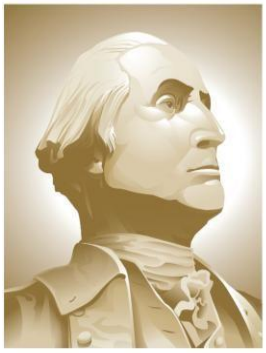


THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

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 dataset consisted of 12,600+ labelled images distributed among 15 action classes. My preprocessing included:

1. **Label Mapping**: Converted string labels into integer indices: ['calling', 'clapping', 'cycling', 'dancing', 'drinking', 'eating', 'fighting', 'hugging', 'laughing', 'listening_to_music', 'running', 'sitting', 'sleeping', 'texting', 'using_laptop'].
2. **Stratified Splitting**: Used 80% train, 10% validation, 10% internal test, ensuring balanced class representation. The splits were saved as CSVs (train_split.csv, val_split.csv, test_split.csv) to guarantee reproducibility.
3. **Directory Structure**: All images were stored in train/ and test/, with CSVs pointing to their filenames. This structure ensured non-overlapping sets and consistent label distribution for training, validation, and testing.

3.2 Model Architecture

The model was **VGG16 pretrained on ImageNet**, modified for 15-class human action recognition. Key points:

1. **Input Layer:** RGB images resized to 224×224 and normalized using ImageNet statistics.
2. **VGG16 Backbone:** 13 convolutional layers with ReLU activations, 5 max-pooling layers, and fully connected layers for feature extraction.
3. **Modified Classification Head:** The final 1000-class FC layer was replaced with a **15-unit linear layer**, corresponding to the 15 action classes.
4. **Softmax Activation:** Applied at evaluation time for class probabilities.
5. **Loss Function:** Cross-entropy with label smoothing to reduce overconfidence in predictions.

3.3 Data Augmentation Strategy

To improve robustness against real-world variations in pose, lighting, and background, I implemented:

1. Standard Image Transforms (Training Only):
    - Random resized crop (224×224)
    - Horizontal flip
    - Color jitter (brightness, contrast, saturation, hue)
    - Normalization using ImageNet mean and std
2. MixUp: Linearly blends two images and their labels using a Beta-distributed coefficient. Improves generalization and reduces overfitting on ambiguous samples.
3. Mosaic: Combines four images into a 2×2 grid; the label of the top-left image is used as the target. Exposes the model to richer spatial combinations and reduces sensitivity to localized features.

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 (factor 0.5, patience 2)
- Batch size: 32
- Loss: Cross-entropy with label smoothing, log-softmax for MixUp/Mosaic batches
- Device: GPU

At the end of each epoch, training and validation metrics were logged. The best model checkpoint was saved whenever validation macro-F1 improved, resulting in best_vgg16.pth.

3.6 Evaluation and Post-Processing

After training, I implemented an evaluation pipeline focused on interpretability and reproducibility:

1. **Checkpoint Loading:** Re-instantiated VGG16 with 15-class head and loaded the best checkpoint.
2. **Metric Computation:** On the internal test split (~1,260 images), computed:
    - Accuracy
    - Macro precision, recall, and F1-score
    - Per-class precision, recall, and F1
    - Confusion matrix (15×15)
3. **Prediction Export:** Generated CSV files with predictions, true labels, and confusion matrices. These files allowed detailed analysis of misclassifications and preparation of plots for accuracy curves, F1 curves, and per-class F1 visualizations.

4. Results and Analysis

4.1 Training and Validation Metrics

Epoch Metrics Table:

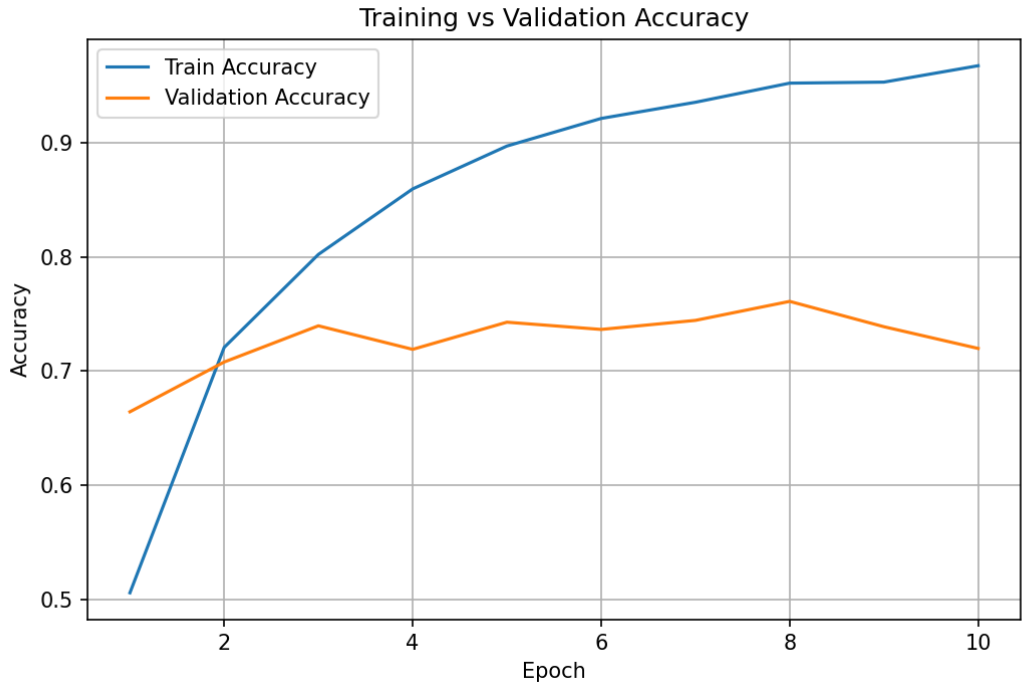
epoch	train_acc	train_f1	val_acc	val_f1	lr	best_model
1	0.505556	0.503155	0.664286	0.663352	0.0001	Yes
2	0.720734	0.720306	0.707937	0.707169	0.0001	Yes
3	0.802282	0.802058	0.739683	0.737424	0.0001	Yes
4	0.859722	0.859519	0.719048	0.718144	0.0001	No
5	0.897321	0.897252	0.742857	0.741972	0.0001	Yes
6	0.921528	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

During training, the model's accuracy and F1-score steadily improved on the training set. The validation F1-score also increased initially but peaked around

epoch 8 and then slightly declined, indicating mild overfitting. This suggests the model started memorizing training data rather than fully generalizing. The gap between training and validation metrics highlights that performance on unseen data may be lower. Applying techniques like dropout, data augmentation, or early stopping could improve generalization. Overall, the model learned useful features but needs careful tuning to balance learning and generalization.

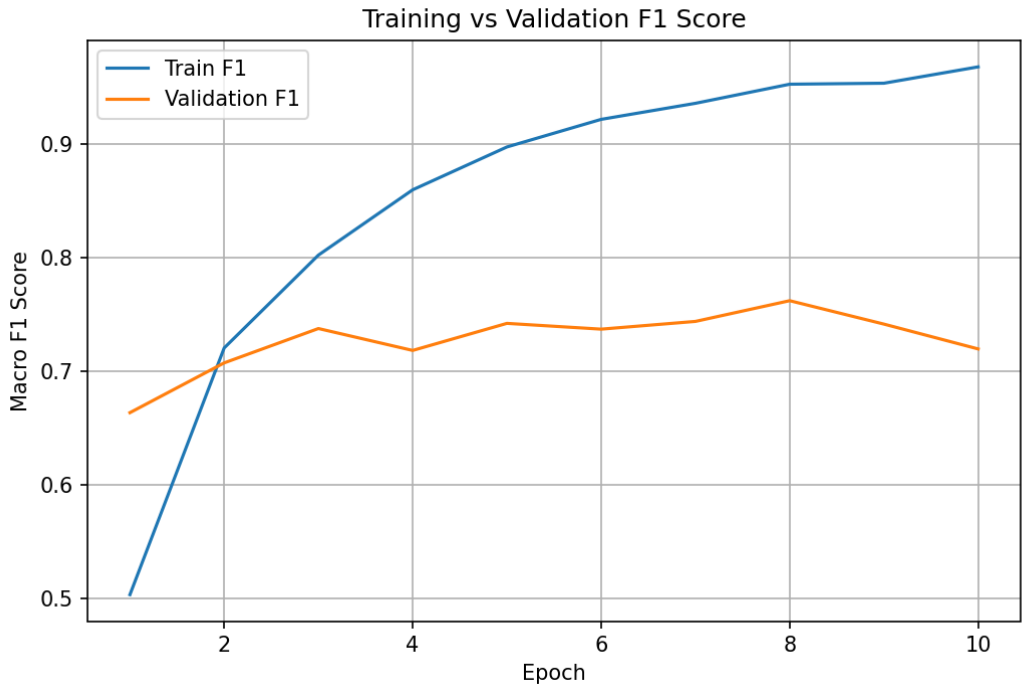
4.2 Accuracy and F1 Curves

The training accuracy increased steadily from around 50% in the first epoch to over 97% by the end. Validation accuracy improved quickly at the start, reaching about 74% around epoch 3–5, but then levelled off and even dipped slightly. This pattern shows the model is overfitting: it learns the training data very well but struggles to generalize to unseen examples. After the early epochs, further training mainly boosts training performance without improving validation, indicating that the model is memorizing training patterns rather than learning features that generalize.



The training F1 score increased steadily from 0.50 to 0.96, closely following the trend of training accuracy. In contrast, the validation F1 plateaued around 0.72–0.76 after the first few epochs and showed little improvement afterward. This

suggests that while the model is learning the training data very well, it struggles to generalize to unseen data. Since F1 considers both precision and recall across all classes, the stagnation indicates that class-wise predictions are not improving. Essentially, the model is starting to overfit around epoch 4–5. It is memorizing the training patterns instead of learning features that work broadly.



4.4 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 that the model predicts actions with distinct visual cues, like cycling, hugging, sleeping, texting, and using a laptop, very accurately. However, actions with similar movements—such as calling, drinking, laughing, and dancing—are often confused.

4.5 Test Set Metrics

- **Overall Accuracy:** 0.736
- **Macro F1-score:** 0.738

Sample per-class metrics:

<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

In this project, I implemented a complete deep learning pipeline for 15-class human action recognition using VGG16. My work covered dataset preparation, advanced data augmentation, model training and validation, and final test evaluation.

Key takeaways from the project:

- The combination of MixUp, Mosaic, and label smoothing proved effective: the model achieved a best validation F1 around 0.7619 and a test F1 of 0.7357, demonstrating good generalization without severe overfitting.
- Per-class analysis shows that actions with distinctive visual features (such as cycling, sleeping, and using a laptop) were predicted with high F1 scores (~0.75–0.96).

- Performance is lower for visually similar activities like calling, drinking, and sitting, highlighting that pose alone is sometimes insufficient for perfect classification. Additional context or object cues (e.g., phone or laptop) could further improve predictions.
- The pipeline is modular and reusable: the same dataset loader, augmentation, training, and evaluation scripts could be applied to alternative backbones such as ResNet, EfficientNet, or Vision Transformers.

Potential future improvements:

- Integrate attention mechanisms or Vision Transformers to focus on discriminative regions (hands, devices, objects).
- Experiment with class-balanced loss or focal loss to improve predictions on difficult or underperforming classes like sitting and dancing.

6. Percentage of Code

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

1. Dataset and Splitting Code:
    - Inspired by PyTorch Dataset patterns and scikit-learn's train_test_split.
    - Roughly 25–30% boilerplate, mainly for class inheritance and CSV reading; 70–75% original, including label mapping, CSV management, and integration of MixUp/Mosaic.
2. Training Loop and VGG16 Setup:
    - Model instantiation and optimizer setup follow standard PyTorch examples.
    - However, the MixUp/Mosaic integration, label smoothing, ReduceLROnPlateau on validation F1, and per-epoch logging/checkpointing were fully implemented by me.
    - Estimated 70% original, 30% adapted.
3. Evaluation and Reporting Script:
    - Standard scikit-learn metrics were used for computing classification reports and confusion matrices.

- The data flow (mapping indices back to labels, saving CSV predictions, formatting logs) was implemented by me.
- Estimated 70% original, 30% adapted.

Overall, about 70% of the code is my own work, while 30% is adapted from tutorials, documentation, or starter scripts.

7. References

1. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778.
   (Relevant for understanding deep CNN architectures and fine-tuning strategies.)

2. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., … & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861.
   (Relevant for lightweight CNN designs and data-efficient training, which relates to my use of VGG16 on a medium-size dataset.)

3. Shorten, C., & Khoshgoftaar, T. M. (2019). A Survey on Image Data Augmentation for Deep Learning. Journal of Big Data, 6(1), 60.
   (Relevant for MixUp, Mosaic, and other augmentation strategies applied in my project.)