COMP 6721 – Applied Artificial Intelligence (Summer 2022)

Project Assignment, Part 1

Team:

1. Almas Saba (40156359) – Data specialist

2. Pradnya Kandarkar (40194666) – Training specialist

3. Vikyath Srinivasulu (40218245) – Evaluation specialist

Project repository: https://github.com/pradnya-git-dev/S22_COMP6721_Project.git

DATASET :

The dataset for this project was built using subsets of images from various publicly available datasets. A compressed version of the primary dataset ("project_dataset.zip") that was built can be found here.

The following main sources were used to make up our dataset:

Dataset 1: Face Mask Detection [1]

Dataset 2 :COVID-19 Medical Face Mask Detection Dataset [2]

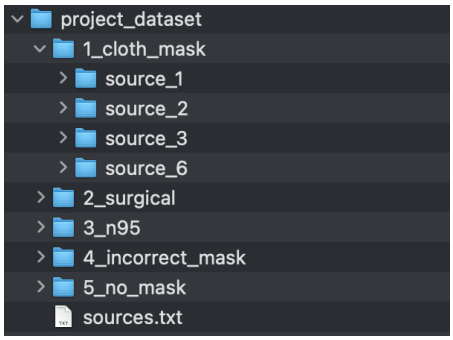
Dataset 3: Face Mask Detection Dataset [3]

Dataset 4: COVID Face Mask Detection Dataset [4]

Dataset 5: Face mask detector(mask ,not mask, incorrect mask [5]

Dataset 6: Face Mask Detection [6]

It has the following directory structure:



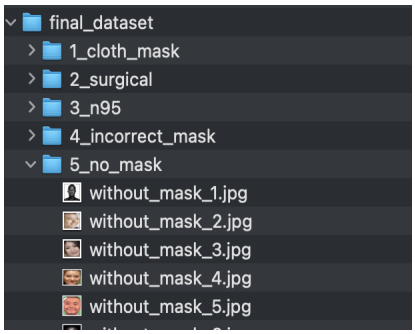


Figure 1: Directory Snapshot 1 Figure 2: Directory Snapshot 2

The "sources.txt" file that provides information about how to reach the sources. However, for ease of use in the project, the images from different sources are combined together and a final dataset is formed. The final dataset is compressed and provided in the "final_dataset.zip" file. When uncompressed, the final project dataset has the following directory structure:

Here, the "final_dataset" directory has 5 sub-directories containing images belonging to one of the output classes of this project.

The total number of images across all classes is 2600. The different classes in the dataset along with their image counts are as follows: 1. Cloth mask - 445 images, 2. Surgical mask - 599 images, 3. N95 mask - 209 images, 4. Incorrect mask - 706 images, 5. No mask - 641 images

The N95 mask category considers any of the “FFP2/N95/KN95"-type masks. Currently, the number of images for the N95 class is less compared to other classes. This shall be tackled in phase 2.

Following tasks are performed for data preprocessing:

- Load the data
- Transformation of the image using the “torchvision.transforms” module. This does the resizing to get the tensors of the same dimension
- Define mappings between classes and labels as follows:
- Automatically split the dataset into training, validation, and testing datasets. The dataset partitions used is shown below

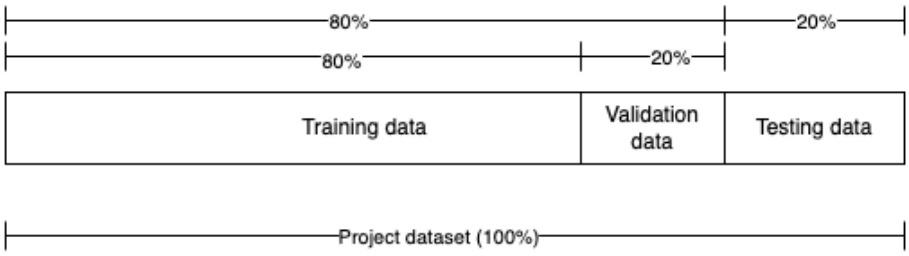


Figure 3: Split of the dataset

From the pytorch library, “torch.utils.data.random_split” has been used that ensures that the data split randomly, without giving any preference to a particular order of data.

CNN ARCHITECTURE :

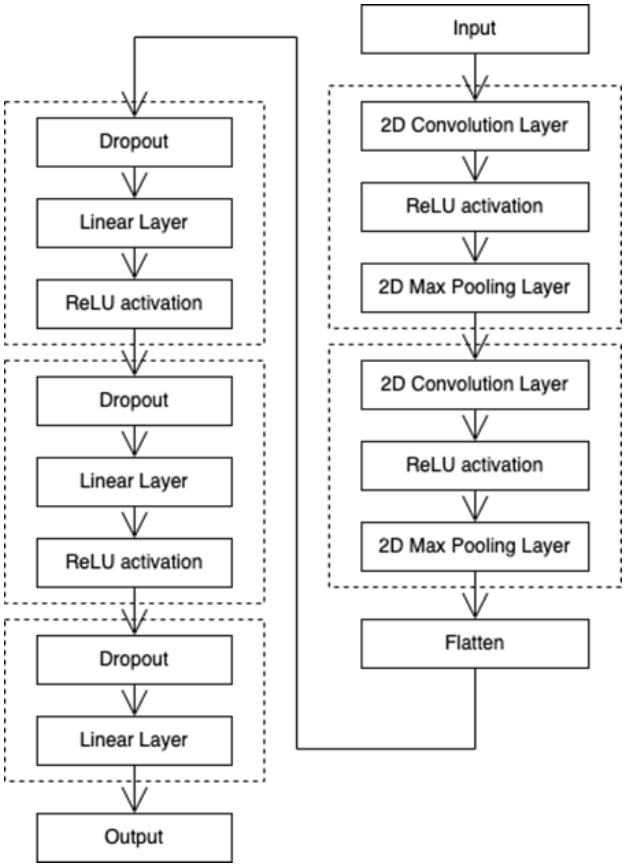
The CNN architecture used for the project is inspired from a PyTorch tutorial [7]. It has two CNN blocks followed by 3 linear layers. Each CNN block has 1 2D convolution layer and 1 2D max pooling layer. The output from CNN blocks is flattened before passing to the linear layers of the architecture. Dropout layers are added before linear layers to add regularization. The activation function used for convolution and linear layers is ReLU. The following figure shows the CNN architecture used for the project:

For variant 1, the pooling layers are removed from the CNN architecture of the project.

For variant 2, a third CNN block is added to the CNN architecture of the project.

The training process involves adjusting the different learnable parameters of the network so that the model learns the best possible mapping between the input and the output. During the training process, the training data is passed through the model for a specified epochs (one epoch = one complete pass through the training data). After every epoch, training and validation loss values

are calculated. Training loss values are used to adjust the values of learnable parameters of the model. Validation loss values can be used to determine whether the model is overfitting and for early stopping. Training process is stopped early if increasing values are observed for the validation loss.



Firgure 4: CNN Architecture

The training process for the project uses the following components:

1. Training and validation data: This is provided by using the data loaders created for training and validation data
2. Model: The model to be trained on the provided data (one of the 3 models described above)
3. Loss function: The loss/objective function gives an idea about how well the model is being trained and whether the model is overfitting to the provided data.

   During training the goal is to minimize the value of the loss function for both training and validation data. The ideal situation for loss function values is lower values of loss function on both the training and validation data. Low training loss values and high validation loss values is an indicator that the model does not generalize well (the model is overfitting).

The loss function used for this project is weighted categorical cross entropy. The weights of each class in the dataset are proportional to the number of samples in the dataset that belong to the class. Weighted categorical cross entropy helps in tackling imbalanced datasets by assigning greater weights to classes with less number of samples. The penalties calculated for incorrect predictions are adjusted according to the weights of classes.

4. Optimizer: Optimizer uses the values of loss function calculated on using training data to adjust the values of different learnable parameters of the network. The optimizer used for this project is Adam.

For training a model, a number of hyperparameters are required which affect the learning of the model. The following hyperparameter values are used - batch size: 64, learning rate: 0.001, dropout: 0.2, epoch: 150 is the default value used for epochs. However, the training stops early if the values for validation loss are observed to be increasing for 3 consecutive epochs. The following figures show an example of training process for the final CNN architecture of the project:

```
Epoch 001: Train_loss: 1.4605, Validation_loss: 1.3854
Epoch 002: Train_loss: 1.2474, Validation_loss: 1.1087
Epoch 003: Train_loss: 0.9429, Validation_loss: 0.9744
Epoch 004: Train_loss: 0.7478, Validation_loss: 0.8324
Epoch 005: Train_loss: 0.7064, Validation_loss: 0.7541
Epoch 006: Train_loss: 0.6446, Validation_loss: 0.7120
Epoch 007: Train_loss: 0.6092, Validation_loss: 0.6904
Epoch 008: Train_loss: 0.6305, Validation_loss: 0.6727
Epoch 009: Train_loss: 0.5526, Validation_loss: 0.6703
Epoch 010: Train_loss: 0.4790, Validation_loss: 0.6461
Epoch 011: Train_loss: 0.4742, Validation_loss: 0.6284
Epoch 012: Train_loss: 0.5557, Validation_loss: 0.6533
Epoch 013: Train_loss: 0.3731, Validation_loss: 0.6430
Epoch 014: Train_loss: 0.3723, Validation_loss: 0.5815
Epoch 015: Train_loss: 0.4117, Validation_loss: 0.6822
Epoch 016: Train_loss: 0.4140, Validation_loss: 0.6130
Epoch 017: Train_loss: 0.3801, Validation_loss: 0.6168
Epoch 018: Train_loss: 0.3144, Validation_loss: 0.6025
Epoch 019: Train_loss: 0.2577, Validation_loss: 0.6189
Epoch 020: Train_loss: 0.3029, Validation_loss: 0.6410
Epoch 021: Train_loss: 0.2523, Validation_loss: 0.6092
Epoch 022: Train_loss: 0.2835, Validation_loss: 0.6785
Epoch 023: Train_loss: 0.2510, Validation_loss: 0.7170
Epoch 024: Train_loss: 0.1766, Validation_loss: 0.7224
Observing increasing values for validation loss. Early stopping.
```

Figure 5: Epochs

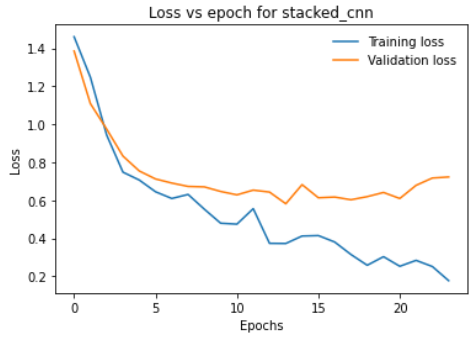


Figure 6: Training vs Validation Loss

Here, it was observed that even though the value of epochs was set to 150, the training process stopped after 24 epochs after increasing values of validation loss were observed for 3 epochs.

EVALUATION:

During the evaluation phase, the test dataset was used to measure the performance of the three models created for the project.

The following section contains the evaluation observations for the final CNN architecture and its two variants:

1. **stacked_cnn** (Model with the final CNN architecture):
    a. Precision, recall, F1-measure, accuracy on testing data:

```
Classification report for stacked_cnn
----------------------------------------------------------------------
Accuracy: 71.88%
Precision: 82.12%
Recall: 71.88%
F1 score: 76.56%
----------------------------------------------------------------------

                precision    recall  f1-score   support

     0_cloth       0.70      0.58      0.64        12
   1_surgical      0.79      0.65      0.71        23
      2_n95        0.00      0.00      0.00         3
 3_incorrect       1.00      0.88      0.93        16
   4_no_mask       1.00      1.00      1.00        10

    accuracy                           0.72        64
   macro avg       0.70      0.62      0.66        64
weighted avg       0.82      0.72      0.77        64
```

Figure 7: Confusion matrix a

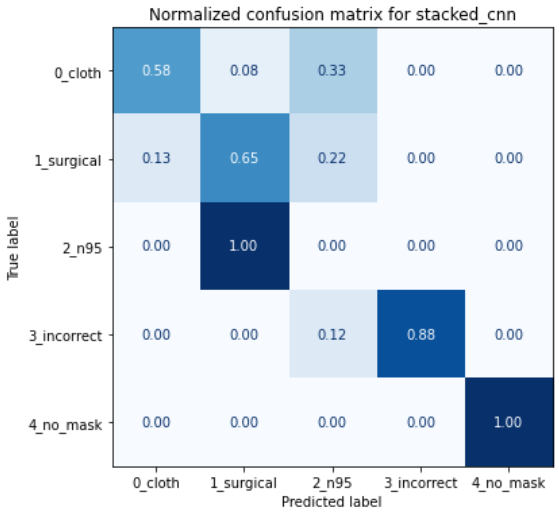


Figure 8: Normalized Matrix a

2. **variant_1** (Model with pooling layers removed from the final architecture):
    a. Precision, recall, F1-measure, accuracy on testing data:

```
Classification report for variant_1
----------------------------------------------------------------------
Accuracy: 81.25%
Precision: 84.65%
Recall: 81.25%
F1 score: 79.74%
----------------------------------------------------------------------


              precision    recall  f1-score   support

     0_cloth       0.67      0.73      0.70        11
   1_surgical      0.63      0.86      0.73        14
       2_n95       1.00      0.25      0.40         8
  3_incorrect      0.95      0.95      0.95        19
    4_no_mask      1.00      1.00      1.00        12

    accuracy                           0.81        64
   macro avg       0.85      0.76      0.75        64
weighted avg       0.85      0.81      0.80        64
```

Figure 9: Confusion matrix b

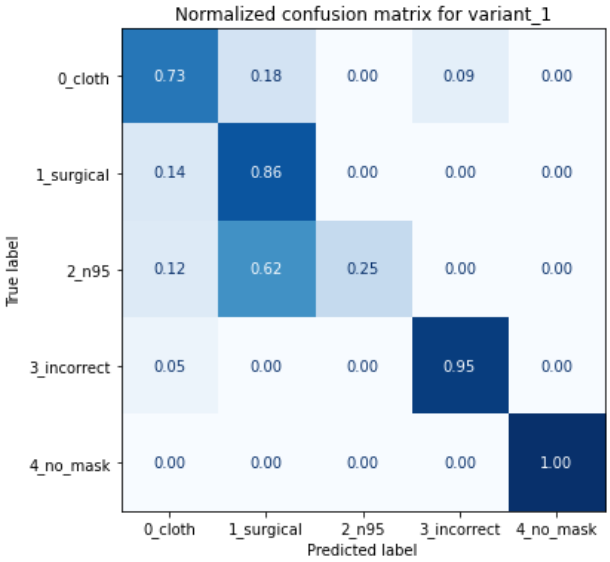


Figure 10: Normalized Matrix b

3. **variant_2** (Different number of convolution layers from the final architecture - 3 convolution layers):
    a. Precision, recall, F1-measure, accuracy on testing data:

```
Classification report for variant_2
---------------------------------------------------------------------
Accuracy: 75.00%
Precision: 78.82%
Recall: 75.00%
F1 score: 76.28%
---------------------------------------------------------------------
```

	precision	recall	f1-score	support
0_cloth	0.76	0.87	0.81	15
1_surgical	0.80	0.63	0.71	19
2_n95	0.14	0.25	0.18	4
3_incorrect	1.00	0.83	0.91	12
4_no_mask	0.80	0.86	0.83	14
accuracy			0.75	64
macro avg	0.70	0.69	0.69	64
weighted avg	0.79	0.75	0.76	64

Figure 11: Confusion matrix c

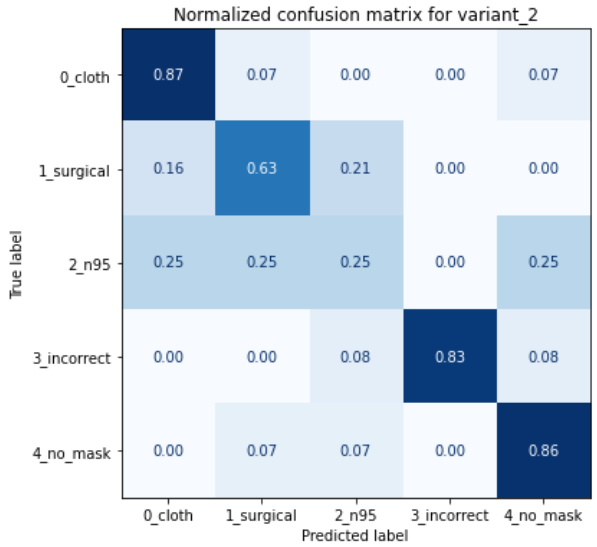


Figure 12: Normalized Matrix c

From the above observations, it can be seen that variant 1 has the highest performance when comparing the performance metrics for all three models. Both variations of the final CNN architecture seem to have a positive impact on the model performance. Variant 1, with pooling layers removed from the final architecture has the most performance gain. However, this may change with further fine-tuning of the hyperparameters.

From the confusion matrices, it can be observed that all three models are able to identify the "incorrect" and "no mask" categories. Their ability to correctly identify "cloth mask" and "surgical mask" examples varies. None of the models perform well for the "N95 mask" category.

During the second phase, we aim to add more samples of the "N95 mask" class. We expect it will help improve the performance for the class which is not good with the current models and dataset. Also, the hyperparameter values were chosen after performing manual hyperparameter search. This could be enhanced in phase 2 by setting up hyperparameter search during training.

REFERENCES:

[1] https://www.kaggle.com/datasets/andrewmvd/face-mask-detection

[2] https://www.kaggle.com/datasets/mloey1/medical-face-mask-detection-dataset

[3] https://www.kaggle.com/omkargurav/face-mask-dataset

[4] https://www.kaggle.com/datasets/prithwirajmitra/covid-face-mask-detection-dataset

[5] https://www.kaggle.com/datasets/prithwirajmitra/covid-face-mask-detection-dataset

[6] https://www.kaggle.com/datasets/prithwirajmitra/covid-face-mask-detection-dataset

[7] https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html