AI SIGN LANGUAGE RECOGNITION

Name: **Anshuman Pratik**

Student ID: **STU62cef4f77a0981657730295**

AICTE Email: **anshuman.pratik2019@vitstudent.ac.in**

Institution: **Vellore Institute of Technology, Vellore**

Abstract

Speech impaired people use hand signs and gestures to communicate. The Problem is translator needs every time to translate sign language. Artificial Intelligence capture hand expression with the help of camera show us what a person speaks, and result shown in the text format. Computer recognition of sign language deals from sign gesture acquisition and continues till text/speech generation. Sign gestures can be classified as static and dynamic. The main obstacles that have prevented more research on American Sign Language from being done in this body of work are incorporated features and local language variance. To communicate with them, sign language should be learned. Peer groups are typically where learning happens. There aren't many study resources accessible for learning signs. The process of learning sign language is therefore a very challenging undertaking. Finger spelling is the first stage of sign learning, and it is also used when there is no corresponding sign or when the signer is unaware of it. Majority of the currently available sign language learning systems rely on pricey external sensors. However static gesture recognition is simpler than dynamic gesture recognition but both recognition systems are important to the human community. The sign language recognition steps are described in this survey. The data acquisition, data pre-processing and transformation, feature extraction, classification and results obtained are examined.

Keywords: Artificial Intelligence, Convolutional Neural Networks, Machine Learning, Sign Recognition, and Image Processing algorithms

Introduction

Overview

Artificial intelligence might be used to translate the signs made by signers in video feeds into spoken language. This could make it easier for sign language users to communicate with their workplace and friends. An Image processing mechanism would combine natural language processing with computer vision, but the first step would be to precisely identify and categorise sign language signs using computer vision. In this notebook, a convolutional neural network will be used to categorise a dataset of American Sign Language alphabet hand signs.

About Data Sets

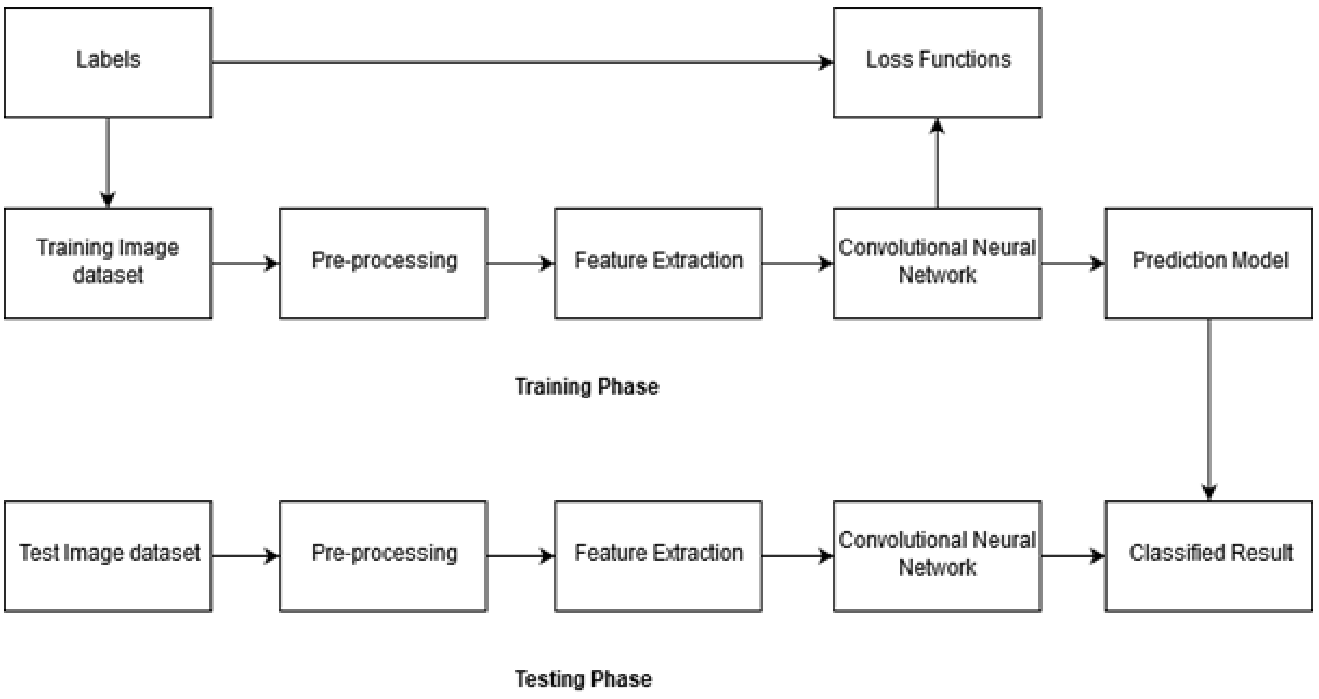
The American Sign Language letter database of hand gestures represent a multi-class problem with 24 classes of letters (excluding J and Z which require motion). The dataset format is patterned to match closely with the classic MNIST. Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (and no cases for 9=J or 25=Z because of gesture motions).

The training data (27,455 cases) and test data (7172 cases) are approximately half the size of the standard MNIST but otherwise similar with a header row of label, pixel1,pixel2….pixel784 which represent a single 28x28 pixel image with grayscale values between 0-255. The original hand gesture image data represented multiple users repeating the gesture against different backgrounds. In this notebook we will be applying a convolutional neural network to classify a dataset of American Sign Language alphabet images. The Sign Language MNIST data came from greatly extending the small number (1704) of the color images included as not cropped around the hand region of interest. To create new data, an image pipeline was used based on ImageMagick and included cropping to hands-only, gray-scaling, resizing, and then creating at least 50+ variations to enlarge the quantity.

Problem Statement

To communicate with other deaf and dumb persons or with the general public, it is most frequently utilised by these individuals. Given a hand gesture, implementing such an application that detects sign language in real time through hand gestures and allowing the user to get the accurate results of the character detected in a live video take as well as allowing such users to build their own customised gesture so that the issues faced by people who are unable to talk vocally can be accommodated with technological assistance and the barrier of expression can be overcome. This could make it easier for sign language users to communicate with their workplace and friends.

Model Diagram



‘Input (Training Data)- ML Algorithms (Testing Data) – Output’

An individual image is imported and processed from the disc for use by the trained model in categorising gestures and pre-processes it by resizing and scaling it, adding filters. The picture data that had been processed was divided into training, validation, and testing data during training and stored. A load dataset.py script that loads the pertinent data divided into a Dataset class is also used during training.

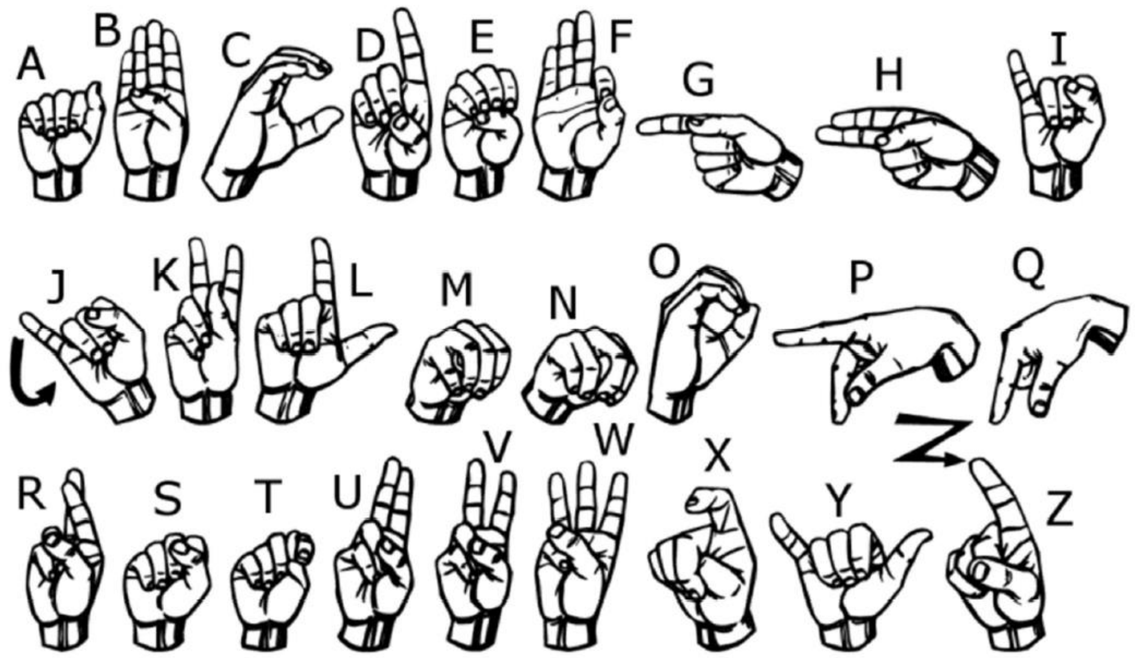
Methodology and Solutions

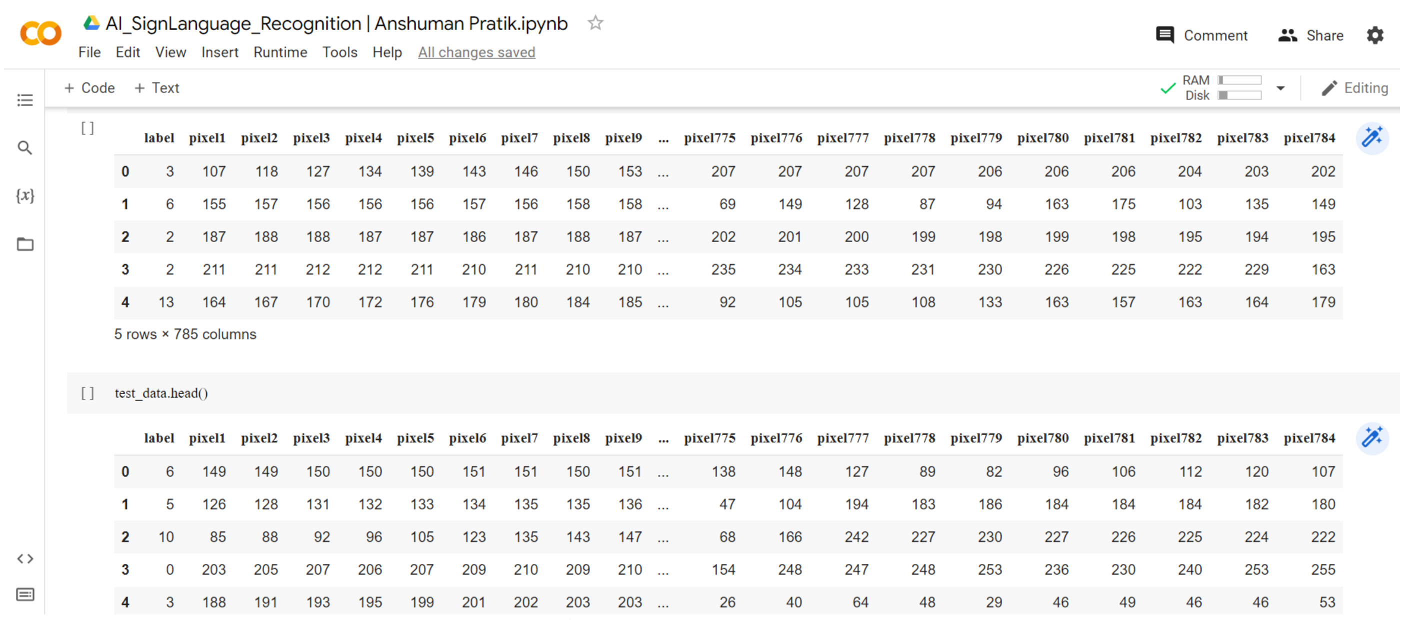
The learning rate, batch size, image filtering, and number of epochs are listed in a configuration file that contains the hyperparameters used to train the model. The architecture of the model and the configuration used to train it are both stored, allowing for further evaluation and optimization of the model's performance. Once trained, a model may be used to categorise new ASL gestures that are stored as files on the filesystem. The file path for the gesture image and the test data are entered by the user. In order to load and pre-process the file in the same manner as the model has been trained, the script will pass a representation of system behaviour, and indication of performance and design constrain, appropriate validate criteria, and other information pertinent to requirements. As the project requires classification of images, a CNN is the go-to architecture. The basis for our model design came from Using Deep Convolutional Networks for Gesture Recognition in Sign Language paper that accomplished a similar Sign Language Gesture Classification task . This model consisted of convolutional blocks containing two 2D Convolutional Layers with ReLU activation, followed by Max Pooling and Dropout layers. These convolutional blocks are repeated 3 times and followed by Fully Connected layers that eventually classify into the required categories. The kernel sizes are maintained at 3 X 3 in the process model. NumPy fully supports an object-oriented approach, starting, once again, with ndarray. For example, ndarray is a class, possessing numerous methods and attributes. Many of its methods are mirrored by functions in the outer-most NumPy namespace, allowing the programmer to code in whichever paradigm they prefer.

Technologies

- AI Algorithms
- MNSIT Datasets
- Image Processing
- CNN Model
- Python
- Google Collaboratory

Result



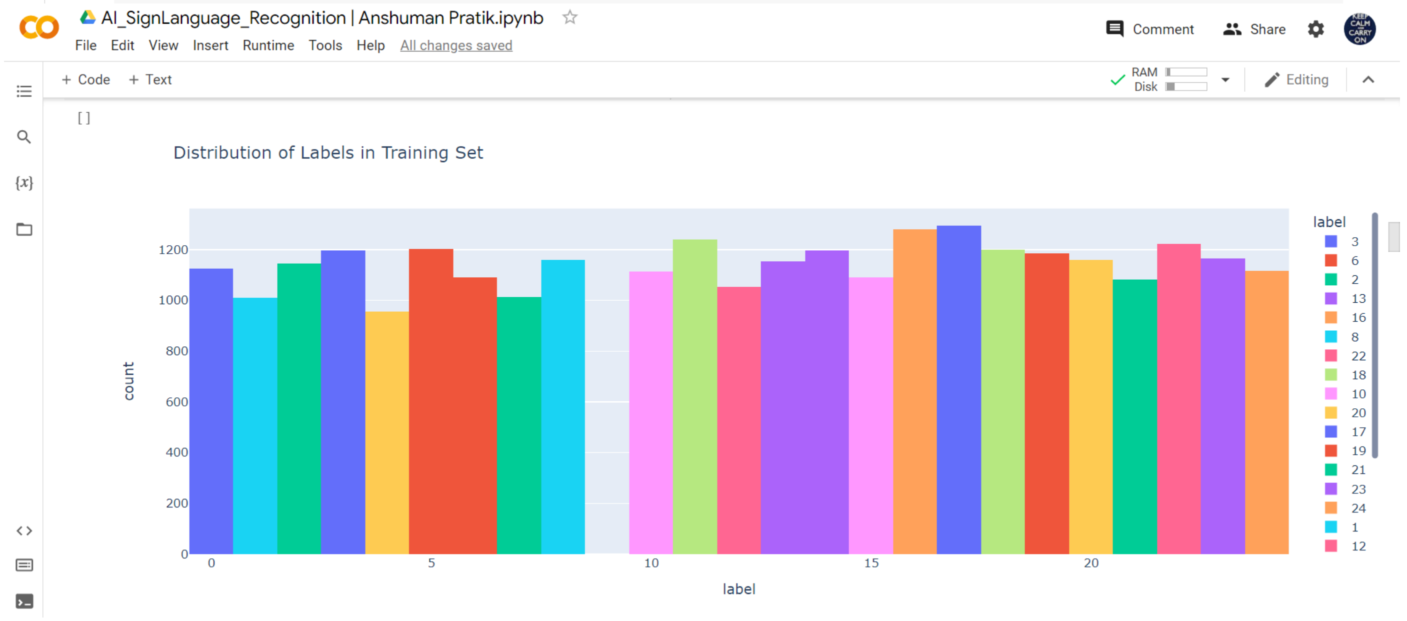

AI_SignLanguage_Recognition | Anshuman Pratik.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Editing

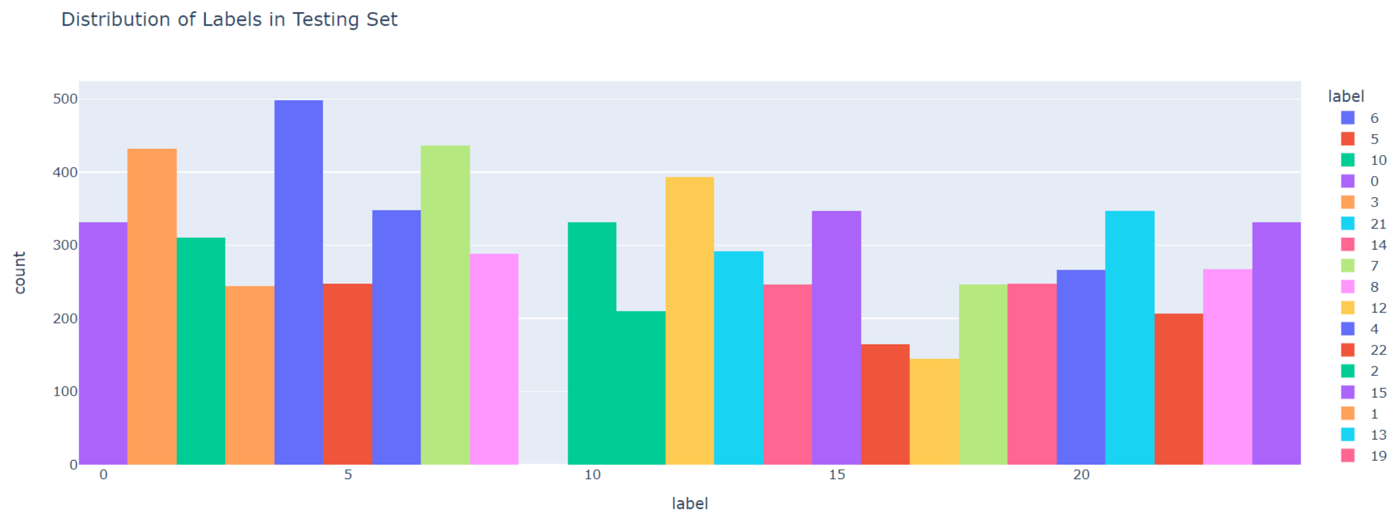
```
   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  pixel9  ...  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  pixel781  pixel782  pixel783  pixel784
0      3     107     118     127     134     139     143     146     150     153  ...       207       207       207       207       206       206       206       204       203       202
1      6     155     157     156     156     156     157     156     158     158  ...        69       149       128        87        94       163       175       103       135       149
2      2     187     188     188     187     187     186     187     188     187  ...       202       201       200       199       198       199       198       195       194       195
3      2     211     211     212     212     211     210     211     210     210  ...       235       234       233       231       230       226       225       222       229       163
4     13     164     167     170     172     176     179     180     184     185  ...        92       105       105       108       133       163       157       163       164       179

5 rows × 785 columns
```

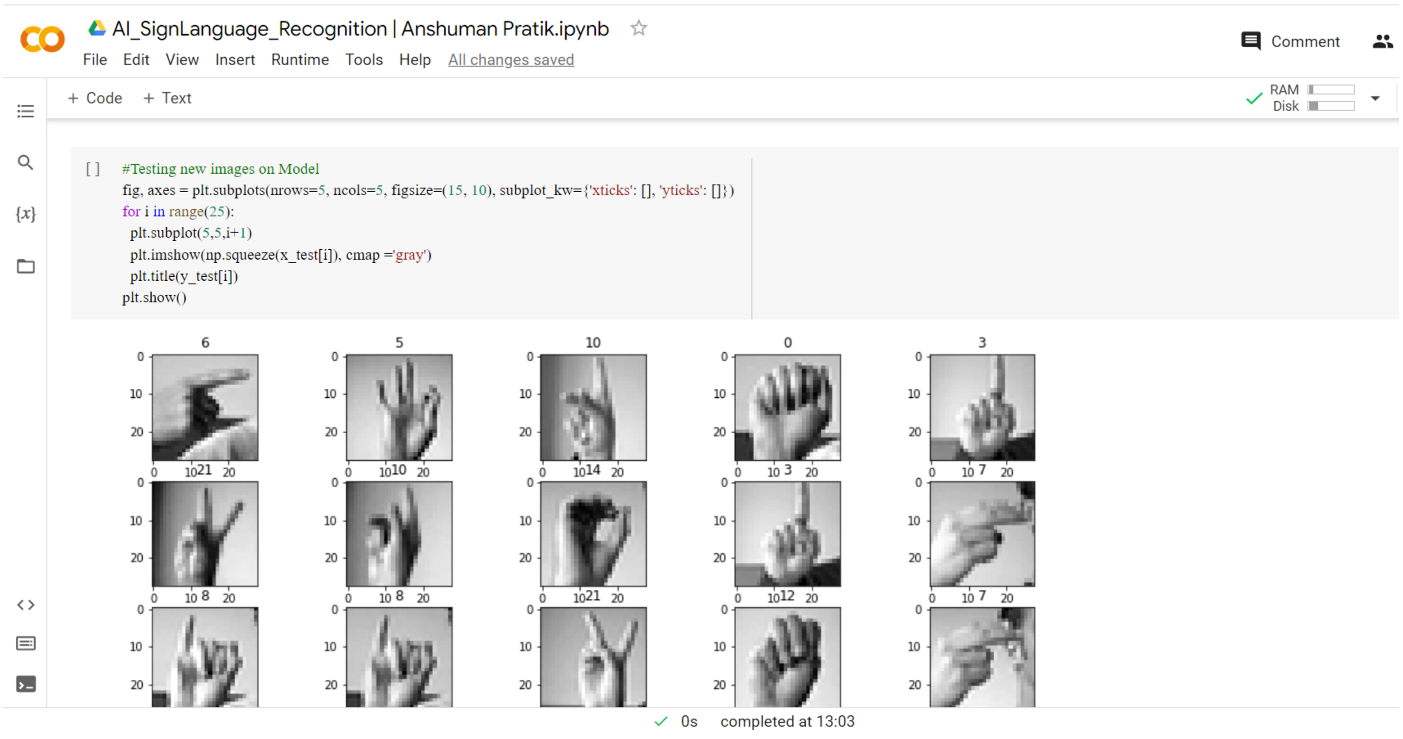
```
test_data.head()
```

```
   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  pixel9  ...  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  pixel781  pixel782  pixel783  pixel784
0      6     149     149     150     150     150     151     151     150     151  ...       138       148       127        89        82        96       106       112       120       107
1      5     126     128     131     132     133     134     135     135     136  ...        47       104       194       183       186       184       184       184       182       180
2     10      85      88      92      96     105     123     135     143     147  ...        68       166       242       227       230       227       226       225       224       222
3      0     203     205     207     206     207     209     210     209     210  ...       154       248       247       248       253       236       230       240       253       255
4      3     188     191     193     195     199     201     202     203     203  ...        26        40        64        48        29        46        49        46        46        53
```

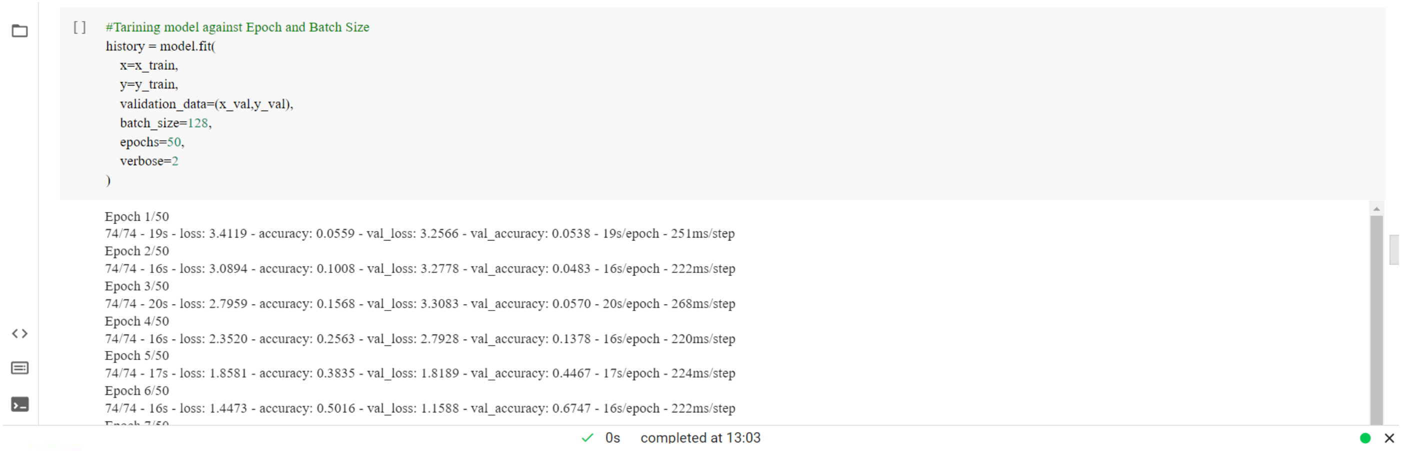




The images were processed and the accuracy of about 98% was obtained after running the batch processes and batch processes the image processing results of all the sign languages such as A to Z work returning accuracy between **95 to 98%.**

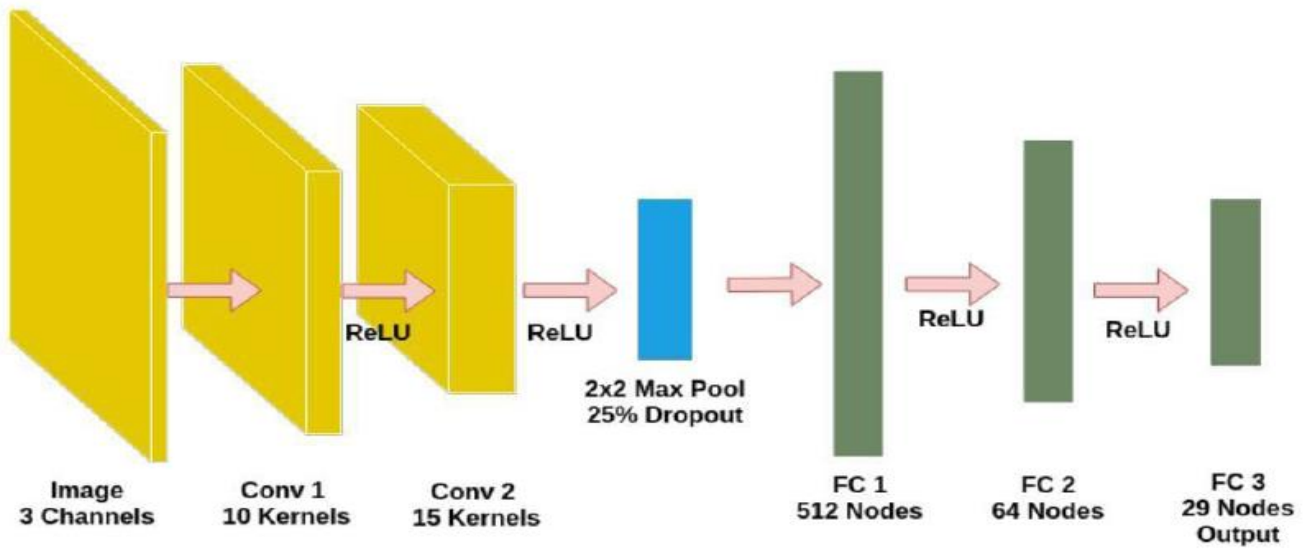

AI_SignLanguage_Recognition | Anshuman Pratik.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
+ Code + Text
RAM
Disk
6
5
10
0
3
21
10
14
3
7
8
8
21
12
7
✓ 0s completed at 13:03

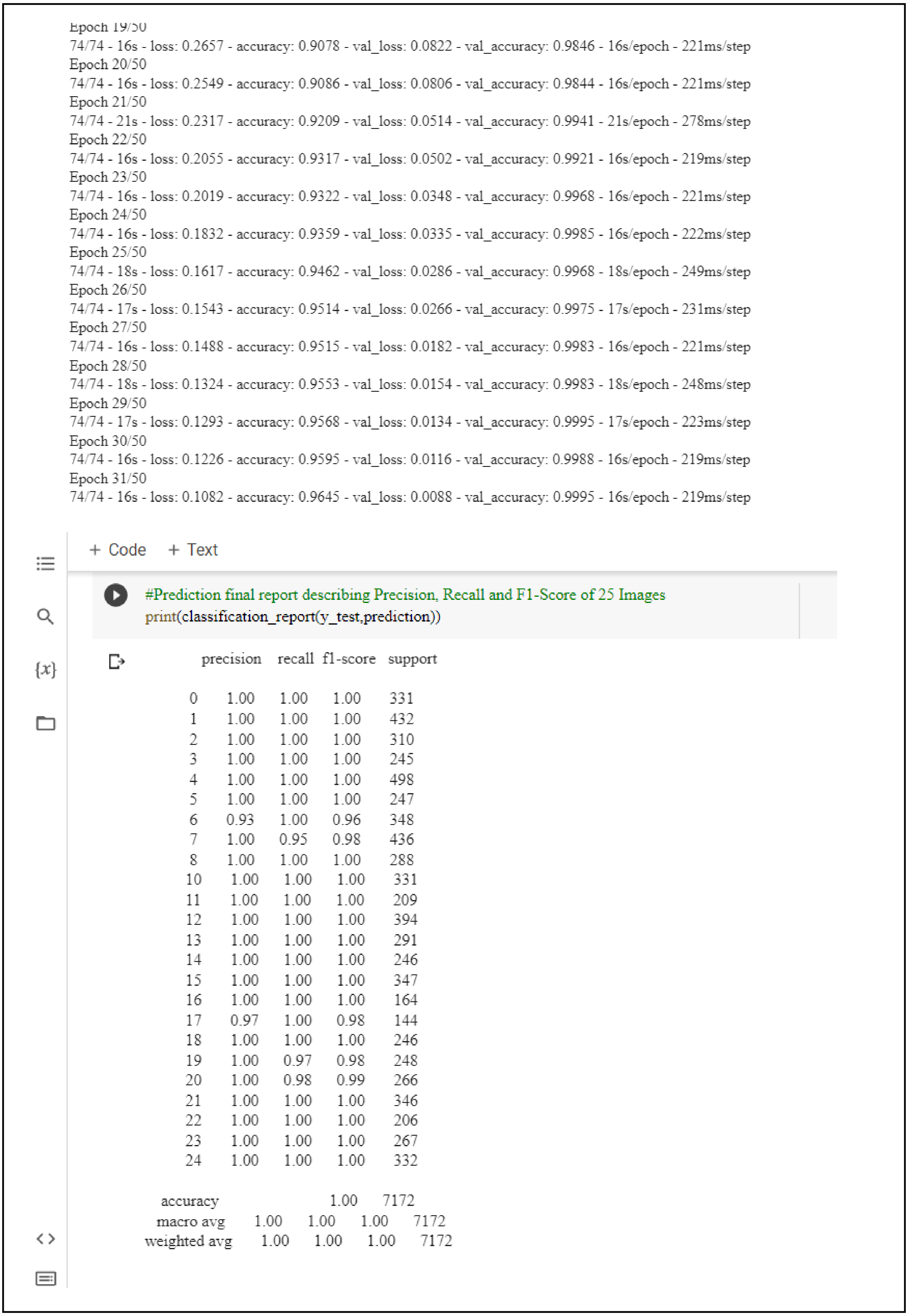
```
#Testing new images on Model
fig, axes = plt.subplots(nrows=5, ncols=5, figsize=(15, 10), subplot_kw={'xticks': [], 'yticks': []})
for i in range(25):
  plt.subplot(5,5,i+1)
  plt.imshow(np.squeeze(x_test[i]), cmap ='gray')
  plt.title(y_test[i])
plt.show()
```



```
#Tarining model against Epoch and Batch Size
history = model.fit(
    x=x_train,
    y=y_train,
    validation_data=(x_val,y_val),
    batch_size=128,
    epochs=50,
    verbose=2
)
```

```
Epoch 1/50
74/74 - 19s - loss: 3.4119 - accuracy: 0.0559 - val_loss: 3.2566 - val_accuracy: 0.0538 - 19s/epoch - 251ms/step
Epoch 2/50
74/74 - 16s - loss: 3.0894 - accuracy: 0.1008 - val_loss: 3.2778 - val_accuracy: 0.0483 - 16s/epoch - 222ms/step
Epoch 3/50
74/74 - 20s - loss: 2.7959 - accuracy: 0.1568 - val_loss: 3.3083 - val_accuracy: 0.0570 - 20s/epoch - 268ms/step
Epoch 4/50
74/74 - 16s - loss: 2.3520 - accuracy: 0.2563 - val_loss: 2.7928 - val_accuracy: 0.1378 - 16s/epoch - 220ms/step
Epoch 5/50
74/74 - 17s - loss: 1.8581 - accuracy: 0.3835 - val_loss: 1.8189 - val_accuracy: 0.4467 - 17s/epoch - 224ms/step
Epoch 6/50
74/74 - 16s - loss: 1.4473 - accuracy: 0.5016 - val_loss: 1.1588 - val_accuracy: 0.6747 - 16s/epoch - 222ms/step
```





```
Epoch 19/50
74/74 - 16s - loss: 0.2657 - accuracy: 0.9078 - val_loss: 0.0822 - val_accuracy: 0.9846 - 16s/epoch - 221ms/step
Epoch 20/50
74/74 - 16s - loss: 0.2549 - accuracy: 0.9086 - val_loss: 0.0806 - val_accuracy: 0.9844 - 16s/epoch - 221ms/step
Epoch 21/50
74/74 - 21s - loss: 0.2317 - accuracy: 0.9209 - val_loss: 0.0514 - val_accuracy: 0.9941 - 21s/epoch - 278ms/step
Epoch 22/50
74/74 - 16s - loss: 0.2055 - accuracy: 0.9317 - val_loss: 0.0502 - val_accuracy: 0.9921 - 16s/epoch - 219ms/step
Epoch 23/50
74/74 - 16s - loss: 0.2019 - accuracy: 0.9322 - val_loss: 0.0348 - val_accuracy: 0.9968 - 16s/epoch - 221ms/step
Epoch 24/50
74/74 - 16s - loss: 0.1832 - accuracy: 0.9359 - val_loss: 0.0335 - val_accuracy: 0.9985 - 16s/epoch - 222ms/step
Epoch 25/50
74/74 - 18s - loss: 0.1617 - accuracy: 0.9462 - val_loss: 0.0286 - val_accuracy: 0.9968 - 18s/epoch - 249ms/step
Epoch 26/50
74/74 - 17s - loss: 0.1543 - accuracy: 0.9514 - val_loss: 0.0266 - val_accuracy: 0.9975 - 17s/epoch - 231ms/step
Epoch 27/50
74/74 - 16s - loss: 0.1488 - accuracy: 0.9515 - val_loss: 0.0182 - val_accuracy: 0.9983 - 16s/epoch - 221ms/step
Epoch 28/50
74/74 - 18s - loss: 0.1324 - accuracy: 0.9553 - val_loss: 0.0154 - val_accuracy: 0.9983 - 18s/epoch - 248ms/step
Epoch 29/50
74/74 - 17s - loss: 0.1293 - accuracy: 0.9568 - val_loss: 0.0134 - val_accuracy: 0.9995 - 17s/epoch - 223ms/step
Epoch 30/50
74/74 - 16s - loss: 0.1226 - accuracy: 0.9595 - val_loss: 0.0116 - val_accuracy: 0.9988 - 16s/epoch - 219ms/step
Epoch 31/50
74/74 - 16s - loss: 0.1082 - accuracy: 0.9645 - val_loss: 0.0088 - val_accuracy: 0.9995 - 16s/epoch - 219ms/step
```

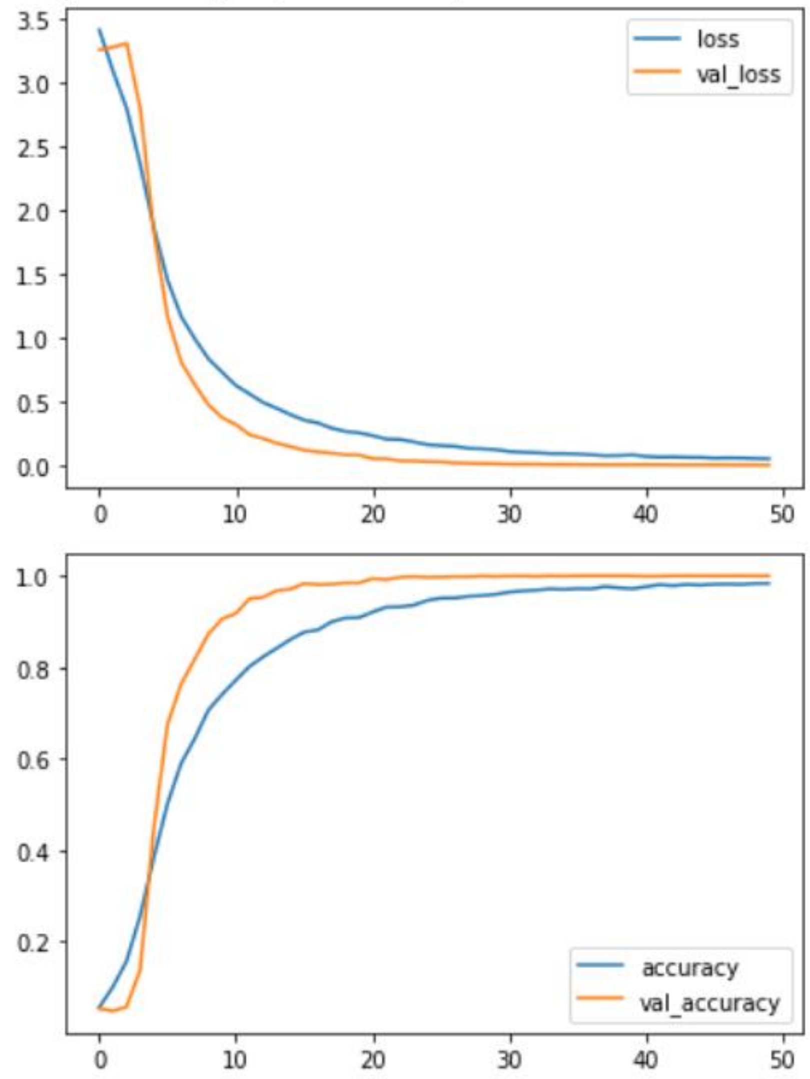
```
#Prediction final report describing Precision, Recall and F1-Score of 25 Images
print(classification_report(y_test,prediction))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       331
           1       1.00      1.00      1.00       432
           2       1.00      1.00      1.00       310
           3       1.00      1.00      1.00       245
           4       1.00      1.00      1.00       498
           5       1.00      1.00      1.00       247
           6       0.93      1.00      0.96       348
           7       1.00      0.95      0.98       436
           8       1.00      1.00      1.00       288
          10       1.00      1.00      1.00       331
          11       1.00      1.00      1.00       209
          12       1.00      1.00      1.00       394
          13       1.00      1.00      1.00       291
          14       1.00      1.00      1.00       246
          15       1.00      1.00      1.00       347
          16       1.00      1.00      1.00       164
          17       0.97      1.00      0.98       144
          18       1.00      1.00      1.00       246
          19       1.00      0.97      0.98       248
          20       1.00      0.98      0.99       266
          21       1.00      1.00      1.00       346
          22       1.00      1.00      1.00       206
          23       1.00      1.00      1.00       267
          24       1.00      1.00      1.00       332

    accuracy                           1.00      7172
   macro avg       1.00      1.00      1.00      7172
weighted avg       1.00      1.00      1.00      7172
```

```
[ ]   <matplotlib.axes._subplots.AxesSubplot at 0x7f727a637210>
```



Conclusion

Training our initial models on less data led to the models quickly overfitting as shown in figure This is likely due to the small amount of samples to train on leading to bad generalization and learning of the sample space. Increasing the size of our dataset to 200 samples/class led to better model results, with peak validation accuracy
of 60.3% in epoch 17. However, looking at our loss function, we see that the validation loss is increasing, indicating overfitting of the model. When applying the CNN model to the images of datset on our test images we could easily **get 98% accuracy**. The mentors were extremely helpful and guided the students in all the possible way The method the method of explanation and teaching was quite amicable and beneficial for us All of our subject queries could also be resolved within the stipulated period and my learning experience was quite amazing. I would like to extend my gratitude to the mentors for supporting me in every step of the internship And I look forward to contributing to the IBM community.

Future Scope

Some of the key advancements in this field could be ***Conversion to Voice, Text, Application for help,*** **interactive Website** to help impaired people at backend through American or Indian Sign Language or Regional Languages as well.

REFERENCES

[1] Md. Moklesur Rahman, Md. Shafiqul Islam, Md. Hafizur Rahman, Roberto Sassi, Massimo W. Rivolta, Md Aktaruzzamank, P2019, “A New Benchmark on American Sign Language Recognition using Convolutional Neural Network”, *IEEE Xplore, 2019.*

[2] https://en.wikipedia.org/wiki/Convolutional_neural_network

[3] https://www.edunetfoundation.org/

[4] https://skills-academy.comprehend.ibm.com/index.php/ibm-skills-academy-home-eng

[5] https://www.kaggle.com/datasets/datamunge/sign-language-mnist

Project Link: https://github.com/Pratik19ap/IBM_AI-SignLanguage-Recognition