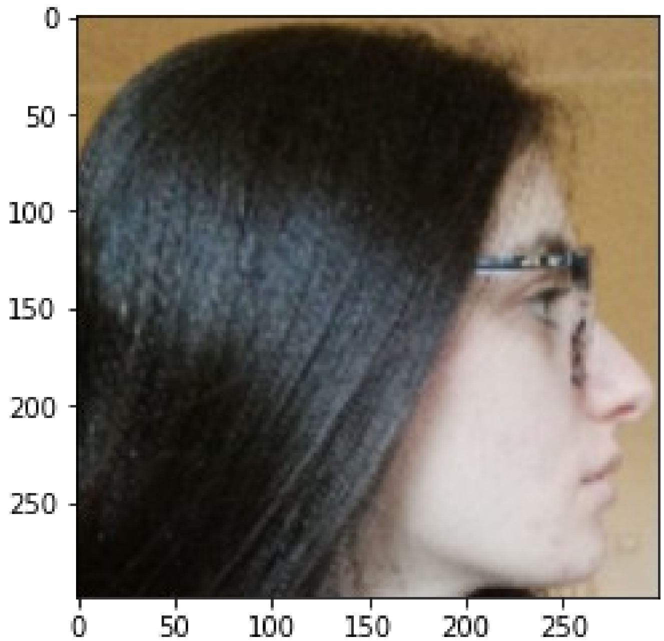
0041_MRNN_SRGM_0000 - predicted: mask_weared_incorrect, ground truth: with_mask



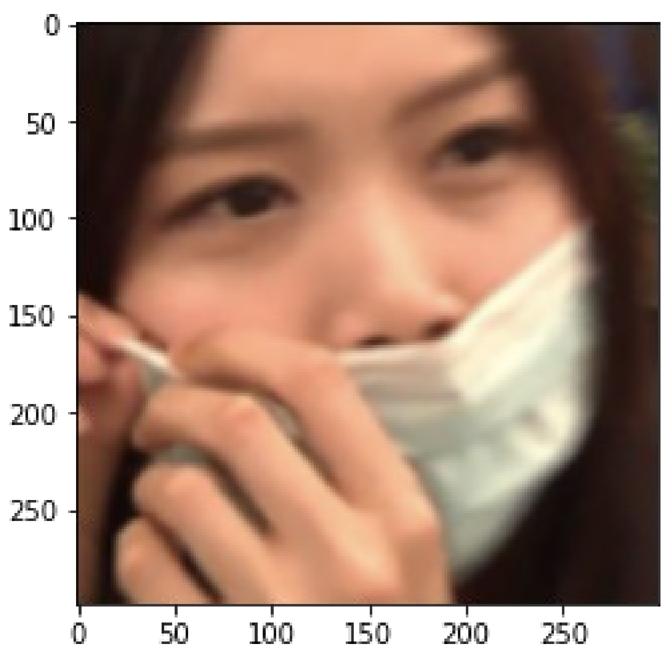
0030_MRFH_DRWV_0000 - predicted: mask_weared_incorrect, ground truth: with_mask



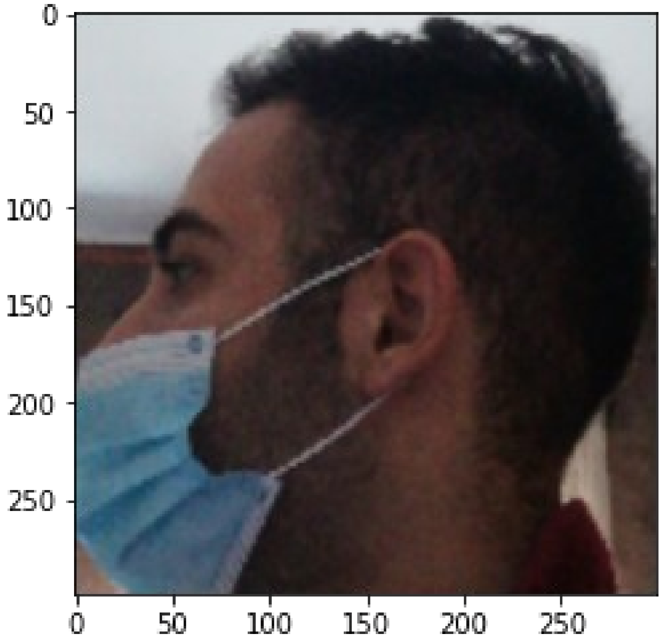
0041_MRFH_SRGM_0090 - predicted: mask_weared_incorrect, ground truth: without_mask



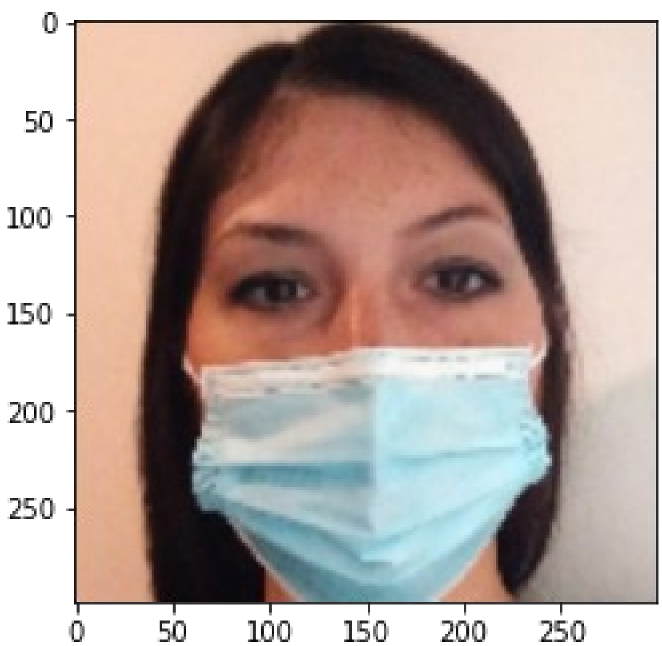
maksssksksss748.png - predicted: with_mask, ground truth: mask_weared_incorrect



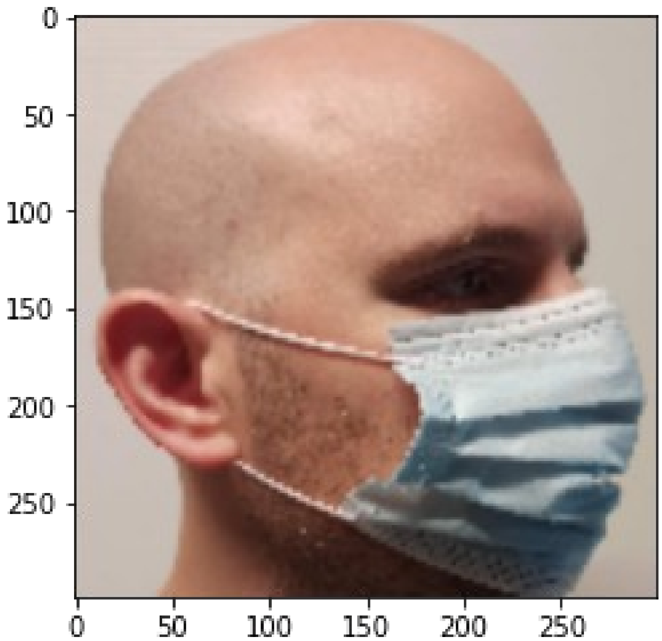
0003_MRFH_SRGM_0045 - predicted: with_mask, ground truth: mask_weared_incorrect



0035_MRNN_NMDM_0045 - predicted: with_mask, ground truth: mask_weared_incorrect



0004_MSFC_NMDM_0000 - predicted: with_mask, ground truth: mask_weared_incorrect



0033_MRNN_SRGM_0045 - predicted: mask_weared_incorrect, ground truth: without_mask