**Machine Learning Project Documentation**

**Model Refinement**

**1. Overview**

The first model that achieved 83% accuracy on test data was Inceptionresnetv2 that had the parameters of:

*base\_model = keras.applications.InceptionResNetV2(include\_top=False, weights='imagenet', input\_shape=(X, Y, 3))*

*for layer in base\_model.layers:  
layer.trainable = False*

*model.add(GlobalAveragePooling2D())  
model.add(Flatten())  
model.add(Dense(256, activation='relu'))   
model.add((Dense(1, activation='sigmoid')))*

But after that the best method was to unfreeze the last 20 layers of the pretrained model (*for layer in base\_model.layers[-20]:* to evaluate and got better results on training and testing data. The purpose of iterating through these layers is typically to unfreeze them for fine-tuning. When a layer is frozen, it means that its weights are not updated during training. By unfreezing a portion of the layers towards the end of the model, you allow these layers to adapt to the specific characteristics of your dataset.

**2. Model Evaluation**

After unfreezing the last 20 layers, and changing the trainable layer to True and adding dropout of 0.5 and regularization of the kernel, we achieved 90.37% accuracy on the testing data in the evaluation of the model. The loss was 0.34% on test data.

6/6 [==============================] - 19s 2s/step - loss: 0.3483 - accuracy: 0.9037  
Test Loss: 0.3483  
Test Accuracy: 90.37%  
6/6 [==============================] - 18s 2s/step  
Precision: 0.2199  
Recall: 0.2053  
Confusion Matrix:  
[[487 110]  
[120 31]]

In model training the training accuracy was 96% and val. Accuracy was 86% of the last epoch.

**3. Refinement Techniques**

The best techniques were to unfreeze the layers. First, we checked the last 20 layers. It had good accuracy but we needed to balance the precision and recall because the low precision and recall indicated that the model was facing difficulty in correct identifying of positive instances.

Describe the techniques used for refining the model. This may include adjusting hyperparameters, trying different algorithms, or incorporating ensemble methods.

**4. Hyperparameter Tuning**

In refinement phase we added dropout layers and regularization to improve the model's generalization and reduce overfitting. our previous layers contained:   
*model.add(base\_model)  
model.add(GlobalAveragePooling2D())  
model.add(Flatten())  
model.add(Dense(256, activation='relu'))  
model.add((Dense(1, activation='sigmoid')))*

Then we added dropout and a dense layer with 512 units, ReLU activation, and L2 regularization with a strength of 0.01. L2 regularization helps prevent overfitting by adding a penalty term to the loss function based on the squared magnitude of the weights.  
*model.add(base\_model)  
model.add(GlobalAveragePooling2D())  
model.add(Dropout(0.5))  
model.add(Dense(512, activation='relu', kernel\_regularizer=keras.regularizers.l2(0.01)))  
model.add(Dropout(0.3))  
model.add(Dense(256, activation='relu'))  
model.add(Dense(1, activation='sigmoid'))*

Detail any additional hyperparameter tuning performed during the refinement phase. Include insights gained and their impact on the model's performance.

**5. Cross-Validation**

Discuss any changes made to the cross-validation strategy during model refinement and explain the reasoning behind those changes.

**6. Feature Selection**

Data Augmentation: A technique that transforms the features of the input data to increase its diversity, reduce overfitting, and improve the model's performance.  
Regularization: L2 regularization with a coefficient of 0.01

**Test Submission**

**1. Overview**

The trained model is loaded from a checkpoint file using the keras.models.load\_model function. The loaded model is evaluated on the test dataset using the evaluate method. The results are stored in the test\_results variable. The test loss and accuracy are extracted from the test\_results variable and printed to the console. The loaded model is used to make predictions on the test dataset using the predict method.

The continuous-valued predictions are converted to binary (0 or 1) by thresholding at 0.5. Predictions above 0.5 are considered as class 1, and predictions below 0.5 are considered as class 0.

Precision, recall, and confusion matrix are calculated using the precision\_score, recall\_score, and confusion\_matrix functions from the sklearn.metrics module.

**2. Data Preparation for Testing**

While downloading the kaggle DFDC dataset, it contains two folders:  
1. test\_videos (contains 400 videos)  
2. train\_sample\_videos (contain 401 videos)

We applied the above test dataset to our model to evaluate.

**3. Model Application**

First we loaded the trained model from the checkpoint file:  
#trained\_model = keras.models.load\_model("/content/drive/MyDrive/DFDC/cnn-lstm")

Then we evaluated the model on test set:  
#test\_results = trained\_model.evaluate(test\_videos)

**4. Test Metrics**

We calculated several metrics to evaluate the model's performance on the test dataset.

1. Test Loss and Test Accuracy: The test loss (test\_loss) and test accuracy (test\_accuracy) provide a measure of how well the model performs on the unseen test dataset.

2. Precision and Recall: Precision measures the accuracy of positive predictions and Recall measures the ability of the model to capture all the positive instances.

3. Confusion Matrix: provides a more detailed breakdown of the model's predictions, showing the counts of true positives, true negatives, false positives, and false negatives.

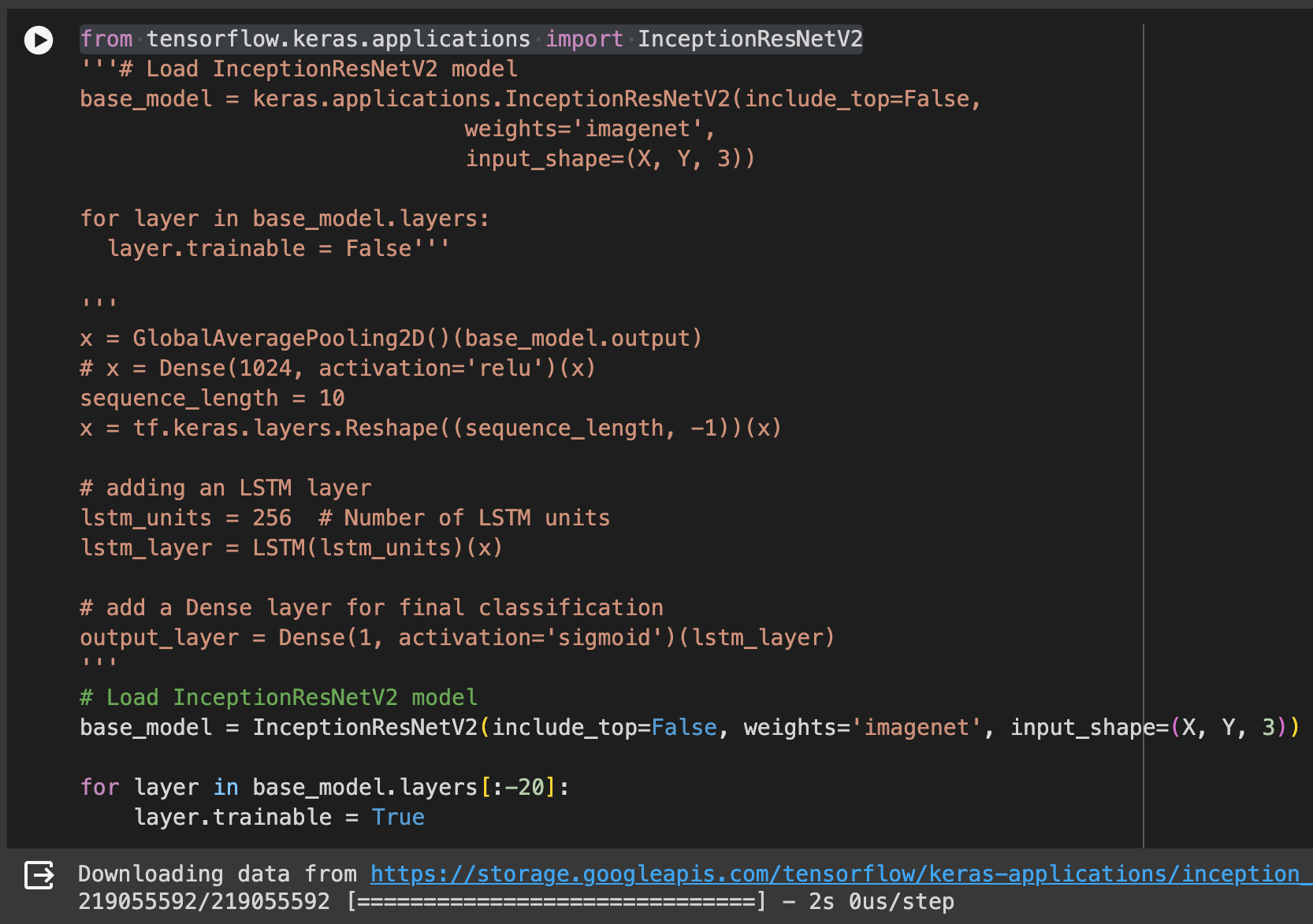
The result we got on test data:  
Test Loss: 0.3483  
Test Accuracy: 90.37%

The result we got while training the model:  
accuracy: 0.96% and val\_accuracy: 0.86%

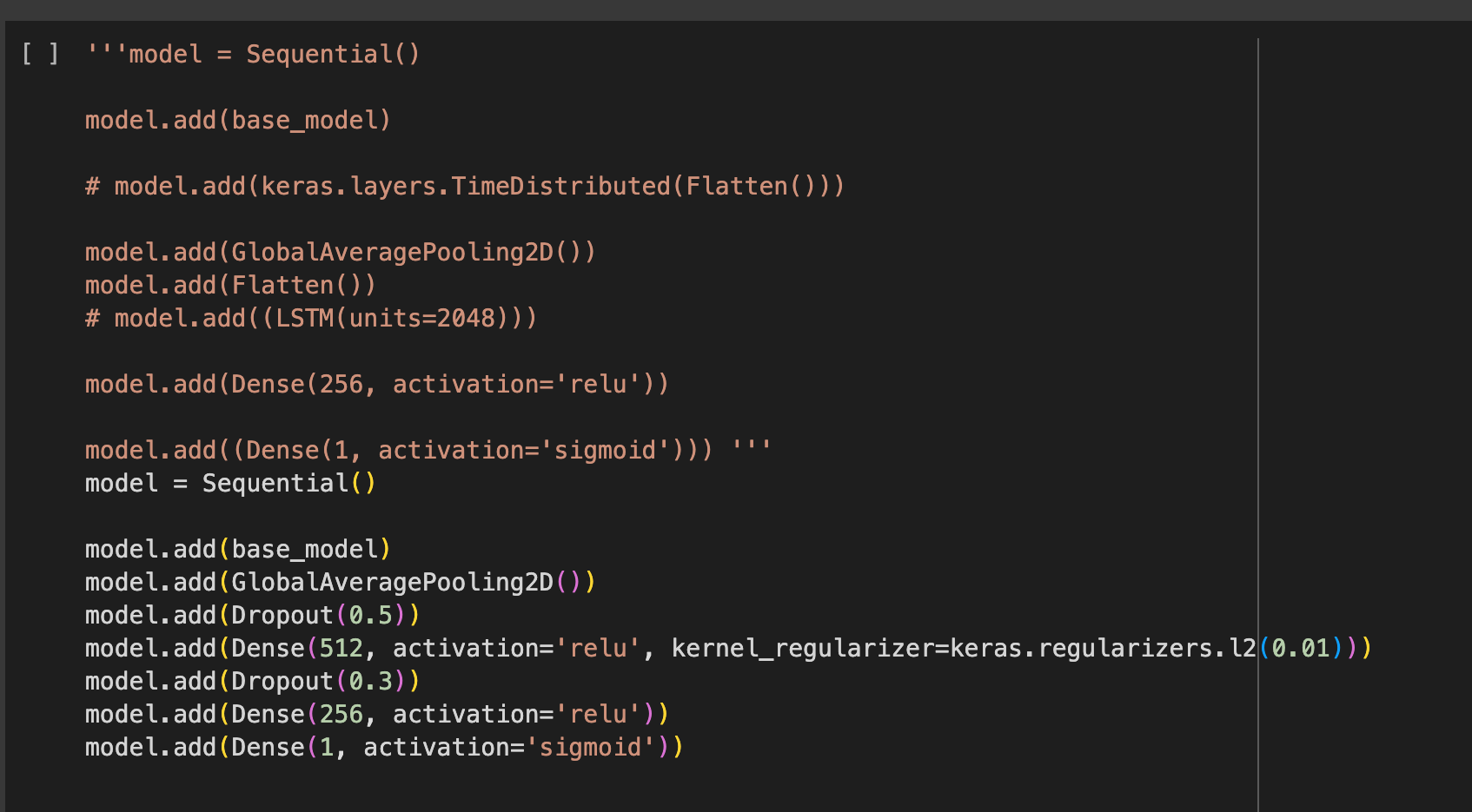
**5. Model Deployment**

If applicable, discuss any steps taken to deploy the model in a real-world setting. This may include integration with other systems or platforms.

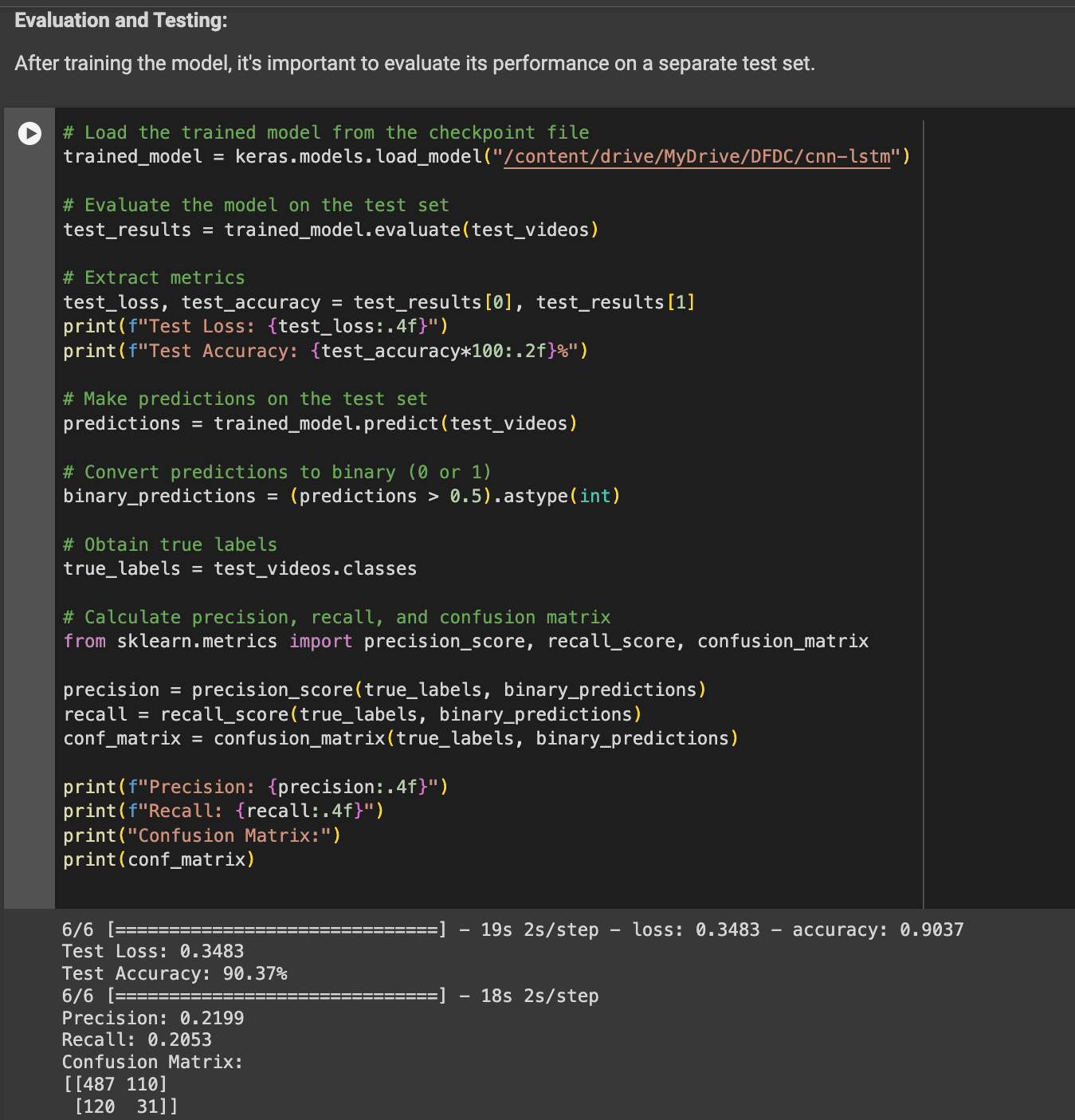
**6. Code Implementation**

**Applying InceptionResNet V2**: The commented part when we were using pretrained and frozen layers, the uncommented code is for pretrained model but the last 20 layers are frozen.  


**Model architecture**: we added the Dropout and L2 Regularization to the model.



**Evaluation:**



**Conclusion**

After Refinement we got almost 90.37% accuracy on test dataset while previously it was 83% on test data. The challenge I encountered was when I tried to unfreeze the last 40 layers of pretrained model, I was getting losses in minus and the model was not performing well and also the biggest issue was the Colab GPU limitation.

**References**

1. <https://www.kaggle.com/competitions/deepfake-detection-challenge/data>
2. <https://ceur-ws.org/Vol-3058/Paper-034.pdf>
3. OpenCV, Dlib, InceptionResNetv2