CS-GY 6643 Project 3: Computer Vision

Total Marks: 100 **Submission Deadline: November 26, 2025 11:59PM**

Marks Distribution:
This project consists of 3 main questions. The first two are **40 points** each, annd the competition question is **20 points**, for a total of **100 points**.

In addition, there are **two optional Bonus Questions** at the end of the project:

- **Bonus Question 1 (23 pts):** Optical Flow–Based Action Classification — each of the 46 test videos contributes 0.5 marks if correctly classified. **Question 3 is also bonus**
- **Bonus Question 2 (15 pts):** Mathematics-Based Questions on Optical Flow theory and derivations.

We also converted **Question 3** to a bonus question, which is worth **25 points**.

Together, the bonus questions provide up to **63 additional points**, allowing a maximum achievable score of **163 points in total**. While these bonus questions are not mandatory, they are strongly encouraged for students who wish to demonstrate deeper conceptual understanding and advanced technical proficiency, as well as to make up for any performance issues on previous assignments.

Submission Instructions

Please read the following instructions carefully to ensure your project is submitted correctly.

- **Single Notebook Format:** Your entire project (all questions except the mathematical bonus) must be consolidated into one single Jupyter or Google Colab notebook (`.ipynb` file), including all code, visual outputs, and written explanations.
- **PDF for Bonus Mathematics Question**: For ease of submission and grading, please submit the bonus mathematics question as a PDF (either handwritten or Latex, Latex strongly preferred).
- **Submission Process (File AND Link):** On Brightspace, you must submit both your final `.ipynb` file (as a file upload) and a shareable link to it (in the text box). Ensure that link sharing is set to "Anyone with the link can view."
- **Report Format:** Your notebook serves as the final report. Use Markdown cells to explain your methods, results, and challenges clearly.
- **LLM Usage Policy:** If you use an LLM (ChatGPT, Claude, Gemini, etc.) for assistance, include a shareable link or PDF of the chat history along with your submission.

Project 3: Computer Vision

Due November 26, 2025 11:59pm

Introduction

In this project, you will explore how deep learning methods can be applied to understand, analyze, and interpret visual information. Building upon the classical computer vision techniques introduced earlier, this project focuses on the use of convolutional neural networks (CNNs) for feature visualization, object detection, tracking, and motion analysis.

You are required to complete multiple tasks that demonstrate both conceptual understanding and practical implementation of modern vision algorithms. These tasks range from visualizing internal CNN activations and Grad-CAM heatmaps to estimating real-world motion through object detection and optical flow. Each question is designed to reinforce your ability to connect theory with application while developing interpretable and efficient computer vision systems.

1 Question 1: CNN Feature Visualization & Interpretation (40 pts)

Question 1 Resources

We have seen that convolutional networks produce increasingly abstract feature maps as we move deeper into the network. In this question, you will **visualize** and **compare** these internal activations for three different architectures and for two different images.

Reference images and example (given)

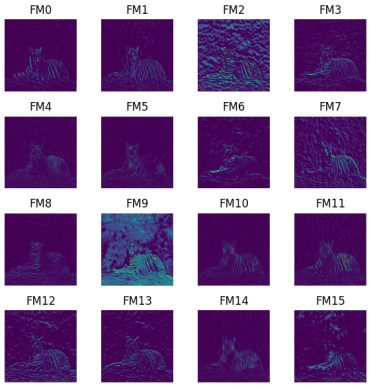
Figure 1 shows two input images you should use throughout this question: (a) a **tiger** image and (b) a **human face** image. Figure 2 shows an *example* visualization already done for you using **VGG16** on the tiger image at three depths (early, mid, deep).

Note that activations from the **last convolutional layer** are often highly abstract and may appear very sparse or uninterpretable when visualized directly. To better understand which image regions drive the network's prediction, you should use **Grad-CAM** (Gradient-weighted Class Activation Mapping), which highlights the most class-relevant spatial locations.

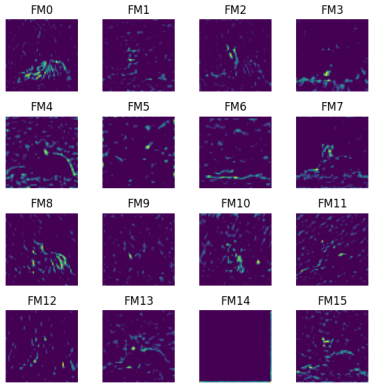
(a) Input image 1: Tiger

(b) Input image 2: Face

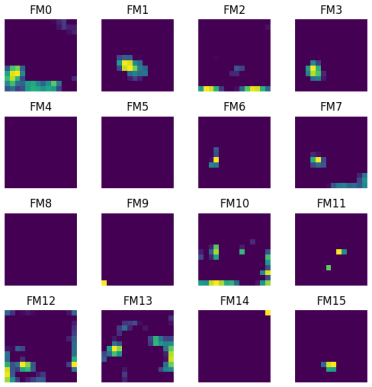
Figure 1: Source images to use for visualization.



(a) VGG16, tiger, early layer (`features[3]`)



(b) VGG16, tiger, mid layer (`features[15]`)



(c) VGG16, tiger, deep layer (`features[28]`)

Figure 2: Example feature maps for VGG16 on the tiger image at different depths.

Grad-CAM Visualization

Gradient-weighted Class Activation Mapping (Grad-CAM) highlights image regions that most influence the model's prediction by weighting each channel of the final convolutional layer using the gradient of the target class score. This produces an interpretable heatmap showing where the network "looks" when classifying an image.

Grad-CAM Implementation Code Example (VGG16)

```
import torch, cv2, numpy as np, matplotlib.pyplot as plt

model.eval()
target_layer = model.features[28]  # last conv layer

fmap, grads = {}, {}
```

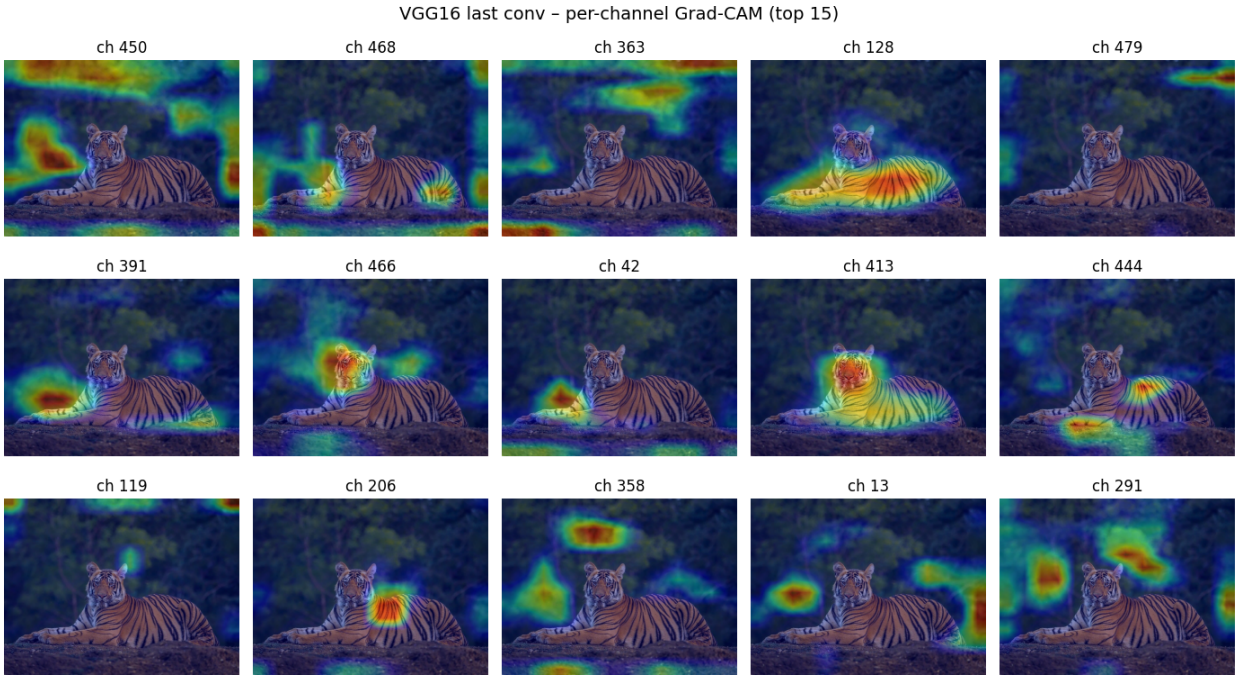


Figure 3: VGG16 last convolutional layer – top 15 per-channel Grad-CAM overlays for the tiger image. Each channel captures a distinct semantic region (e.g., stripes, torso, face).

```
def fwd_hook(m, i, o): fmap["value"] = o.detach()
def bwd_hook(m, gin, gout): grads["value"] = gout[0].detach()

# register hooks
h1 = target_layer.register_forward_hook(fwd_hook)
h2 = target_layer.register_backward_hook(bwd_hook)

# forward & backward
out = model(input_tensor)
pred_class = out.argmax(dim=1).item()
model.zero_grad()
out[0, pred_class].backward()

# feature maps (A) & gradients (G)
A, G = fmap["value"], grads["value"]
alpha = G.mean(dim=(2,3)).squeeze(0)        # channel importance
topk = alpha.topk(15).indices               # pick top 15 channels

img_np = np.array(img).astype(np.float32) / 255.0
H, W = img.size[1], img.size[0]

fig, axes = plt.subplots(3,5,figsize=(14,8))
for idx, ax in enumerate(axes.flat):
    ch = topk[idx].item()
```

```
    cam = torch.relu(alpha[ch] * A[0,ch,:,:]).cpu().numpy()
    cam = (cam - cam.min()) / (cam.max() + 1e-5)
    cam_resized = cv2.resize(cam, (W, H))
    heatmap = plt.cm.jet(cam_resized)[:,:,:3]
    overlay = np.clip(0.4*heatmap + 0.6*img_np, 0, 1)
    ax.imshow(overlay)
    ax.set_title(f"ch {ch}")
    ax.axis("off")

plt.suptitle("VGG16 last conv - per-channel Grad-CAM (top 15)")
plt.tight_layout()
plt.show()
h1.remove(); h2.remove()
```

Your tasks

(a) ResNet50 activations (6 pts).
Using a pretrained **ResNet50** and the **same two input images** (tiger and face), extract and visualize feature maps at three depths:

- *Early layer:* e.g. output of `layer1`
- *Mid layer:* e.g. output of `layer2` or `layer3`
- *Deep layer:* e.g. output of `layer4` (use Grad-CAM if the maps are not interpretable)

For each depth, display at least **16** channels as a 4×4 grid (like in Fig. 2). Do this for *both* the tiger and the face image. Clearly label each figure with: model name, image name, and layer name.

(b) DenseNet121 activations (6 pts).
Repeat the same procedure using a pretrained **DenseNet121**. Extract and visualize feature maps from:

- *Early block:* `features.denseblock1`
- *Mid block:* `features.denseblock2` or `features.denseblock3`
- *Last block:* `features.denseblock4` (use Grad-CAM for interpretation)

Again, show at least 16 channels (4×4) for both images.

(c) Analysis and comparison (13 pts).

i. **Abstraction (3 pts):** Describe how the feature maps evolve from *early* → *mid* → *deep* layers in both ResNet50 and DenseNet121.

ii. **Sparsity (3 pts):** A sparse activation map has only a few bright, active regions (most values near zero), indicating that the network focuses on specific discriminative areas rather than the entire image. Discuss whether the tiger or face image yields sparser activations, and relate it to each model's architecture and feature reuse.

iii. **Smoothness / resolution (2 pts):** Deeper feature maps often appear "blocky" or coarse. This occurs because spatial dimensions progressively shrink through pooling or strided convolutions (e.g., $224 \times 224 \rightarrow 14 \times 14$), meaning each activation corresponds to a large image region (a large receptive field). Explain how this reduced spatial resolution limits fine-grained detail but helps capture global semantic context.

iv. **Semantic focus (3 pts):** Using Grad-CAM, compare which semantic regions the networks emphasize in their final layers (e.g., tiger's face vs body, human facial center vs hair or background). Note whether ResNet50 and DenseNet121 focus on similar or different regions, and hypothesize why.

v. **Image dependence (2 pts):** Compare interpretability between the two input images. Which input (tiger vs face) produces more structured or meaningful deep activations? Discuss why one image might yield clearer activations.

2 Question 2: Multi-Car Speed Estimation with Detection & Optical Flow (40 pts)

Question 2 Resources

In this question you will work with a short **highway video** (approximately 60 seconds long, **1501 frames at 25 FPS**) where multiple cars move through a fixed camera view. Your goal is to (1) detect cars, (2) track them across frames, and (3) estimate their relative motion speed using dense optical flow.

Your tasks

(a) Car detection (7 pts).
Run an object detector on every frame of the video and keep only vehicle classes (`car`, `truck`, `bus`). Draw bounding boxes and class labels on each detected vehicle.

(b) Tracking (8 pts).
Maintain a consistent ID for each vehicle across frames (i.e., the same car should have the same ID as it moves through the video). Draw tracking lines for each vehicle and track it throughout the video.

(c) Speed estimation (10 pts).
For every frame, estimate the motion of each tracked vehicle and express it as a **mean displacement in pixels per frame**. Convert this value to **pixels per second**

(px/s) using the known frame rate of the video:

$$v_{\text{px/s}} = (\text{displacement in px/frame}) \times 25$$

For example, a car with 8.46 px/frame corresponds to $8.46 \times 25 = 211.5$ px/s.

Display the computed **px/s** value above or next to each vehicle in the video frames.

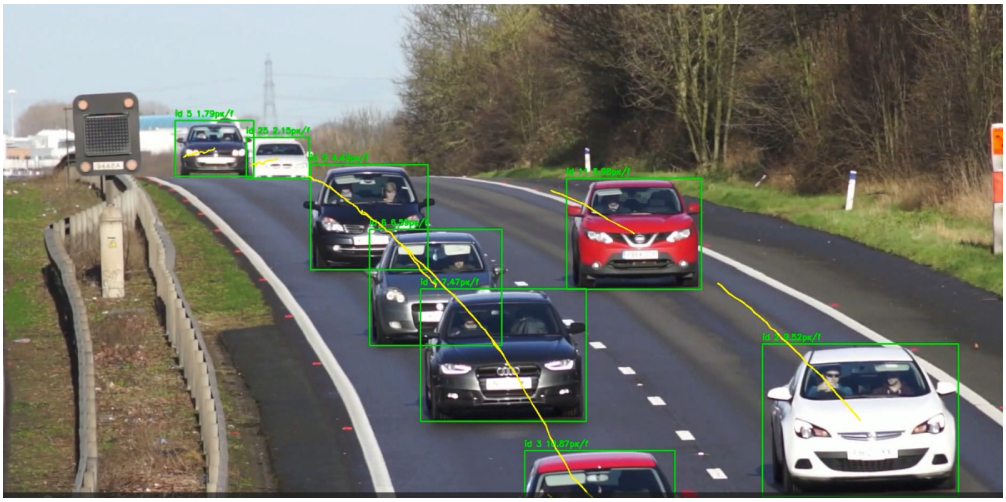


Figure 4: Example multi-car tracking and speed overlay. Each label shows a tracker ID and the corresponding motion in pixels per second (px/s).

3 Bonus Question 3: Person & Dog Tracking with Object Detection & Motion Analysis (25 pts) (Bonus)

Question 3 Resources

In this question, you will analyze a short video of a **Dog running towards a Person**. Your goal is to detect and track both the person and the dog across all frames, visualize their trajectories, and estimate their motion magnitudes over time.

Your tasks

(a) Detection (5 pts).
Run an object detector on each frame and retain only the `person` and `dog` classes. Draw bounding boxes and class labels in different colors (for example, red for the person and yellow for the dog). Save 3–4 representative annotated frames.

(b) Tracking (6 pts).
Maintain consistent IDs for both the person and the dog throughout the video. Show that each keeps the same ID across frames even when they partially overlap or move apart.

(c) Motion estimation (6 pts).
For each frame, estimate the motion of both tracked objects and express it as **mean displacement in pixels per frame**.

(d) Trajectory visualization (4 pts).
For the entire video, record the center of the bounding box of the dog and plot it's trajectory as a continuous line over time. Ensure the dog's path is visible from the first frame to the last frame.

(e) Analysis (4 pts).
Ensure that the dog remains correctly tracked even when it becomes partially or fully occluded, demonstrating robust ID consistency throughout the sequence.

(a) Person and dog tracking (early frame).

(b) Person and dog tracking (later frame).

Figure 5: Example of person and dog tracking with trajectory lines. Each label shows tracker ID and motion in pixels per second (px/s).

4 Baseball Pitch Prediction Kaggle Competition (20 pts)

Predict the Strike Zone... Before the Ball Gets There

Kaggle Invitation Link

Baseball pitchers throw a ball over 60 feet toward home plate, and human umpires must decide—within milliseconds—whether a pitch is a strike or a ball. But what if we try to make that decision before the pitch even reaches the plate?

In this competition, your goal is to build a model that predicts whether a pitch will end up inside the strike zone using only a short video clip of the ball's early flight.

You are given two types of clips:

- **Full Pitch clips (1.2s)** — roughly the entire pitch.
- **Trimmed clips (0.26–0.75s)** — short physics-based clips aligned to ∼80% of the ball's time-of-flight.

For each clip, you must predict two things:

1. Whether the pitch ends in the strike zone (`strike` or `ball`)
2. Which Gameday zone (1–14) the pitch ultimately crosses.

Description

Predict the Strike Zone Before the Ball Arrives

In baseball, a pitch travels over 60 feet toward home plate in less than half a second. Umpires must decide instantly whether the pitch is a strike or a ball.

In this competition, your task is to make that same decision using only short video clips containing the first portion of the pitch's trajectory.

No pitch-tracking data.
No coordinates.
Only the early flight of the baseball.

What Are You Predicting?

Your goal is:

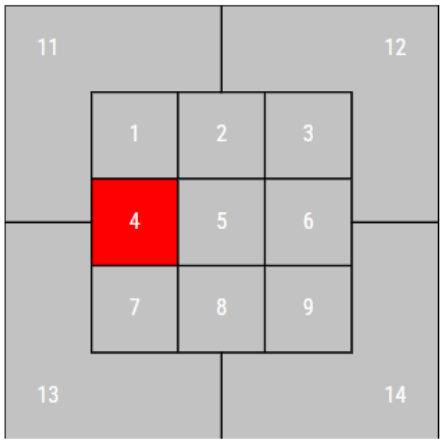
- **pitch_class**: `strike` or `ball`
- **zone**: integer from 1–14

The labels are derived from real Statcast trajectory measurements, not the umpire call. Each pitch includes the following Statcast fields:

- `plate_x` — horizontal plate-crossing location
- `plate_z` — vertical plate-crossing location
- `sz_top` — top of batter's strike zone
- `sz_bot` — bottom of batter's strike zone

A pitch is labeled strike if the center of the baseball intersects the strike-zone volume (expanded slightly by the ball radius). Otherwise it is labeled ball.

Gameday Strike Zone



-- Zone from **Catcher's** Perspective

MLB's standard Gameday grid defines 14 labeled regions:

- Zones 1–9: true strike-zone
- Zones 11–14: "shadow" regions just outside

PLEASE NOTE IT IS FROM CATCHER'S PERSPECTIVE. Your model sees only the early-flight segment but must infer the final location.

Evaluation

Submissions in this competition are evaluated using a weighted multi–target accuracy, combining:

1. Pitch Class Prediction (70%)

Participants must predict whether the pitch ends up in the strike zone or not.

For each test video, `pitch_class` must be either: `strike` or `ball`.

Class accuracy is defined as:

$$AccuracyClass = \frac{1}{N}\sum_{i=1}^{N} I(\hat{c}_i = c_i)$$

where:

- $\hat{c}_i$ is the predicted class
- c_i is the true class
- $I(\cdot)$ is the indicator function (1 if the condition is true, 0 otherwise)
- N is the number of test samples

2. Zone Classification (30%)

Each pitch is associated with a Gameday strike zone region:

$$zone \in \{1, 2, \ldots, 14\}$$

Zone accuracy is defined as:

$$AccuracyZone = \frac{1}{N}\sum_{i=1}^{N} I(\hat{z}_i = z_i)$$

where:

- $\hat{z}_i$ is the predicted zone
- z_i is the true zone

Final Metric

The final competition score is a weighted combination of class and zone accuracy:

$$Score = 0.7 \cdot AccuracyClass + 0.3 \cdot AccuracyZone$$

Submission Format

Your submission must be a CSV with:

```
file_name, pitch_class, zone
pitch1.mp4, strike, 5
pitch2.mp4, ball, 14
...
```

Please refer to the sample template csv file provided **Constraints:**

- `file_name` must match the test set exactly.
- `pitch_class` must be `strike` or `ball`.
- `zone` must be an integer 1–14.
- Every test file must appear exactly once.

Leaderboard Split

The leaderboard is divided into:

- **Public LB:** 50% of test set
- **Private LB:** 50% of test set (used for final ranking)

Dataset Description

Statcast-Based Ground Truth

Each pitch in the dataset uses MLB Statcast tracking to determine the true strike or ball outcome.

`plate_x` and `plate_z` These describe where the ball crossed the front plane of home plate:

- Positive `plate_x`: catcher's right
- Negative `plate_x`: catcher's left
- `plate_z`: vertical crossing height

sz_top and sz_bot Define the personalized vertical strike-zone limits for each batter. A pitch is considered a strike when:

$$(\texttt{sz\_bot} - r) \leq \texttt{plate\_z} \leq (\texttt{sz\_top} + r)$$

and

$$-\left(\frac{17}{24} + r\right) \leq \texttt{plate\_x} \leq \left(\frac{17}{24} + r\right),$$

where r is the baseball radius (~ 1.5 inches).

Time-of-Flight (ToF)

Trimmed clips represent approximately 0.26–0.75 seconds of early ball flight (about 80% of total ToF). Estimated ToF is:

$$\text{ToF} = \frac{60.5 - \texttt{release\_extension}}{\texttt{release\_speed} \times 1.46667}.$$

Only the start of the pitch trajectory is shown; the ball never reaches the plate.

Dataset Structure

- `train_full/`: 1.2-second broadcast-style clips (3.0s–4.2s of the feed)
- `train_trimmed/`: physics-aligned clips (0.26–0.75s), roughly 80% of ToF
- `test/`: trimmed ToF clips, same format as `train_trimmed/`

All clips are anonymized: `pitch1.mp4`, `pitch2.mp4`, ...

Pitch Physics Inputs

- `release_speed`: velocity at release (mph)
- `effective_speed`: perceived velocity
- `release_spin_rate`: spin rate (RPM)
- `release_pos_x`, `release_pos_y`, `release_pos_z`: 3D release point
- `release_extension`: release distance toward home plate (ft)
- `pfx_x`, `pfx_z`: horizontal and vertical break (inches)

Batter and Pitcher Context

- `stand`: batter stance (L or R)
- `p_throws`: pitcher throwing hand (L or R)

CSV Files

Three CSV files define the dataset splits.

`train_ground_truth.csv` Contains all metadata plus true labels.

Column	Description
file_name	Anonymized filename (e.g., `pitch1.mp4`)
plate_x, plate_z	Plate-crossing location
sz_top, sz_bot	Personalized strike-zone limits
release_speed	Speed at release (mph)
effective_speed	Perceived velocity
release_spin_rate	Spin rate (RPM)
release_pos_x/y/z	3D release point
release_extension	Release distance toward plate (ft)
pfx_x, pfx_z	Pitch break (inches)
stand	Batter stance
p_throws	Pitcher throwing hand
pitch_class	Strike or ball
zone	Gameday zone (1–14)

`test_features.csv` Contains the same metadata as training *except* the fields that leak labels: plate_x, plate_z, pitch_class, zone.

`test_template.csv` Required submission format:

- `file_name`
- `pitch_class`
- `zone`

Example:

```
file_name,pitch_class,zone
pitch7001.mp4,strike,5
pitch7002.mp4,ball,14
pitch7003.mp4,strike,3
```

Prediction Targets

Models must predict:

- `pitch_class`: `strike` or `ball`
- `zone`: integer 1–14

Both predictions are required for every test sample.

5 Question Bonus Section: Optical Flow–Based Action Classification (23 pts)

Bonus Question Resources

In this question, you will classify a collection of short human-action videos into the correct action category using any approach of your choice. You may use classical computer vision methods, handcrafted motion features, or deep learning–based models. The only requirement is that your final classification pipeline must be implemented as a **single unified script or notebook code cell** capable of automatically classifying **all 46 unlabeled test videos**.

Each correctly classified test video is worth **0.5 marks**, contributing up to **23 marks**.

You are provided with:

- **2 labeled training videos per class** (a total of 6 labeled clips) from the three action categories below.
- **46 unlabeled test videos** drawn from the same three categories.

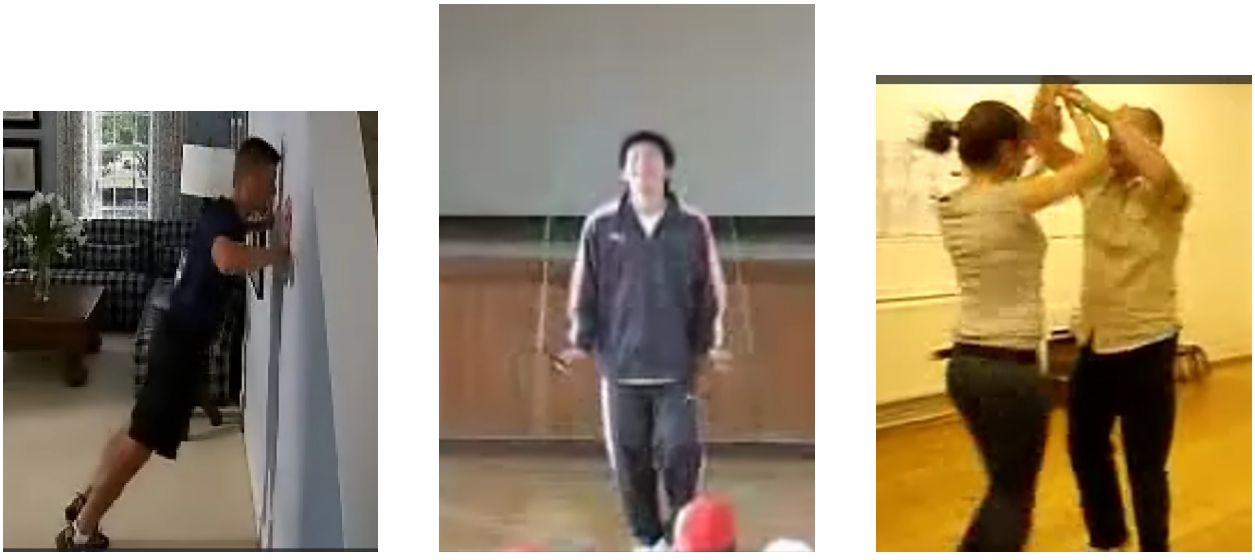
The three action classes are:

(i) **Wall Pushups** – repetitive horizontal upper-body motion.

(ii) **Jump Rope** – strong cyclic vertical and circular arm motion.

(iii) **Salsa Spin** – rotational full-body motion.

Figure 6 shows representative frames from each class.



(a) Wall Pushups (b) Jump Rope (c) Salsa Spin

Figure 6: Representative frames from the three action categories. Two labeled training videos per class are provided, along with 46 unlabeled test videos for classification.

Note: You must use the same algorithm or model for all test videos. Submitting multiple scripts or per-video methods is not allowed.

6 Question Bonus section: Mathematics-Based Questions (15 pts)

1. The core of the **Lucas-Kanade** optical flow method involves solving the overdetermined system $\mathbf{Au} = \mathbf{b}$ for the flow vector $\mathbf{u}$ in a local window. The least-squares solution is given by:

$$\mathbf{u} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$$

Let $\mathbf{G} = \mathbf{A}^T\mathbf{A}$ be the 2×2 **Structure Tensor** (or Gram Matrix) of the window.

Prove that a reliable and unique solution for the flow vector $\mathbf{u}$ exists **if and only if** the Structure Tensor $\mathbf{G}$ is **invertible**.

2. The change in image intensity $E(u, v)$ when a window is shifted by a vector $\mathbf{d} = [u, v]^T$ is approximated using the Second Moment Matrix $\mathbf{M}$:

$$E(u, v) \approx \mathbf{d}^T\mathbf{M}\mathbf{d}$$

Prove that if a region is an ideal, straight, uniform edge (not a corner), one of the eigenvalues of $\mathbf{M}$ must be zero.

3. In the **Horn-Schunck** optical flow method, the smoothness term $\mathcal{E}_s$ penalizes the squared magnitude of the gradient of the flow field. For a vector $\mathbf{v} = [v_1, v_2, \ldots, v_n]^T$, the squared Frobenius Norm is defined as $||\mathbf{v}||_F^2 = \sum_{i=1}^{n} v_i^2$.

Prove that the term $\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2$ can be expressed as the squared Frobenius Norm of the flow gradient vector $\nabla u = \left[\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right]^T$.