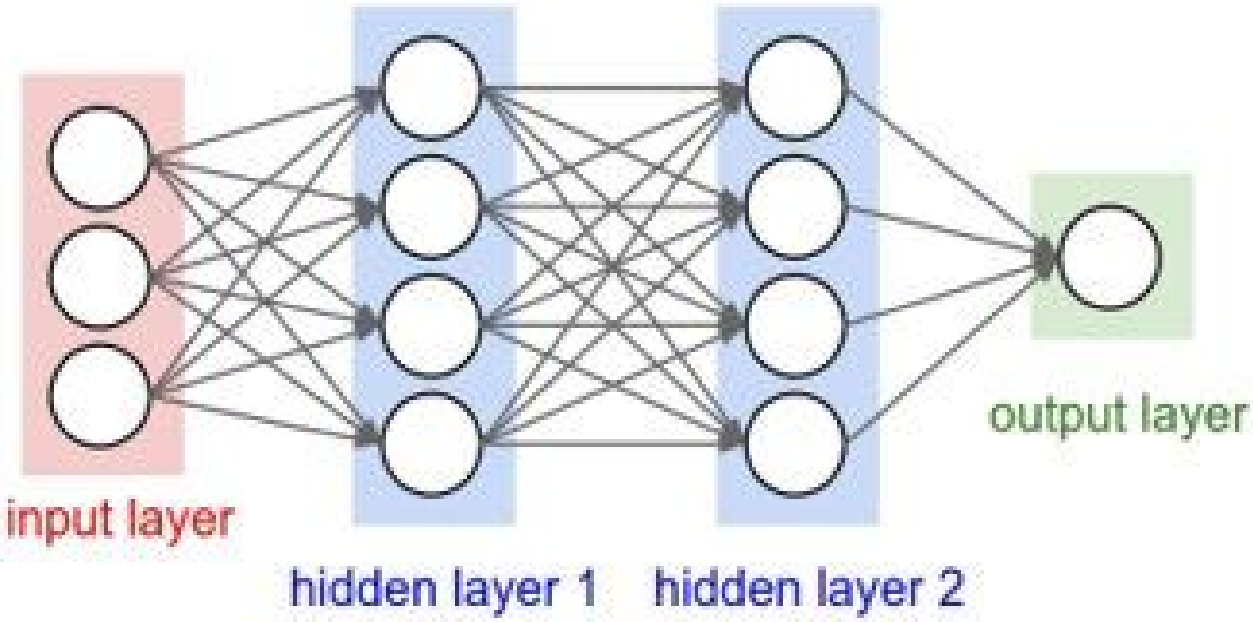
MACHINE LEARNING 2

PROJECT REPORT



TejasviSharma

Deepak Agarwal

INTRODUCTION

Data used in this project is hand gesture recognition data taken from Kaggle.
Hand gesture recognition database is presented, composed by a set of near infrared images acquired by the Leap Motion sensor.
The data is composed by 10 different hand-gestures that were performed by 10 different subjects (5 men and 5 women).
Gestures in data are- Palm, L shape, Fist, Fist moved, Thumb, Index, Ok, Palm moved, C shape, Hand down.
DataSource:-
https://www.kaggle.com/benenharrington/hand-gesture-recognition-database-with-cnn/data

OBJECTIVE

The objective of our project is to develop a Neural Network, which will be trained on hand gesture data. Net should be able to predict unseen data with good accuracy. Trained model will then be deployed to amazon web services. AWS architecture will be paralleled, so that multiple images can be predicted simultaneously. We have used tools provided by Amazon web services to build an architecture on cloud, which is scalable, easily deployable and have parallelized classification of images through it.

TOOLS & LIBRARIES

- Python 3
- Libraries used :
    1. matplotlib - for plotting graphs
    2. sklearn, torch - for modeling
    3. PTL - for images
    4. pandas, numpy - for data manipulation
- Flask web App
- AWS Lambda
- AWS EMR
- Docker Container

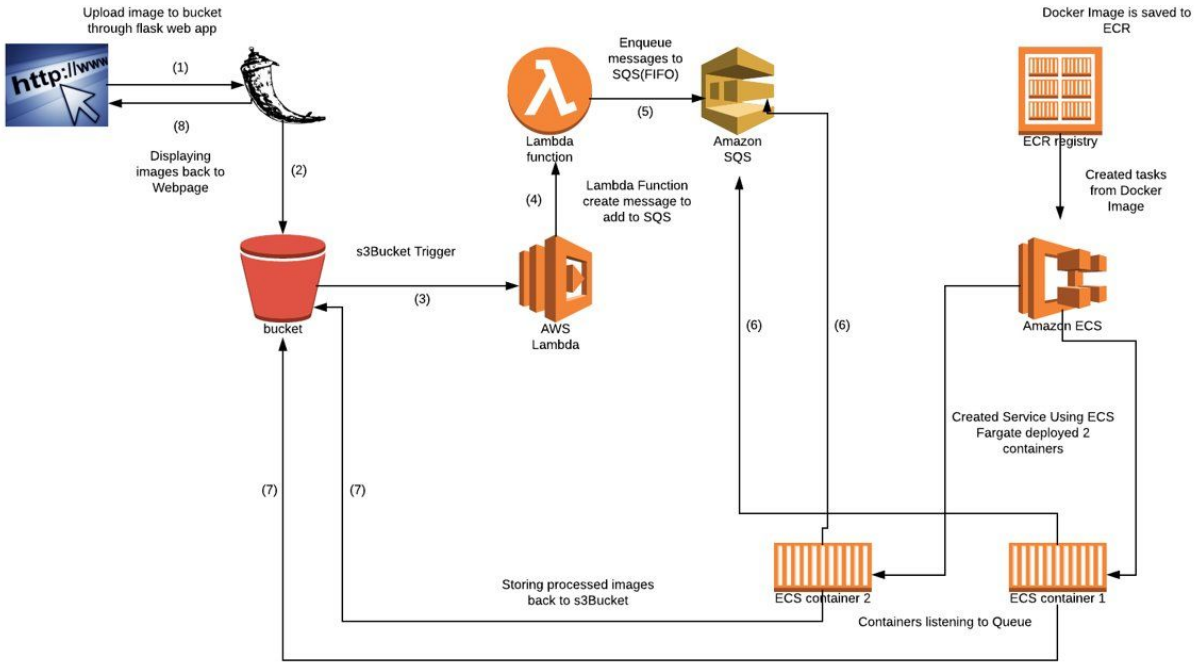
DATA SET

Data used in this project is hand gesture recognition data taken from Kaggle.
Hand gesture recognition database is presented, composed by a set of near infrared images acquired by the Leap Motion sensor.
The data is composed by 10 different hand-gestures that were performed by 10 different subjects (5 men and 5 women).
Gestures in data are- Palm, L shape, Fist, Fist moved, Thumb, Index, Ok, Palm moved, C shape, Hand down.
DataSource:-
https://www.kaggle.com/benenharrington/hand-gesture-recognition-database-with-cnn/data

WORKFLOW

We have deployed industry scalable and parallel architecture on cloud using Flask Web App, AWS Lambda, S3, SQS, EMR and Docker container.

Below is the architecture diagram :



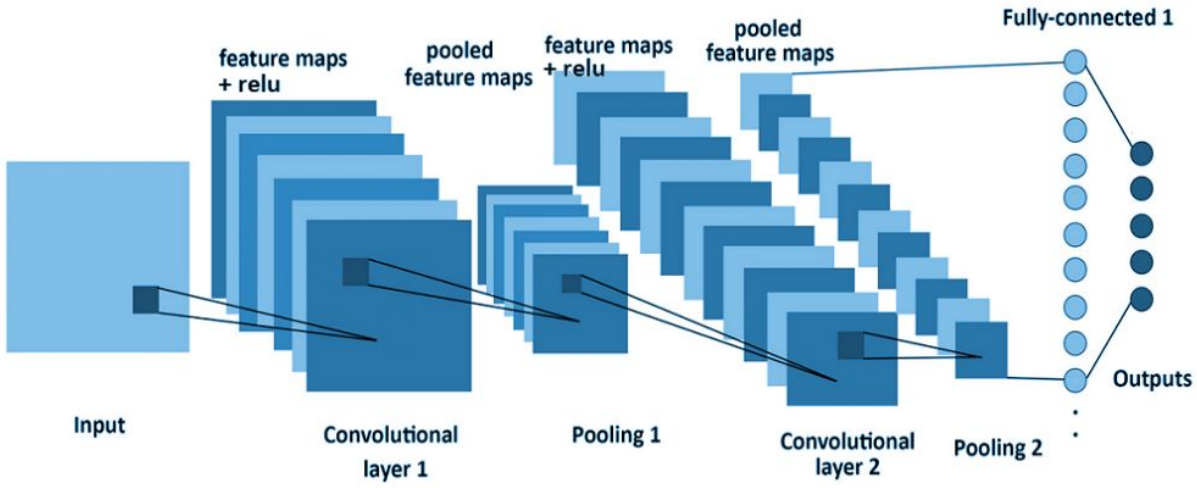
Steps are as follows :

1. Front End Web Page provides image input to Flask App
2. Flask App saves the image to S3 bucket
3. S3 Bucket data save triggers an event on AWS Lambda
4. AWS Lambda function processes the trigger

5. AWS Lambda enque the messages to AWS SQS
6. AWS SQS message is consumed by EMR running trained model in docker container
7. Predicted results are transferred to S3 bucket
8. Web app reads the predicted label from S3 bucket and displays it along with the image

DEEP LEARNING FRAMEWORK & ALGORITHM

We have used CNN with three layers deep learning network on Pytorch. Network details are as follows :



1. Layer 1 - Conv2d layer with number of channels as 1, feature maps as 32, kernel size 3 and stride 2 followed by relu activation function and max pooling layer of kernel size 2

2. Layer 2 - Conv2d layer with input size as 32, feature maps as 64, kernel size 3 and stride 2 followed by relu activation function and max pooling layer of kernel size 2

3. Layer 3 - Conv2d layer with input size as 64, feature maps as 64, kernel size 3 and stride 2 followed by relu activation function and max pooling layer of kernel size 2

4. Layer 4 - Fully connected linear layer with relu activation function

EXPERIMENTAL SETUP

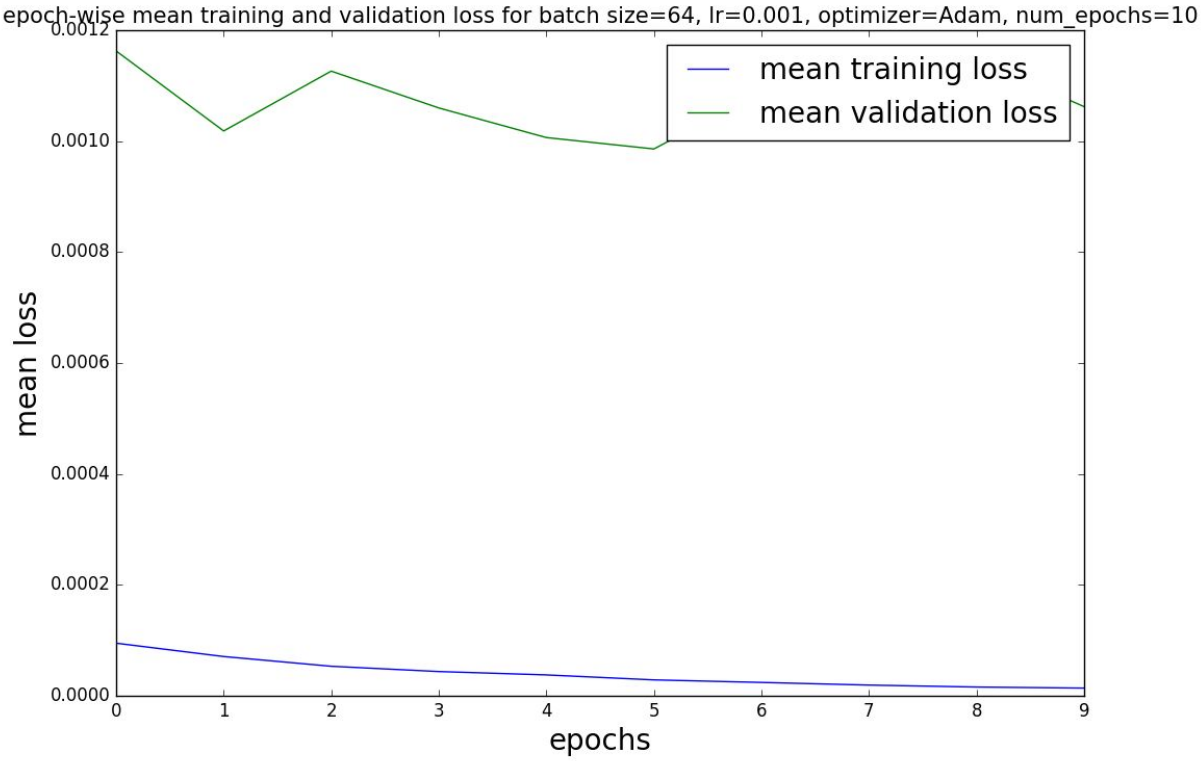
To determine optimum parameters for the model (mini-batch size, learning rate, optimizer function, number of epochs) we have tried our model on 54 different combinations of the above mentioned parameters. Below are the results of each one of them :

batch_ size	learning _rate	optimizer_ method	num_epo chs	time	accuracy
128	0.1	Adam	5	6.93	92.4
128	0.1	Adam	10	6.96	97.8
128	0.1	Adam	2	6.93	94.73
128	0.1	Adadelta	2	7.06	98.53
128	0.1	Adadelta	5	7.05	99.8
128	0.1	Adadelta	10	7.06	99.87
128	0.01	Adam	10	7.01	99.93
128	0.01	Adam	2	7.02	78.2
128	0.01	Adam	5	7	99.8

128	0.01	Adadelta	2	7.06	62.67
128	0.01	Adadelta	5	7.1	89.8
128	0.01	Adadelta	10	7.07	98.73
128	0.001	Adam	2	7.01	99.83
128	0.001	Adam	5	7.02	99.93
128	0.001	Adam	10	7.05	99.93
128	0.001	Adadelta	2	7.09	13.17
128	0.001	Adadelta	10	7.1	51.43
128	0.001	Adadelta	5	7.07	32.4
64	0.1	Adam	10	7.44	98.47
64	0.1	Adam	2	7.4	96.23
64	0.1	Adam	5	7.4	85.37
64	0.1	Adadelta	10	7.61	99.93
64	0.1	Adadelta	5	7.63	99.87
64	0.1	Adadelta	2	7.54	99.43
64	0.01	Adam	2	7.55	99.1
64	0.01	Adam	5	7.53	99.93

64	0.01	Adam	10	7.68	99.93
64	0.01	Adadelta	2	7.6	70.2
64	0.01	Adadelta	5	7.63	95.13
64	0.01	Adadelta	10	7.68	99.47
64	0.001	Adam	2	7.57	99.93
64	0.001	Adam	5	7.65	99.93
64	0.001	Adam	10	7.69	99.9
64	0.001	Adadelta	10	7.67	57.9
64	0.001	Adadelta	2	7.68	14.53
64	0.001	Adadelta	5	7.71	36.47
32	0.1	Adam	5	8.65	75.57
32	0.1	Adam	10	8.81	75.67
32	0.1	Adam	2	8.89	87.6
32	0.1	Adadelta	5	9.05	99.9
32	0.1	Adadelta	10	9.12	99.9
32	0.1	Adadelta	2	9.11	99.53
32	0.01	Adam	2	8.93	98.5

32	0.01	Adam	10	8.93	99.9
32	0.01	Adam	5	8.86	91.63
32	0.01	Adadelta	5	9.12	97.97
32	0.01	Adadelta	2	9.18	74.23
32	0.01	Adadelta	10	9.15	99.7
32	0.001	Adam	2	9.1	99.8
32	0.001	Adam	5	9.06	99.97
32	0.001	Adam	10	8.89	99.97
32	0.001	Adadelta	5	9.24	47.23
32	0.001	Adadelta	10	9.33	75.23
32	0.001	Adadelta	2	9.38	19.97

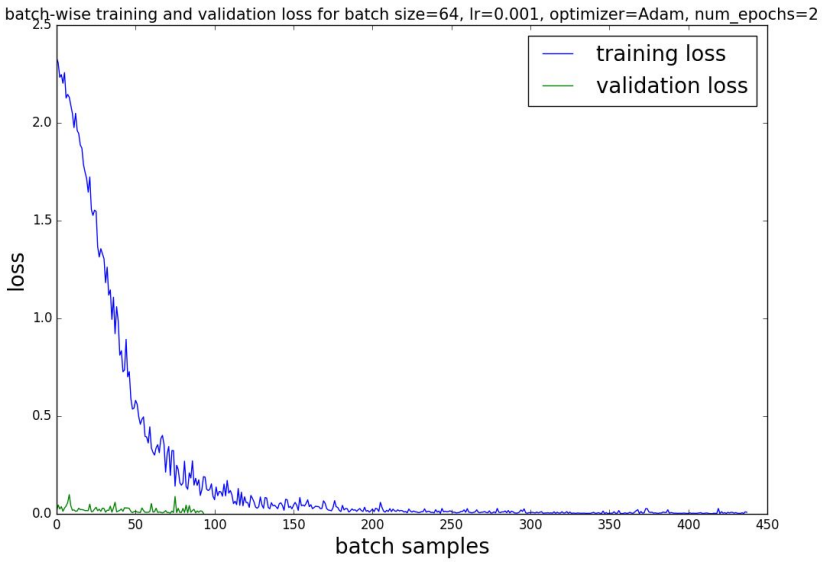


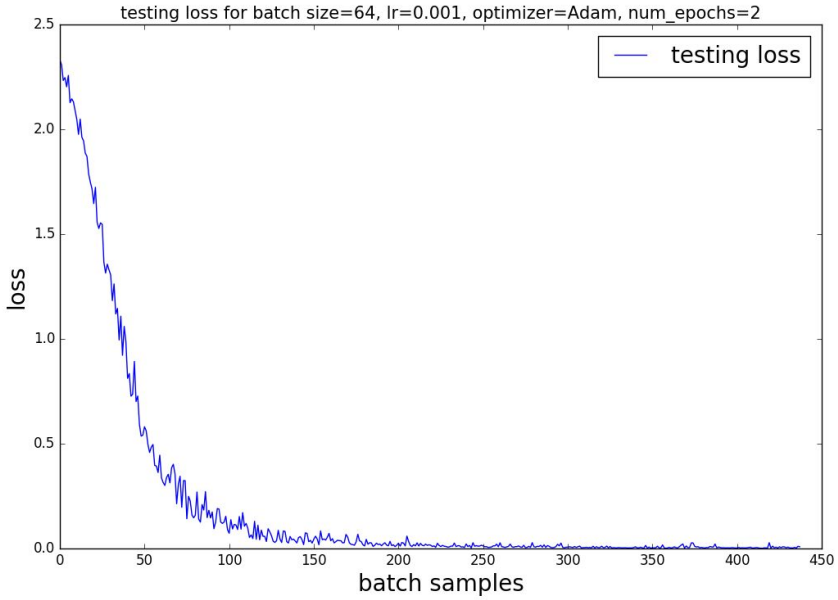
As we can see from above results that Adam optimizer is giving us consistently better results and after two epochs the training loss doesn't decrease significantly.

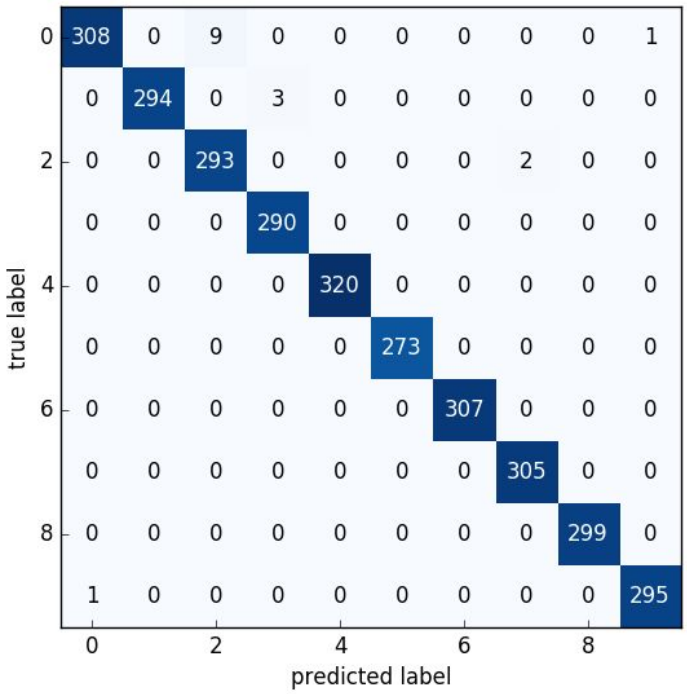
RESULTS

For our data set we have found best accuracy/ f-score for below parameters:

batch size	learning rate	optimizer method	num epochs	time((sec)	accuracy
64	0.001	Adam	2	7.26	99.47




testing loss for batch size=64, lr=0.001, optimizer=Adam, num_epochs=2
testing loss
loss
batch samples
0.0
0.5
1.0
1.5
2.0
2.5
0
50
100
150
200
250
300
350
400
450


true label
predicted label
0 308 0 9 0 0 0 0 0 0 1
0 294 0 3 0 0 0 0 0 0
2 0 0 293 0 0 0 0 2 0 0
0 0 0 290 0 0 0 0 0 0
4 0 0 0 0 320 0 0 0 0 0
0 0 0 0 0 273 0 0 0 0
6 0 0 0 0 0 0 307 0 0 0
0 0 0 0 0 0 0 305 0 0
8 0 0 0 0 0 0 0 0 299 0
1 0 0 0 0 0 0 0 0 295
0 2 4 6 8