**TECHNICAL REPORT**

**Function Based Hand Gesture Detection Using Images**

***Submitted By***

***Udhaya Kumar Rajendran (1134990)***

***Vairag Parikh (1153939)***

**Hardware Requirements:**

The project is based on deep learning neural network architecture. Below are the hardware requirements for the project to successfully execute.

RAM: 16GB (minimum)

CPU: i7 11th gen or higher

GPU (recommended): NVIDIA GeForce GTX 960 or higher graphics card - 8GB

Web camera: To predict the gesture live

**Software Requirements:**

The following software/packages/bundles should be available/installed in the execution environment.

Programming Language: Python 2.7 or higher

Deep Learning Framework: Pytorch

Computer Vision: OpenCV-Python library

**Architecture Diagram**

Pre-processing

(scaling, resizing)

Pre-processing

(scaling, resizing)

**Train Data**

**Test Data**

**Webcam**

**(Live Images)**

Binary Image Conversion

Histogram Equalizer

**Detected Gesture**

Feature Extraction/Detection and Classification

Training Knowledge

Convolutional Neural Network

Fully Connected **Classification Layer**

Feature Extraction **Convolution Layer**

**Dataset:**

We have used 6 gestures for training and testing the built convolutional neural network model. There are 1200+ images in each of the gesture category for training the model.

Following are the 6 gestures.

|  |  |
| --- | --- |
| **Image** | **Gesture** |
|  | Fist |
|  | L |
|  | OK |
|  | Palm |
|  | Thumbs Up |
|  | Thumbs Down |

**Pytorch:**

Pytorch is an open source machine learning library based on the torch library useful for the applications such as computer vision, natural language processing etc. Since the project is based on convolutional neural network architecture, we need deep learning framework which provide useful abstractions speed up model development as well as reducing the boilerplate code. We have few more deep learning frameworks available such as keras and tensorflow. We have used pytorch in our project since it is easier to customize the layers in the neural network as well as to interact with the model.

A screenshot of a computer

Description automatically generated

**OpenCV:**

OpenCV-Python library provides APIs such as imread(), imshow() and other necessary APIs which are used to read an image, write an image that are really useful when live images are being taken from the webcam.

Also, the cv2 library is used to work with the camera, to turn on or to capture a frame etc.

Text

Description automatically generated

**Project Workflow:**

Since this is a classification problem which comes under supervised learning paradigm, each image should be associated with its gesture label.

**Conversion of Image to Pixel**

* There are two different folders that contains images for training the model and testing the knowledge of the model respectively.
* For the model to work with the numbers, the images are converted to their corresponding pixels.
* Since we only have images collected under a folder (name of the folder is the label) for every category, we iterate over every folder, convert the images into pixel and associate it with the name of the folder as label.
* While loading the image with the help of torchvision package, the image can be transformed such as the image can be scaled, resized etc.

Text

Description automatically generated

* Now we have ‘pixel of an image’ as ***X*** and ‘folder name (label)’ as ***y***
* It is not advisable to do this step (conversion of image to pixel) every time when the dataset is not changed. Hence it is recommended to save the converted pixels in the local space and load whenever needed. Pytorch supports saving the conversion with torch.save() api and loading the saved file with torch.load().

Text

Description automatically generated

**Preparing Dataloader:**

It is essential to keep the data manageable especially when working with large dataset for a machine learning project. Pytorch supports preparing custom dataloader with the dataset we have which can be batched and iterated while training the deep network model.

Graphical user interface, text

Description automatically generated

**Convolutional Neural Network:**

Convolutional neural network (CNN) is a deep learning network very useful for image recognition or classification since the features from the images are extracted in the convolution layer of the network which provides improved performance even when the target object is present anywhere in the image. Basically, the CNN has convolution layers, then pooling layers and then fully connected layers for classification.

Below are the parameters used for the building Convolutional Neural Network in this project.

|  |  |
| --- | --- |
| Parameter | Value |
| Input Channels | 1 (binary image) |
| Kernel Size | 3 |
| Stride | 1 |
| Padding | 1 |
| Dropout | 0.2 |
| Pooling | MaxPooling with Kernel size 2 |
| Batch Size of training dataloader | 32 |
| Learning Rate | 0.01 |
| Number of epochs (iterations) | 3, 10, 25, 50 (different combinations) |
| Loss Function | Cross Entropy Loss |
| Optimizer | Adam |

**Kernel Size** – filter used to extract features from the image

**Stride** – controls how the kernel is moved around the input image pixel matrix

**Padding** – number of pixels added to an image on the boundary sides of the image

**Dropout** – certain neurons are randomly selected and are ignored during training

**Pooling** – in every convolution layer the number of channels is increased. With pooling, the size of output of convolutional layer is decreased considerably.

**Training and Testing:**

1. The convolutional neural network is trained for the number of epochs defined in the program.
2. For every epoch,
   1. the training data is allowed to pass through the convolution layers which extract the features and the flatten layer will convert the m\*n\*d array to one dimension vector which is passed to the fully connected dense layer. Finally, the output layer will produce the output values which is correlated with the class label of an image
   2. The predicted labels are compared with the original target labels and the loss is calculated with the help of Loss function.
   3. The loss is back propagated in the network so that the weights and bias in the layers of the deep network gets adjusted to minimize the loss function.
3. The trained model is saved with the help of torch.save(). The same trained model can be loaded next time which will use the available learned weights (from the training) for prediction.

The model is tested with around 160 images (approximately 25 in each category). Confusion matrix is recorded for the testing phase.

**Testing Live Gestures:**

With the help of OpenCV-Python library, every frame captured from webcam is converted, pre-processed, and then allowed to utilize the knowledge of the trained model for prediction.

The frame that is captured via webcam is converted to binary image followed by histogram equalization. The image is then resized and converted to pixels before using it in convolution layers. The testing phase is same as that of the testing scenarios as said in Training and Testing section.

**Global Constants:**

To organize and have all the constants in one place, we have created a dedicated file for global constants which can be imported in any python file and used.

**HOW TO RUN THE PROJECT:**

The following files (submitted in D2L) should be available in the folder structure as submitted.

Unzip the dataset zip file and project zip file - ***HandGestureRecognition\_Group12, Leap\_Group12***

***capture\_image.py***

***classification\_cnn.py***

***dataloader.py***

***global\_constants.py***

***img\_to\_pixel.py***

***knowledge.py***

***label\_rename.py***

***main.py***

Please Open the file ***global\_constants.py*** and change the below constants to set the base path of the dataset folder.

**BASE\_PATH = *<<saved root path of dataset>>***

**TARGET\_FOLDER = *<<saved root path of dataset>>***

Run the ***main.py*** python file which has the sequential steps to execute the project.

**SAMPLE OUTPUT SCREENSHOTS:**

The training is carried out for 50 epochs and the training loss is printed for every epoch.

A screenshot of a computer

Description automatically generated

Below is the training loss curve for the 50 epochs

Chart, histogram

Description automatically generated

During Testing Phase, the original labels and predicted labels are printed

A screenshot of a computer

Description automatically generated with medium confidence

Below is the confusion matrix for the test set

Graphical user interface, application, Teams

Description automatically generated

CChart, histogram

Description automatically generated

Graphical user interface

Description automatically generated

Chart

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

Chart, histogram

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence