**COMP 6721 Project**

**AI Face Mask Detector - CNN**

TEAM NAME: **NS\_09**

|  |  |  |
| --- | --- | --- |
| **Name** | **ID** | **Work** |
| Smit Desai | 40120178 | Data collection & preprocessing.  Model building and training |
| Aayush Khandelwal | 40156633 | Model building and training |
| Mansajan Singh | 40160310 | Evaluation |

**1. DATASET**

**1.1 COLLECTING DATA**

As outlined in the project description, the model had to classify images into 4 different classes: Cloth-Mask, Surgical-Mask, FFP2-Mask, and No-Mask. The dataset has a total of 1988 images. Here are some statistics about the dataset:

Chart, pie chart

Description automatically generatedChart, pie chart

Description automatically generatedChart, pie chart

Description automatically generated

Source:

1. Cloth-Mask, Surgical-Mask images were collected from Google images.
2. FFP2-Mask images were collected from Shutterstock.
3. No-Mask images were collected from 2 different Kaggle datasets
   1. <https://www.kaggle.com/vinaykudari/facemask> by Vinay Kudari
   2. <https://www.kaggle.com/spandanpatnaik09/face-mask-detectormask-not-mask-incorrect-mask> by Spandanpatnaik

(Please see the attached reference files to view every image source)

**1.2 PREPROCESSING**

The following preprocessing steps were taken:

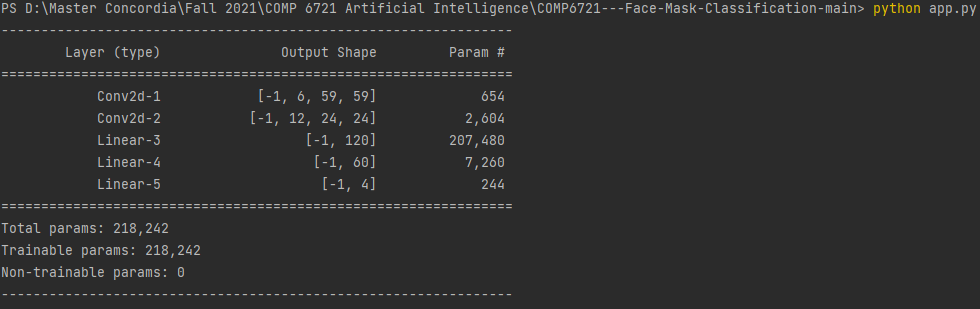
1. **Resize:** In order to feed the images to the CNN network, they were resized (64 px, 64 px)
2. **Horizontal-flips:** To increase randomness, images were flipped horizontally with a probability of 0.5
3. **Conversation to tensor:** Converts the images into an array of numbers, called torch tensor. Each pixel of the input RGB image is divided into three different pixels- red, blue, and green. This creates three different images. For each generated image, the pixel value is divided by 255 to range the pixel range from [0 255] to [0 1]
4. **Train-Test split:** The data set is split into 2 categories during runtime. The training dataset has 1690 images (85%) whereas the test dataset has 298 images (15%)
5. **Batch-loading:** Finally, images from both categories are loaded into a batch of size 32 and are randomly shuffled.

**2. CNN ARCHITECTURE**

Taking account of CNN architecture we have created a Convolution Neural Network which primarily consists of 3 non-linear convolution layers and 2 linear fully connected layers. The architecture initiates with producing a feature map when the input tensor of shape [32, 3,64,64] is passed through this layer. The stack normalization is then applied to all the feature maps. This helps reduce a wider weight range by normalizing with the standard deviation.

Here, PyTorch describes the beta and gamma hyperparameters of batch normalization. For this architecture, we have chosen Relu as our activation function because it sets the negative value to 0. After that pass the output tensor through the dropout layer which will deactivate randomly selected activations according to probability. This gives the model considerable robustness and reduces the possibility of overfitting. At Last, performing MaxPooling with a kernel size of 2X2 to reduce network complexity. This model used a 3x3 size kernel to extract features. To further optimize the memory, we implemented the Relu feature directly on the output sensor without allocating additional memory. For reshaping the formula we have used is (width\_of\_input –kernel\_size + 1)/stride so by placing the value in the above formula gives us the output 32 after applying operation.

Looping the above-mentioned process for the next 2 convolution layers produces an output tensor of shape which then passes it to the Fully connected linear layer, followed by flatting the output tensor first, producing the linear output tensor. The output is shown in Figure 3 given below:



Here, all layers are classes in PyTorch's nn module, and model classes are inherited from nn. Module classes that handle the entire complexity of initializing model parameters (weighting), manipulating them, and storing them in memory.

**3. CNN Model Training**

Here, we have selected five model training epochs and used batch input processing to overcome the customization of large amounts of data in memory. (batch size for input during training is 32).

CNN model training has two phases: forward pass and backward pass. Repeated at each epoch, the input batch goes through a series of layers defined by the forward method of the CNN model class. After performing this forward pass, the loss value is calculated for the specified prediction (at this point model.train() allows gradient manipulation so you can update the model weights). The loss value or gradient is typically the following error calculation: Each output node and inner layer node. This is done using the CrossEntropyLoss function of the Nn module. This function finally uses the SoftMax activation function to narrow the output probability between 0 and 1 and calculate the loss value. Next, loss.backward () performs the backpropagation phase. In this phase, the difference in total weighting is calculated and optimized and optimized. Step () updates the model weights. Here, Adam is used as the optimizer and a learning rate of 0.01 is selected for backward execution. Here, the output prediction gets the probabilities of all four output classes. Among them, the class with the highest probability value was selected as the class predicted by the model.

**4. EVALUATION**

The initial step in evaluating our model involved fine tuning the hyperparameters of the CNN model i.e.the number of epochs in our case. This was determined by applying our generated model to the validation dataset and examining the output. The validation dataset consists of 1988 images split evenly among the four classes namely “Cloth-Mask”, “Surgical-Mask” , “FFP2-Mask” , and “No-Mask”. The output was observed and it indicated that after 5 epochs the improvement in accuracy and loss was unnoticeable

Results of each epoch are as follows.

**Epoch 1:** Correct predictions: 7

Accuracy: 0.46094674556213017

Loss: 8.298946174855768

**Epoch 2:** Correct predictions: 801/1690

Accuracy: 0.47396449704142013

Loss: 0.038740938302327894

**Epoch 3:** Correct predictions: 801/1690

Accuracy: 0.47396449704142013

Loss: 0.038791874174535626

**Epoch 4:** Correct predictions: 801/1690

Accuracy: 0.47396449704142013

Loss: 0.03877918755514382

**Epoch 5:** Correct predictions: 801/1690

Accuracy: 0.47396449704142013

Loss: 0.03880976057616917

When model was applied for testing, it predicted 151 images as correct class out of 298. **Accuracy comes out to be 0.5067114093959731 and loss was 0.04058477662553723.**