National College of Ireland

Word-Based American Sign Language Recognition Using 2D CNN with Temporal Shift Module and Attention

MSc Research Project
MSc in Data Analytics

Semanta Das
Student ID: 23338806

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Tomer

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Semanta Das
Student ID:	23338806
Programme:	MSc in Data Analytics **Year:** 2024-25
Module:	Msc Research Project
Supervisor:	Prof. Vikas Tomer
Submission Due Date:	11/08/2025
Project Title:	Word-Based American Sign Language Recognition Using 2D CNN with Temporal Shift Module and Attention
WordCount:	**525**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Semanta Das
Date:	11/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

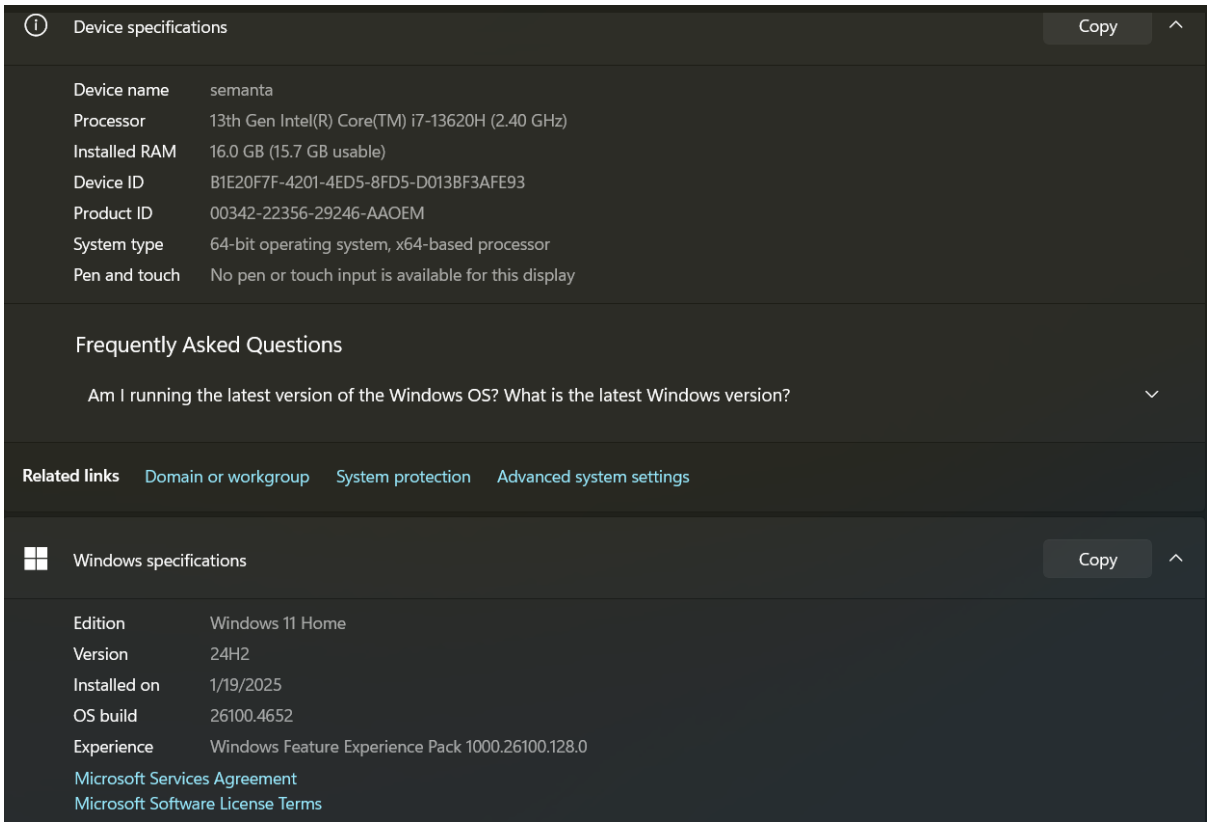
Word-Based American Sign Language Recognition Using 2D CNN with Temporal Shift Module and Attention

Semanta Das
23338806

1) Introduction

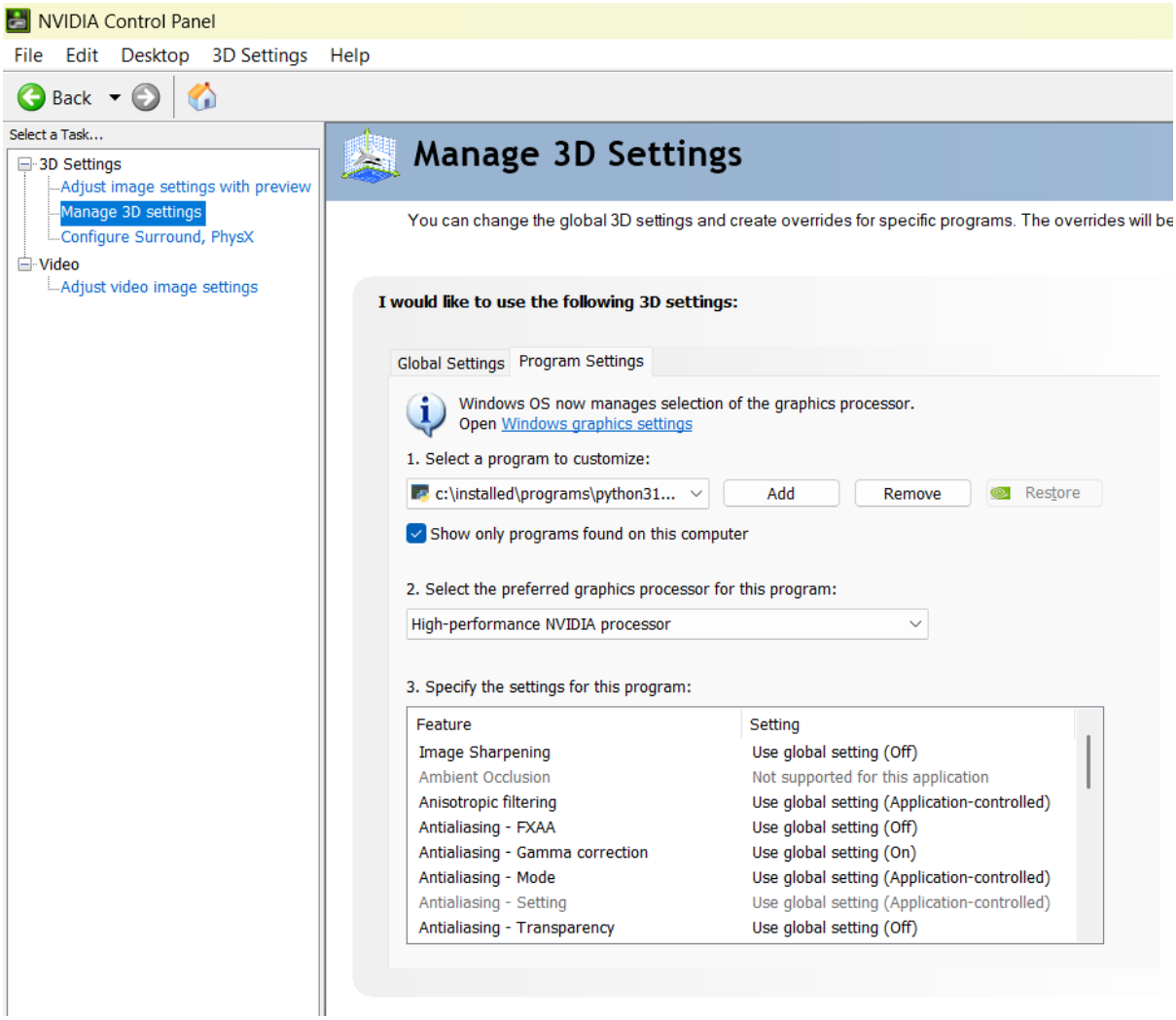
The goal of the document is to mitigate the gap between thesis report paper and code artifact.

2) Hardware Requirements



This is our GPU configuration, Using this GPU we trained, and evaluated our models.

I added this line of code to active my GPU for each python script as every script was involved with doing some with frames.



We need to go to NVDIA CONTROL PANEL then Program settings then need to choose the python exe. We used python language to made our project. In my case I have given the pathway of python 3.12. Thus, we have given full access to GPU to python interpreter. Thus, we are ready to use GPU.

3) Libraries Used:

- Core Python libraries: os, math, time, functools, pathlib (Path), tempfile, collections (deque), typing (Dict, Any, List)

- Data Handling & Processing: json, numpy, tabulate, tqdm
- Image and Video Processing: cv2 (OpenCV), PIL (Image), mediapipe (mp), ffmpeg, yt_dlp
- Machine Learning & Deep Learning: torch, torch.nn (nn), torch.nn.functional (F), torch.utils.data (Dataset, DataLoader, WeightedRandomSampler), torchvision.models (mobilenet_v2), torchvision.transforms (Compose, Resize, RandomResizedCrop, RandomHorizontalFlip, ColorJitter, GaussianBlur, ToTensor, Normalize, RandomErasing)
- Evaluation and Metrics: sklearn.metrics
- Deployment and User Interface: streamlit (st)

.

4) Library installation process

To run this project, the required Python libraries must be installed in the environment. All libraries mentioned in Section 3 can be installed together using a single pip command:

"pip install torch torchvision opencv-python mediapipe numpy tqdm scikit-learn tabulate Pillow streamlit ffmpeg-python yt-dlp"

5) Dataset's link

I used MS-ASL (Microsoft American Sign Language) dataset and collected the dataset form the official site of microsoft. This dataset has more than 25000 annotated videos for 1000 class . I worked with 100 MS-ASL class.

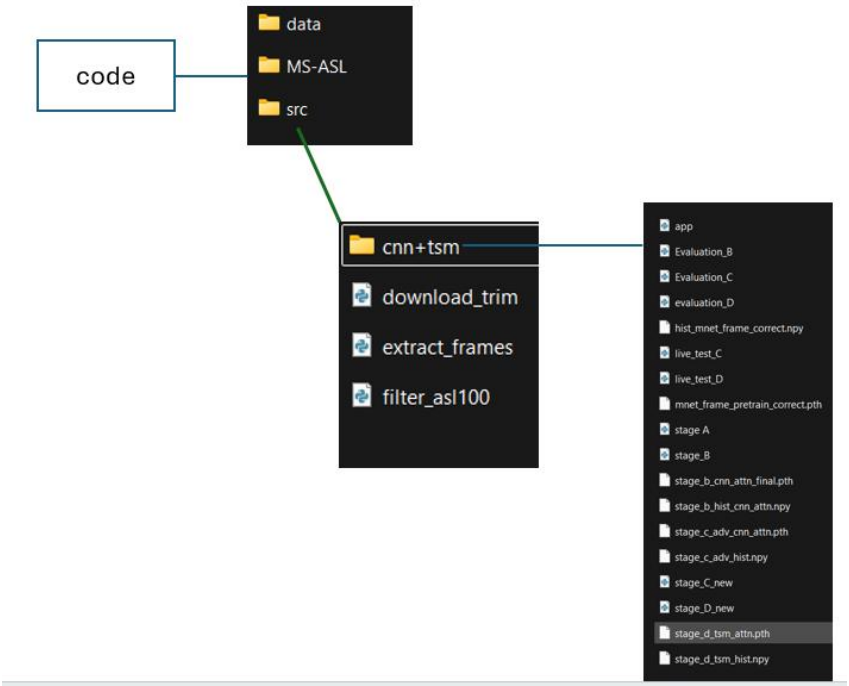
Here is the link of the dataset:
https://www.microsoft.com/en-us/research/project/ms-asl/

6) Software Used

Pycharm IDE with python 3.12

7) Zip folder Overview

Pycharm IDE with python 3.12



In the zip file, I have already added the main data folder where json files are there. I could not add the prepared data as data is huge.

After downloading raw videos, we need trim it then we need to extract frames maintaining 16fps. The size of the extracted sequence frames for all 100 classes were 26 GB, which i have used in this project. This is our main dataset which were used for training and testing our models.

I have given in the zip file all the python script and saved model's weight.

For running the model to see how it works, simply use 'live_test_D.py' script as this model got the best accuracy. This script will load the model and used webcam. Or write in the terminal "streamlit run app.py", which will run a web application in local host. There you upload any MS-ASL100 classes' video, it will give you prediction.

If you want to run full training process, you need to run from the starting using given script.

1. First you need to run 'filter_asl100.py', it will filter the json file and created 3 separate json file which will be stored in the 'data' -> 'list' folder. Such away it will do everything by itself.
2. Run 'download_trim.py'
3. Extract_frames.py

For training the model the models:

4. Run 'stage A'
5. stage_B
6. stage_C_New
7. stage_D_New
8. then evaluation scripts

8) Model Snippet:

Stage A's model:

```
# model
class MobileNetFrame(nn.Module):  1 usage  seman
    def __init__(self, n_cls):  seman
        super().__init__()
        m = mobilenet_v2(weights="IMAGENET1K_V1")
        m.classifier[1] = nn.Linear( in_features: 1280, n_cls)
        self.net = m
    def forward(self, x): return self.net(x)  seman
```

Stage B's model (MobileNet-V2+Attention):

```
################## Model
class TemporalAttn(nn.Module):  3 usages  seman *
    #MobileNet-V2 per-frame -> soft attention -> logits
    def __init__(self, n_cls, emb_dim=256):  seman *
        super().__init__()
        m = mobilenet_v2(weights=None)              ## grab a readymade Mo
        self.backbone = m.features                 ## keeping only the fea
        self.pool  = nn.AdaptiveAvgPool2d(1)
        self.proj  = nn.Linear( in_features: 1280, emb_dim, bias=False) #A sm
        self.bn    = nn.BatchNorm1d(emb_dim)                #BatchNorm: k
        self.attn  = nn.Linear(emb_dim,  out_features: 1)      #A tiny layer
        self.fc    = nn.Linear(emb_dim, n_cls)          ##Final layer that

    def forward(self,x):                                    # x (B,3,T,H,
        B,C,T,H,W = x.shape
        x = x.permute(0,2,1,3,4).reshape(B*T,C,H,W)
        feat = self.pool(self.backbone(x)).flatten(1)       # (B·T,1280)
        feat = F.relu(self.bn(self.proj(feat)))             # (B·T,emb)
        feat = feat.view(B,T,-1)                            # (B,T,emb)
        attn_w = F.softmax(self.attn(feat).squeeze(-1),dim=1) # (B,T)
        emb = (attn_w.unsqueeze(-1)*feat).sum(1)            # (B,emb)
        return self.fc(emb)
```

Stage C(MobileNet-V2 + Attention + Argumentation):

```
class TemporalAttn(nn.Module):  7 usages  seman *
    def __init__(self,n_cls,emb_dim=256):                # in this cons
        super().__init__()
        m = mobilenet_v2(weights=None)                     # grab a rea
        self.backbone = m.features                           # keeping
        self.pool     = nn.AdaptiveAvgPool2d(1)
        self.proj     = nn.Linear( in_features: 1280,emb_dim,bias=False)
        self.bn       = nn.BatchNorm1d(emb_dim)                  #BatchN
        self.dropout  = nn.Dropout(0.5)                       #Randomly tu
        self.attn     = nn.Linear(emb_dim, out_features: 1)   #A tiny laye
        self.fc       = nn.Linear(emb_dim,n_cls)              #Final layer

    def forward(self,x):  seman
        B,C,T,H,W = x.shape
        x = x.permute(0,2,1,3,4).reshape(B*T,C,H,W)
        f = self.pool(self.backbone(x)).flatten(1)
        f = F.relu(self.bn(self.proj(f)))
        f = self.dropout(f).view(B,T,-1)
        w = F.softmax(self.attn(f).squeeze(-1),dim=1).unsqueeze(-1)
        emb = (w*f).sum(1)
        return self.fc(emb)
```

Stage D: (MobileNet-V2+TSM + Attention):

```
#MODEL (TSM + attn)
class TSM_Attn(nn.Module):  9 usages  seman *
    def __init__(self, n_cls, emb_dim=256):  seman *
        super().__init__()
        m = mobilenet_v2(weights=None)
        self.backbone = m.features
        self.shift_div = 8
        self.pool = nn.AdaptiveAvgPool2d(1)
        self.proj = nn.Linear( in_features: 1280, emb_dim, bias=False)
        self.bn   = nn.BatchNorm1d(emb_dim)
        self.attn = nn.Linear(emb_dim,  out_features: 1)
        self.fc   = nn.Linear(emb_dim, n_cls)
    def tsm(self, x):  seman
        B_T, C, H, W = x.shape; B = B_T//NUM_FRAMES
        x = x.view(B, NUM_FRAMES, C, H, W)
        fold = C // self.shift_div; out = torch.zeros_like(x)
        out[:, :-1, :fold] = x[:, 1:, :fold]
        out[:, 1:, fold:2*fold] = x[:, :-1, fold:2*fold]
        out[:, :, 2*fold:] = x[:, :, 2*fold:]
        return out.view(B_T, C, H, W)
    def forward(self, x):  seman
        B,C,T,H,W = x.shape
        x = self.tsm(x.permute(0,2,1,3,4).reshape(B*T, C, H, W))
        f = self.pool(self.backbone(x)).flatten(1)
        f = F.relu(self.bn(self.proj(f))).view(B,T,-1)
        w = F.softmax(self.attn(f).squeeze(-1), dim=1).unsqueeze(-1)
        emb = (w*f).sum(1)
        return self.fc(emb)
```