Neural Network Presentation

SIGN LANGUAGE RECOGNITION

Submitted by:

SRISTI MITRA
ROLL: 2K19/CO/389

ZISHNENDU SARKER
ROLL: 2K19/CO/450

Submitted to:

RHEA SACHDEVA

DELHI TECHNOLOGICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

Table Of Content

INTRODUCTION

Approximately 18.5 million people around the world are deaf or have a voice or language impairment. They find it difficult to communicate with those around them. We can only think how much they could have communicated if they had had an easy communication medium. Speech-impaired people typically utilize ASL (American Sign Language) to communicate and make gestures with their hands. But what if they have to coordinate with someone who isn't proficient in ASL or isn't familiar with it? With this in mind, we trained our own model that has live detection of hand gestures and converts them to English language in a much faster and accurate way as possible

Objective of Our Project

The ASL alphabets are recognised in this project using convolutional neural networks (CNN). The ASL Alphabet Classification challenge calls for a deep network, hence this approach is helpful for recognition. The first phase consists of pre-processing operations for the MNIST dataset. The pre-processed hand gesture picture is computed for a number of significant characteristics after the first phase. The accuracy and AUC Score of the network model's ability to recognise the sign language alphabet were discovered in the final phase, depending on the attributes estimated or calculated in the early phases. ASL alphabet recognition will also be accomplished in this project using an ANN (Artificial Neural Network). Likewise, compute accuracy and evaluate it against the CNN model. Finally, utilising YOLOV5, we will construct a further unique sign recognition detection model.

Motivation

This project was inspired by the stories of numerous people who have been marginalized in various sectors due to a flaw that was not their fault. There are no fields in which a person with a speech handicap cannot succeed. Just because they don't speak doesn't mean they don't have valuable insights that can improve the world. Let's not forget the 20 million people who have contributed to make the world a better place.

Project Description

01 Creating Dataset

In this project we use the photos of American Sign Language (ASL). We build the dataset with ASL and level them with yolov5 format, ANN, CNN.

02 Deep Learning Model

Convolutional neural networks, a type of artificial neural network used to analyze visual information, will be the foundation for the deep learning model. The model will be trained on a set of data and is expected to achieve an accuracy of at least 80%. The Python programming language will be used to create this model.

03 Visualization of the Project

The model we trained with CNN, ANN, YOLOV5 will be checked as a computer vision where we use a webcam to take the videos and after visualizations the model will provide the information about the sign detection.

04 Model Dataflow

Images used for training model. Using image processing to better visualize the feature if the image .Train the model and download the weight..Applying gaussian blur to the input image for better visualization. Using the weight for predicting the user’s sign language by the trained model

Project Requirements

# Base:-
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.2
Pillow>=7.1.2
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
torch>=1.7.0
torchvision>=0.8.1
tqdm>=4.41.0

# Logging :-
tensorboard>=2.4.1
# wandb

# Plotting :-
pandas>=1.1.4
seaborn>=0.11.0

# Export :-
#coremltools>=4.1 # CoreML export
#onnx>=1.9.0 # ONNX export
#onnx-simplifier>=0.3.6 # ONNX simplifier
#scikit-learn==0.19.2 #CoreML quantization

# Extras :-
albumentations>=1.0.3
# Cython # for pycocotools https://github.com/cocodataset/cocoapi/issues/172
# pycocotools>=2.0 # COCO mAP
# roboflow
thop # FLOPs computation

Tools and Technologies

Python	01	05	Google Colab
Keras	02	06	Jupyter Notebook
Maxpolling	03	07	CUDA
Kaggle	04	08	Roboflow

METHODOLOGY

01 Loading and Preprocessing Dataset

We started by importing the dataset to work upon and making it suitable to perform various algorithms to predict sign language alphabet labels correctly. In the given dataset, we already have separate training and tests. We visualized our dataset, and it came out to be balanced. We also store the labels of each image, which is mapped with their respective labels in a list, and generates a random image to perform class label verification. We performed a grayscale normalization to reduce the effect of illumination's differences

02 Data Augmentation

In Data augmentation from one image, we generate more images. Therefore, we can have a variety in our dataset, and it is also used to increase the size of the dataset to avoid overfitting. Our data augmentation includes random rotation ranging 10 degrees and random height and width shift in the range of 0.1, random zooming, and random flips. This was implemented using Keras image pre-processing module.

.

METHODOLOGY

03 Model Building

We build the model using the combination of convolution, max pooling, and flattening layers in Keras. We also used batch normalization and dropout layers to prevent the model from overfitting the data. We used a callback to determine if our validation set Accuracy did not increase even after 2 (patience level) consecutive epochs during the model fitting stage. Then, we will alter our learning rate by a factor of 0.5.

04 Independent variables and Dependent variables:

Independent variable- number of images which is 704, pixel distribution of images which is stored in NumPy array. Dependent variables are- feature map after each convolution layer output, the volume of feature map which is input to the next layer, probabilities of labels which are used for classification.

METHODOLOGY

05 Parameters and Hyper Parameters

Parameters like kernel size, astride of filter, padding, number of features/filters, filter size, activation function, a connection between two layers, pooling type, number of labels. Hyperparameters like coefficients in the learning rate, dropout rate, batch size, number of the hidden and dense layer in architecture, number of epochs.

06 Performance Measure

Performance measures are used to check the correctness of our model. The majors we are going to use are Accuracy on both training and validation dataset, loss in both validation and training dataset, Precision, Recall, and F1 measure of all 25 classes.

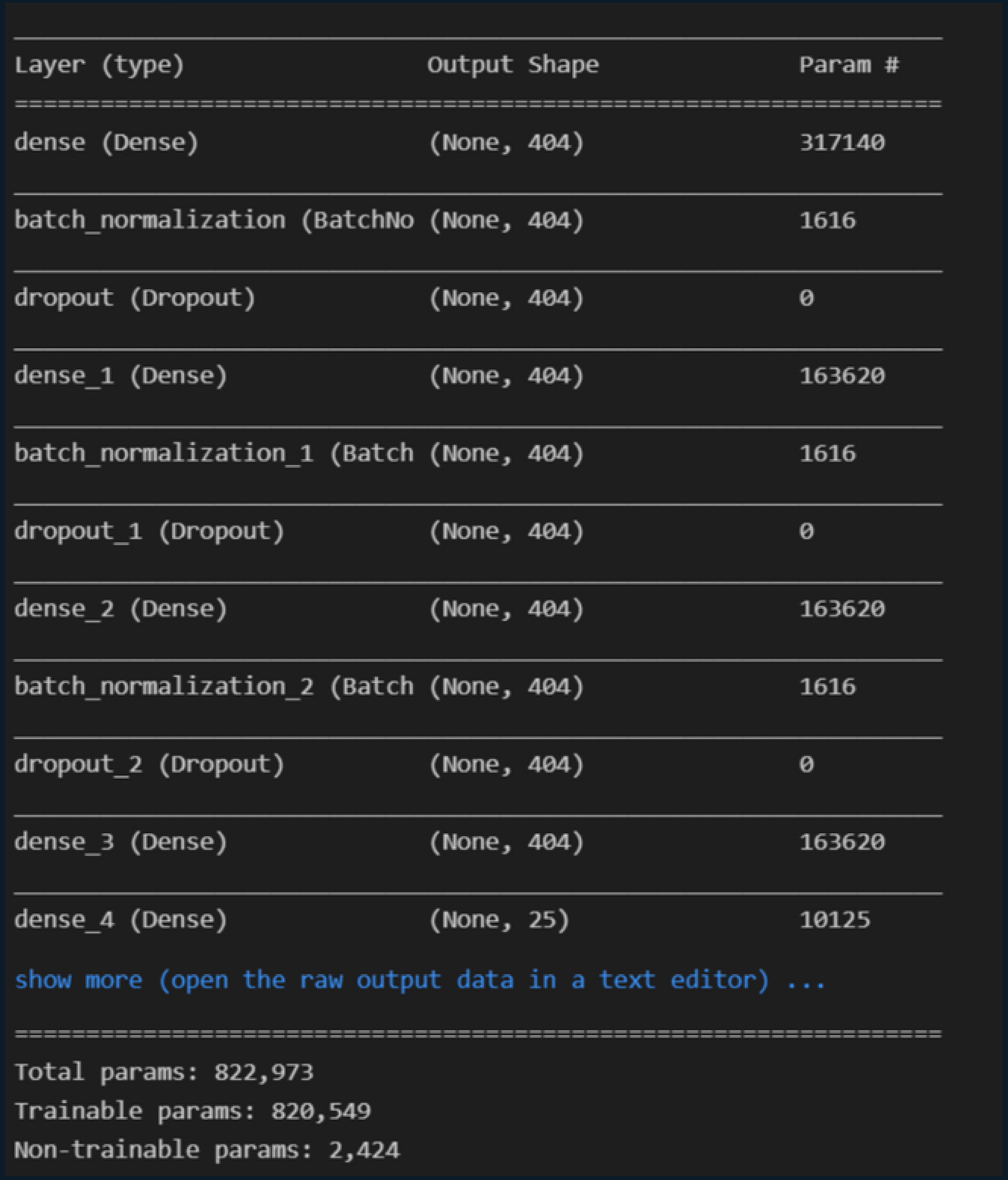
METHODOLOGY

07 Validation Method

We are using the Holdout validation technique to evaluate our model. We initially have a training and testing set. Our dataset contains 704 images. 20% of total images (144) are present in our testing set and the rest in the training se. We split our training set into two parts, one for training our model and another part which is a validation set for validating our model. Later we will analyze our model based on predictions predicted by our trained model on the test set and obtain its accuracy, precision, recall, F1 Score and AUC Score to see its performance.

ANN Model:

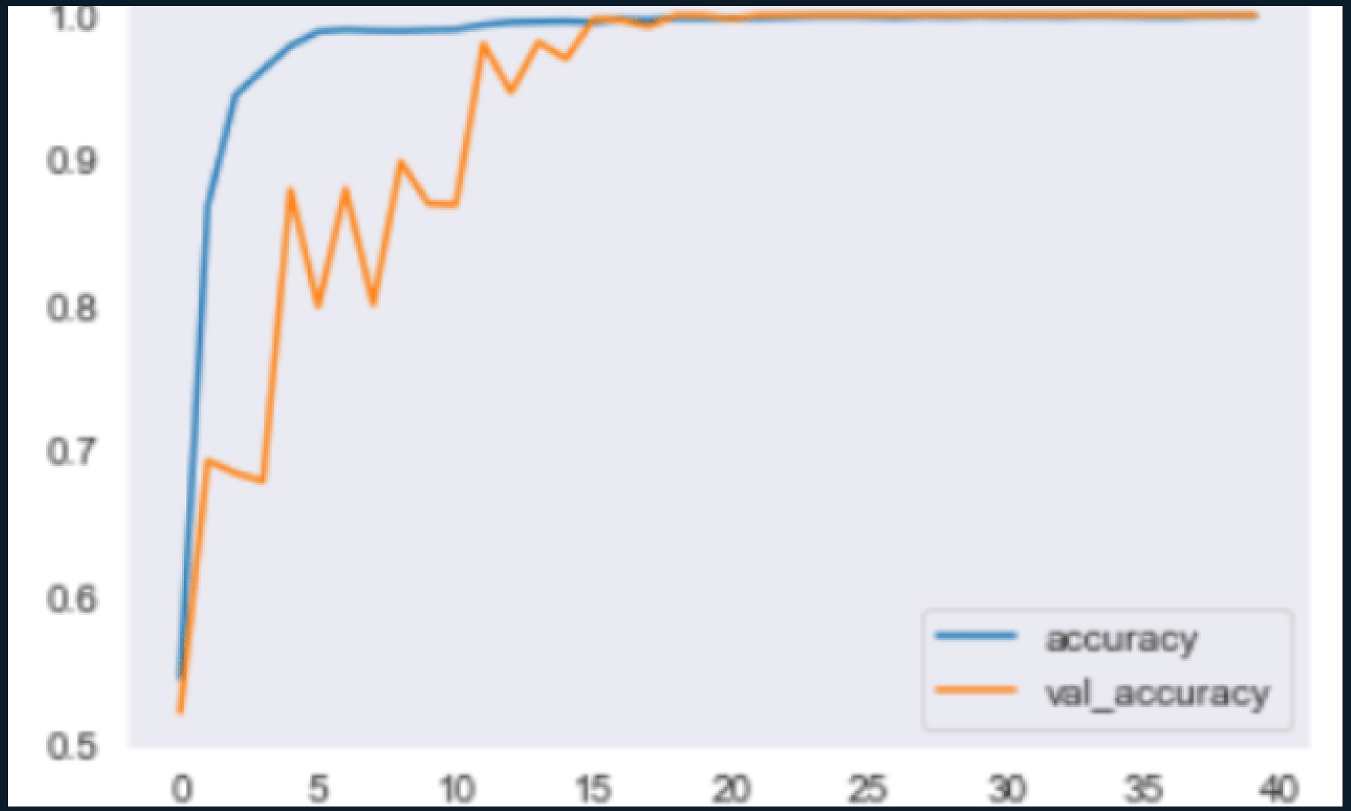
- We are using keras which is an open-source library to build our model. So, initially we imported keras and all the other dependencies.
- Then, we have initialized our model as sequential model. Input layer of our network expects rows of data with 784 variables.
- We have input dimension as 784 which is equal to the number of columns in the dataset .
- We have used four hidden layers each of which has 404 nodes and uses the Rectified Linear Unit (ReLU) Activation Function because of its fast computation capabilities.
- we have also added batch normalization between the layers for faster working of our model. Batch Normalization adds a few new layers in our neural network which normalizes the values corresponding to the input layer
- At the output layer , we have 25 nodes and we have used Softmax activation function .We have used sparse categorical cross-entropy because we have labelled them as targets
- we compile our model, and for minimization of cost function



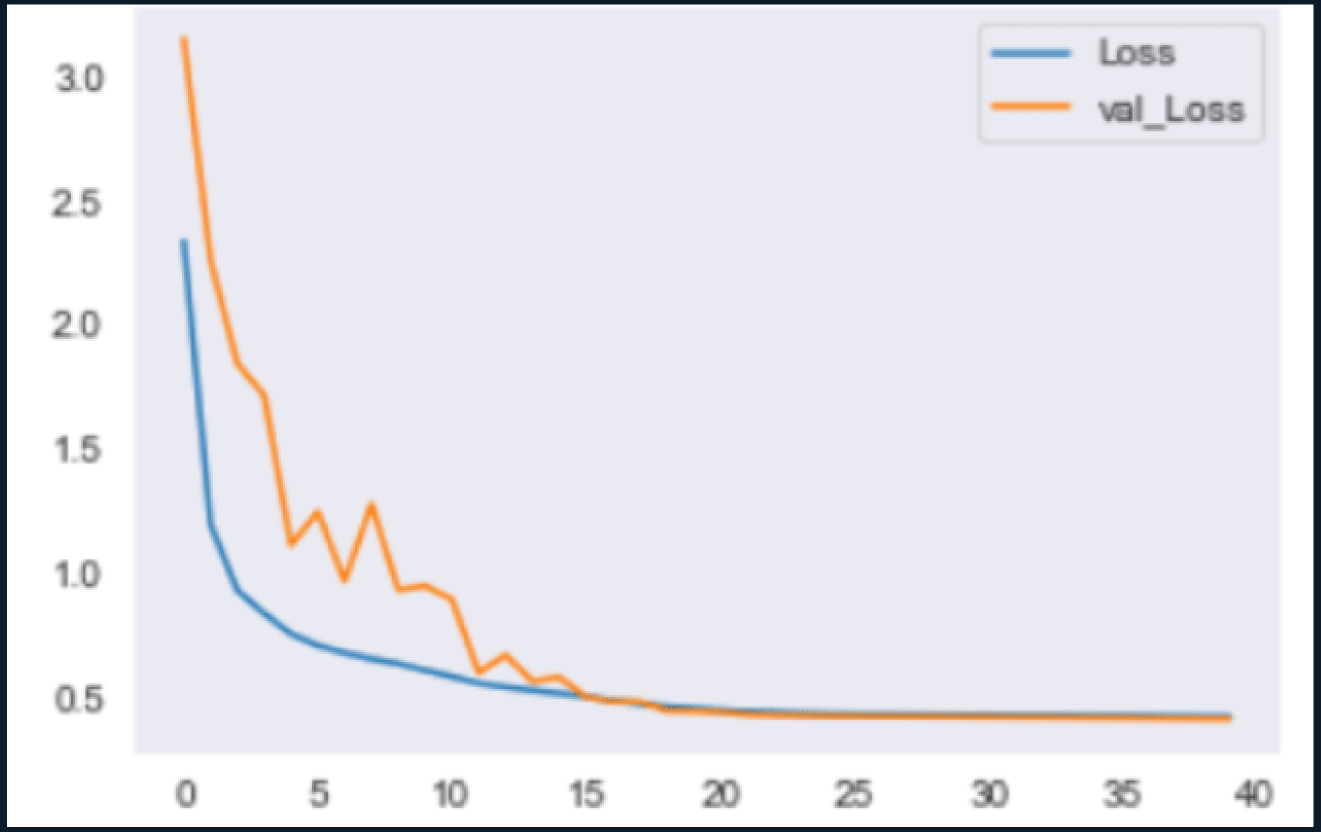
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 404)               317140
_________________________________________________________________
batch_normalization (BatchNo (None, 404)               1616
_________________________________________________________________
dropout (Dropout)            (None, 404)               0
_________________________________________________________________
dense_1 (Dense)              (None, 404)               163620
_________________________________________________________________
batch_normalization_1 (Batch (None, 404)               1616
_________________________________________________________________
dropout_1 (Dropout)          (None, 404)               0
_________________________________________________________________
dense_2 (Dense)              (None, 404)               163620
_________________________________________________________________
batch_normalization_2 (Batch (None, 404)               1616
_________________________________________________________________
dropout_2 (Dropout)          (None, 404)               0
_________________________________________________________________
dense_3 (Dense)              (None, 404)               163620
_________________________________________________________________
dense_4 (Dense)              (None, 25)                10125
show more (open the raw output data in a text editor) ...

=================================================================
Total params: 822,973
Trainable params: 820,549
Non-trainable params: 2,424
```

Reasult of ANN:

	Accuraacy	**Precision**	**Recall**	**AUC Score**	**F1 Score**
Result	0.8353	0.83	0.82	0.9142	0.83

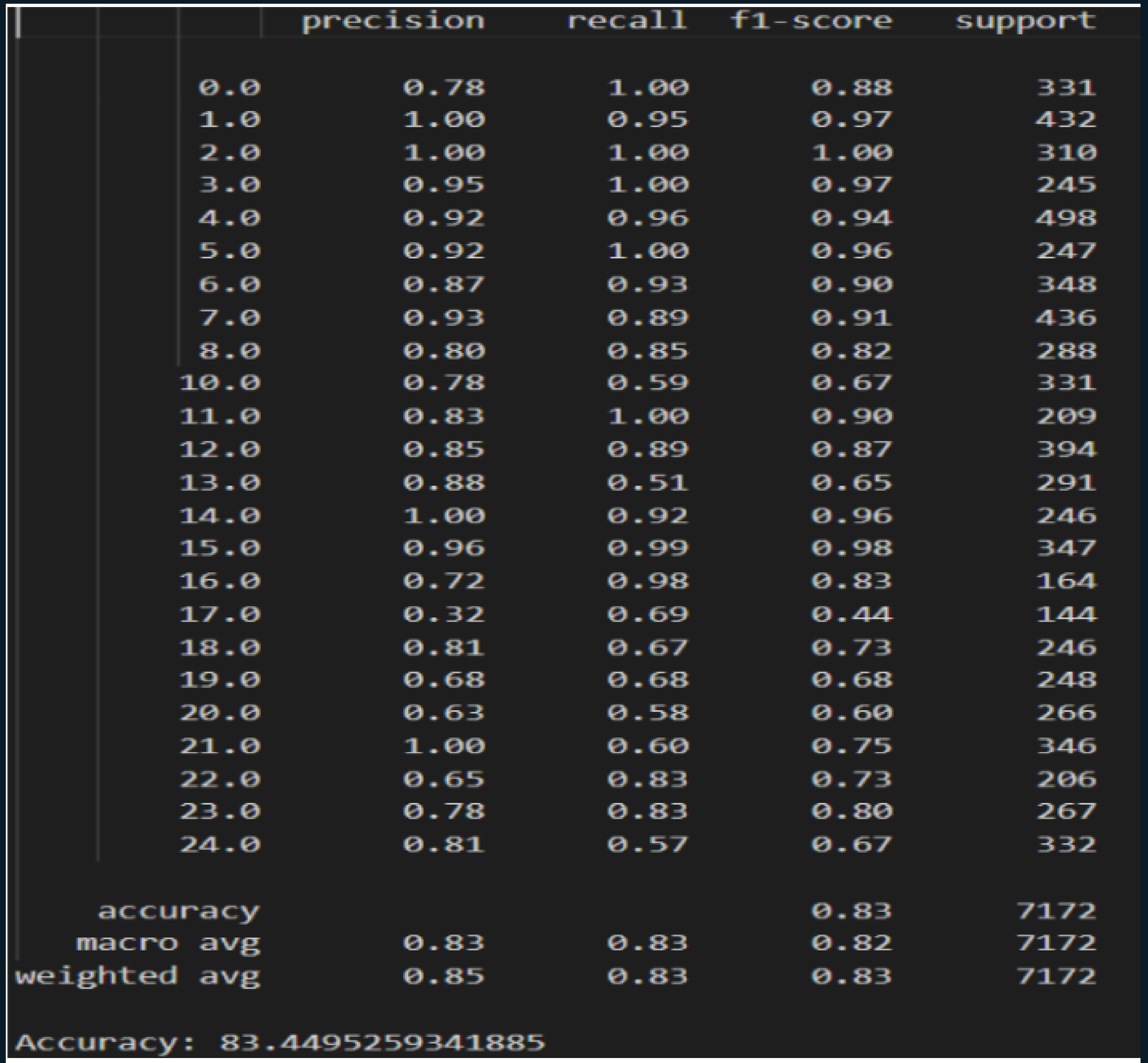


Accuracy evaluation



Loss Evaluation

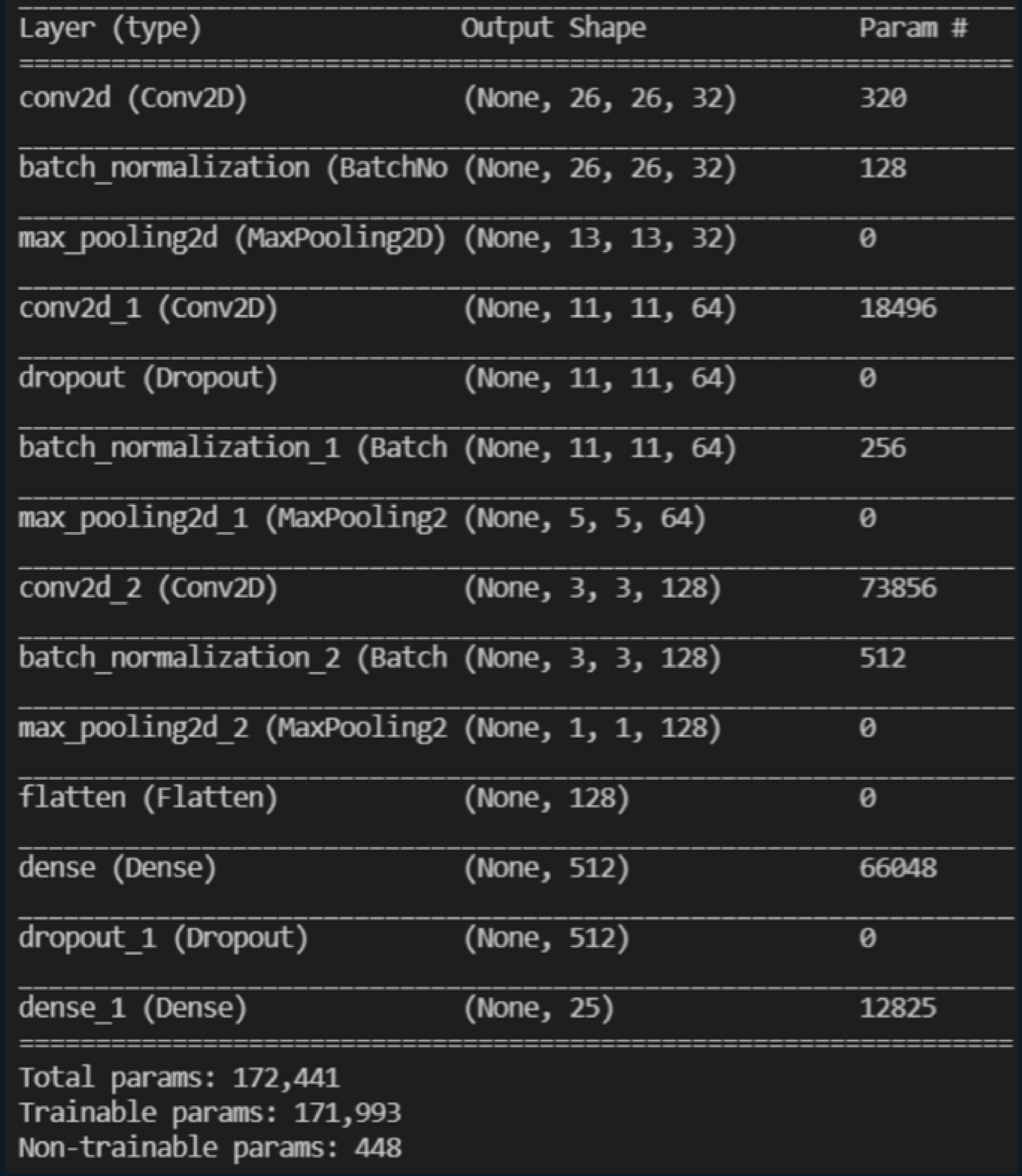
Classification Report



```
              precision    recall  f1-score   support

         0.0       0.78      1.00      0.88       331
         1.0       1.00      0.95      0.97       432
         2.0       1.00      1.00      1.00       310
         3.0       0.95      1.00      0.97       245
         4.0       0.92      0.96      0.94       498
         5.0       0.92      1.00      0.96       247
         6.0       0.87      0.93      0.90       348
         7.0       0.93      0.89      0.91       436
         8.0       0.80      0.85      0.82       288
        10.0       0.78      0.59      0.67       331
        11.0       0.83      1.00      0.90       209
        12.0       0.85      0.89      0.87       394
        13.0       0.88      0.51      0.65       291
        14.0       1.00      0.92      0.96       246
        15.0       0.96      0.99      0.98       347
        16.0       0.72      0.98      0.83       164
        17.0       0.32      0.69      0.44       144
        18.0       0.81      0.67      0.73       246
        19.0       0.68      0.68      0.68       248
        20.0       0.63      0.58      0.60       266
        21.0       1.00      0.60      0.75       346
        22.0       0.65      0.83      0.73       206
        23.0       0.78      0.83      0.80       267
        24.0       0.81      0.57      0.67       332

    accuracy                           0.83      7172
   macro avg       0.83      0.83      0.82      7172
weighted avg       0.85      0.83      0.83      7172

Accuracy: 83.4495259341885
```

CNN Model:

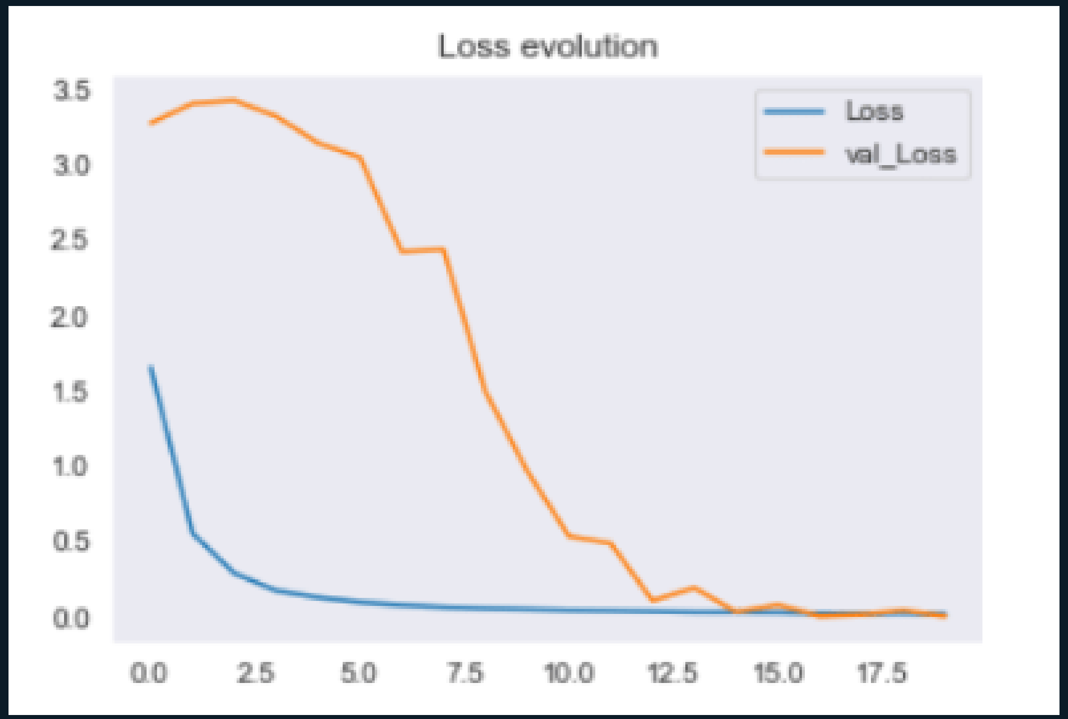
- The first step towards building a model for our work is to initialize the model. Since our model is sequential,we initialized it as sequential.
- The next step is to create our first layer of CNN to perform convolution operations. We created the input layer of our CNN. We have taken 32 feature detectors of 3x3. The layer takes input into a batch of images. In our case, the batch size is 32. The shape of our image for the input layer is (28,28,1), where the height and width of the image are 28, and there is one channel as we grayscaled our images.
- After the convolution operation, the ReLU activation function is applied to increase non-linearity in our CNN model. If the value is greater than 0, it passes the input; otherwise, it returns 0.
- Now we have performed Batch Normalization to keep the output of the layer of the same range to get an unbiased result. After this, the output of our first convolution layer of volume 26x26x32 is passed for the max-pooling operation to the next layer.
- the output volume will be 13x13x32. Now we have a second layer, which consists of 1 convolution 2d layer and one max pooling. The input of this layer is the output of the previous layer, which is 13x13x32. The kernel size is 3x3, with 64 different feature detectors. So, the output volume is 11x11x64.



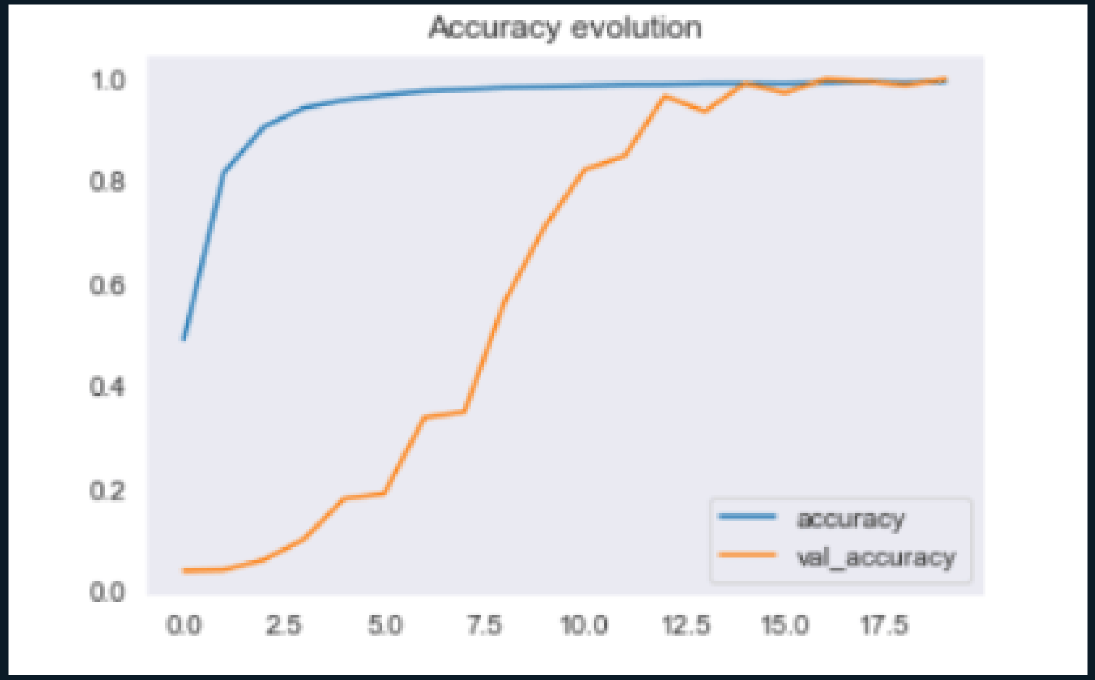
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
batch_normalization (BatchNo (None, 26, 26, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
dropout (Dropout)            (None, 11, 11, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 11, 11, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 3, 3, 128)         512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 512)               66048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 25)                12825
=================================================================
Total params: 172,441
Trainable params: 171,993
Non-trainable params: 448
```

Result of CNN:

	Accuraacy	Precision	Recall	F1 Score	AUC Score
Result	0.9963	0.99	1	0.9981	1.00

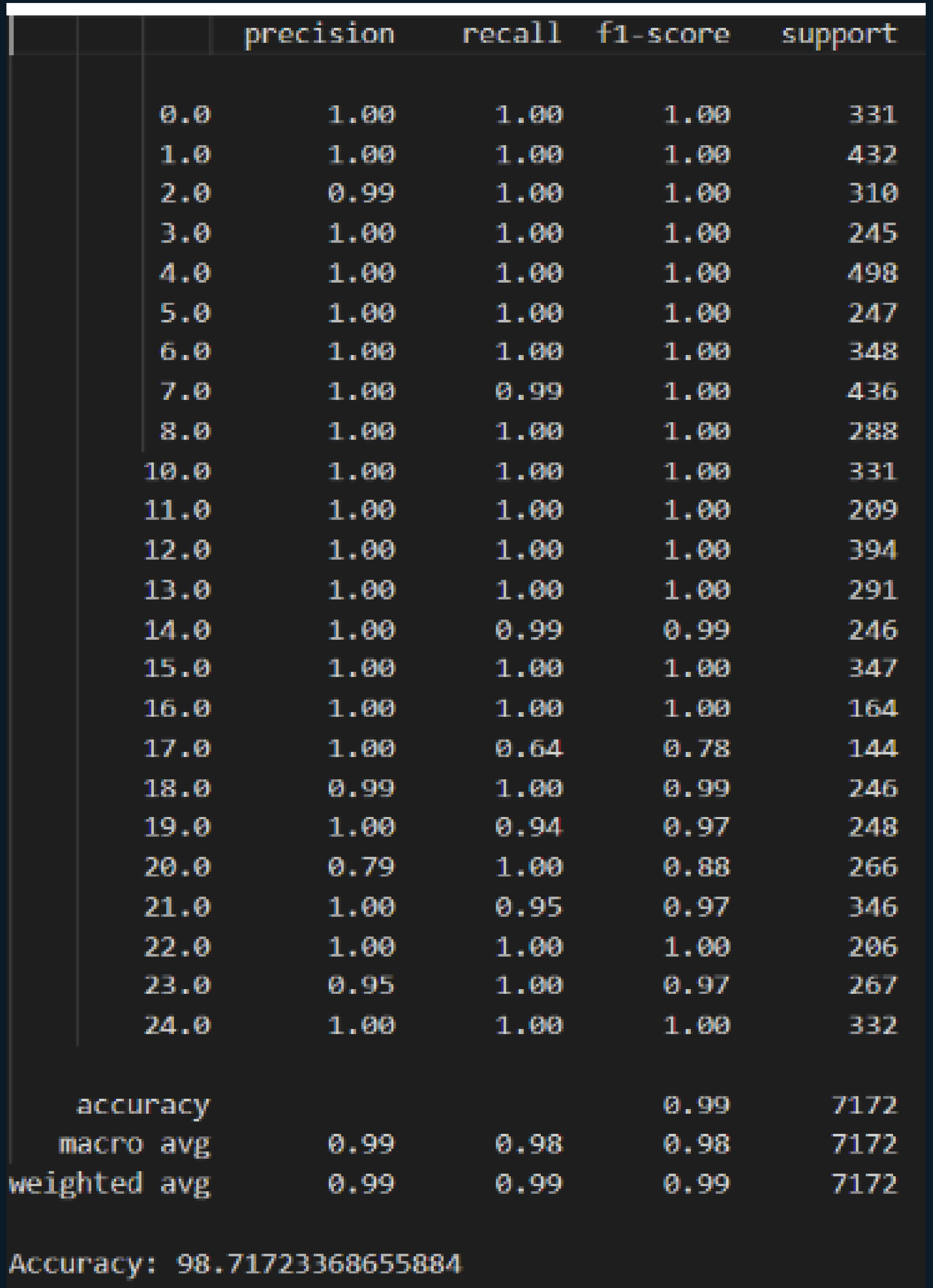


Loss Evalution



Accuracy Evalution

Classification report



```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       331
         1.0       1.00      1.00      1.00       432
         2.0       0.99      1.00      1.00       310
         3.0       1.00      1.00      1.00       245
         4.0       1.00      1.00      1.00       498
         5.0       1.00      1.00      1.00       247
         6.0       1.00      1.00      1.00       348
         7.0       1.00      0.99      1.00       436
         8.0       1.00      1.00      1.00       288
        10.0       1.00      1.00      1.00       331
        11.0       1.00      1.00      1.00       209
        12.0       1.00      1.00      1.00       394
        13.0       1.00      1.00      1.00       291
        14.0       1.00      0.99      0.99       246
        15.0       1.00      1.00      1.00       347
        16.0       1.00      1.00      1.00       164
        17.0       1.00      0.64      0.78       144
        18.0       0.99      1.00      0.99       246
        19.0       1.00      0.94      0.97       248
        20.0       0.79      1.00      0.88       266
        21.0       1.00      0.95      0.97       346
        22.0       1.00      1.00      1.00       206
        23.0       0.95      1.00      0.97       267
        24.0       1.00      1.00      1.00       332

    accuracy                           0.99      7172
   macro avg       0.99      0.98      0.98      7172
weighted avg       0.99      0.99      0.99      7172

Accuracy: 98.71723368655884
```

YOLOV5:

The object detection method YOLO, which stands for "You Only Look Once," separates images into a grid structure. Each grid cell is in charge of detecting items within itself. Because of its speed and precision, YOLO is one of the most well-known object recognition algorithms.

REQUIREMENTS:

We will need Python >= 3.8 and PIP in order to follow this guide. The rest of the requirements are listed in './requirements.txt'

INSTALLATION:

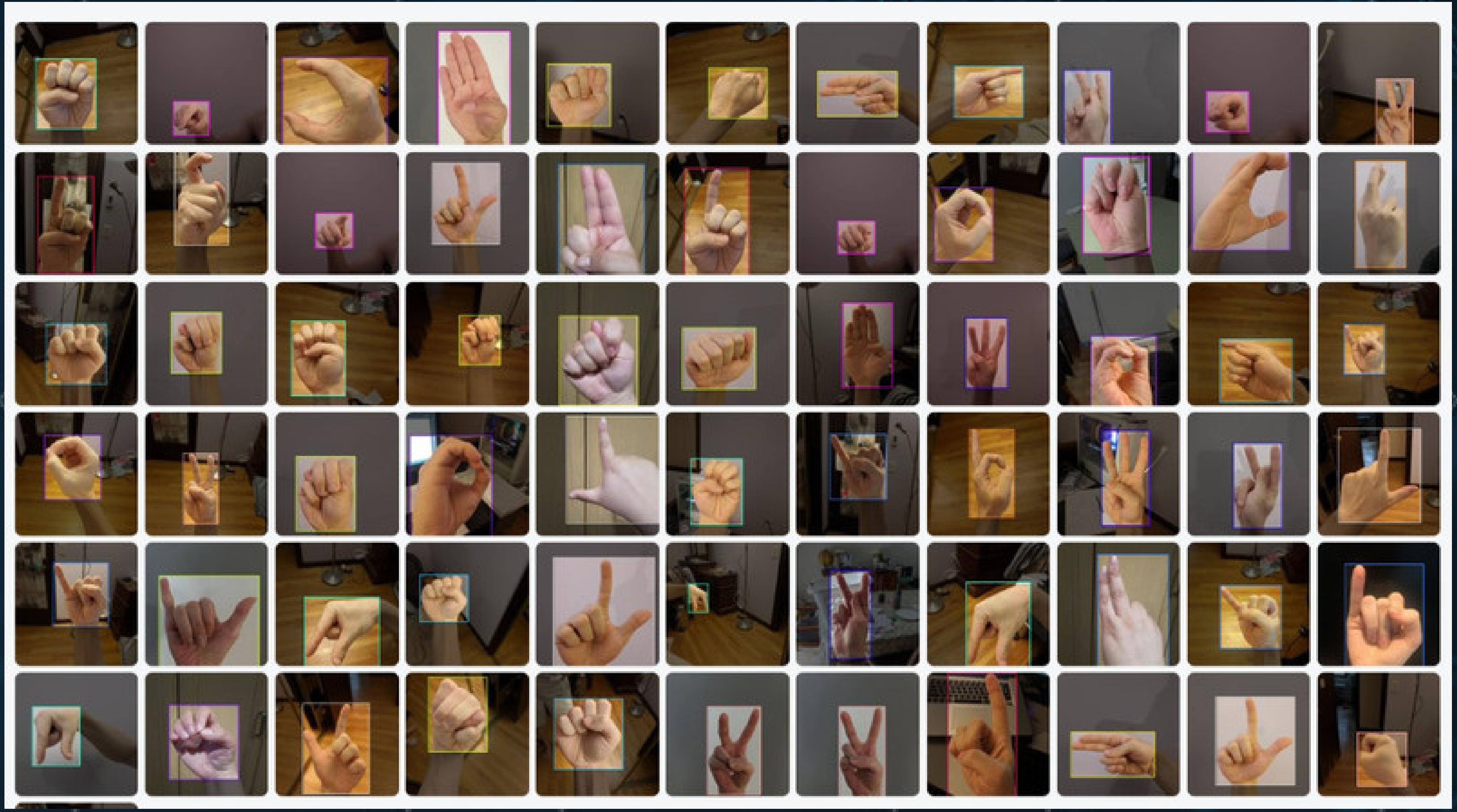
We first have to clone the repository then enter the root directory and lastly install the required packages from your cloned repository root directory.

PACKAGED ENVIRONMENT

A quick and hassle free setup YOLOv5 has been packaged with all dependencies* for the following environments, *including CUDA/CUDNN, Python and PyTorch. Google Colab and Kaggle notebooks with free GPU, Amazon Deep Learning AMI. Docker Image.

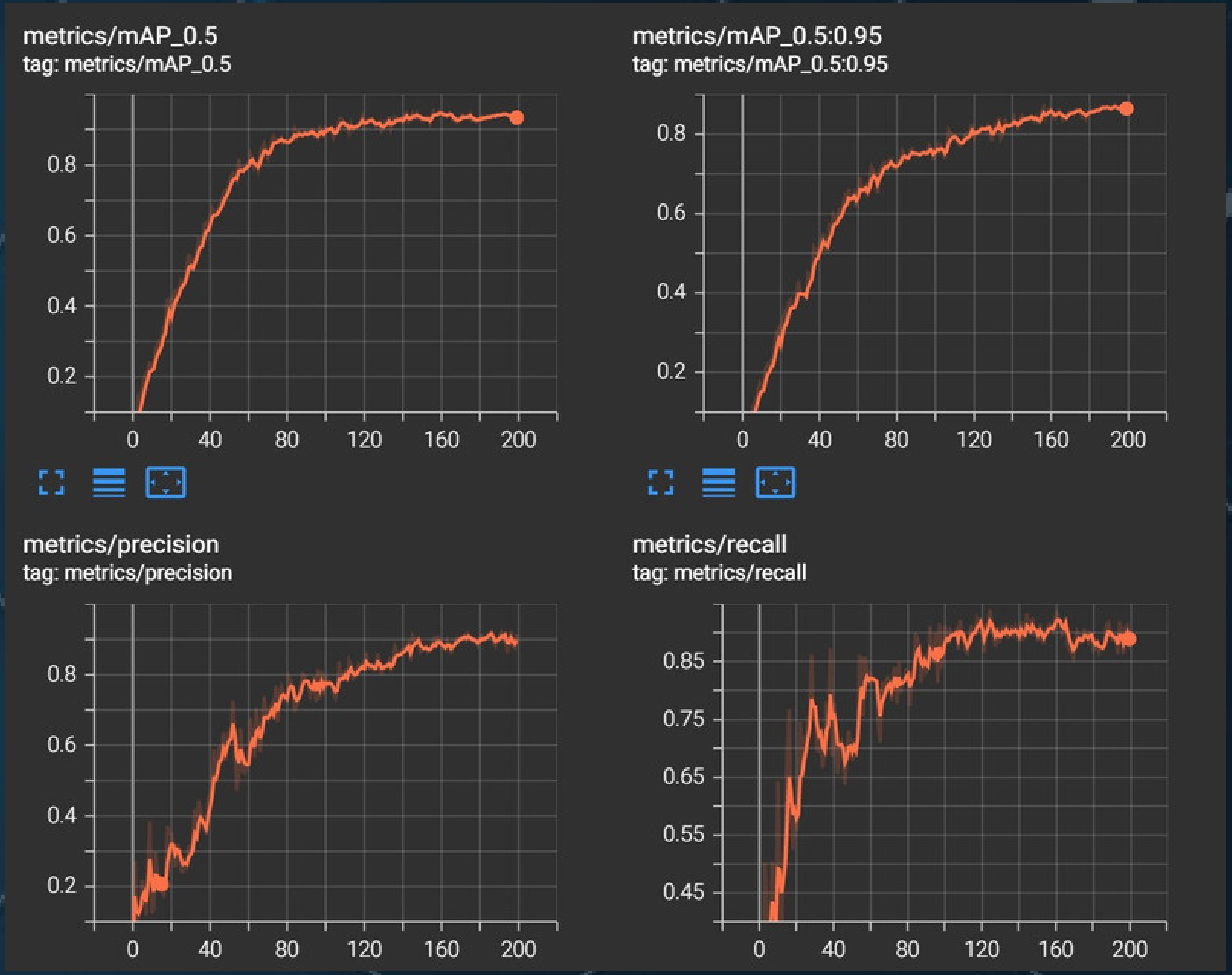
IMPLEMENTATION

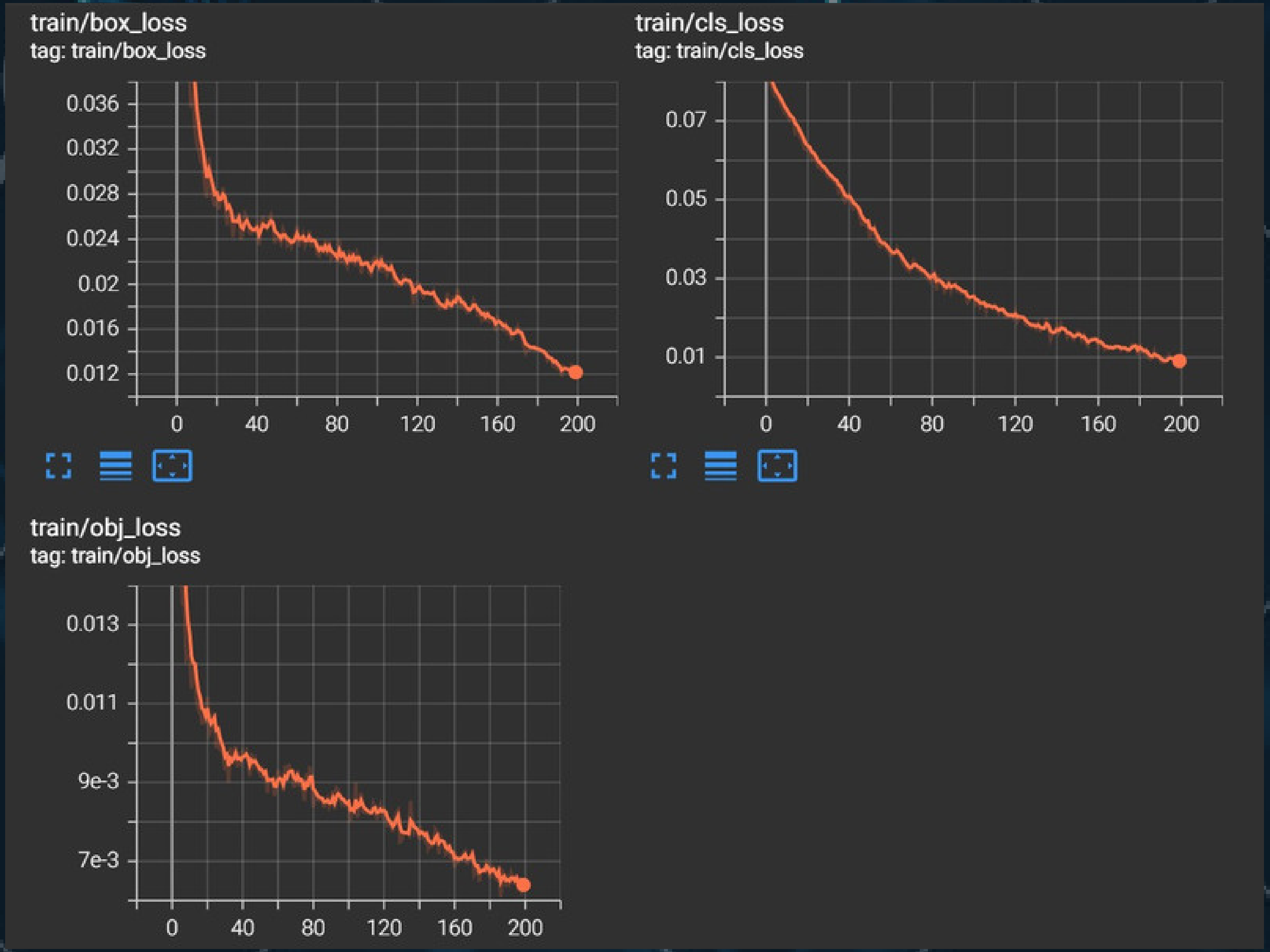
First, we take the dataset of American Sign Language (ASL). and we take the picture from there and we labeled the pictures for train the dataset . We use the yolov5 format for that, it's a faster algorithm method to detect sign language.

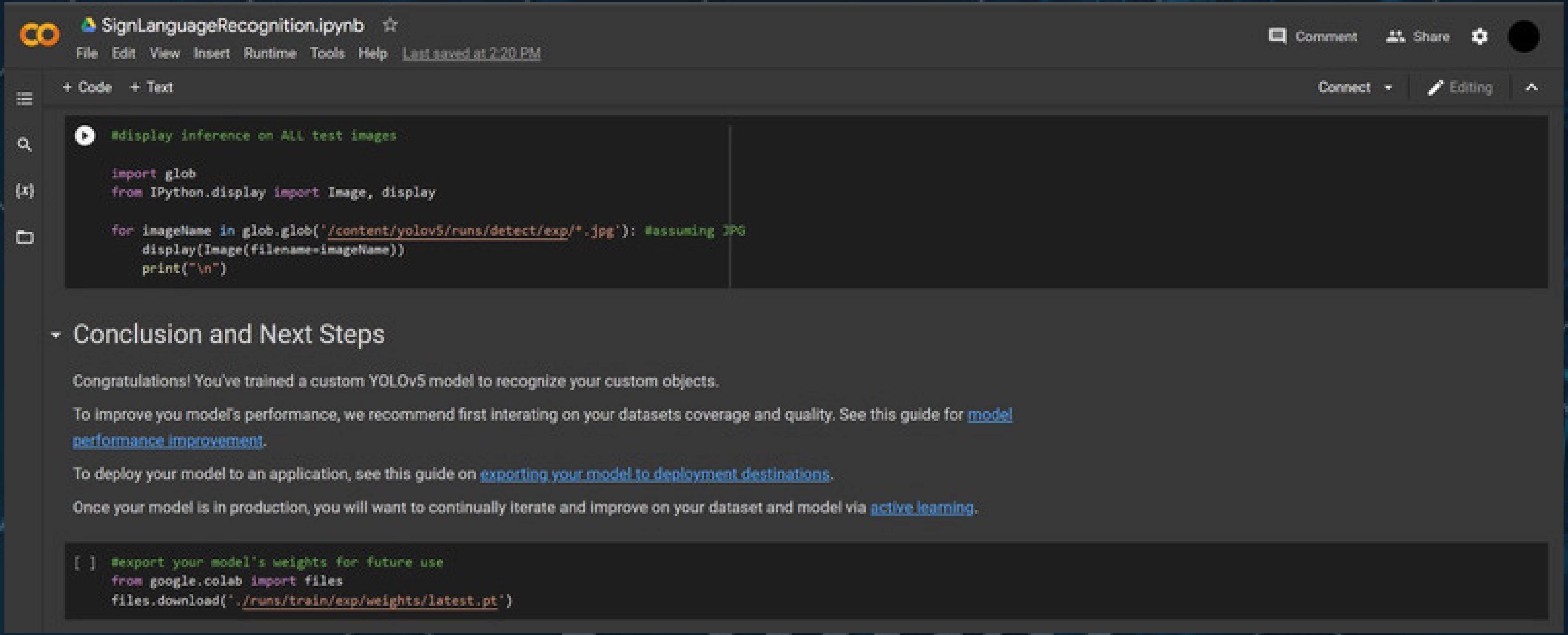


RESULT

TThe graphs are given below comes from the tensorboard from our model . As we can see that, here the model has 4 graphs metrics where they are presenting the mAP_0.5, mAP_0.5 to 0.95, precision and recall for 200 epochs. Where we can see that , as the number of epochs reaches 200 the value we get is near 1 and that means that our model gives more precise values here.
And also we can see that, when we recall the datas it also fluctuates between .90 to 1 .






SignLanguageRecognition.ipynb
File Edit View Insert Runtime Tools Help Last saved at 2:20 PM
Comment Share
+ Code + Text
Connect Editing

```
#display inference on ALL test images

import glob
from IPython.display import Image, display

for imageName in glob.glob('/content/yolov5/runs/detect/exp/*.jpg'): #assuming JPG
    display(Image(filename=imageName))
    print("\n")
```

Conclusion and Next Steps

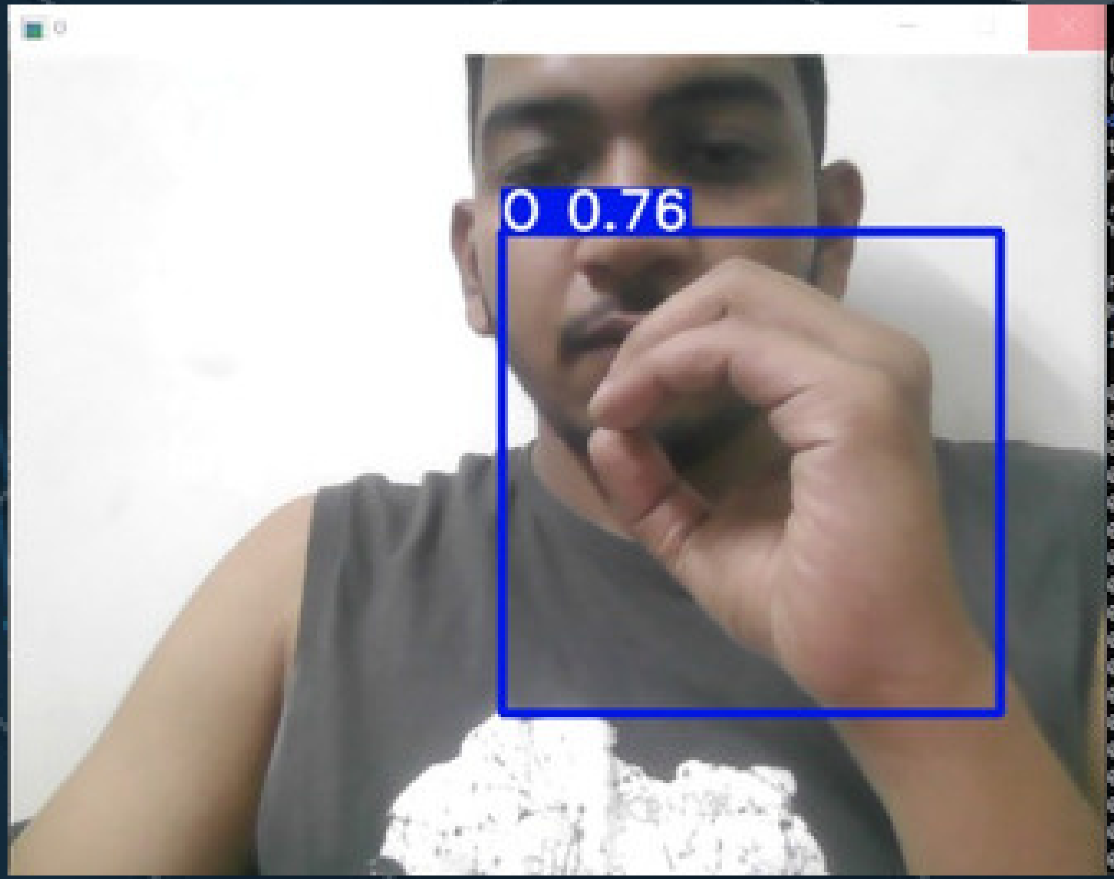
Congratulations! You've trained a custom YOLOv5 model to recognize your custom objects.

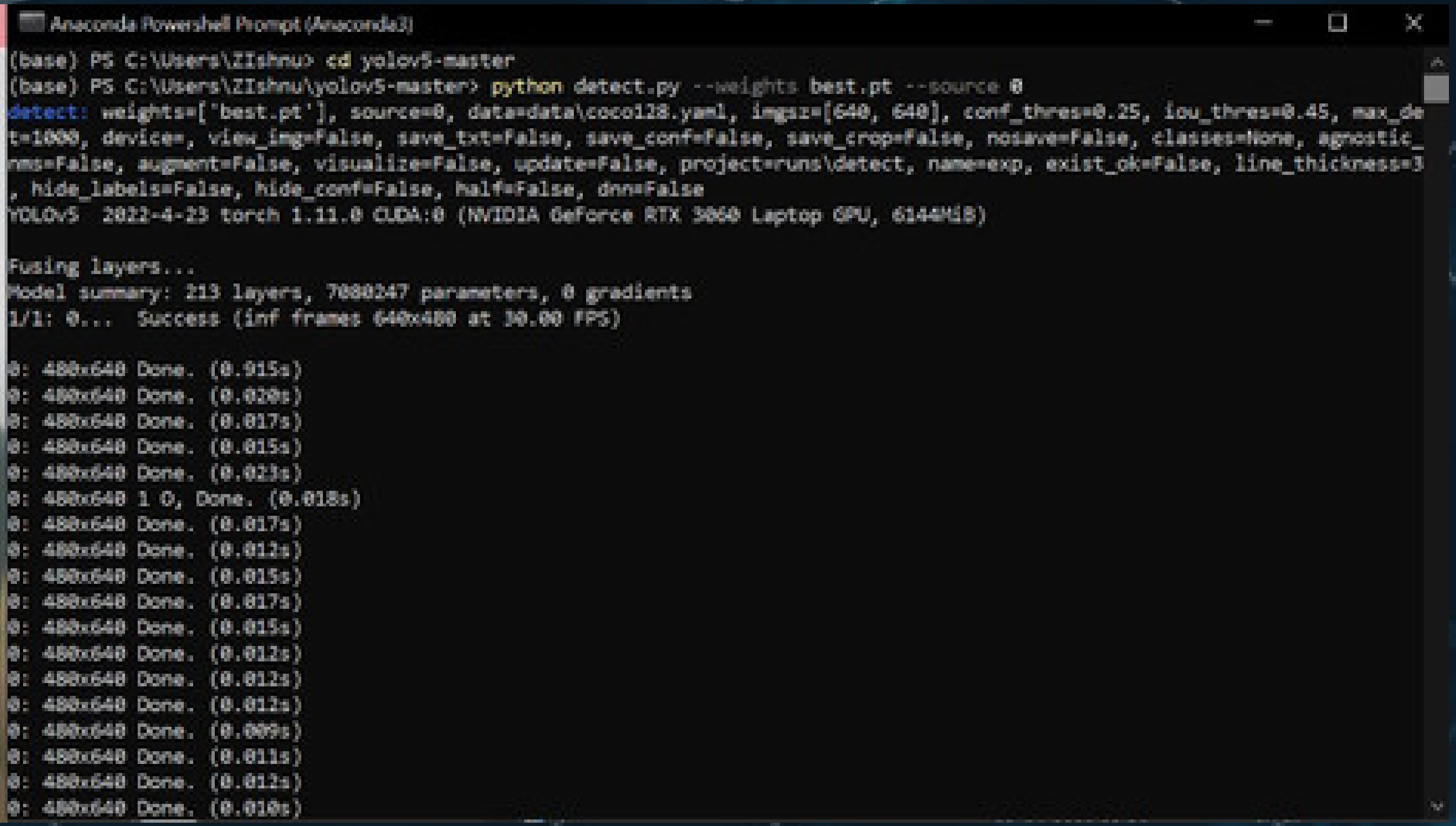
To improve you model's performance, we recommend first interating on your datasets coverage and quality. See this guide for model performance improvement.

To deploy your model to an application, see this guide on exporting your model to deployment destinations.

Once your model is in production, you will want to continually iterate and improve on your dataset and model via active learning.

```
[ ] #export your model's weights for future use
    from google.colab import files
    files.download('./runs/train/exp/weights/latest.pt')
```

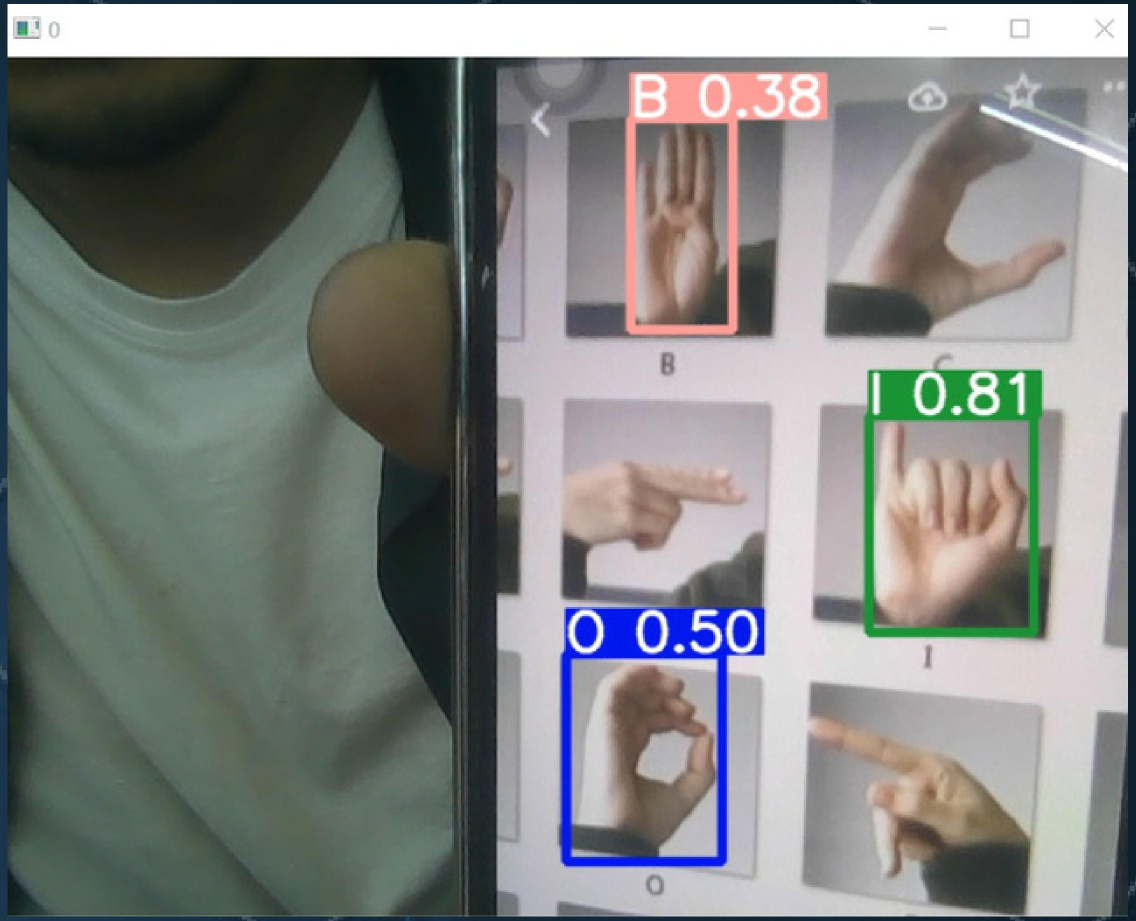


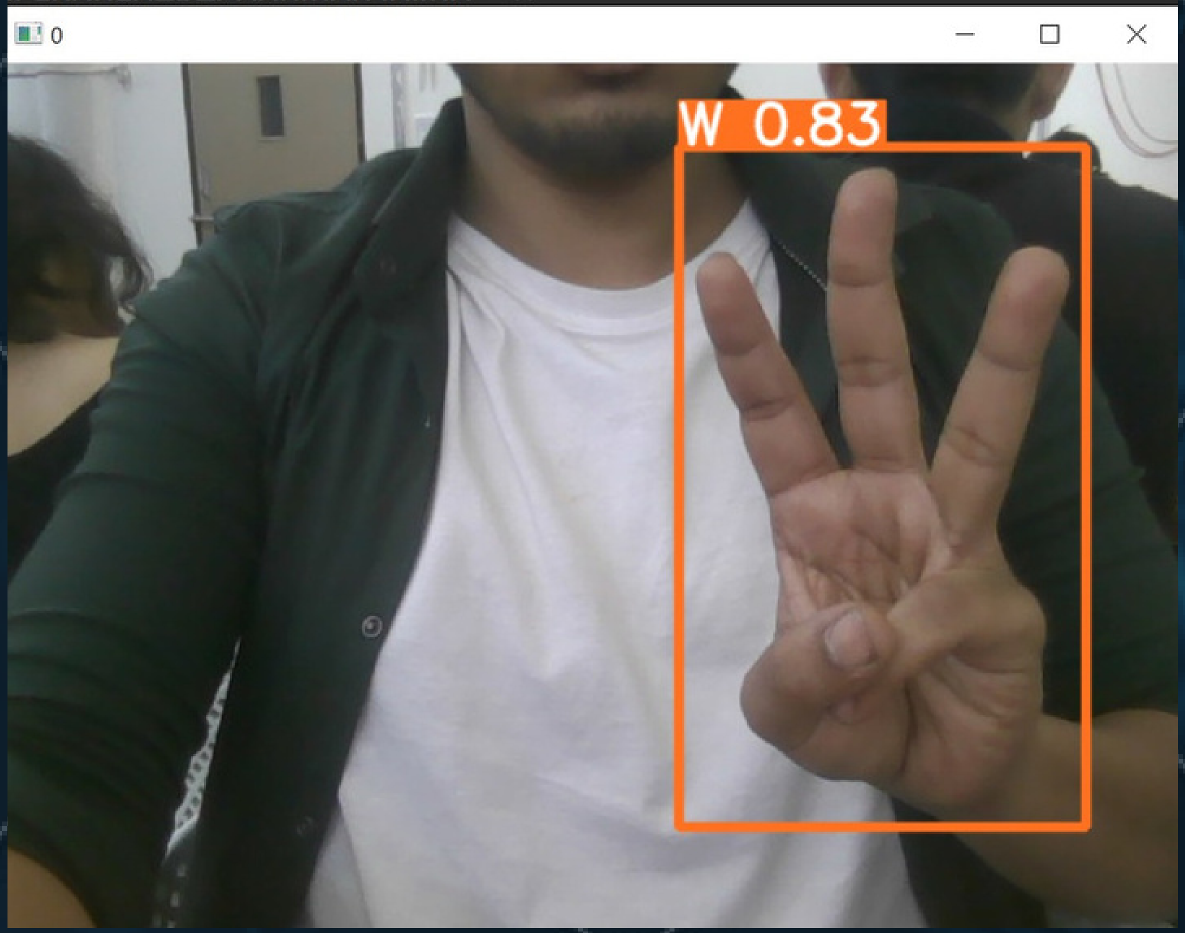


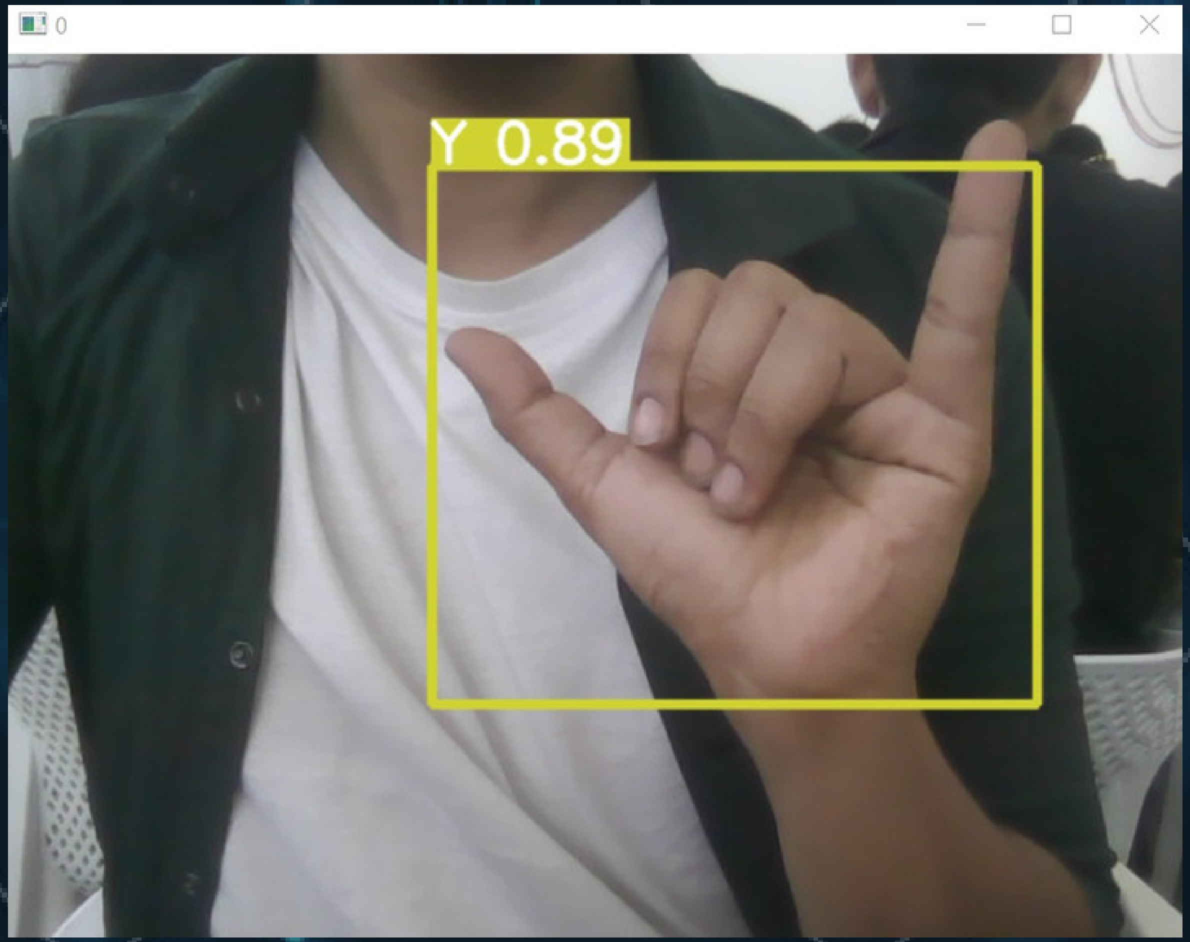
```
(base) PS C:\Users\ZIshnu> cd yolov5-master
(base) PS C:\Users\ZIshnu\yolov5-master> python detect.py --weights best.pt --source 0
detect: weights=['best.pt'], source=0, data=data\coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45, max_de
t=1000, device=, view_img=False, save_txt=False, save_conf=False, save_crop=False, nosave=False, classes=None, agnostic_
nms=False, augment=False, visualize=False, update=False, project=runs\detect, name=exp, exist_ok=False, line_thickness=3
, hide_labels=False, hide_conf=False, half=False, dnn=False
YOLOv5  2022-4-23 torch 1.11.0 CUDA:0 (NVIDIA GeForce RTX 3060 Laptop GPU, 6144MiB)

Fusing layers...
Model summary: 213 layers, 7080247 parameters, 0 gradients
1/1: 0...  Success (inf frames 640x480 at 30.00 FPS)

0: 480x640 Done. (0.915s)
0: 480x640 Done. (0.020s)
0: 480x640 Done. (0.017s)
0: 480x640 Done. (0.015s)
0: 480x640 Done. (0.023s)
0: 480x640 1 O, Done. (0.018s)
0: 480x640 Done. (0.017s)
0: 480x640 Done. (0.012s)
0: 480x640 Done. (0.015s)
0: 480x640 Done. (0.017s)
0: 480x640 Done. (0.015s)
0: 480x640 Done. (0.012s)
0: 480x640 Done. (0.012s)
0: 480x640 Done. (0.012s)
0: 480x640 Done. (0.009s)
0: 480x640 Done. (0.011s)
0: 480x640 Done. (0.012s)
0: 480x640 Done. (0.010s)
```

RESULT

Below there we can see, our model took almost 7 hours to train the model weight for 200 epochs where can see all the classes for the alphabets. We took 144 valid image set for every alphabet and levels them. Also the below graph is showing the precision values, recalling values and also the values of mAP 0.5 and mAP_0.5 to 0.95 . Were are getting almost 90 percent precision values on average and also more than 92 % accuracy in average from the model







CONCLUSION AND FUTURE SCOPE

If you have a deaf or hard-of-hearing friend, sign language is the only way to communicate with them effectively. People all over the world are starting to use this type of sign language to communicate effectively with the deaf these days. It is possibly the most effective way of communication for the deaf. Learning American sign language translation, on the other hand, may be difficult due to the fact that it is a motor rather than a cognitive skill. The goal of our project is to create a sign language for text translators to solve this problem.

For the future perspective, we can also try to measure accuracy of this model in TensorFlow and compare the accuracy for both the models. In this way, we allow a new field to boarden up as then the accuracy of the sign language recognition will be effective

REFERENCES

[1] S. Y. Kim, H. G. Han, J. W. Kim, S. Lee, and T. W. Kim, “A hand gesture recognition sensor using reflected impulses,” IEEE Sens. J., vol. 17, no. 10, 2975–2976,2017,doi: 10.1109/JSEN.2017.2679220.

[2] Y. Cui and J. Weng, “A learning-based prediction-and-verification segmentation scheme for hand sign image sequence,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 21, no. 8, pp. 798–804, 1999, doi: 10.1109/34.784311.

[3] A. Kumar and A. Kumar, “Dog Breed Classifier for Facial Recognition using Convolutional Neural Networks,” pp. 508–513, 2020.

[4] K. L. Bouman, G. Abdollahian, M. Boutin, and E. J. Delp, “A low complexity sign detection and text localization method for mobile applications,” IEEE Trans. Multimed., vol. 13, no. 5, pp. 922–934, 2011, doi: 10.1109/TMM.2011.2154317.

[5] J. Wu, L. Sun, and R. Jafari, “A Wearable System for Recognizing American Sign Language in Real-Time Using IMU and Surface EMG Sensors,” IEEE J. Biomed. Heal. Informatics, vol. 20, no. 5, pp. 1281–1290, 2016, doi: 10.1109/JBHI.2016.2598302.

[6] C. Zhang, W. Ding, G. Peng, F. Fu, and W. Wang, “Street View Text Recognition With Deep Learning for Urban Scene Understanding in Intelligent Transportation Systems,” IEEE Trans. Intell. Transp. Syst., pp. 1–17, 2020, doi: 10.1109/tits.2020.3017632.

Thank You!

ANY QUESTIONS?