American Sign Language Classification Using Convolutional Neural Networks

Project by-

Sagarika Limaye(N15838975)

Samadnya Kalaskar(N13360675)

Rishabh Bahuguna(N19715191)

Problem Formulation

- Classification of American Sign Language Alphabet.
- The training data set contains 87,000 images which are 200x200 pixels.
- There are 29 classes, of which 26 are for the letters A-Z and 3 classes for Space Delete and Nothing.
- The test data set contains a mere 29 images, to encourage the use of real world test images.

Dataset

- https://www.kaggle.com/grassknoted/asl-alphabet/kernels

Motivation

- Sign Language is an important communication tool that is understood by a very few people outside the deaf community.
- The need to bridge the communication gap caused by deafness and speech impairment has been a motivational force for this project.
- One can build autonomous translators to overcome this gap with the help of various Machine Learning tools.
- This project is focused at using American Sign Language recognition with the help of Deep Learning.

Literature Survey

- In Algorithms like Support Vector Machine based image classification, we need to select the features(local, global) and classifiers.
- In some cases, global features work well and in some cases, local features work well.
- This way of extracting the features and further applying different classification techniques may lead to results with less accuracy.

Approach

- ➔ In order to avoid this, our approach was to perform Deep Learning, CNN performs well and it gives better accuracy.

- ➔ It covers local and global features. It also learns the different features from images during training.

- ➔ CNNs effectively use adjacent pixel information to effectively downsample the image first by convolution and then uses a prediction layer at the end. Hence, using Neural networks was our choice of approach.

Algorithm

- A Tensorflow - Keras, GPU implementation is done.
- A sequential model is used.
- Multiple 2D Convolutional Layers with Dropout and MaxPooling have been used.
- Use the model to train and test the predicted outputs.

Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 64, 64, 64)        4864
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 64)        102464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 128)       204928
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       409728
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 256)         819456
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 29)                118813
=================================================================
Total params: 1,660,253
Trainable params: 1,660,253
Non-trainable params: 0
```

Model Fitting and Results

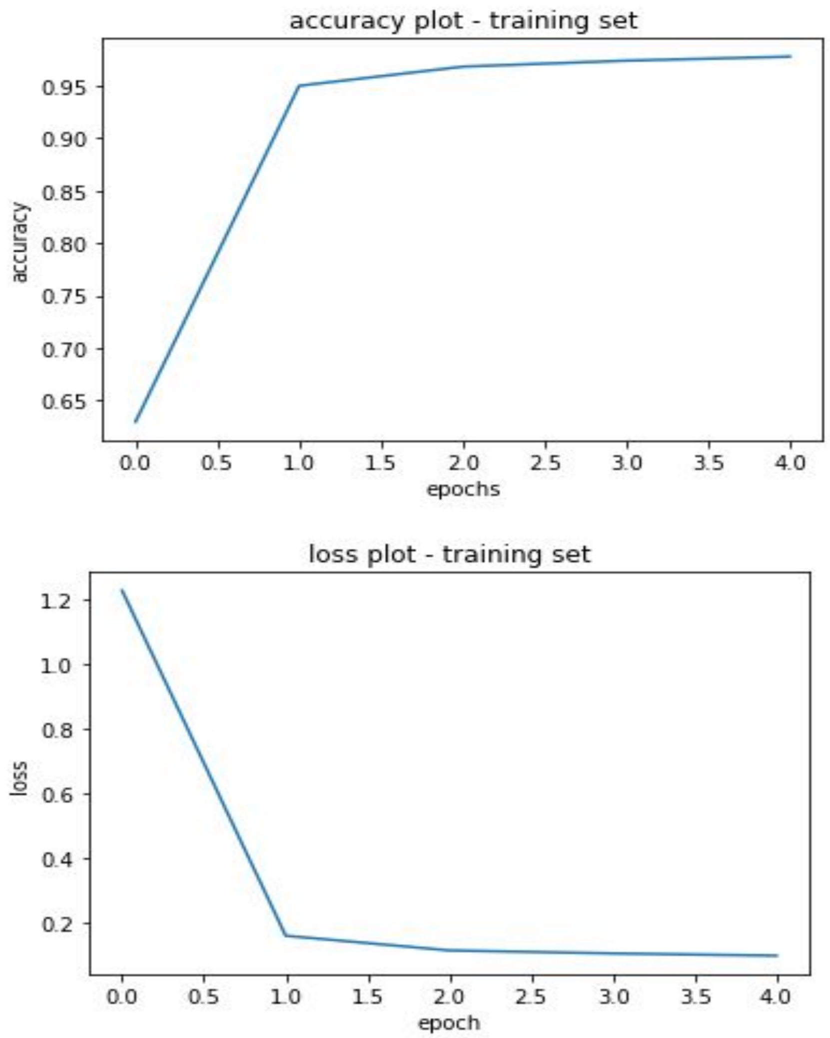
```
Epoch 1/5
78299/78299 [==============================] - 109s 1ms/step - loss: 1.2291 - acc: 0.6297
Epoch 2/5
78299/78299 [==============================] - 105s 1ms/step - loss: 0.1597 - acc: 0.9497
Epoch 3/5
78299/78299 [==============================] - 105s 1ms/step - loss: 0.1143 - acc: 0.9680
Epoch 4/5
78299/78299 [==============================] - 105s 1ms/step - loss: 0.1049 - acc: 0.9737
Epoch 5/5
78299/78299 [==============================] - 105s 1ms/step - loss: 0.0980 - acc: 0.9775
```

```
# Printing scores for test and evaluation images

score = model.evaluate(x = Xts, y = yts, verbose = 0)
print('Accuracy for validation images:', round(score[1]*100, 3), '%')
score = model.evaluate(x = images_test, y = labels_test, verbose = 0)
print('Accuracy for test images:', round(score[1]*100, 3), '%')
```

```
Accuracy for validation images: 99.161 %
Accuracy for test images: 100.0 %
```

Accuracy and Loss



Results and Conclusion

- We were able to accurately classify the test images
- Validation accuracy is 99.161
- For 29 test images(1 for each label) the accuracy obtained is 100%
- Using a GPU improved the speed and performance