Capstone Report

Shadman Ansari

AWS Machine Learning Engineer Nanodegree

American Sign Language hand gesture recognition using CNN Transfer learning

Domain Background:

Sign languages are an important means of communication for individuals with hearing impairments or speech disabilities. However, access to sign language education and resources can be limited, especially for people with special needs. This project aims to use computer vision technology to make sign language education more accessible and inclusive for people with special needs.

The project focuses on developing a Convolutional Neural Network (CNN) model for recognizing sign language gestures. The model will be trained using a pre-trained model as a base, which will be fine-tuned to better recognize the specific sign language being used. The use of a pre-trained model will minimize the amount of data required to train the model and make the process more efficient.

The developed model will be made available online for individuals with special needs to use for learning sign language. The user-friendly interface and real-time gesture recognition will provide an interactive and engaging learning experience, making sign language education more accessible and inclusive.

This project has the potential to improve the lives of individuals with special needs by empowering them to communicate effectively using sign language. By leveraging cutting-edge technology, this project aims to bridge the gap in access to sign language education and promote inclusivity for people with special needs.

Problem Statement:

Individuals who are specially enabled often face difficulties in accessing resources for learning sign languages, particularly American Sign Language (ASL) alphabets. This can pose challenges to their ability to communicate effectively and limit their full participation in society.

The aim of this project is to develop a computer vision model that can accurately recognize ASL alphabets, A through Z, to make sign language education more accessible to these individuals. The model will be based on a pre-trained ResNet architecture and fine-tuned to recognize the 29 classes of ASL alphabets. The challenge is to fine-tune the pre-trained model to accurately recognize the subtle differences between ASL alphabets and minimize the error rate in their recognition.

The goal is to create a model with high accuracy and minimal error in recognizing ASL alphabets, which can be made available online for individuals who are specially enabled to use for learning sign language. This model has the potential to play a crucial role in promoting inclusivity and access to sign language education for all.

Datasets and Inputs:

Data is collected from Kaggle . It has two folders – Train and Test, each folder has 29 classes – A through Z, Del, Space, nothing. Each training class has 2400 images, and each testing class has 600 images.

Data distribution:

Test

-- A (600)

-- B (600)

.

.

.

-- Z (600)

Train

-- A (2400)

-- B (2400)

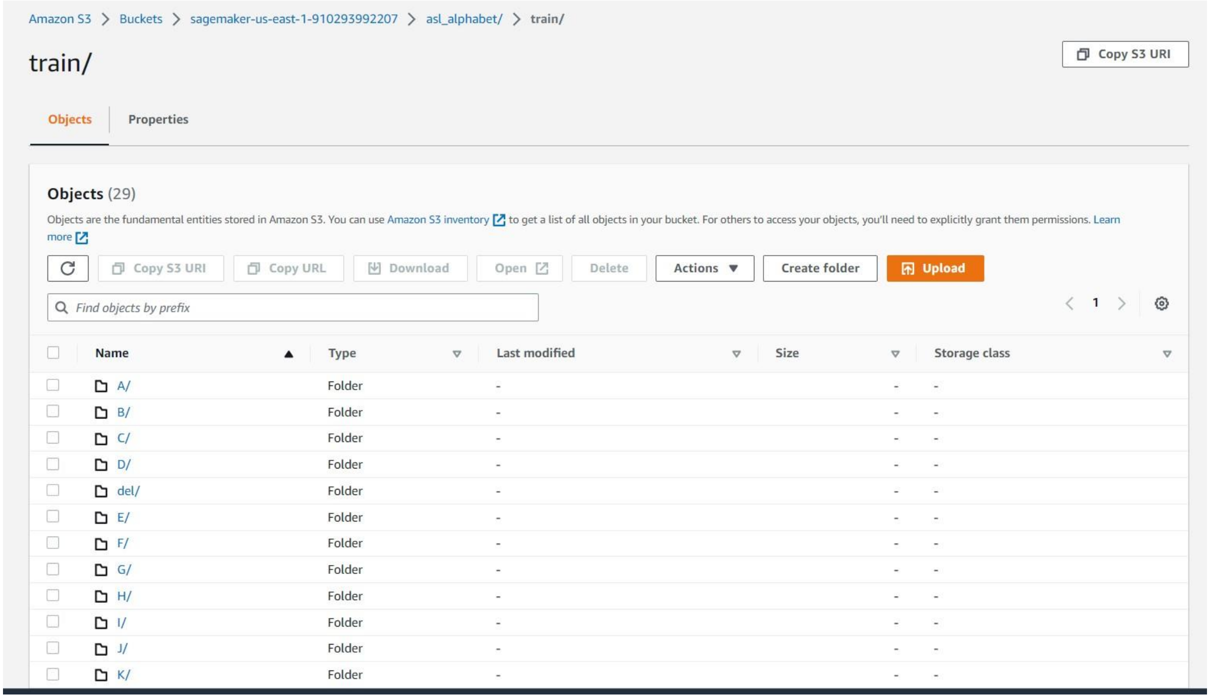
.

.

.

-- Z (2400)

S3 Bucket for training and testing data:



Solution:

Data Preprocessing:

1- Cleaning the image
2- Cropping the image size to (240, 240, 3) to feed them as input to the Convolutional Neural Network
3- Applied Normalization techniques as we have scaled pixel values in the range 0 -1

Benchmark Model:

A convolutional Neural Network (CNN) model will be used in this problem. CNN is a robust algorithm for images and video processing. It is currently one of the best algorithms for the automated processing images. Application involving object detection, image recognition, image segmentation, etc, are some of the tasks of CNN models

Convolutional Neural Network process data with a grid structure [6, 7]. CNN consists of three-layer: Convolutional Layer, Pooling layer, and Fully Connected Layer. The convolutional layer is the most essential and the primary layer in the CNN architecture. Next, to reduce the dimensionality of the feature map, the pooling layer comes into play, then outputs from the final pooling layer or the convolutional layer are fed as inputs to the full connected layer after flattening.

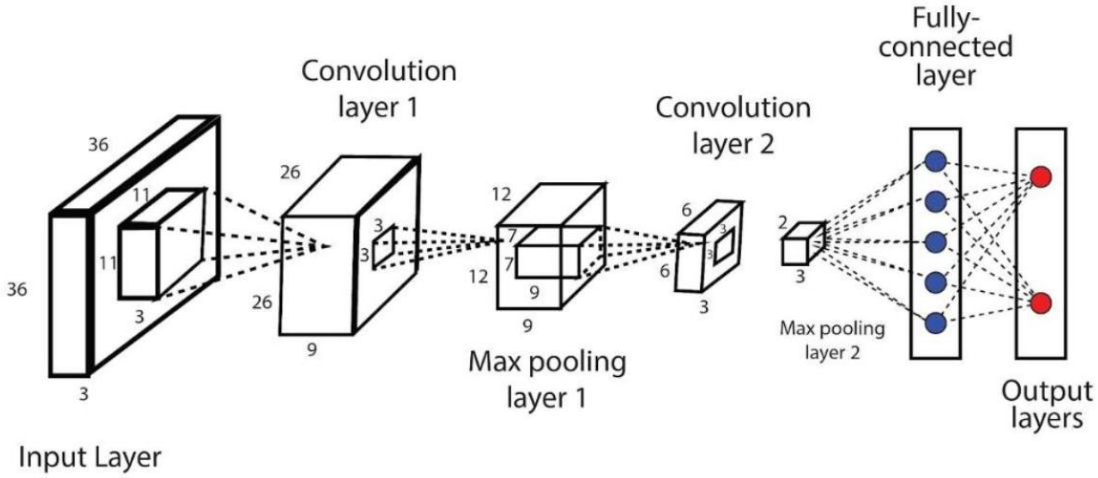
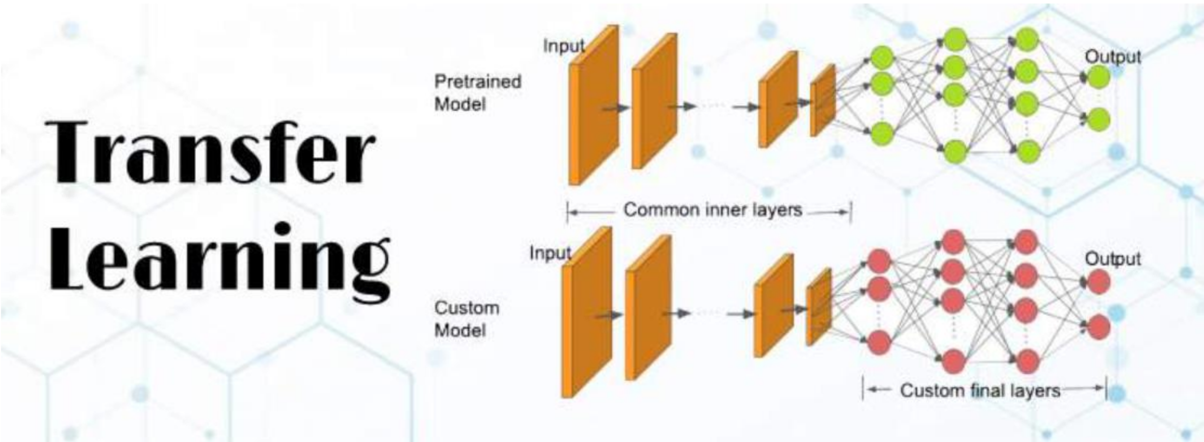
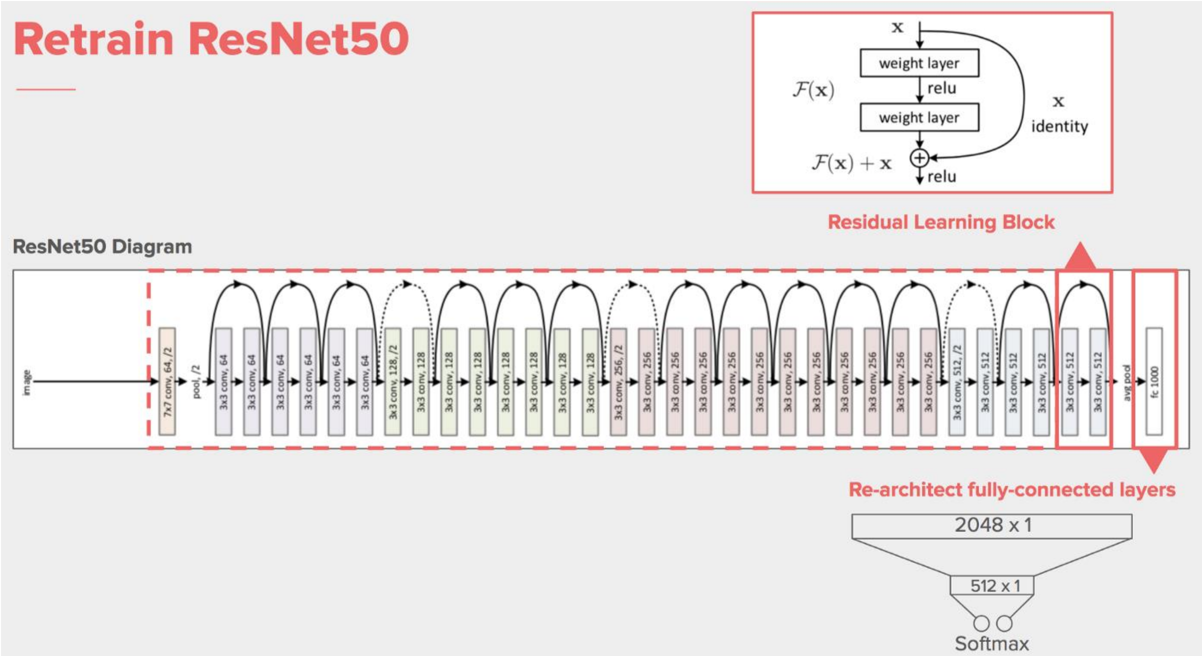


Fig. 3. CNN Architecture [3]

We have used Transfer learning using Resnet-50 Model, Transfer learning is a process to utilize a pre-trained model and only training the outer layer to save computational costs, and time.



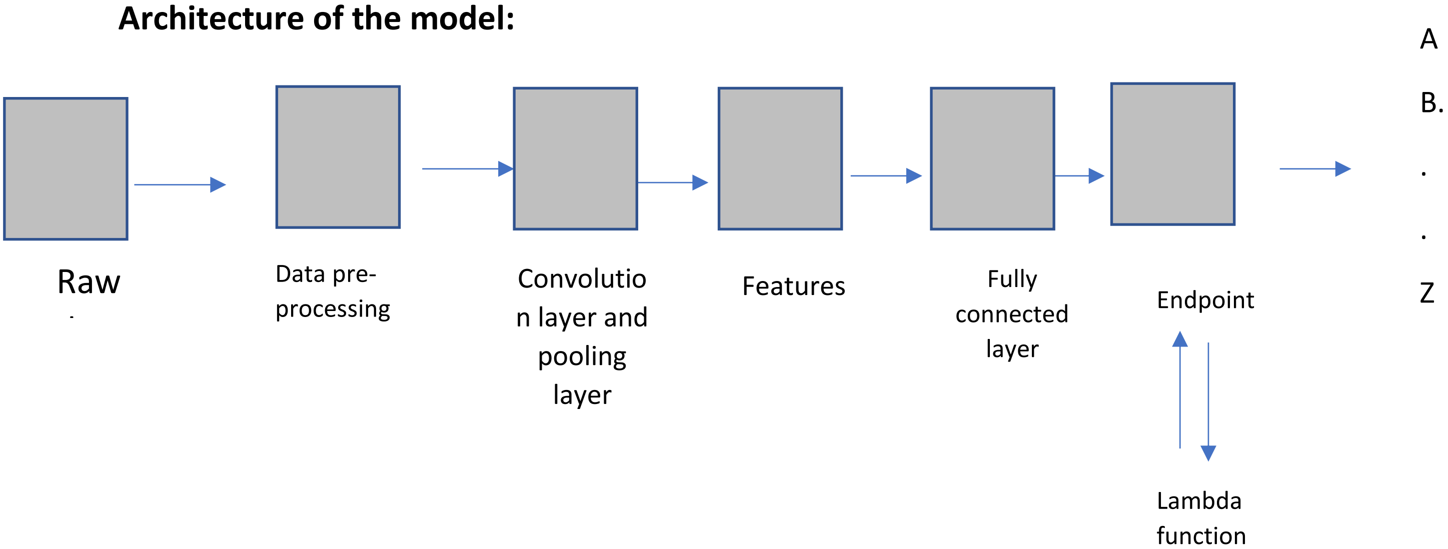


Project Design:

A Convolution Neural Network (CNN) model will be trained using Resnet-50 Pretrained model, hyperparameter tuning job will be trained to determine hyperparameters. The Best hyperparameters will be used to train an estimator and deploy the model.

Then A lambda function will be used to invoke the endpoint and print the output.

Architecture of the model:



Model training – Utilization of Pre-trained Resnet50 CNN model using Transfer learning

Overview of Project steps:

- We have used data from Kaggle, link is mentioned [here](#):

- We will be using a pretrained Resnet50 model from pytorch vision library here
- We will add two Fully connected Neural Network layers on top of the above Resnet50 model We will use concept of Transfer learning therefore we will be freezing all the existing Convolutional layers in the pretrained Resnet50 model and only change the gradients for the two fully connected layers
- We perform Hyperparameter tuning, to get the optimal best hyperparameters to be used in our model
- We have added configuration for Profiling and Debugging our training model by adding relevant hooks in Training and Testing (evel) phases
- We will then deploy our Model, for deploying we have created inference script. The inference script will be overriding a few functions that will be used by our deployed endpoint for making inferences/predictions.

Files used:

- hpo.py - This script file contains code that will be used by the hyperparameter tuning jobs to train and test the models with different hyperparameters to find the best hyperparameter
- train_model.py - This script file contains the code that will be used by the training job to train and test the model with the best hyperparameters that we got from hyperparameter tuning
- endpoint_inference.py - This script contains code that is used by the deployed endpoint to perform some preprocessing (transformations) , serialization-deserialization and predictions/inferences and post-processing using the saved model from the training job.
- train_and_deploy.ipynb -- This jupyter notebook contains all the code and steps that we performed in this project and their outputs.

Hyperparameter Tuning

The Resnet50 Model with two fully connected Neural network layers are used for the image classification problem. Resnet-50 is 50 layers deep NN and is trained on million images of 1000 categories from the ImageNet Database.

The optimizer that we will be using for this model is AdamW (For more info refer

Hence, the hyperparameters selected for tuning were:

Endpoint:

-- Learning rate - default(x) is 0.001 , so we have selected 0.01x to 100x range for the learing rate

-- eps - defaut is 1e-08 , which is acceptable in most cases so we have selected a range of 1e-09 to 1e-08

-- Weight decay - default(x) is 0.01 , so we have selected 0.1x to 10x range for the weight decay

-- Batch size -- selected only two values [64, 128]

Evaluation: Training and testing accuracy: 60% model is trained using hyperparameter tuning, using evaluation: accuracy loss.

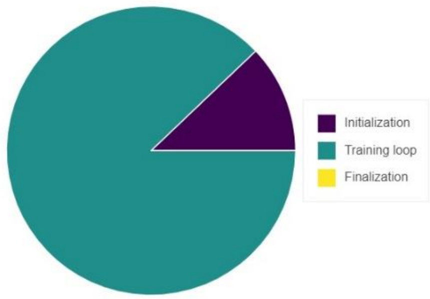
Average test loss

Loss is a value that represents the summation of errors in our model. **It measures how well (or bad) our model is doing**. If the errors are high, the loss will be high, which means that the model does not do a good job. Otherwise, the lower it is, the better our model works.

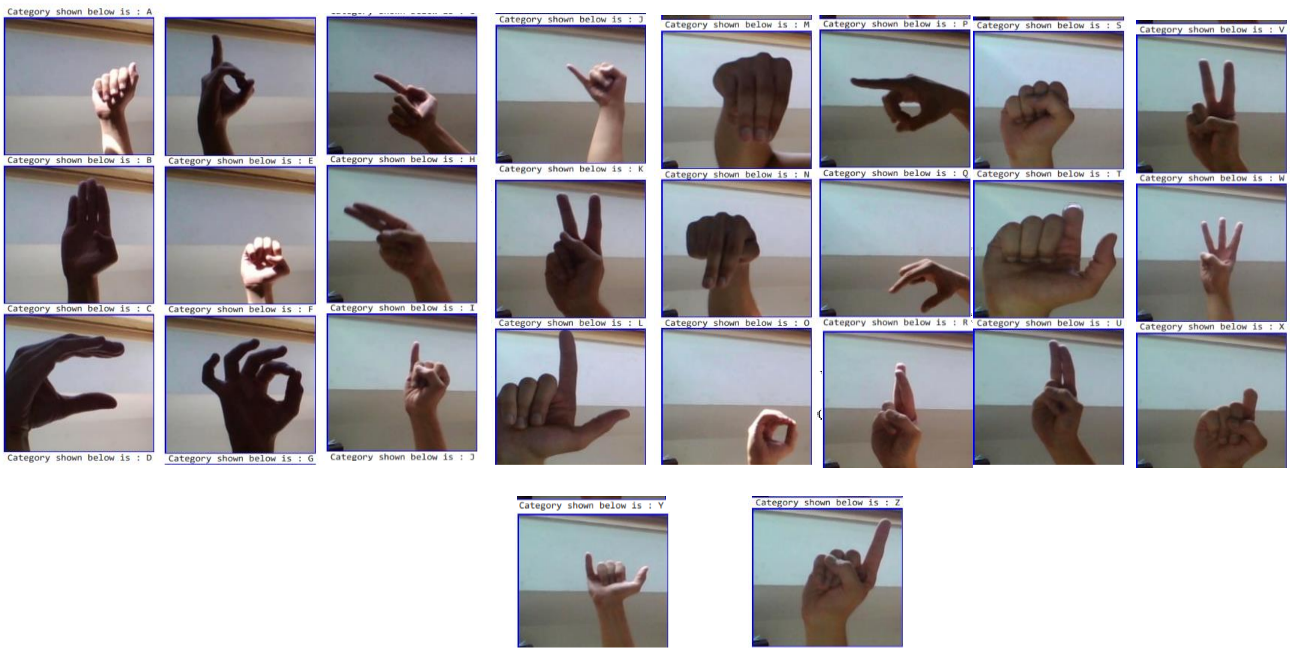
Training job summary

The following table gives a summary about the training job. The table includes information about when the training job started and ended, how much time initialization, training loop and finalization took. Your training job started on 02/10/2023 at 05:02:51 and ran for 1328 seconds.

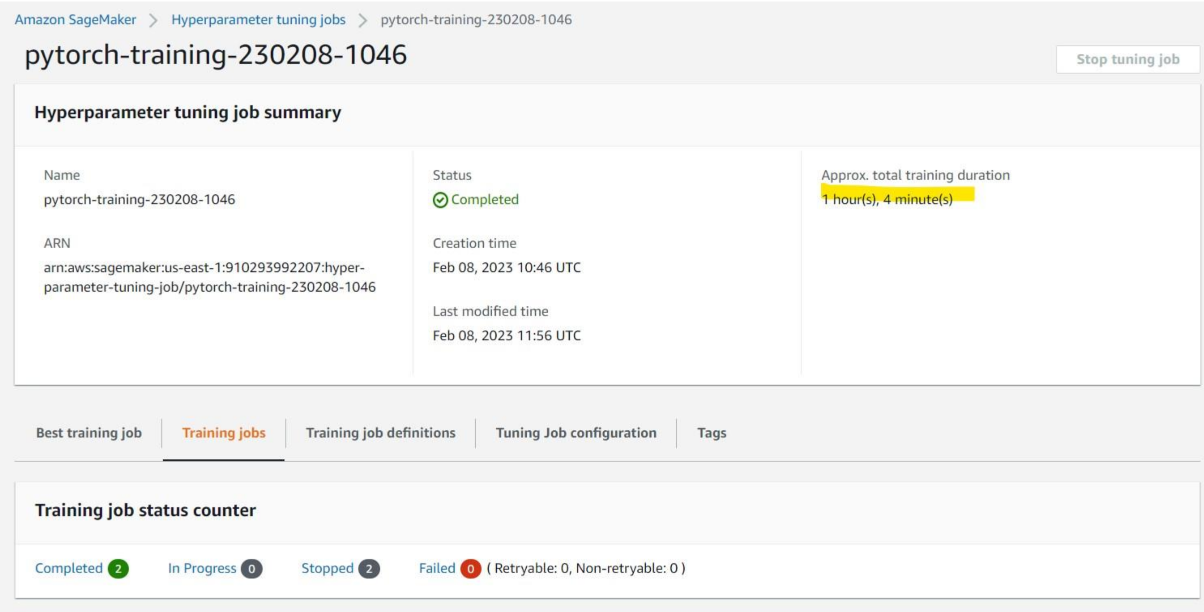
#		Job Statistics
0	Start time	05:02:51 02/10/2023
1	End time	05:24:59 02/10/2023
2	Job duration	1328 seconds
3	Training loop start	05:05:38 02/10/2023
4	Training loop end	05:24:59 02/10/2023
5	Training loop duration	1160 seconds
6	Initialization time	167 seconds
7	Finalization time	0 seconds
8	Initialization	12 %
9	Training loop	87 %
10	Finalization	0 %



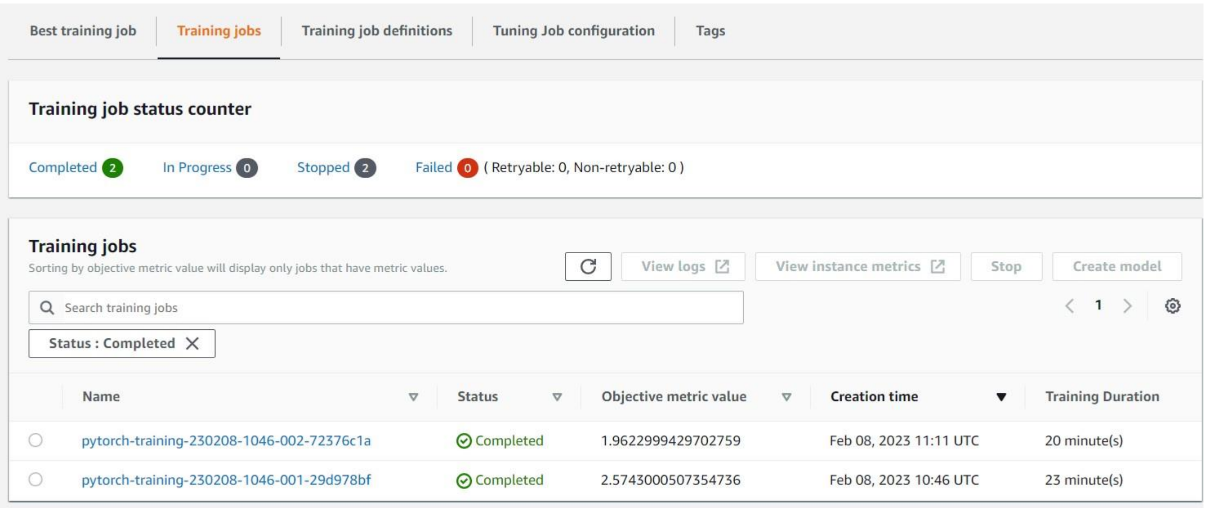
Data exploration:



Hyperparameter Tuning job:



Multiple training jobs triggered by the Hyperparameter tuning job:



Best Hyperparameter job status:

Name	Status	Objective metric	Value
pytorch-training-230208-1046-002-72376c1a	Completed	average test loss	1.9622999429702759

Best training job hyperparameters

Name	Type	Value
_tuning_objective_metric	FreeText	average test loss
batch_size	Categorical	"64"
eps	Continuous	8.894659223977433e-09
lr	Continuous	0.0009034645151607949
sagemaker_container_log_level	FreeText	20
sagemaker_estimator_class_name	FreeText	"PyTorch"
sagemaker_estimator_module	FreeText	"sagemaker.pytorch.estimator"

Endpoint:

Active endpoint:

Endpoints

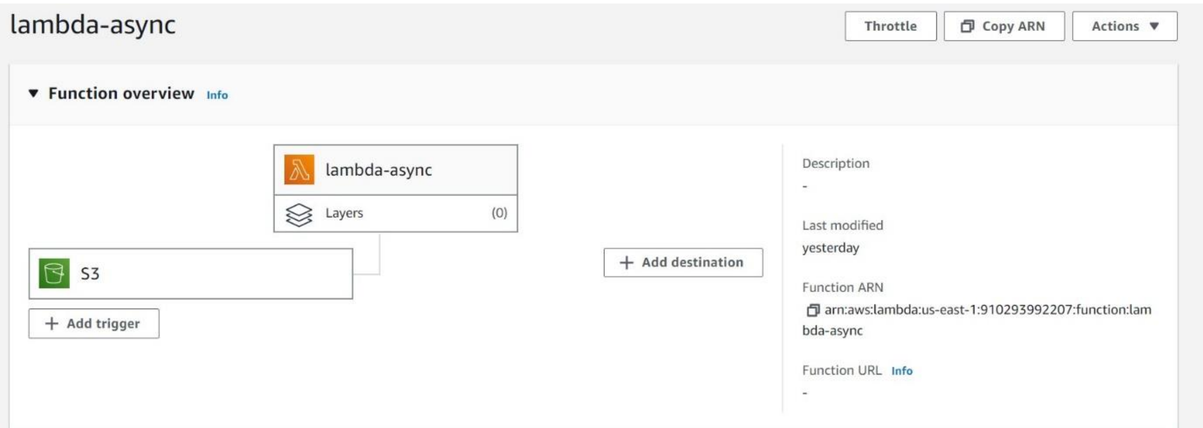
Update endpoint | Action

Search endpoints

Name	ARN	Creation time
pytorch-training-2023-02-10-05-33-56-757	arn:aws:sagemaker:us-east-1:910293992207:endpoint/pytorch-training-2023-02-10-05-33-56-757	Feb 10, 2023 05:34 UTC

After deploying endpoint, I have created a lambda function and created asynchronous trigger, so whenever a input test file is upload in the below location:

Lambda function S3 asynchronous – trigger:



Lambda function: once file is upload in the linked S3 bucket, s3_uri is obtained for endpoint invocation

Go to Anything (Ctrl-P)

lambda-async - /
lambda_function.py
predictor.py

lambda_function × | predictor.py ×

```python
import json
import urllib
from predictor import preprocess

def lambda_handler(event, context):
    #for r in event['query']['Records']
    for r in event['Records']:
        bucket = r['s3']['bucket']['name']
        key = urllib.parse.unquote_plus(r['s3']['object']['key'], encoding='utf-8')
        uri = "/".join([bucket, key])
        output = preprocess(uri)

    return {
        'statusCode': 200,
        'body': {

            "s3_bucket": bucket,
            "s3_key": uri ,
            "predicted_category": output
                }
            }
```

Lambda- predictor .py file:

Go to Anything (Ctrl-P)

lambda-async - /
lambda_function.py
predictor.py

lambda_function × | predictor.py ×

```python
import boto3
import os
import io
import boto3
import json
import base64
ENDPOINT_NAME = 'pytorch-inference-2023-02-10-05-41-51-640'

runtime= boto3.client('runtime.sagemaker')
def preprocess(s3_input_uri):

    with open(s3_input_uri, "rb") as f:
        image = base64.b64encode(f.read())
        # image = base64.b64encode(image_data)
        # Make a prediction:
        response = runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
                                           ContentType='image/png',
                                           Body=image)
        predicted_category = np.argmax(response,1) + 1
        return predicted_category
```

S3 bucket used for Lambda asynchronous invocation:

Amazon S3 > Buckets > sagemaker-us-east-1-910293992207 > Input-Test-Prediction-Folder/

Input-Test-Prediction-Folder/

Copy S3 URI

Objects | Properties

Objects (4)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload

Find objects by prefix

1

Name	Type	Last modified	Size	Storage class
A5.jpg	jpg	February 10, 2023, 12:31:15 (UTC+05:30)	11.3 KB	Standard
B11.jpg	jpg	February 10, 2023, 12:31:17 (UTC+05:30)	11.5 KB	Standard
C30.jpg	jpg	February 10, 2023, 12:31:18 (UTC+05:30)	12.0 KB	Standard
D10.jpg	jpg	February 10, 2023, 12:31:20 (UTC+05:30)	12.2 KB	Standard

As soon as the file is uploaded in the above S3 bucket, lambda event is triggered, S3 uri is send to lambda function invoke an established endpoint

Output is shown below:

lambda_function × Execution result×

Execution results — Status: Succeeded | Max memory used: 36 MB | Time: 1.26 ms

Test Event Name

```
test
```

Response


```
{
  "statusCode": 200,
  "body": {
    "s3_bucket": "sagemaker-us-east-1-910293992207",
    "s3_key": "s3://sagemaker-us-east-1-910293992207/Input-Test-Prediction-Folder/A5.jpg",
    "predicted_category": 1
  }
}
```

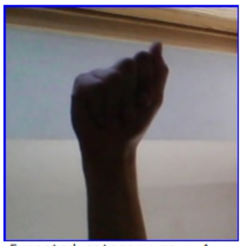

Function Logs

```
START RequestId: 4d026fa1-46da-4ba2-8e1b-a5926b8899ac Version: $LATEST
END RequestId: 4d026fa1-46da-4ba2-8e1b-a5926b8899ac
REPORT RequestId: 4d026fa1-46da-4ba2-8e1b-a5926b8899ac  Duration: 1.26 ms   Billed Duration: 2 ms   Memory Size: 128 MB Max Memory Used: 36 MB  Init Dur
```

Request ID

```
4d026fa1-46da-4ba2-8e1b-a5926b8899ac
```

Some of the outputs generated using endpoint invocation in the jupyter notebook:



```
Expected category no : A
Response: [[7.688182353973389, 3.9646029472351074, 0.9547290802001953, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.5011566877365112, 0.0, 3.56672835
3500366, 2.2505133152008057, 0.0, 0.0, 0.0, 0.0, 0.310312956571579, 4.681624412536621, 4.327795505523682, 0.0, 0.0, 0.0, 4.6362524032592
77, 2.499082326889038, 0.0, 0.0, 0.0, 2.8500635623931885, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]
--------------------------------------------------------------------
--------------------------------------------------------------------
--------------------------------------------------------------------
Response/Inference for the above image is : 'A'
--------------------------------------------------------------------
Test image no: 2
Below is the image that we will be testing:
```

```
Expected category no : B
Response: [[0.0, 10.367291450500488, 0.0, 0.0, 0.0, 0.0, 0.0, 0.598381757736206, 0.0, 0.8182090520858765, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.3108114004135132, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]
--------------------------------------------------------------------
--------------------------------------------------------------------
Response/Inference for the above image is : 'B'
--------------------------------------------------------------------
```

```
--------------------------------------------------------------------
Test image no: 3
Below is the image that we will be testing:
```

```
Expected category no : C
Response: [[0.0, 0.0, 10.699620246887207, 3.797069549560547, 0.0, 0.0, 0.0, 0.4872783422470093, 0.0, 0.0, 0.0, 5.143039703369141, 0.0,
0.0, 0.0, 0.3721250891685486, 1.5564589500427246, 0.0, 0.0, 1.8282265663146973, 0.0, 0.0, 0.0, 0.7803218960762024, 0.0, 0.0, 0.0, 0.0,
4.661263942718506, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]
--------------------------------------------------------------------
Response/Inference for the above image is : 'C'
--------------------------------------------------------------------
--------------------------------------------------------------------
--------------------------------------------------------------------
Test image no: 4
Below is the image that we will be testing:
```

```
Expected category no : D
Response: [[0.0, 0.0, 3.358764410018921, 8.803892135620117, 0.0, 2.419361114501953, 0.0, 0.39468228816986084, 0.0, 3.692802667617798, 0.
3554805815219879, 0.0, 0.0, 0.0, 0.21633651852607727, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.438459694385528
56, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]
Response/Inference for the above image is : 'D'
--------------------------------------------------------------------
--------------------------------------------------------------------
--------------------------------------------------------------------
--------------------------------------------------------------------
```

Future scope:

Implementation of the model and creating an app using RestAPI to create real time predictions, I have got training and testing accuracy around 60%, but that can be increased by increasing the number of testing data images.

Conclusion:

ASL gesture recognition model is helpful for translating sign language into English language, and this model will be helpful for the Specially abled people to express themselves and make other people understand. This model can be improved in terms of Accuracy by incorporating more sign language images for each class, high number of images are not used in this model due to budget constraints.

The endpoint can be linked to Rest API and can be incorporated into an Mobile app to create real-time conversation solution.

References

[1] Accuracy, Precision, and Recall in Deep Learning | Paperspace Blog. (2022). Retrieved 2 July 2022, from https://blog.paperspace.com/deep-learning-metrics-precision-recallaccuracy/#:~:text=Accuracy%20is%20a%20metric%20that,the%20total%20number%20of %20predictions

[2] AUC-ROC Curve in Machine Learning Clearly Explained - Analytics Vidhya. (2022). Retrieved 1 July 2022, from https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curvemachine-learning/

[3] Balodi, T. (2022). Convolutional Neural Network with Python Code Explanation | Convolutional Layer | Max Pooling in CNN. Retrieved 1 July 2022, from https://www.analyticssteps.com/blogs/convolutional-neural-network-cnn-graphicalvisualization-code-explanation

[4] Brain Tumor: Symptoms, Signs & Causes. (2022). Retrieved 2 July 2022, from https://my.clevelandclinic.org/health/diseases/6149-brain-cancer-brain-tumor

[5] Confusion Matrix, Accuracy, Precision, Recall, F1 Score. (2022). Retrieved 30 June 2022, from https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-scoreade299cf63cd#:~:text=F1%20Score%20becomes%201%20only,0.857%20%2B%200.75)%2 0%3D%200.799

[6] Danukusumo, K., Pranowo, & Maslim, M. (2017). Indonesia ancient temple classification using convolutional neural network. 2017 International Conference on Control, Electronics, Renewable Energy and Communications (ICCREC). doi: 10.1109/iccerec.2017.8226709

[7] Febrianto, D., Soesanti, I., & Nugroho, H. (2020). Convolutional Neural Network for Brain Tumor Detection. IOP Conference Series: Materials Science and Engineering, 771(1), 012031. doi: 10.1088/1757-899x/771/1/012031

[8] Kumar, S., Dhir, R., & Chaurasia, N. (2021). Brain Tumor Detection Analysis Using CNN: A Review. 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS). doi: 10.1109/icais50930.2021.9395920

[9] VGG-19 convolutional neural network. (2022). Retrieved 1 July 2022, from https://www.mathworks.com/help/deeplearning/ref/vgg19.html;jsessionid=61f4e9473920 165c667ac9639843#:~:text=VGG%2D19%20is%20a%20convolutional,%2C%20pencil%2 C%20and%20many%20animals

[10] Bansal, S. (2022). CNN Architectures: VGG, ResNet, Inception + TL. Retrieved 1 July 2022, from https://www.kaggle.com/code/shivamb/cnn-architectures-vgg-resnet-inception-tl

[11] Bardhi, M. (2022). Image Detection Using the VGG-19 Convolutional Neural Network.

Retrieved 2 July 2022, from https://medium.com/mlearning-ai/image-detection-usingconvolutional-neural-networks-89c9e21fffa3

[12] Baheti, P. (2022). A Newbie-Friendly Guide to Transfer Learning. Retrieved 1 July 2022, from

https://www.v7labs.com/blog/transfer-learning-guide