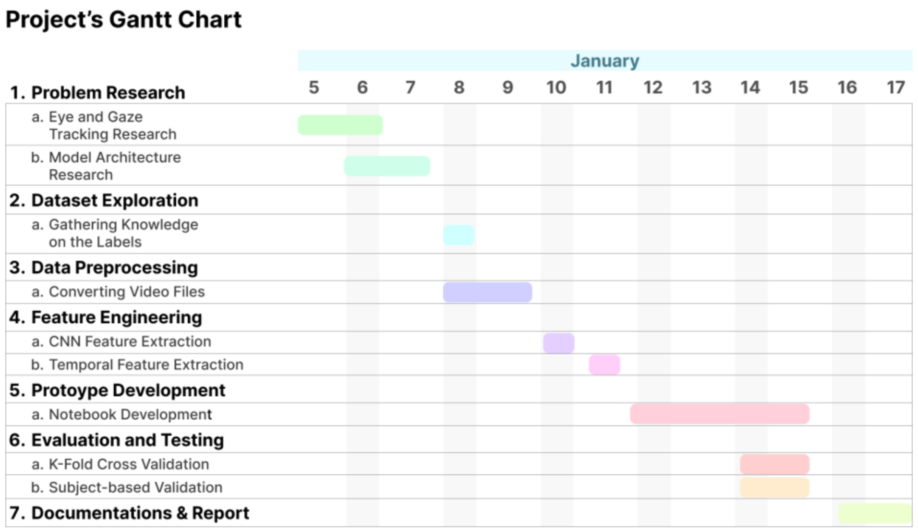
Proctoring System: Cheating-Behavior Detection System

HeyHi Technical Test Project

Daffa Bilnadzary

1. Project Overview

This project is done as the submission for the HeyHi Technical Test project. The project aims to create a model that can classify students and test takers' unusual behavior such as cheating. This project will be divided into several parts.



Project's Gantt Chart, Planned for 2 Weeks

2. Background and Data Overview

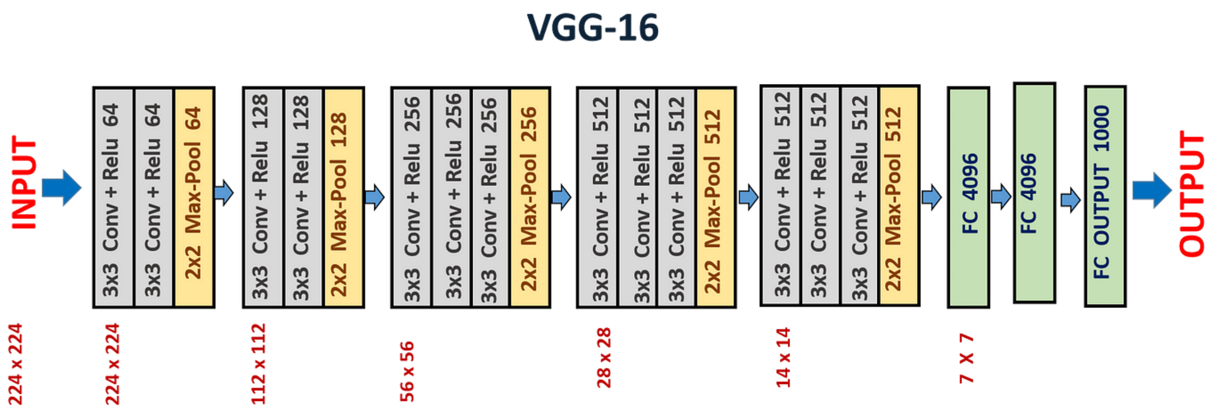
A proctoring system is a system that aims to observe any unusual behaviors such as cheating on a test/exam. With the transition from paper-based exams to computer-based exams, cheating can also evolve into new forms, such as internet browsing, voice call, or any digital based approach. A dataset is provided to build the proctoring model. This dataset contains recordings from across 10 different unique subjects, each containing 3 different files recorded from 3 different devices.

The first one is a webcam, recording the subject/test taker during the exam process. The second one is a wearcam, recording the point-of-view of the test taker. This view mostly consists of the monitor screen displaying the exam UI, as well as any objects used during any cheating period. The third one is a microphone, used to capture sound from the subject. All the videos are taken in 25 Frames Per Second (FPS), with the dimension of 480x640.

3. Feature Engineering

Since all the data are in video format, the features will be captured by first extracting the frames from the video. While this can be done automatically by extracting all the frames in one video, it would lead to memory overload due to the size of the data created. Hence, a sampling process is employed. Since each video is taken in 25 FPS, it will be resampled first to a reasonable 5 FPS. Each frames captured will also be resized to half its original size before putting it into the array.

Out of all three sources available in the dataset, the wearcam is best suited to be extracted, since the recordings mainly show not only the monitor as its object, but also any objects that might appear during the duration of the cheating behavior, such as phone, textbooks, or even the screen change on the monitor (indicating browsing internet sources). Hence, only the wearcam is used as our source data.

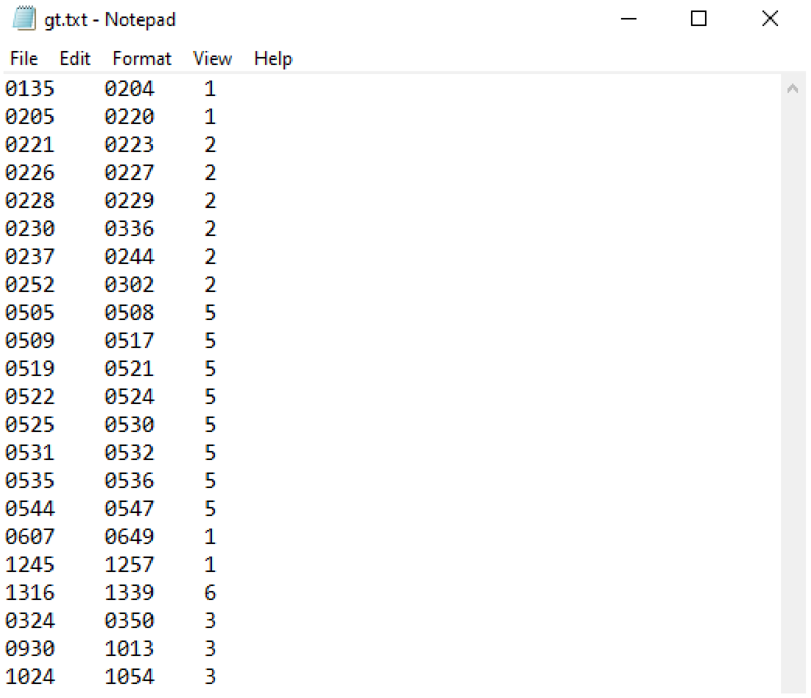


VGG16 Architecture

Instead of putting all of the extracted frames into the model immediately, we will use the pretrained VGG16 model to extract features from the image. The VGG16 is trained on Imagenet, and can be a good source of object detection model. However, we will only use the architecture up until before the output's dense layers. This way, the model can return features obtained from the MaxPooling layer, which can then be used further.

The output of the VGG16 model is an array of size of 51178 rows and over 30000 columns. These number of features need to be reduced, since training with this large amount of data can be exhausting. A technique called Principal Component Analysis (PCA) is employed. PCA is used to reduce the complexity of the data by reducing the number of features on the dataset. We set our threshold to be 90%, which means we want to retain all the features that can already explain 90% of the dataset complexity, which is a fairly good trade off between memory consumption, speed, and accuracy. The result is a dataset with the size of 51178 rows and 100 columns/features.

Lastly, to make sure the model is able to learn effectively, the raw ground truth file needs to be processed. The original ground truth file is a .txt file containing all the range of timestamps a cheating behavior has occurred as well as the cheating class labelled. For example, “0135 0140 1” means every frames that are inside the timestamp range between 01:35 to 01:40 will be marked as 1. This process will be done to every single timestamp. All the timestamps that are not labelled as any of the ground truth labels will be labelled as 0, indicating a non-cheating behavior. The output of this process is a fully concatenated dataset, as well as its labels, which can be exported into a single CSV file.



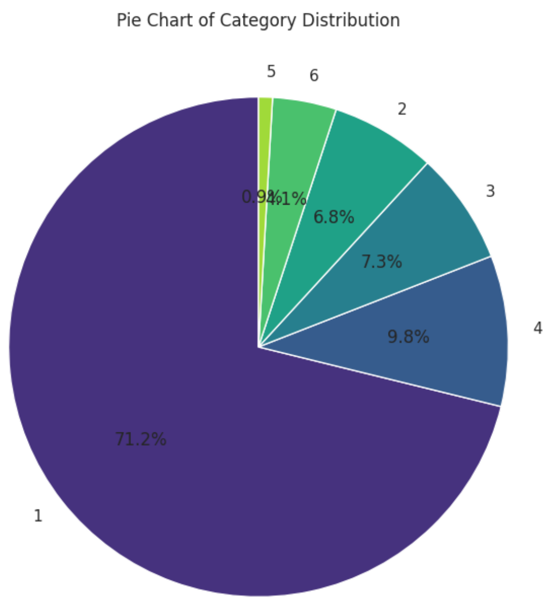
```
0135    0204    1
0205    0220    1
0221    0223    2
0226    0227    2
0228    0229    2
0230    0336    2
0237    0244    2
0252    0302    2
0505    0508    5
0509    0517    5
0519    0521    5
0522    0524    5
0525    0530    5
0531    0532    5
0535    0536    5
0544    0547    5
0607    0649    1
1245    1257    1
1316    1339    6
0324    0350    3
0930    1013    3
1024    1054    3
```

Example of the ground truth file.

4. Exploratory Data Analysis

An Exploratory Data Analysis step is employed, to look further into the processed dataset, as well as the label distributions. Out of 51178 rows extracted from all the videos, we want to first check the label distributions. For the sake of modelling purposes, the labels are remapped by assuming 1 as the non-cheating behavior, while the rest of the labels indicate the cheating behavior of different classes.

Based on the figure below, it is clear that around 70% of the rows are labelled as “non-cheating behavior”, while around 30% are indicated as cheating. The order of frequency of the cheating types from highest to lowest are class 4, 3, 2, 6, and 5, respectively. Keep in mind that these distributions are taken generally among all the dataset, not individually. Individual subject might have more different and unique label distributions since cheating behaviors can be completely independent.

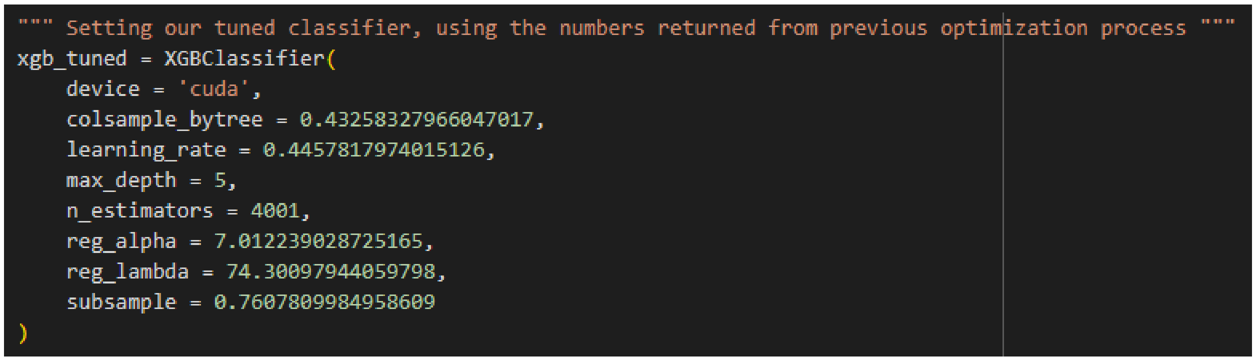


Another thing to analyze is related to the core subject of the problem, which is human activity recognition. Since the problem is correlated to activity recognition, we might want to see whether or not a temporal feature extraction will boost the performance of our classifier significantly. These features will capture the temporal aspect of the features since as we can see in the dataset, some activities (both cheating and non-cheating) can last for a period of time. Instead of treating every single row as a completely independent observation point, we want to treat one data as a bunch of time-related data within a short span of time. To determine the right size of the temporal window, an investigation about the average duration of all cheating activities is done.

Based on the observation, the total average cheating duration on all subjects is 75 frames (or 15 seconds). This means that on average, a cheating behavior might last around 10-15 seconds. We will use 10 seconds as the fixed length of the sliding window to extract the temporal features of the dataset.

5. Prototype (Model) Development

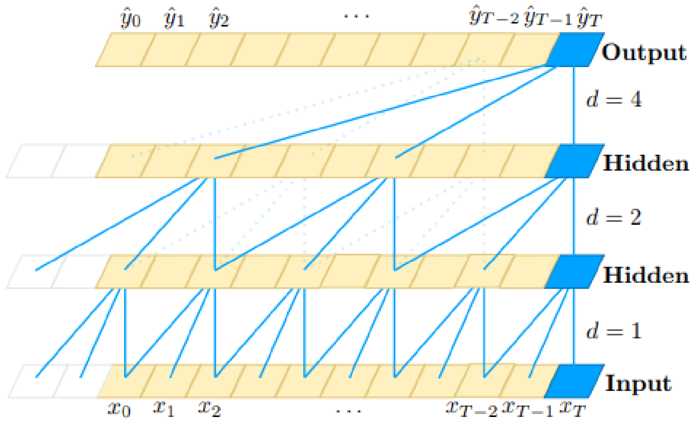
In this project, two models with two evaluation methods will be developed. The first model is an XGBClassifier model, which is a relatively powerful tree-based machine learning model. Since the input of the model is in the shape of 2D, we will not employ a temporal feature extraction process while developing this model. Along with the model training, a Hyperparameter Optimization (HPO) process is also done using the Bayesian Optimization algorithm. Code below shows the model instantiation process using the HPO parameter values result.



```
""" Setting our tuned classifier, using the numbers returned from previous optimization process """
xgb_tuned = XGBClassifier(
    device = 'cuda',
    colsample_bytree = 0.43258327966047017,
    learning_rate = 0.4457817974015126,
    max_depth = 5,
    n_estimators = 4001,
    reg_alpha = 7.012239028725165,
    reg_lambda = 74.30097944059798,
    subsample = 0.7607809984958609
)
```

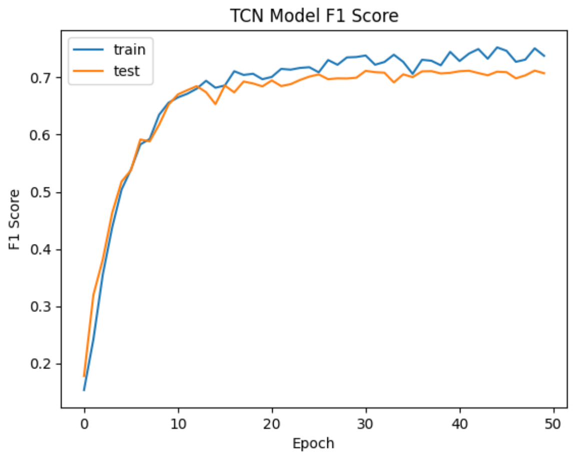
A code snippet of the tuned XGBClassifier model instantiation.

The second model is the Temporal Convolutional Network (TCN) model. While TCN can be used along with generic dense layers to produce outputs form singular data just like XGBClassifier, we can also use a TimeDistributed() wrapper to apply the layer to every temporal slice of the input. This means instead of using the dataset we used to train the XGBClassifier, we use the data that has been extracted even further. During the development, we use a window size of 50, with an overlap of 40. Meaning that the first row will be a window data of size 50, capturing the dataset from row 1 to 50, while the second one capturing the dataset from row 20 to 60 and so on. The result is a 3D numpy array of shape (5074, 50, 100).


$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ … $\hat{y}_{T-2}$ $\hat{y}_{T-1}$ $\hat{y}_T$
Output
$d = 4$
Hidden
$d = 2$
Hidden
$d = 1$
Input
x_0 x_1 x_2 … x_{T-2} x_{T-1} x_T

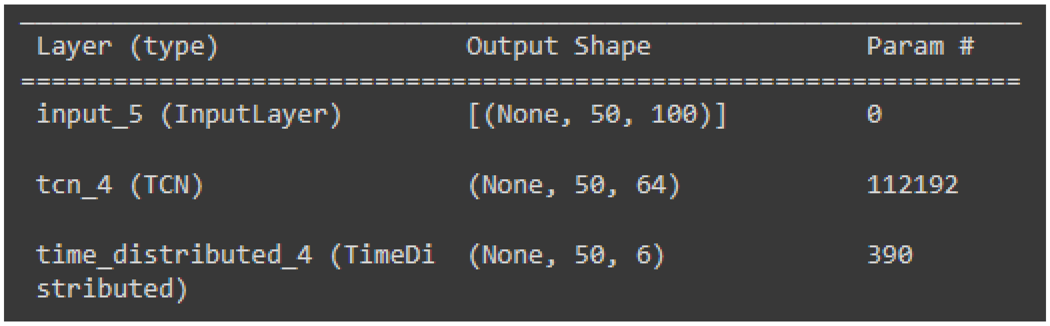
An illustration of how dilated convolution works in TCN.

The TCN has various parameters that can be set. In this observation, we use ReLU as the activation function and softmax as the output activation function, since the problem lies under the scope of multiclass classification. The model is compiled with Adam as the optimizer, with learning rate of 0.001. Each model is trained on 50 epochs, with 32 batch size.



TCN Model Training History F1 Score.

Both of the model will be trained using a dataset that has been splitted before to gain both the train and test set. The proportion of the split is 80:20, where 80% of the data is split into training set, while the remaining 20% is for the testing process. A fixed seed is also employed to make sure each splitting process is repeatable, producing the same splitting results.



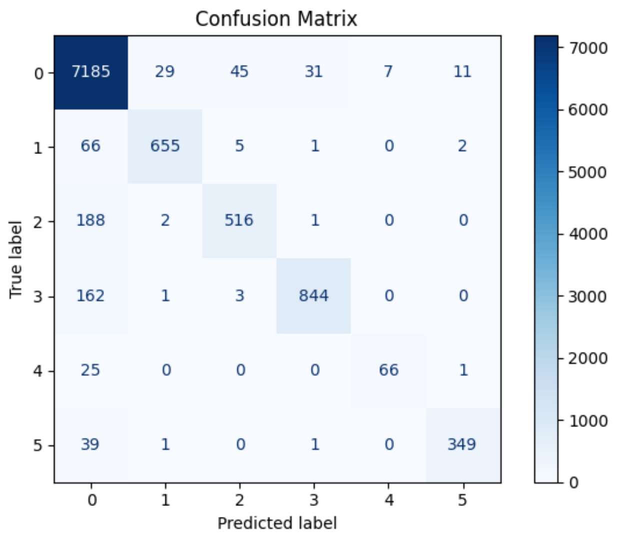
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 50, 100)]         0

 tcn_4 (TCN)                 (None, 50, 64)            112192

 time_distributed_4 (TimeDi  (None, 50, 6)             390
 stributed)
```

TCN Model Summary.

6. Model Evaluation and Testing

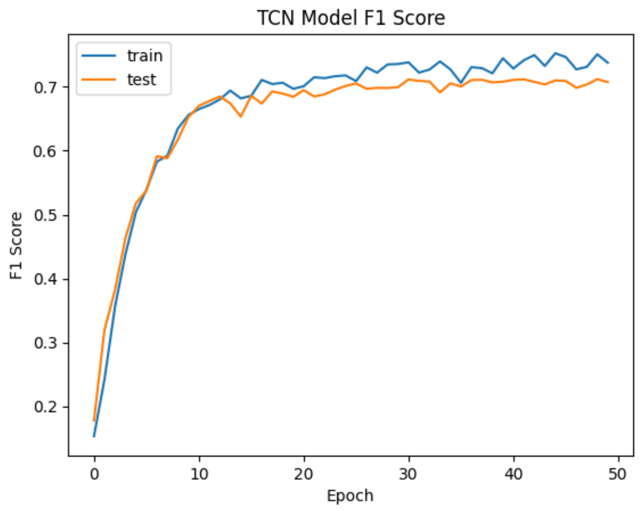
The trained models will be evaluated using two evaluation methods, K-Fold Cross Validation and Subject-Based Validation/Evaluation. The first one, K-Fold, is a generally widely used method for model evaluation. In this project, the K is set to 5, which means there will be 5 models created. Two metrics will be used, accuracy and F1 Score. Accuracy measures the average percentage a label is predicted correctly, while F1 Score is a better metric that calculates both the precision and recall of the model. Since the model's output lies under the scope of multiclass classification, F1 Score is generally more recommended especially for unbalanced data.

For XGBClassifier, the model was able to perform relatively good with an average accuracy of 92.9%, while the F1 Score is 88.5. While it is a bit lower than the accuracy, the performance is still good since it does not drop significantly. A model that does not generalize well across all labels will perform much worse. Keep in mind that we only use the tabular 2D data for our XGBClassifier, without the temporal features. Figure below shows the confusion matrix of the model's prediction. The label 0 indicates the non-cheating behavior class, while the rest of the classes are types of cheating behaviors. We can see that for class 0 (non-cheating), around 98.3% of the labels are predicted correctly. While the rest of the labels have accuracy of 89.8%, 72.9%, 83.5%, 71.7%, and 89.4%, respectively.



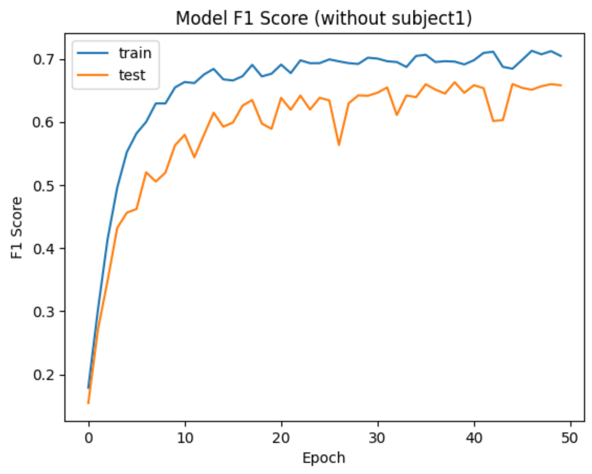
Confusion matrix for XGBClassifier's prediction.

Our second model, TCN, was able to perform much better than XGBClassifier. Our TCN model is trained on the dataset that has its temporal features extracted. Using a window size of 50 and an overlap of 40, the shape of the dataset is (5074, 50, 100), which means there are 5074 windows of shape (50, 100). The TCN model is able to perform with 98.7% accuracy and with an F1 Score of 97.2. We can see that there's a significant difference not only the accuracy but also the F1 score of our TCN model compared to the traditional XGBClassifier. Figure below shows the F1 Score history from the training process in one of the folds.



F1 score training history of TCN using K-Fold CV.

The second evaluation method is Subject-based Evaluation. While K-fold generally is a great and widely used evaluation method, we can see that our dataset consists of data taken from different random subjects. This means that we have to take into account the behavior of each individuals, which might result in a more unique data from each subject. For example, subject A might tend to check their phone more often during the exam session, while subject B might tend to browse the internet directly from the same monitor the exam is taken on. This evaluation approach will create a loop according to the amount of subjects participating in the dataset. Suppose there are 10 subjects in the dataset, we will create a loop of 10 where in each iteration a model will be trained using the data from 9 subjects, while the remaining 1 will be used as the test data. This process is done 10 times, resulting 10 different models. The aim of this is so that our model can generate labels from unseen data, especially for new subjects (test takers). Figure below shows the F1 Score history from the training process in one of the subjects.



F1 score training history of TCN using subject-based evaluation.

In contrast to the first evaluation method, while the training process went smoothly with relatively good performance, the testing process indicates a bad performace between all the subjects. This shows that all of our models cannot generalize good enough on unseen data. This also shows why a subject-based evaluation method is important.

7. Conclusions and Considerations

Based on the experiments done, we can conclude a few things:

1. While the raw data is in form of a video, the preprocessing and modelling step can be done by converting it into a series of images.
2. Instead of using vanilla 2D-CNN as the classifier, we can use TCN as the classifier because of its capability to support multirow predictions while also being causal.
3. The VGG16 pre-trained model can be used as a feature extractor for our images, where the number of features can also be reduced by performing PCA.
4. The processed data can be put straight away into traditional ML models such as XGBoost, or can be further explored by extracting its temporal features to be fed into more advanced models such as TCN.
5. Subject-based evaluation shows that all of the trained models cannot generalize well enough to the unseen data. A further development is suggested to collect more data from more subjects, or expanding the image extraction frequency (from 5 FPS to its original 25 FPS).
6. To handle the data imbalance even further, joining all the cheating category classes into a single class can also work. By sacrificing the ability of the model to classify types of cheatings, the performance can be improved not only on the K-Fold evaluation but more importantly on the subject-based evaluation.

Some considerations that should be conveyed while working on this project:

1. The project's development is mostly done in Python Notebook Environment in Google Colab. This is done so since it is still a prototype development. Google Colab also offers relatively good enough GPUs to train all the models as well as enough memory to store all the data.
2. All the model developments are done using the NVIDIA T4 and V100 GPU, each with 52 GB RAM and 16 GPU RAM.
3. While extracting all the frames in all videos is possible, it would make the memory overload simply because using the VGG16 alone as the feature extractor would produce over 30000 columns for one row of data.

4. Since this prototype is mostly focused on one part of the dataset which is the wearcam video, a further development using all of the sources available is suggested.
5. This project also employs the usage of subject-based evaluation, which addresses the bias in each data by checking whether or not the model can perform good on unseen data and new subjects.