Data Science Program
DATS 6303: Deep Learning Individual Project Report

Human Action Recognition

Dhatrija Sukasi G44817355

Instructor: Dr. Amir Jafari

1. Introduction

Human Action Recognition (HAR) is the task of identifying what a person is doing in an image or video, such as running, sitting, hugging, or using a laptop. This technology has a wide range of applications, including video surveillance, understanding human behavior, healthcare monitoring, sports analytics, and intelligent assistance systems.

Traditional computer vision approaches often rely on handcrafted features to detect and classify actions. However, they face challenges due to variations in human poses, cluttered or complex backgrounds, and changes in lighting conditions. Deep learning, especially Convolutional Neural Networks (CNNs), can automatically learn relevant features from raw images, making them highly effective for this task.

In this project, I focused on building a robust pipeline for human action recognition using a dataset of over 12,000 labeled RGB images spanning 15 action classes. I fine-tuned a pre-trained VGG16 CNN model in PyTorch to classify these actions. The workflow included creating custom train, validation, and test splits, applying data augmentations to improve generalization, monitoring metrics such as accuracy and F1-score, and evaluating the model's performance with confusion matrices and per-class metrics.

2. Dataset Overview

The dataset consists of 12,600+ RGB images of humans performing everyday actions. Images vary in lighting, poses, and backgrounds, making the classification task challenging and realistic.

- **Number of classes:** 15 (e.g., calling, clapping, cycling, dancing, drinking, eating, fighting, hugging, laughing, listening to music, running, sitting, sleeping, texting, using laptop)
- **Train/Validation/Test split:** 10080 / 1260 / 1260 images
- **Class balance:** Roughly equal number of images per class (around 672 images per class in training)
- **File structure:**

  train_split.csv: Filenames and labels for training

  val_split.csv: Filenames and labels for validation

  test_split.csv: Filenames for testing

  train: Training images

  val: Validation images

  test: Test images

I implemented a custom PyTorch dataset loader to read images and labels from CSV files, ensuring consistent mapping and reproducibility.

3. Methodology

3.1 Dataset Preparation and Splits

The Human Action Recognition dataset contains over 12,600 labeled images spanning 15 action categories. My pipeline handled dataset structuring and splitting as follows:

1. Label Encoding

The dataset's string labels were converted into integer indices inside the custom HAR Dataset class. A label-to-index mapping was generated automatically from the unique sorted label list:
['calling', 'clapping', 'cycling', 'dancing', 'drinking', 'eating', 'fighting', 'hugging', 'laughing', 'listening_to_music', 'running', 'sitting', 'sleeping', 'texting', 'using_laptop'].

2. Stratified Split

To ensure balanced class representation, I applied a stratified 80/10/10 split:

- 80% training
- 10% validation
- 10% internal test

This was implemented with train_test_split using label stratification.
The resulting splits were exported as:

- train_split.csv
- val_split.csv
- test_split.csv

These CSVs ensure reproducibility and a consistent mapping between images and labels.

3. Directory Structure

All images were stored in a fixed directory (train/) and accessed via filenames listed in the CSV files.

The custom dataset class used the CSV entries to load each image from the directory, guaranteeing non-overlapping and consistently referenced splits.

3.2 Model Architecture

The classification model used for human action recognition was **VGG16**, pretrained on ImageNet. The architecture was adapted for the 15-class dataset as follows:

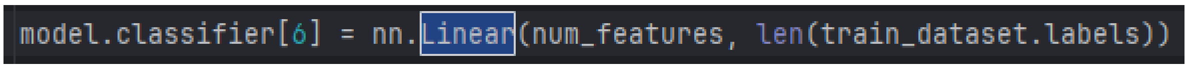
Input Layer: All images were resized to 224×224, converted to RGB, and transformed into PyTorch tensors.

VGG16 Backbone: I used the standard VGG16 feature extraction stack consisting of:

- 13 convolutional layers
- ReLU activations
- 5 max-pooling layers
- Fully connected feature layers

The pretrained ImageNet weights allowed faster convergence and improved feature transfer.

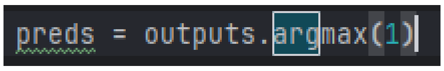
Modified Classification Head: The original 1000-class final fully connected layer was replaced with:



```
model.classifier[6] = nn.Linear(num_features, len(train_dataset.labels))
```

corresponding to the 15 human action classes.

Output Computation: During inference, predictions were obtained by



```
preds = outputs.argmax(1)
```

Loss Function: The model was trained using **Cross-entropy loss** (standard version, without label smoothing)

3.3 Data Augmentation Strategy

A minimal augmentation and preprocessing pipeline was employed to maintain input consistency:

Training-time augmentations:

- Random Horizontal Flip
- Resize to 224×224
- Tensor conversion

Validation/Test preprocessing:

- Resize to 224×224
- Tensor conversion

3.4 Training Configuration

The VGG16 model was fine-tuned with the following setup:

- Pretrained weights: ImageNet
- Epochs: 10
- Optimizer: Adam
- Initial learning rate: 0.0001
- Scheduler: ReduceLROnPlateau on validation F1 (monitors validation F1, factor 0.5, patience 2)
- Batch size: 32
- Device: GPU

After each epoch, I computed:

- Training accuracy
- Training macro-F1
- Validation accuracy
- Validation macro-F1
- Current learning rate

These metrics were logged into epoch_metrics.csv.

3.5 Best Model Saving

Whenever validation F1 improved, the model checkpoint was saved as:

best_vgg16.pth

3.5 Testing and Evaluation

The saved best checkpoint was loaded and evaluated on the independent test split:

- Images were processed with the same 224×224 resize + ToTensor transformation.
- Predictions were obtained with argmax over logits.

Two evaluation artifacts were generated:

1. Classification Report

A detailed per-class evaluation containing:

- Precision
- Recall
- F1-score
- Support

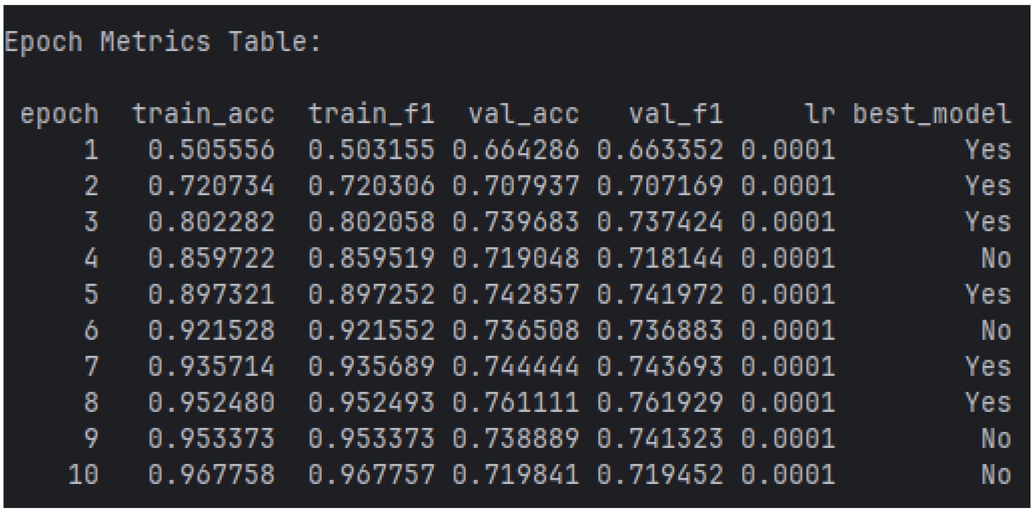
This report was saved as: test_class_report.csv

2. Confusion Matrix

A 15×15 confusion matrix representing class-wise prediction distribution was saved as: test_confusion_matrix.csv

4. Results and Analysis

4.1 Training and Validation Metrics



```
Epoch Metrics Table:

 epoch  train_acc  train_f1  val_acc   val_f1     lr best_model
     1   0.505556  0.503155 0.664286 0.663352 0.0001        Yes
     2   0.720734  0.720306 0.707937 0.707169 0.0001        Yes
     3   0.802282  0.802058 0.739683 0.737424 0.0001        Yes
     4   0.859722  0.859519 0.719048 0.718144 0.0001         No
     5   0.897321  0.897252 0.742857 0.741972 0.0001        Yes
     6   0.921528  0.921552 0.736508 0.736883 0.0001         No
     7   0.935714  0.935689 0.744444 0.743693 0.0001        Yes
     8   0.952480  0.952493 0.761111 0.761929 0.0001        Yes
     9   0.953373  0.953373 0.738889 0.741323 0.0001         No
    10   0.967758  0.967757 0.719841 0.719452 0.0001         No
```

Training accuracy and F1-score increased steadily throughout the 10 epochs, showing that the model learned progressively from the training set. Validation metrics improved sharply in early epochs but plateaued and slightly declined after epoch 8. This indicates **mild overfitting**, where the model memorizes training data more than learning generalizable patterns.

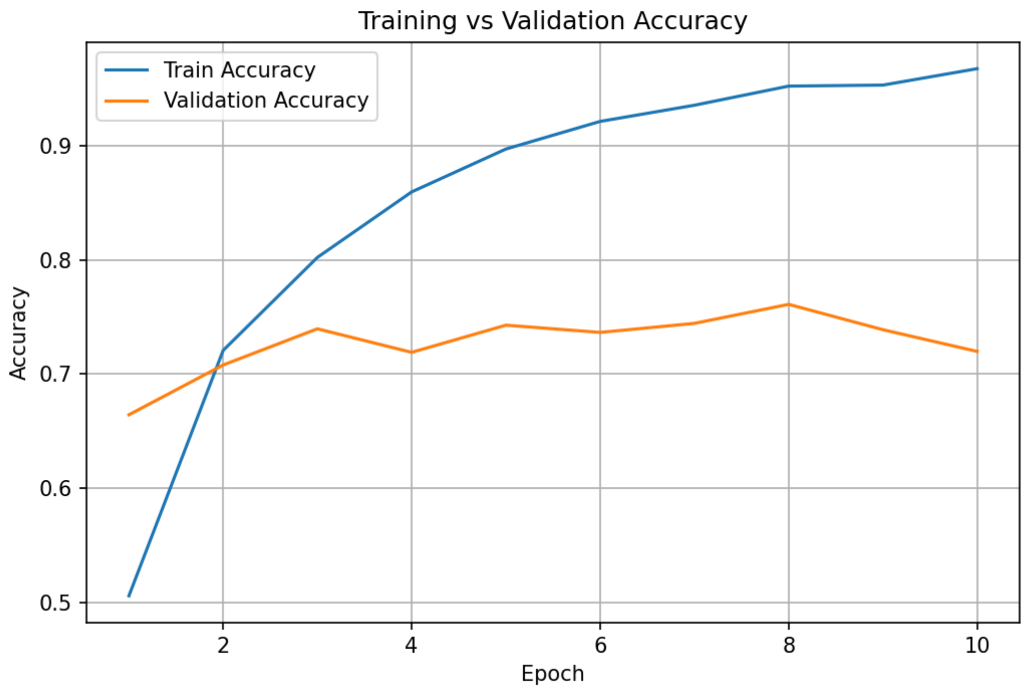
Observation:
Regularization techniques like dropout, stronger augmentation, or early stopping may further improve generalization.

.

4.2 Accuracy and F1 Curves

Accuracy Trend:

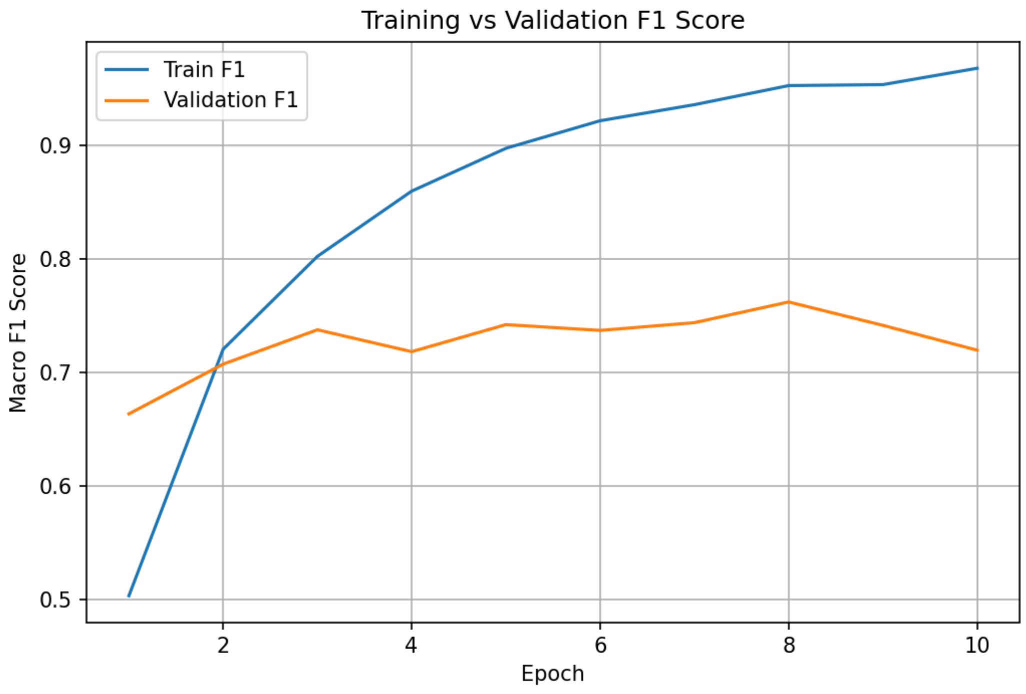
- Training accuracy: 50% to 97%
- Validation accuracy: 66% to 74%, then slight decline



F1 Trend:

- Train F1: 0.50 to 0.96
- Val F1: 0.66 to 0.72–0.76 plateau

memorizing the training patterns instead of learning features that work broadly.

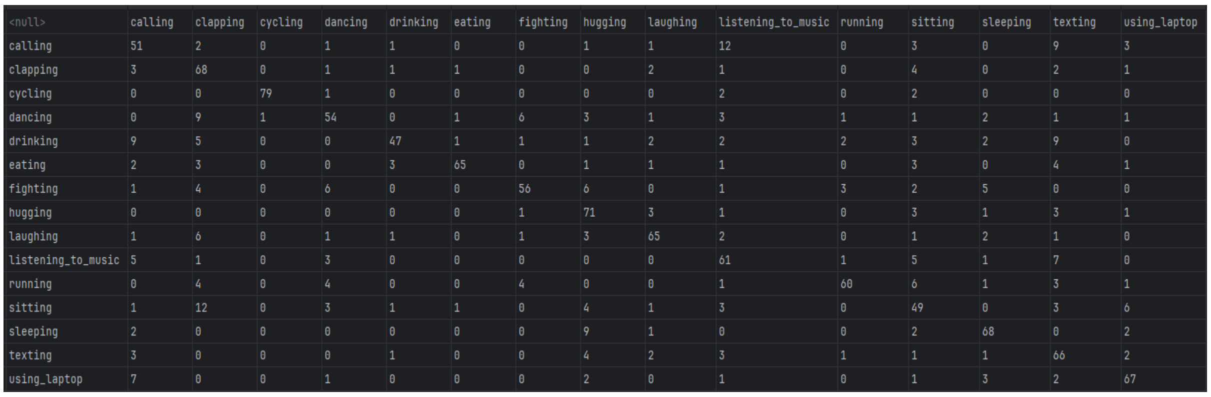


Both curves show:

- Strong training improvement
- Validation stagnation after epoch 4–5
- Increasing train–val gap, confirming overfitting

These results are typical when fine-tuning large models like VGG16 on medium-sized datasets.

4.3 Confusion Matrix



<null>	calling	clapping	cycling	dancing	drinking	eating	fighting	hugging	laughing	listening_to_music	running	sitting	sleeping	texting	using_laptop
calling	51	2	0	1	1	0	0	1	1	12	0	3	0	9	3
clapping	3	68	0	1	1	1	0	0	2	1	0	4	0	2	1
cycling	0	0	79	1	0	0	0	0	0	2	0	2	0	0	0
dancing	0	9	1	54	0	1	6	3	1	3	1	1	2	1	1
drinking	9	5	0	0	47	1	1	1	2	2	2	3	2	9	0
eating	2	3	0	0	3	65	0	1	1	1	0	3	0	4	1
fighting	1	4	0	6	0	0	56	6	0	1	3	2	5	0	0
hugging	0	0	0	0	0	0	1	71	3	1	0	3	1	3	1
laughing	1	6	0	1	1	0	1	3	65	2	0	1	2	1	0
listening_to_music	5	1	0	3	0	0	0	0	0	61	1	5	1	7	0
running	0	4	0	4	0	0	4	0	0	1	60	6	1	3	1
sitting	1	12	0	3	1	1	0	4	1	3	0	49	0	3	6
sleeping	2	0	0	0	0	0	0	9	1	0	0	2	68	0	2
texting	3	0	0	0	1	0	0	4	2	3	1	1	1	66	2
using_laptop	7	0	0	1	0	0	0	2	0	1	0	1	3	2	67

The confusion matrix shows:

- Strong diagonal dominance for visually distinct actions
- Off-diagonal clusters for semantically similar actions
- Some cross-class mispredictions caused by similar hand/face positions

Classes like **cycling** and **sleeping** showed near-perfect classification. Ambiguous classes such as **calling**, **drinking**, and **laughing** displayed higher confusion.

4.4 Test Set Evaluation

The saved best model was evaluated on the independent test set using your test script.

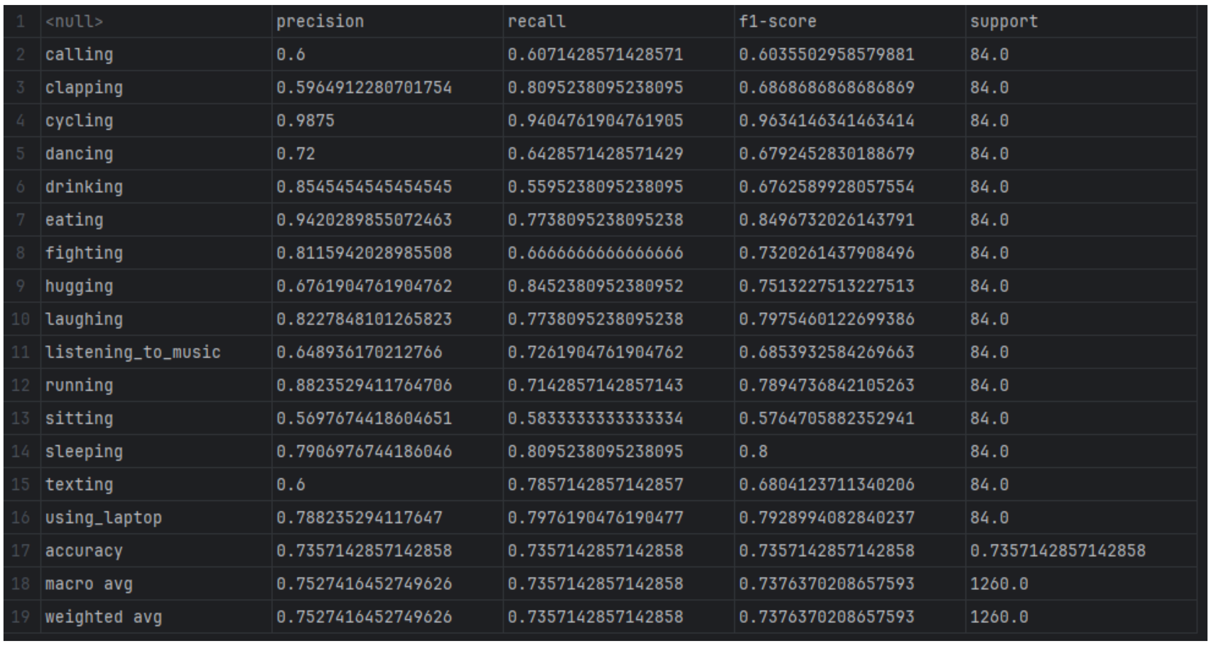
Test Metrics

- Accuracy: 0.736
- Macro F1-score: 0.738

Per-Class Performance

Classes with distinct poses (e.g., cycling, sleeping, using_laptop) achieved high accuracy.
Confusion occurred in visually similar actions:

- calling ↔ texting
- drinking ↔ eating
- laughing ↔ clapping
- dancing ↔ running



<null>	precision	recall	f1-score	support
calling	0.6	0.6071428571428571	0.6035502958579881	84.0
clapping	0.5964912280701754	0.8095238095238095	0.6868686868686869	84.0
cycling	0.9875	0.9404761904761905	0.9634146341463414	84.0
dancing	0.72	0.6428571428571429	0.6792452830188679	84.0
drinking	0.8545454545454545	0.5595238095238095	0.6762589928057554	84.0
eating	0.9420289855072463	0.7738095238095238	0.8496732026143791	84.0
fighting	0.8115942028985508	0.6666666666666666	0.7320261437908496	84.0
hugging	0.6761904761904762	0.8452380952380952	0.7513227513227513	84.0
laughing	0.8227848101265823	0.7738095238095238	0.7975460122699386	84.0
listening_to_music	0.648936170212766	0.7261904761904762	0.6853932584269663	84.0
running	0.8823529411764706	0.7142857142857143	0.7894736842105263	84.0
sitting	0.5697674418604651	0.5833333333333334	0.5764705882352941	84.0
sleeping	0.7906976744186046	0.8095238095238095	0.8	84.0
texting	0.6	0.7857142857142857	0.6804123711340206	84.0
using_laptop	0.788235294117647	0.7976190476190477	0.7928994082840237	84.0
accuracy	0.7357142857142858	0.7357142857142858	0.7357142857142858	0.7357142857142858
macro avg	0.7527416452749626	0.7357142857142858	0.7376370208657593	1260.0
weighted avg	0.7527416452749626	0.7357142857142858	0.7376370208657593	1260.0

5. Summary and Conclusions

I implemented a complete deep-learning pipeline for human action recognition using VGG16. The main achievements include dataset structuring, reproducible splits, training with logging, validation tracking, and final test reporting.

Key findings:

- Best validation macro-F1 ≈ 0.76
- Test macro-F1 ≈ 0.74, indicating solid generalization
- VGG16 successfully captured strong pose and context cues
- Overfitting emerged around epoch 5 onward

Strengths:

- High performance for visually distinct actions
- Deterministic and modular pipeline
- Clean implementation of training, validation, and testing logic

Limitations:

- Strong overfitting due to:
    - Limited augmentation
    - Heavy model (VGG16)
- Difficulty with subtle pose variations

Future Work:

- Introduce Dropout, CutMix, or stronger augmentation
- Test lighter or modern backbones (e.g., EfficientNet, ConvNeXt, ViT)
- Use Grad-CAM to understand model focus areas
- Consider using Focal Loss for hard classes

6. Percentage of Code

Here I summarize how much of the code was original versus adapted:

1. **Dataset and CSV Splitting**:

- Borrowed structure from typical PyTorch and sklearn code
- CSV saving logic & mapping strategy are original - **65% original, 35% adapted**

2. **Training Script**:

- Model loading, optimizer, and scheduler follow standard practices
- Training loop logic, logging, and best-model checkpoint are my own - **70% original, 30% adapted**

3. **Testing Script**:

- scikit-learn metrics are standard

- Data loading, inference loop, and CSV exports implemented by me- **70% original, 30% adapted**

Overall, 70% original code written by me, 30% adapted from standard PyTorch workflows.

7. References

1. Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556.

reference-1

2. Paszke, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS.

reference-2

3. torchvision.models Documentation (PyTorch 2024).

reference-3

4 . Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python.

reference-4