
# COMP 6721 Applied Artificial Intelligence
# Project, Phase I

## Face Mask Detection using CNN in PyTorch

**Department of Computer Science and Engineering**

**Instructor: Dr Rene Witte**

**Team Number: RN_01**

**Team Members:**
**Vashisht Marhwal (40158888)**
**Enrique Martin del Campo Hernandez (40173409)**
**Pouria Roostaei Ali Mehr (40172684)**

# Dataset

The dataset consists of 1372 images in total and split into 4 classes: cloth(373), FFP2(316), surgical(341) and no_mask(342) as shown in Figure 1.
The images were recollected from existing datasets in Kaggle [1][2][3][4][5] and github [6].

For the training we use a K-fold = 4 cross validation from the whole dataset (explained in more detail at Evaluation section).Then the images are resized into 128x128 as the initial input for the CNN.

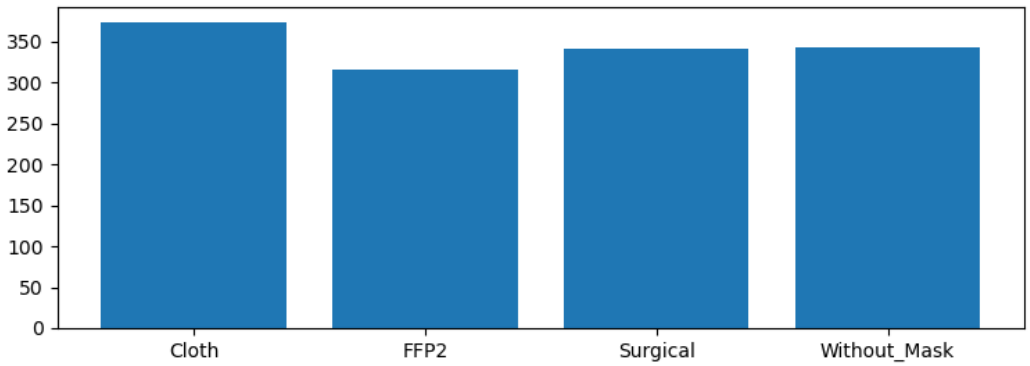


Figure 1: Dataset Information

The dataset needs to be pre-processed so that our Neural Network module can understand the input data. To that end, the data is loaded using a DataLoader module which then converts the input data into tensors. A tensor essentially stores data in a NxN matrix. This is then normalized so that each pixel value is in a similar range. This ensures that while training our gradients do not go out of control thus needing only one learning rate.

## CNN Architecture

Our CNN model consists of 3 blocks[8]. Each block has 2 convolutional layers followed by a batch normalization and a single max-pooling layer. LeakyReLu activation function is used to make sure that none of the neurons reach the dead state. The activation function ensures that all negative values are removed from the feature map since pixel values cannot be negative. Stride of 2 is used while padding is 1. After convolution and extracting the features, a flatten layer is defined to convert tensors in 3 dimensions to one dimension. The 4 linear layers added help reduce the size of the tensor and learn the features. General CNN Architecture is in Figure 2.

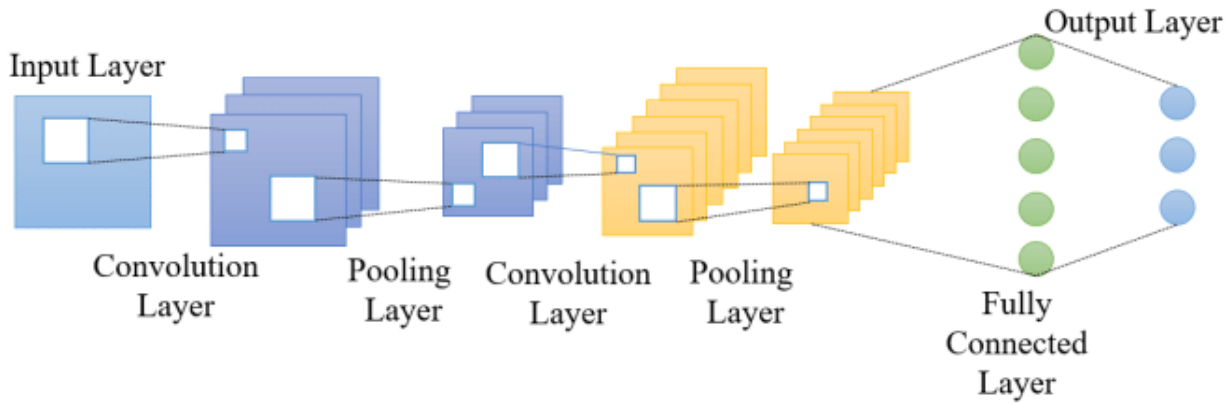


Figure 2: CNN Architecture (src)

Detailed Architecture for our CNN is given below[7]:

1. Input image of size 128 x128 with 3 input channels i.e RGB channels is converted to a 3D matrix.
2. A filter of size 3x3 with padding = 1 is used which results in learning of 32 filters from our 32 x 32 input matrix.
3. Dot product or Convolution is performed with the above filter to make the output matrix.
4. Batch Normalization is done to improve the learning rate.
5. Leaky ReLu is used as an activation function ensuring no negative values are present in the output matrix.
6. Max Pooling layer ensures that the result obtained from the convolution layer (feature map) is present in a lower dimension.
7. Point 2 to 6 is repeated two more times to obtain the final single numpy array which is then fed to a fully-connected layer.
8. Fully connected layer is a feed forward neural network which takes a flattened input from the last pooling layer.

9. Each Artificial Neuron performs the required calculations and then the final layer uses ReLU to get probabilities of the input belonging to a particular class.
10. A total of 30 such epochs are done for training these images.
11. The obtained probability values are used to evaluate accuracy on the train data after every epoch.

**Model Summary Output**

```
convolutional_neural_network(
  (clayer): Sequential(
        (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): LeakyReLU(negative_slope=0.01, inplace=True)
        (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (5): LeakyReLU(negative_slope=0.01, inplace=True)
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (8): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (9): LeakyReLU(negative_slope=0.01, inplace=True)
        (10): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (11): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (12): LeakyReLU(negative_slope=0.01, inplace=True)
        (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        (14): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (15): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (16): LeakyReLU(negative_slope=0.01, inplace=True)
        (17): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (18): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (19): LeakyReLU(negative_slope=0.01, inplace=True)
        (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (complete_connected_layer): Sequential(
        (0): Dropout(p=0.1, inplace=False)
        (1): Linear(in_features=32768, out_features=1000, bias=True)
        (2): ReLU(inplace=True)
        (3): Linear(in_features=1000, out_features=512, bias=True)
        (4): ReLU(inplace=True)
        (5): Dropout(p=0.1, inplace=False)
        (6): Linear(in_features=512, out_features=128, bias=True)
        (7): ReLU(inplace=True)
        (8): Dropout(p=0.1, inplace=False)
```

```
        (9): Linear(in_features=128, out_features=5, bias=True)
    )
)
```

# Evaluation

In this project we evaluated the model by using 4-folds cross validation for images[9] as test data,which was selected randomly and consisted of four classes I(without mask, surgical, ffp2 and cloth). For the evaluation we used a confusion matrix to describe the performance of the classification model on the test data. And classification_report in order to show the accuracy, precision, recall and F1-score.

As you can observe in Figure 3 we provided the confusion matrix for fold-4, which indicates that this model predicted 6 times the class “Without Mask” as “Cloth” and 3 times “Cloth” as “FFP2” and 7 times predicted class “Surgical” as “Without Mask”.

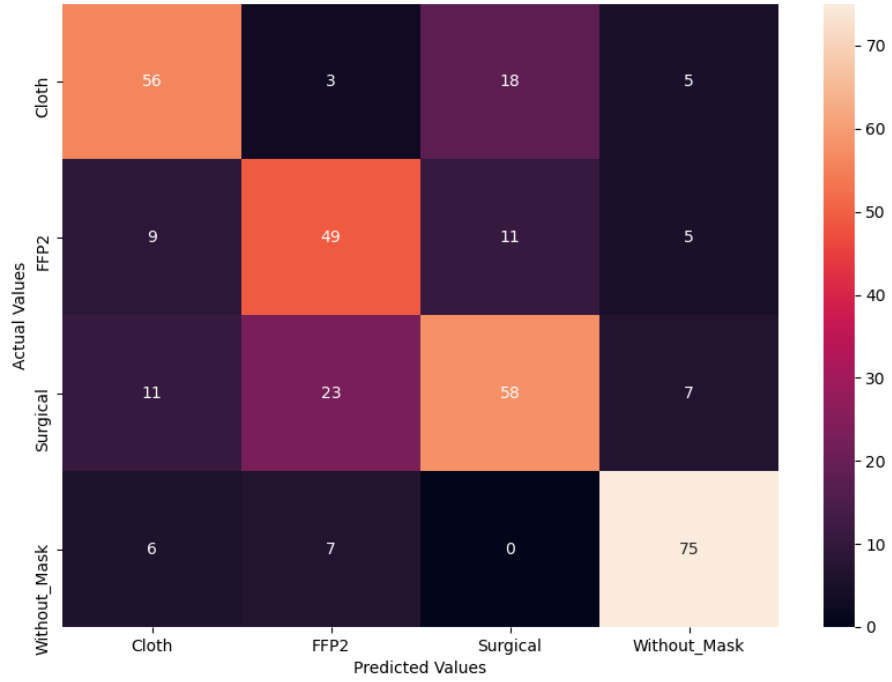


Figure 3: Confusion Matrix in Fold-4

In Figure 4 we illustrated the classification report, which contains the accuracy of the model for the test data on fold-4, and also includes precision, recall and F1-score for all classes separately. In the table below we got the best precision of 82% for “Without Mask”

and 68% for “Cloth” then 67% for “Surgical”. However if we take a look at the recall and f1-score, we can see that the better result was for the “Without Mask” with 85% for recall and 83% for f1-score.

In terms of accuracy of the model, we received an accuracy of 69% for our testing data in fold-4, which needs to be improved[10] and we will discuss it in this section.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Cloth | 0.68 | 0.68 | 0.68 | 82 |
| FFP2 | 0.60 | 0.66 | 0.63 | 74 |
| Surgical | 0.67 | 0.59 | 0.62 | 99 |
| Without_Mask | 0.82 | 0.85 | 0.83 | 88 |
| accuracy | | | 0.69 | 343 |
| macro avg | 0.69 | 0.70 | 0.69 | 343 |
| weighted avg | 0.69 | 0.69 | 0.69 | 343 |

Figure 4: Classification report for Fold-4

As we mentioned above we trained the images with 30 epochs in each fold. Looking at the logs of our training accuracy and training loss we observed that after each epoch the train accuracy and train loss has been improved.

In addition we used 4-folds cross validation in our project. The model accuracy for test dataset on Fold 1 to 4 were: 72%, 69%, 72% and 70%. The average model accuracy was: 70.69%.

In order to improve the performance of the model for the next phase we have different approaches. The first approach, which could lead us to improve the accuracy is to improve some parameters such as, increase the number of epochs. Running on the GPU is one of the targets. The next approach is to use augmented data, which is not that significant but it could be helpful.

# References

[1] https://www.kaggle.com/aditya276/face-mask-dataset-yolo-format
[2] https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset
[3] https://www.kaggle.com/omkargurav/face-mask-dataset?select=data
[4] https://www.kaggle.com/vijaykumar1799/face-mask-detection
[5] https://www.kaggle.com/andrewmvd/face-mask-detection
[6] https://github.com/X-zhangyang/Real-World-Masked-Face-Dataset (only RFMD_part_1 & RFMD_part_3)
[7] https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05
[8] https://medium.com/thecyphy/train-cnn-model-with-pytorch-21dafb918f48
[9] https://medium.com/dataseries/k-fold-cross-validation-with-pytorch-and-sklearn-d094aa00105f
[10] https://towardsdatascience.com/the-quest-of-higher-accuracy-for-cnn-models-42df5d731faf