A DEEP LEARNING APPROACH TO CONTROL A GAME BASED ON HAND GESTURES USING A LOW-RESOLUTION CAMERA

GroupNumber:31 **PanelNumber:19**

S.No	Reg.No	Name of the Student	Section
1	CB.EN.U4CSE17420	GAYATHRI E	CSE E
2	CB.EN.U4CSE17441	NISHANTH M	CSE E
3	CB.EN.U4CSE17457	SRIHARI S	CSE E
4	CB.EN.U4CSE17468	BHASKAR REDDY Y	CSE E

Project Guide : Dr B Rajathilagam

Problem Definition

To present the details of design and implementation of gesture-controlled game application using an optimized deep learning and neural network model.

Previous Review Comments

- Everybody should be contributing to building deep learning model which is used for interpreting human hand gestures
- Technical content in PPT is lacking
- What makes our CNN model yield better results than previous works

Actions taken

- Everybody has equally contributed to building deep learning model which is used for interpreting human hand gestures
- Added technical content in PPT
- Training a convolutional neural network on raw images will probably lead to bad classification performances, so pre-processing is carried out before constructing the model and the CNN model we have proposed has Conv3D and ConvLSTM layer to extract and learn both spatial features as well as temporal features respectively from the video.

Challenges Ahead

Gesture based controls is a thriving domain in gaming environment, but the devices use a high-resolution cameras to capture gesture and use it for controlling the system. There are several challenges in gesture recognition based on the video streams:

1. Handling the massive dataset which contains 148,092 small videos of various gestures. Each of this small clips has 37 to 40 frames, that sums up to 5479404 frames so there occurs starved GPU problem during training phase of the model.

2. Using a low-resolution camera to capture image frames from the video stream.

Starved GPU problem

The utilization of the GPUs on any machine will be often fluctuating between being utilized and completely being idle, resulting starving of the GPUs. This is a term coined in the CPU context, describes the fact that we are unable to saturate the compute unit (CPU/GPU) with data.

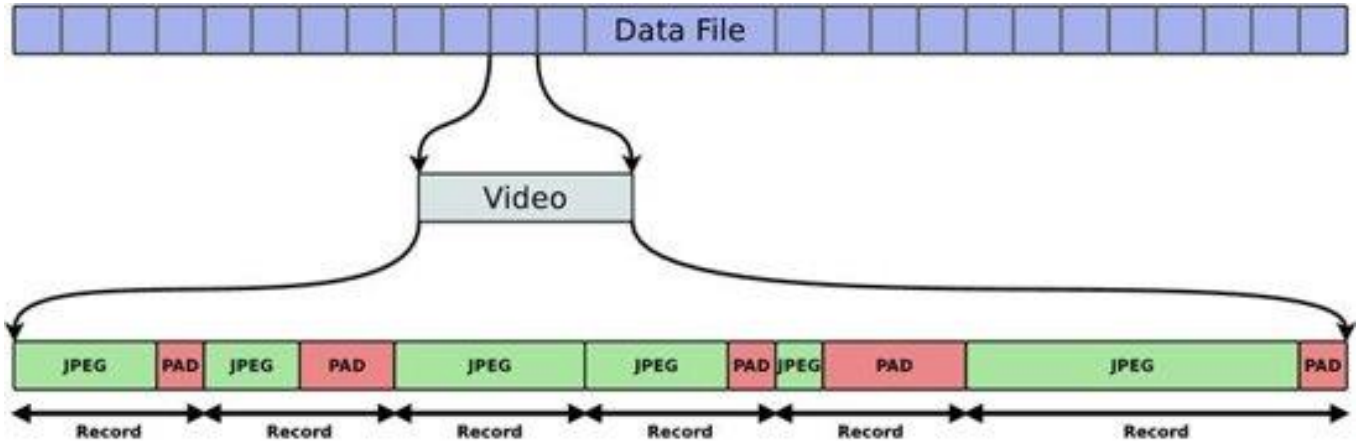
Dataset used in the project: TwentyBN Jester Dataset

- Deep learning compute-resources of most of the TwentyBN's are equipped with around 128 to 512 GB of RAM.
- Training will be very slow during the first epoch. After that, all the images will be cached in memory and the later epocs will no longer need to access the disk to load samples. Eventually cache becomes full.

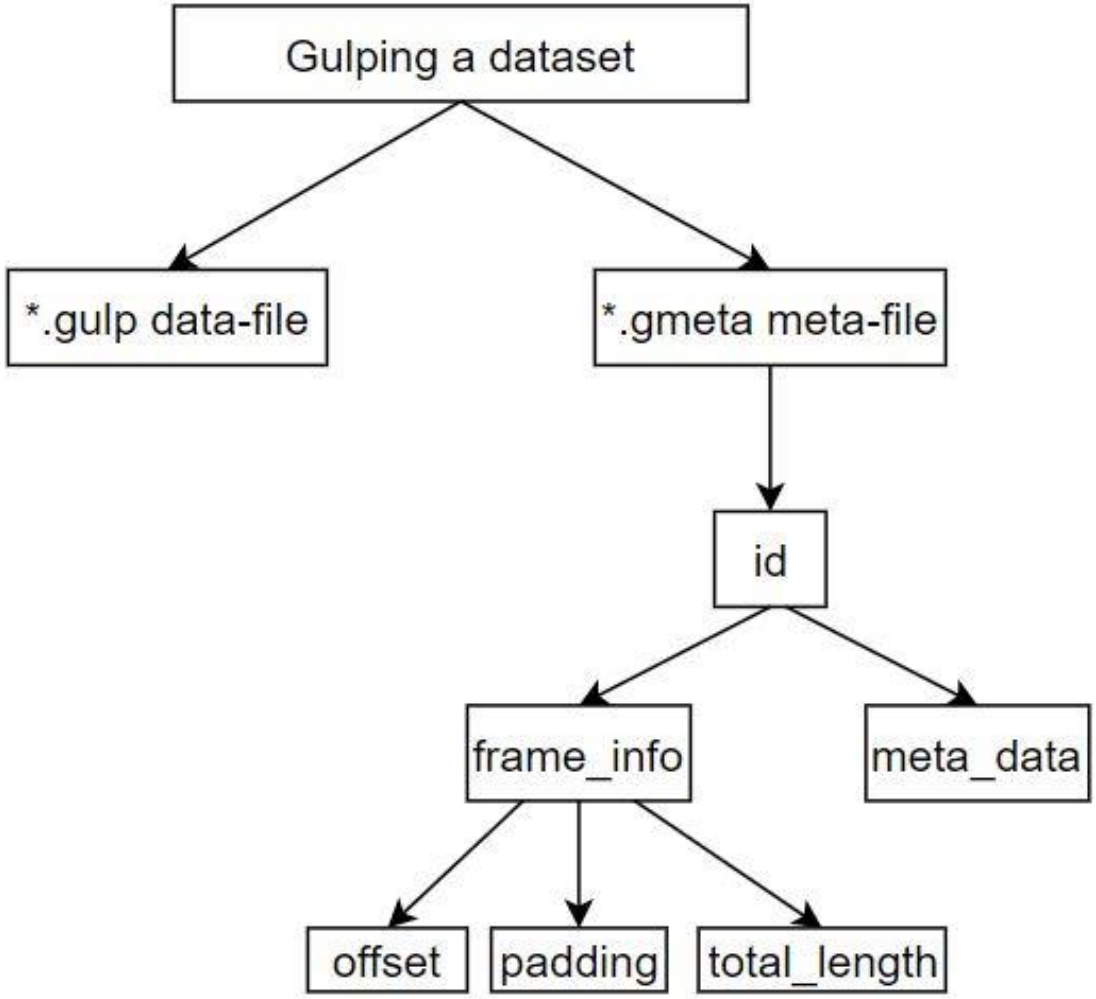
Proposed method to overcome starved GPU problem

Use of Gulp for starved GPU problem:

Gulp is a toolkit to automate & enhance the workflow. An important feature of why we use Gulp for better utilization of the GPUs is that it supports fast data loading from disk to memory.



Multiple videos can be stored in a single *gulp* chunk. This means when gulping a dataset, we can end up with a handful of around 20 or so gulp chunks.


Gulping a dataset
*.gulp data-file
*.gmeta meta-file
id
frame_info
meta_data
offset
padding
total_length

Problems of using low resolution camera

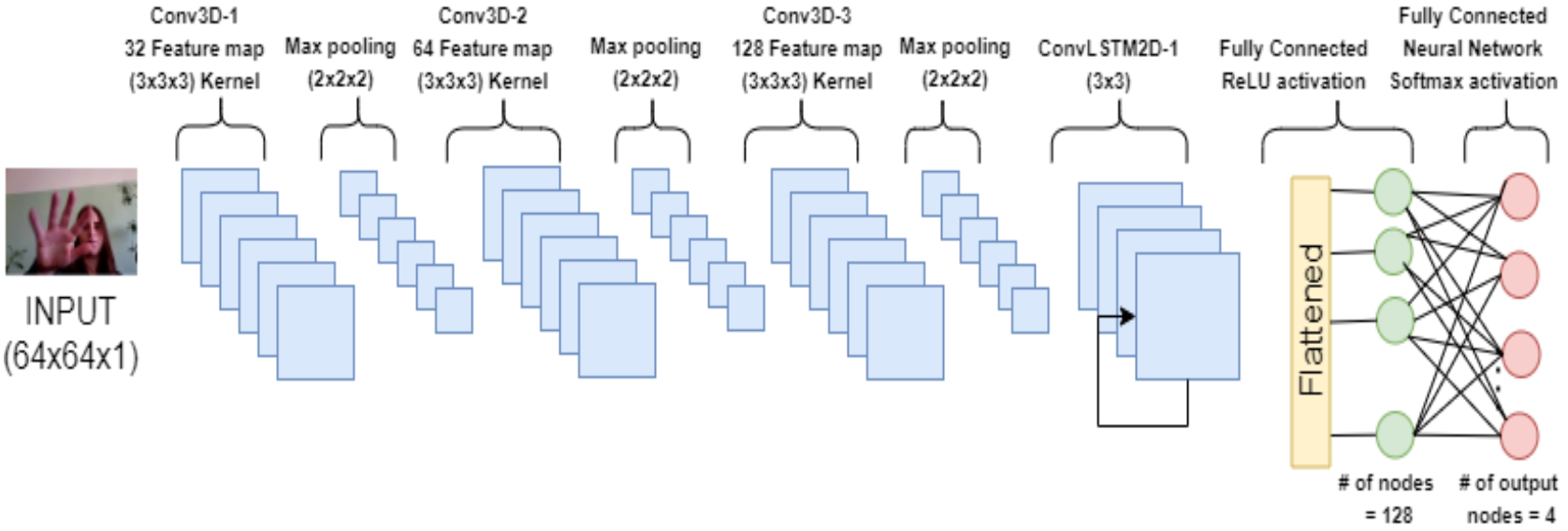
The existing solution proposed in "An efficient method for human hand gesture detection and recognition using deep learning convolutional neural networks" uses a process flow that consists of hand region of interest segmentation using mask image, fingers segmentation, normalization of segmented finger image and finger recognition using CNN classifier. This approach uses CNN to detect gestures based on number of fingers rather than the gesture.

In "Hand Gesture Recognition With 3D Convolutional Neural Networks", they propose an algorithm for driver's hand gesture recognition from challenging depth and intensity data using 3D convolutional neural networks. Their solution combines information from multiple spatial scales for the final prediction.

Solution to handle low resolution image frames

We propose a solution to use all the frames of a video to pass it into a CNN which has convLSTM layer , it is similar to an LSTM layer, but the input transformations and recurrent transformations are both convolutional. And also, we intend to use conv3d layer for spatial convolution over volumes. Thus, not only allowing us to predict the gesture but also the direction of movement(left to right, top to bottom), which in turn makes the gesture recognition system more robust.

Architecture Diagram



Explanation of Architecture diagram

- Two 3D convolutional layers.
- Each convolutional layer is followed by a max-pooling layer.
- One convolutional LSTM layer.
- One Flatten Layer.
- Two fully connected layers with 128 and 5 nodes respectively, following the flatten layer.
- SoftMax activation at the end to predict the class probabilities for 5 gesture classes.

श्रद्धावान् लभते ज्ञानम्

Modules

- Module 1: Handling the dataset
- Module 2: Constructing a gesture recognition model
- Module 3: Deployment of the model

Module 1 description

Importing data

- Jester dataset has 148,092 small videos of various gestures.
- Each of this small clips has 37 to 40 frames, that sums up to 5479404 frames.
- This dataset has CSV file with columns as filename and its corresponding gesture. While using our personal computer, often we will not have the performance needed to work on all this data. To simplify we will have to import only those gestures corresponding to the game that we have chosen.

Preprocessing of the video

Preprocessing is one of the most important steps in our project. We will do a multi-staged preprocessing.

- Stage 1 : RGB to Gray scale conversion

Convert all the frames from color images to black and white.

- Stage 2 : Unifying Frames

The videos do not have the same number of frames, here we try to unify.

For each gesture we try to duplicate last frame if video is shorter than necessary, else if there are more frames, then sample starting offset.

Module 1 description - continued

- Stage 3 : Resize the frames

Resize all the frames to a fixed height and width.

- Stage 4 : Normalization

Normalize each frame. Data normalization is an important step which ensures that each input parameter (pixel, in this case) has a similar data distribution. This makes convergence faster while training the network.

Labels and Loading Data

We consider only the following four gestures necessary for the game we have considered. So from the dataset containing 27 gestures we extract the image frames corresponding to these 4 gestures

```
LABELS = {
    "Swiping Right": 0,
    "Swiping Left": 1,
    "Stop Sign": 2,
    "Thumb Up": 3,
}
```

Loading Data

```
BASE_PATH = 'D:\Jester'
TRAIN_DATA_CSV = BASE_PATH + '/Train.csv'
TEST_DATA_CSV = BASE_PATH + '/Test.csv'
VAL_DATA_CSV = BASE_PATH + '/Validation.csv'

TRAIN_SAMPLES_PATH = BASE_PATH + '/Train/'
TEST_SAMPLES_PATH = BASE_PATH + '/Test/'
VAL_SAMPLES_PATH = BASE_PATH + '/Validation/'
```

```
train_csv=pd.read_csv(TRAIN_DATA_CSV)
```

```
train_csv.head(10)
```

	video_id	label	frames	label_id	shape	format
0	1	Doing other things	37	0	(100, 176)	JPEG
1	3	Pushing Two Fingers Away	37	6	(100, 176)	JPEG
2	6	Drumming Fingers	37	1	(100, 176)	JPEG
3	11	Sliding Two Fingers Down	37	10	(100, 176)	JPEG
4	14	Pushing Hand Away	37	5	(100, 176)	JPEG
5	17	Shaking Hand	37	9	(100, 176)	JPEG
6	20	Doing other things	37	0	(100, 176)	JPEG
7	28	Pulling Two Fingers In	37	4	(100, 176)	JPEG
8	31	Stop Sign	37	14	(100, 176)	JPEG
9	34	Zooming In With Two Fingers	37	24	(100, 176)	JPEG

Preprocessing – Part 1

Stage 1 : RGB to grayscale conversion

Stage 2 : Resizing the image frames

Stage 3 : Unifying the number of frames in each folder

Preprocessing

```python
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])
```

```python
def resize_frame(frame):
    frame = img.imread(frame)
    frame = cv2.resize(frame, (64, 64))
    return frame
```

```python
hm_frames = 30 # number of frames
def get_unify_frames(path):
    offset = 0
    # pick frames
    frames = os.listdir(path)
    frames_count = len(frames)
    # unify number of frames
    if hm_frames > frames_count:
        # duplicate last frame if video is shorter than necessary
        frames += [frames[-1]] * (hm_frames - frames_count)
    elif hm_frames < frames_count:
        # If there are more frames, then sample starting offset
        # diff = (frames_count - hm_frames)
        # offset = diff-1
        frames = frames[0:hm_frames]
    return frames
```

Preprocessing – Part 2

Stage 4 : Adjusting data, feature scaling and normalization of image frames

```
# Adjust training data
train_targets = [] # training targets
test_targets = [] # testing targets

new_frames = [] # training data after resize & unify
new_frames_test = [] # testing data after resize & unify

for idx, row in tqdm(targets.iterrows(), total=len(targets)):
    if idx % 4 == 0:
        continue

    partition = []
    # Frames in each folder
    frames = get_unify_frames(TRAIN_SAMPLES_PATH + str(row['video_id']))
    if len(frames) == hm_frames:
        for frame in frames:
            frame = resize_frame(TRAIN_SAMPLES_PATH + str(row['video_id']) + '/' + frame)
            partition.append(rgb2gray(frame))
            if len(partition) == 15: # partition each training on two trainings.
                if idx % 6 == 0:
                    new_frames_test.append(partition) # append each partition to training data
                    test_targets.append(row['label'])
                else:
                    new_frames.append(partition) # append each partition to test data
                    train_targets.append(row['label'])
                partition = []

train_data = np.asarray(new_frames, dtype=np.float16)
del new_frames[:]
del new_frames

test_data = np.asarray(new_frames_test, dtype=np.float16)
del new_frames_test[:]
del new_frames_test

gc.collect()
```

Preprocessing – Part 2 (contd.,)

```
print(f"Training = {len(train_data)}/{len(train_targets)} samples/labels")
print(f"Test = {len(test_data)}/{len(test_targets)} samples/labels")
print(f"Validation = {len(cv_data)}/{len(cv_targets)} samples/labels")
```

```
Training = 9538/9538 samples/labels
Test = 1192/1192 samples/labels
Validation = 1462/1462 samples/labels
```

Feature scaling

```
# Normalisation: training
print('old mean', train_data.mean())

scaler = StandardScaler(copy=False)
scaled_images  = scaler.fit_transform(train_data.reshape(-1, 15*64*64))
del train_data
print('new mean', scaled_images.mean())

scaled_images  = scaled_images.reshape(-1, 15, 64, 64, 1)
print(scaled_images.shape)
```

```
old mean 115.6
new mean -2.4e-07
(9538, 15, 64, 64, 1)
```

```
x_train
```

```
array([[[[[-7.1436e-01],
          [-5.3711e-01],
          [-5.6445e-01],
          ...,
          [-1.0010e+00],
          [-9.9316e-01],
          [-9.7559e-01]],

         [[-6.5234e-01],
          [-5.5029e-01],
          [-5.6152e-01],
          ...,
          [-9.9902e-01],
          [-1.0820e+00],
          [-9.8340e-01]],
```

Module 2 description

Constructing a model

Construct a custom model, where we use 3D convolutional layer, Max pooling layer, Convolutional LSTM layer (to learn from the sequence of frames), flatten layer, dropout layer and dense layer. Then, we will have to do hyperparameter tuning, as this will directly impact the performance of the model.

Compiling the model

Compile the model using loss as sparse categorical cross entropy, then go about trying various optimizers for our model.

Module 2 description - continued

Training the model
Train the model with the training data and the appropriate parameters.

Validating the model
Use the validation data to analyze the performance of the model and make the further modifications to make the right model.

Saving the weights and architecture
Save the weights and architecture of this trained model to be used in our driver application.

Conv3D Model

```
class Conv3DModel(tf.keras.Model):

  def __init__(self):
      super().__init__()

      # Convolutions
      self.conv1 = tf.compat.v2.keras.layers.Conv3D(32, (3, 3, 3), activation='relu', name="conv1", data_format='channels_last')
      self.pool1 = tf.keras.layers.MaxPool3D(pool_size=(2, 2, 2), data_format='channels_last')
      self.conv2 = tf.compat.v2.keras.layers.Conv3D(64, (3, 3, 3), activation='relu', name="conv2", data_format='channels_last')
      self.pool2 = tf.keras.layers.MaxPool3D(pool_size=(2, 2,2), data_format='channels_last')
      self.conv3 = tf.compat.v2.keras.layers.Conv3D(128, (3, 3, 3), activation='relu', name="conv3", data_format='channels_last')
      self.pool3 = tf.keras.layers.MaxPool3D(pool_size=(2, 2,2), data_format='channels_last')

      # LSTM & Flatten
      self.convLSTM =tf.keras.layers.ConvLSTM2D(40, (3, 3))
      self.flatten =  tf.keras.layers.Flatten(name="flatten")

      # Dense layers
      self.d1 = tf.keras.layers.Dense(128, activation='relu', name="d1")
      self.out = tf.keras.layers.Dense(4, activation='softmax', name="output")

  def call(self, x):
      x = self.conv1(x)
      x = self.pool1(x)
      x = self.conv2(x)
      x = self.conv3(x)
      x = self.pool3(x)
      x = self.convLSTM(x)
      x = self.flatten(x)
      x = self.d1(x)
      return self.out(x)
```

Model compilation and Saving the weights

```
model = Conv3DModel()
```

```
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics = ['accuracy'])
```

```
history = model.fit(x_train, y_train,
                    validation_data=(x_val, y_val),
                    batch_size=32,
                    epochs=5)
```

```
Train on 9538 samples, validate on 1462 samples
Epoch 1/5
9538/9538 [==============================] - 891s 93ms/sample - loss: 1.1885 - accuracy: 0.4234 - val_loss: 0.9447 - val_accura
cy: 0.6293
Epoch 2/5
9538/9538 [==============================] - 837s 88ms/sample - loss: 0.7349 - accuracy: 0.7031 - val_loss: 0.7059 - val_accura
cy: 0.7100
Epoch 3/5
9538/9538 [==============================] - 1451s 152ms/sample - loss: 0.6194 - accuracy: 0.7535 - val_loss: 0.6080 - val_accu
racy: 0.7585
Epoch 4/5
9538/9538 [==============================] - 3675s 385ms/sample - loss: 0.5281 - accuracy: 0.7901 - val_loss: 0.6051 - val_accu
racy: 0.7544
Epoch 5/5
9538/9538 [==============================] - 936s 98ms/sample - loss: 0.4615 - accuracy: 0.8183 - val_loss: 0.5973 - val_accura
cy: 0.7811
```

```
model.save_weights('weights/w.tf', save_format='tf')
```

Module 3 Description

Creating an end application

Make a web application using flask which will capture users' gesture and use the model to recognize the gesture and control the game.

Creating a model

Create a model with the same architecture .

Loading the saved weights

Load the weight from the previously saved data into this newly created model.

Module 3 Description - continued

Capturing video through user's camera

Capture the video through user's low-resolution camera and preprocess the frames.

Recognizing the corresponding gesture

Predict the gesture of the current action.

Converting the prediction to a keyboard action

The predicted output will be mapped to a keyboard action, which in turn gets reflected on the game

Third party game under consideration

Application name: Bubble trouble

Keyboard keys used:

A) Up arrow key
B) Down arrow key
C) Left arrow key
D) Spacebar

Implementations done so far

```
train_csv.head(10)
```

	video_id	label	frames	label_id	shape	format
0	1	Doing other things	37	0	(100, 176)	JPEG
1	3	Pushing Two Fingers Away	37	6	(100, 176)	JPEG
2	6	Drumming Fingers	37	1	(100, 176)	JPEG
3	11	Sliding Two Fingers Down	37	10	(100, 176)	JPEG
4	14	Pushing Hand Away	37	5	(100, 176)	JPEG
5	17	Shaking Hand	37	9	(100, 176)	JPEG
6	20	Doing other things	37	0	(100, 176)	JPEG
7	28	Pulling Two Fingers In	37	4	(100, 176)	JPEG
8	31	Stop Sign	37	14	(100, 176)	JPEG
9	34	Zooming In With Two Fingers	37	24	(100, 176)	JPEG

In [8]:
```
targets = pd.read_csv(TRAIN_DATA_CSV)
targets = targets[targets['label'].isin(LABELS.keys())]
targets['label'] = targets['label'].map(LABELS)
targets = targets[['video_id', 'label']]
targets = targets.reset_index()
targets
```

Out[8]:

	index	video_id	label
0	8	31	2
1	16	51	0
2	20	59	2
3	34	95	0
4	35	100	0
...	...	...	...
7149	50404	148053	0
7150	50406	148059	3
7151	50408	148061	0
7152	50410	148070	0
7153	50418	148090	1

Implementations done so far

```
In [23]: x_train
```

```
Out[23]: array([[[[-7.1436e-01],
                  [-5.3711e-01],
                  [-5.6445e-01],
                  ...,
                  [-1.0010e+00],
                  [-9.9316e-01],
                  [-9.7559e-01]],

                 [[-6.5234e-01],
                  [-5.5029e-01],
                  [-5.6152e-01],
                  ...,
                  [-9.9902e-01],
                  [-1.0820e+00],
                  [-9.8340e-01]],

                 [[-6.2988e-01],
                  [-5.5566e-01],
                  [-5.9668e-01],
```

Results obtained so far

In [27]:
```
history = model.fit(x_train, y_train,
                    validation_data=(x_val, y_val),
                    batch_size=32,
                    epochs=5)
```

```
Train on 9538 samples, validate on 1462 samples
Epoch 1/5
9538/9538 [==============================] - 891s 93ms/sample - loss: 1.1885 - accuracy: 0.4234 - val_loss: 0.9447 - val_accura
cy: 0.6293
Epoch 2/5
9538/9538 [==============================] - 837s 88ms/sample - loss: 0.7349 - accuracy: 0.7031 - val_loss: 0.7059 - val_accura
cy: 0.7100
Epoch 3/5
9538/9538 [==============================] - 1451s 152ms/sample - loss: 0.6194 - accuracy: 0.7535 - val_loss: 0.6080 - val_accu
racy: 0.7585
Epoch 4/5
9538/9538 [==============================] - 3675s 385ms/sample - loss: 0.5281 - accuracy: 0.7901 - val_loss: 0.6051 - val_accu
racy: 0.7544
Epoch 5/5
9538/9538 [==============================] - 936s 98ms/sample - loss: 0.4615 - accuracy: 0.8183 - val_loss: 0.5973 - val_accura
cy: 0.7811
```

CONCLUSION

» We have proposed a model to simulate a game application using human gestures with reduced latency and low-resolution camera. This model aims to reduce the latency with-out having the need for a high-resolution camera with help of efficient pre-processing and deep learning-based model architecture. The GulpIO package is playing an important role in overcoming the starved GPU problem by caching the image frames during the training phase of the model. The LSTM layers present in the model help the model to remember and distinguish ambiguous gestures like swipe-left and swipe-right. Sowe have provided a cost-efficient and time-efficient solution to control any application through human gestures.

श्रद्धावान् लभते ज्ञानम्

References (in IEEE format)

- Pavlo Molchanov, Shalini Gupta, Kihwan Kim, and Jan Kautz NVIDIA, Santa Clara, California, USA ,Hand Gesture Recognition with 3D ConvolutionalNeuralNetworks, https://research.nvidia.com/sites/default/files/pubs/2015-06_Hand-Gesture-Recognition/CVPRW2015-3DCNN.pdf
- V. John, A. Boyali, S. Mita, M. Imanishi and N. Sanma, "Deep Learning-Based Fast Hand Gesture Recognition Using Representative Frames" *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, Gold Coast, QLD, 2016, pp. 1-8, doi: 10.1109/DICTA.2016.7797030.
- Kim, J.-H., Hong, G.-S., Kim, B.-G., and Dogra, D. P. (2018), DeepGesture: Deep Learning-based Gesture Recognition Scheme using Motion Sensors, doi:10.1016/j.display 2018.08.001.
- Marco Roccetti, Gustavo Marfia, Angelo Semeraro, Playing into the wild: A gesture-based interface for gaming in public spaces, Journal of Visual Communication and Image Representation,Volume 23, Issue 3,2012,Pages 426-440,ISSN 1047-3203, https://doi.org/10.1016/j.jvcir.2011.12.006.
- Neethu, P., R. Suguna and Divya Sathish. “An efficient method for human hand gesture detection and recognition using deep learning convolutional neural networks.” *Soft Computing* (2020): 1-10.

THANK YOU