IN3060/INM460 Computer Vision Coursework report

Data

The dataset consists of respectively 2394 training and 458 testing images of human faces. Each image has a corresponding .txt file which consists of the following labels: 0 – no mask, 1 – mask with proper wear, 2 – mask with improper wear. When creating a sampling distribution function for the train data, there are 376 images with label 0, 1940 images with label 1, 78 images with label 2. As there is huge disparity between the number of images with different classes, to prevent biases, rebalancing is made. As the file is in a form of .zip, following lab 7, the file was copied and unzipped directly to the Colab server while also making accessing the data much faster. The .zip file consisted of 2 folders, test and train and both had folders of images and labels. The images folders included all the .jpeg images for the train and test datasets with the name of each image file. The labels folders contained the same amount of .txt files with the same name as the images. Inside each .txt file there is a number label 0,1,2. To organize the code for this coursework, a function dataFolder was developed where it first creates a folder Data, which then includes a folder for test and train images and then both test and train folder include 3 folders with the labels of the images (0,1,2). The function then matches the names of the images and the text files which then reads the text file and puts the image into the corresponding label.

For the personal dataset, 7 images for each label (21 total) were downloaded and put inside the Personal_Dataset folder, Within that folder, similarly to the organization of the images for the previous implementation, 3 folders of names 0,1,2 were created which then was used for "testing in the wild".

Implemented methods

Following the example of image classification with Bag of Visual Words in session 7, 3 combinations of feature descriptors and classifiers were developed. The combinations created via this method were SIFT with SVM and SIFT with MLP. The reasoning for choosing these models is because we need to differ between the images, and each image has a certain set of features. These features consist of keypoints and descriptors which is what this model excels at and can reach the goal of matching the images to labels.

For both the models then a function import_selected_data was created following lab 7 which loaded the images and the respective labels and using it data was imported. To check the distribution among different classes Counter was used to distinguish the number of images for each label in the training set. This is also how the distribution was identified in the Data section above. As the dataset was largely unbalanced, to prevent bias at test time, stratify parameter was used from sklearn.model_selection.train_test_split.

Once that was done, SIFT was used for both models to initiate the first two steps of BoVW which is interest point detection and feature descriptor extraction. Once keypoints were identified from the images, the feature descriptors were extracted from each. These points were then mapped into the 128-dimensional feature space and a list of descriptors and labels were appended.

The next step was the clustering of the descriptors in the feature space which was done via K-means. This algorithm looks for the clusters in the given space and identifies them as features. Those clusters for images are then plotted into the histogram of codewords. This is done to ensure uniform dimensionality across images regardless of the amount of keypoints detected.

At test time (including search in the wild dataset), we keep the coordinates from the feature space and the position and orientation of the decision boundary. As images are unseen previously, we first detect the interest points, apply a feature descriptor to find the vectors and create the histogram using the codebook generated during training.

Everything is now ready to be fed into the classifiers (SVM, MLP), where the hyperparameters adjustments were tested and are discussed in the results section below for qualitative and quantitative analysis.

The next 2 models developed were by using HOG feature descriptor (presented in lab 6) and a combination of SVM and MLP classifier. Similarly, to the BoVW example in lab 7 and SIFT implementation the first thing that was done was to load the data. However, here the images had to be first converted into grayscale images in order to compute the HOG features. Then padding was done before loading the data.

For dataset rebalancing, instead of using stratify, like it was done for the BoVW models, I have applied SMOTE from the lab 7 website. First, I had to reshape the train data and convert it into an array to apply SMOTE. After that was done, the train data was changed back into a 1d array to prepare it for the classifiers.

Additionally, to increase the accuracy of the HOG extractor feature, PyTorch padding was applied.

After applying SVM and MLP classifiers to the HOG respectively, accuracy of the model was calculated and then 2 types of confusion matrix were plotted. First is the same as for the SIFT models comparing the actual label with the predicted one, and second type is the matrix with: true positive (TP), true negative (TN), false positive (FP), false negative (FN).

For CNN implementation, first GPU availability was checked for faster computing. Then following previous implementations, the data was imported and organized the same way. The libraries used were imported similarly to lab 8 and 9. For loading the data, first the means and standard deviations variables of three colour channels of the datasets were defined, which then were used to transform the data. The next step was to create directories for the train and test data following the lab 9 lab.

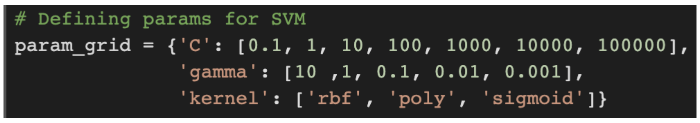
Once the data was prepared, the size of the first input layer was determined: [4, 3, 224, 224], where 4 is the match size, 3 is the input channel number, and 224 are the width and height. Learned from the lab8 CNN implementation on CIFAR-10 dataset, IN3063 module and YouTube tutorial, testing on different convolutional layers was done to determine the values for the neural network. Once the optimal number of layers and the values have been determined a ConvolutionalNeuralNetwork class was defined, with the parameters for layers tested, which was then applied in the forward pass, followed by flattening the view to pass the fully connecting layers and then by applying ReLU to fully connected layers. After that, the model was ready to be trained with adjustable hyperparameters following the lab 9 example. To test the model qualitatively and quantitatively the training model was run which allowed for comparison between different parameters, which then allowed us to visualise by implementing ground truth labels and plotting graphs.

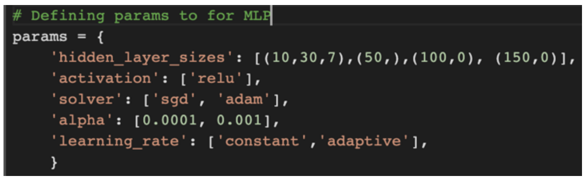
Results (for more detailed images, please see Code folder for specific models and test_functions)

For the BoVW models, the first optimization was done by splitting the train and test datasets to handle class imbalances which could lead to bias. As the dataset is very unbalanced, this ensured that the label distribution was maintained for both data splits. The data split leading to best accuracy for SIFT_SVM model was with a 90/10 split, whereas for SIFT_MLP model the best split was 80/20, without any sacrifices in time.

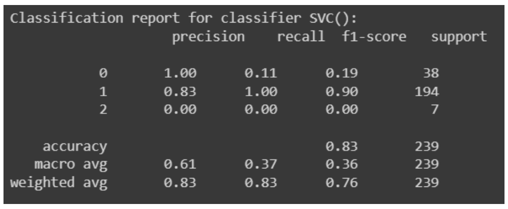
The next parameter to consider was the variables for the clustering of descriptors K-means. As the sample size was less than 10000, when comparing both methods using KMeans and MiniBatchKMeans, no significant difference in accuracy was observed, but the computation time was increased by 5-10 seconds. When choosing different batch sizes, the more batch sizes, the shorter the computation time, with slight decrease in accuracy, therefore the value was left at 4 where it was found to be most optimal for both time and accuracy. When testing a different number of centroids, when increasing the value of it the processing time has increased significantly with no increase in any other aspect, whereas when decreasing it, the accuracy has decreased. Therefore, it was kept at the rule of thumb of *10.

For the classifier optimization, for SVM in a grid search was attempted from GridSearchCV to find best parameters following the guide on a blog. For MLP the hyper parameters were adjusted from here.

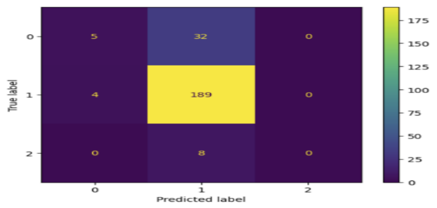

```
# Defining params for SVM
param_grid = {'C': [0.1, 1, 10, 100, 1000, 10000, 100000],
              'gamma': [10 ,1, 0.1, 0.01, 0.001],
              'kernel': ['rbf', 'poly', 'sigmoid']}
```


```
# Defining params to for MLP
params = {
    'hidden_layer_sizes': [(10,30,7),(50,),(100,0), (150,0)],
    'activation': ['relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.001],
    'learning_rate': ['constant','adaptive'],
    }
```

Best performing kernel for SVM was rbf as it is most suitable for images when compared to others (poly/ sigmoid). The gamma value that led to best accuracy was 10. Best value of C was 10.

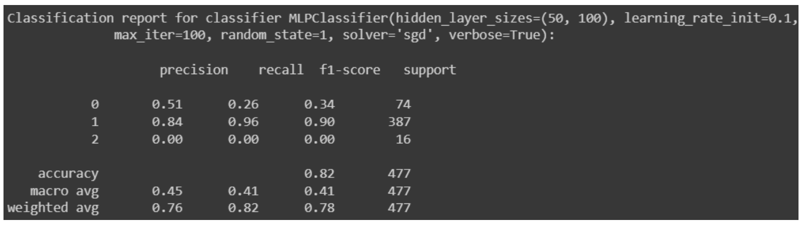


```
Classification report for classifier SVC():
              precision    recall  f1-score   support

           0       1.00      0.11      0.19        38
           1       0.83      1.00      0.90       194
           2       0.00      0.00      0.00         7

    accuracy                           0.83       239
   macro avg       0.61      0.37      0.36       239
weighted avg       0.83      0.83      0.76       239
```

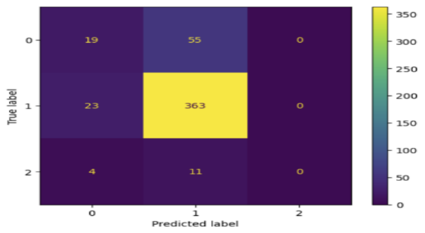


	0	1	2
0	5	32	0
1	4	189	0
2	0	8	0

For MLP, the best solver was stochastic gradient descent (SGD), with hidden layer =50,100. For the number of maximum iterations, the bigger it was, the better was the accuracy. However, the computation time also increased. At some point, the increase in accuracy was negligible compared to the increase in time. The learning rate, the smaller the more accurate the model, but similarly to max iteration, the processing time also increased. That's why max_iter = 100, learning_rate= .01

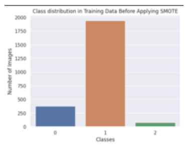


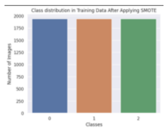
```
Classification report for classifier MLPClassifier(hidden_layer_sizes=(50, 100), learning_rate_init=0.1,
              max_iter=100, random_state=1, solver='sgd', verbose=True):

              precision    recall  f1-score   support

           0       0.51      0.26      0.34        74
           1       0.84      0.96      0.90       387
           2       0.00      0.00      0.00        16

    accuracy                           0.82       477
   macro avg       0.45      0.41      0.41       477
weighted avg       0.76      0.82      0.78       477
```



	0	1	2
0	19	55	0
1	23	363	0
2	4	11	0

When comparing the rebalancing method, SMOTE gave better results, as instead of stratifying the dataset. Stratifying only partially prevented bias issues by splitting the test and train data by maintaining the data distribution, however in SMOTE the classes with less samples were matched which led to better accuracies.

Before SMOTE: 

After SMOTE: 

For HOG, following results were calculated with SVM.
```
HOG_SVM Classifier Accuracy: 0.8471615720524017
```

```
Confusion Matrix:
[[   0  51    0]
 [   0 388    0]
 [   0  19    0]]
```

```
Class 0: TP = 0, TN = 407, FP = 0, FN = 51
Class 1: TP = 388, TN = 0, FP = 70, FN = 0
Class 2: TP = 0, TN = 439, FP = 0, FN = 19
```

For HOG, the following results were calculated with MLP.
```
HOG_MLP Classifier Accuracy: 0.8209606986899564
```

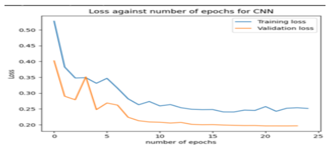
```
Confusion Matrix:
 [[ 14  36   1]
 [ 23 362   3]
 [  4  15   0]]
```

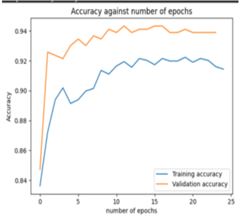
```
Class 0: TP = 14, TN = 380, FP = 27, FN = 37
Class 1: TP = 362, TN = 19, FP = 51, FN = 26
Class 2: TP = 0, TN = 435, FP = 4, FN = 19
```

As HOG models used SMOTE, it led to better accuracy than using stratify like in BoVW models. However, the computation time was much longer due to having more samples to go through.

For CNN, the parameters tested were learning rate, number of epochs, and optimizers used. Generally, the smaller the learning rate chosen, the larger was the loss value, leading to lower accuracy and computation size being larger. The bigger the number of epochs the bigger was the accuracy. The optimizer that gave better results was Adam, when compared to SGD. Following results were obtained for 25 epochs (from 0/0)

```
Epoch 24/24
train loss: 0.2514 acc: 0.9144
test loss: 0.1964 acc: 0.9389
CNN average accuracy: 91.19%
Training complete in 5m 23s
```





Once the model was finished training, 2 graphs were plotted. First was loss against number of epochs and second was accuracy against epochs.

The accuracies range from 75% to slightly over 90% depending on mode. The best performing model is CNN, followed by SMOTE HOG models, followed by BoVW.

Overall, all the models guess correctly majority of the time, when tested on ground truth labels matching the image to the label. This is the case for both the images in the test set and for the images for the personal dataset when "testing in the wild".

Ground truth label on test data:

Ground truth label on personal dataset: