**Training Plan: Beginner Computer Vision Tasks and Datasets**

To prepare an undergraduate student for the **CeleST worm project**, it’s helpful to first build foundational skills in four key computer vision tasks: **(1) image classification**, **(2) image segmentation**, **(3) video object tracking**, and **(4) video-based movement analysis**. Each task will be introduced with **user-friendly, open-source datasets** and practical Python examples, with a focus on using **OpenCV** alongside other libraries. Below, we outline recommended datasets (with licensing and sources), example code resources, and documentation for each task. This will provide hands-on experience and confidence before tackling the more complex worm tracking problem.

A close-up of a sign

AI-generated content may be incorrect.

**1. Image Classification**

**Goal:** Learn to build a model that assigns labels to images.\ **Recommended Datasets:** We suggest starting with simple, well-known datasets that are easy to load and have ample tutorials:

* **MNIST** – classic handwritten digit dataset (10 classes, 28×28 grayscale). This “hello world” of CV has 60k training and 10k test images[[1]](https://huggingface.co/datasets/p2pfl/MNIST). It’s extremely beginner-friendly: many frameworks include it by default (e.g. keras.datasets.mnist.load\_data() or torchvision.datasets.MNIST)[[2]](https://keras.io/api/datasets/mnist/). The dataset is open-source (MIT License on HuggingFace version)[[1]](https://huggingface.co/datasets/p2pfl/MNIST), so there are no usage restrictions.
* **CIFAR-10** – 10 classes of tiny 32×32 color images (e.g. cat, dog, plane, truck). It has 50k train + 10k test images[[3]](https://huggingface.co/datasets/p2pfl/CIFAR10). CIFAR-10 is slightly more complex than MNIST (color, more varied content) but still small enough for quick training. It’s also freely available (MIT licensed)[[3]](https://huggingface.co/datasets/p2pfl/CIFAR10) and accessible via library loaders (e.g. TensorFlow Datasets or PyTorch).

**Practice Approach:**

1. **Loading the Data:** Use built-in loaders for convenience. For example, with Keras: <pre loop-creation-data="{"dataType":"Code","data":{"codeLanguage":"Python","code":"from tensorflow import keras\n(x*train, y*train), (x*test, y*test) = keras.datasets.mnist.load\_data()\n`` &quot;}}" unfurl="true" lang="en-us">from tensorflow import keras (x\_train, y\_train), (x\_test, y\_test) = keras.datasets.mnist.load\_data() ``` </pre><span></span></span> This one-liner downloads and returns the MNIST data as NumPy arrays[2](https://keras.io/api/datasets/mnist/). Similarly, PyTorch’storchvision.datasets.CIFAR10(root, download=True)` will fetch CIFAR-10. Using these utilities eliminates manual download hassle, letting students focus on modeling.
2. **Exploring with OpenCV:** Even though frameworks handle data loading, the student can integrate OpenCV for image manipulations. For example, use cv2.imshow to visualize samples, or convert images between color spaces. OpenCV’s cv2.resize could be used to upscale CIFAR images for display, or cv2.equalizeHist to normalize MNIST contrast – practicing OpenCV while working with the dataset.
3. **Training a Simple Model:** Start with a basic algorithm:
   * A **k-Nearest Neighbors (k-NN)** classifier on raw pixels (as shown in [PyImageSearch’s tutorial][6]): this requires minimal code and helps the student understand basic feature vectors[[4][4]](https://pyimagesearch.com/2021/04/17/your-first-image-classifier-using-k-nn-to-classify-images/). For instance, flatten the 28×28 images into 784-dimensional vectors and use sklearn.neighbors.KNeighborsClassifier – a great intro to machine learning on images.
   * Then progress to a simple **Neural Network**. For MNIST, a small multilayer perceptron or a single-layer CNN (convolutional neural network) can achieve high accuracy. There are many open tutorials: e.g., GeeksforGeeks shows MNIST classification with Keras and PyTorch side-by-side. Using those, the student can try both frameworks and even deploy an OpenCV DNN inference (by exporting a model).
   * For CIFAR-10, experiment with a deeper CNN (like those in Keras examples or PyTorch’s CIFAR10 tutorial). The student can also use a **pre-trained model** via OpenCV’s DNN module to classify CIFAR images. For example, OpenCV DNN can load a pre-trained ImageNet model (like MobileNet) and predict on CIFAR images, just to see how a larger model perceives them[[5]](https://colab.research.google.com/github/bigvisionai/upgrad_alumni_workshop_day2/blob/master/opencv_dnn_image_classification/OpenCV_DNN_Image_Classification.ipynb).
4. **Documentation & Examples:**
   * **Keras Docs**: The official Keras documentation for mnist.load\_data() succinctly describes the dataset and usage[[2]](https://keras.io/api/datasets/mnist/). It’s a good starting point for the student to read about shapes and returns.
   * **PyTorch/Torchvision**: The torchvision.datasets docs for MNIST and CIFAR show how to download and transform data[[1]](https://huggingface.co/datasets/p2pfl/MNIST)[[3]](https://huggingface.co/datasets/p2pfl/CIFAR10). They can practice using transforms (like normalization, random flips on CIFAR) via Torchvision – these use PIL by default, but OpenCV can be integrated by converting arrays.
   * **Example Code**: Refer them to a **PyTorch MNIST tutorial** or a **Keras CIFAR-10 example**. For instance, TensorFlow’s official tutorial “Basic classification” uses Fashion-MNIST (very similar to MNIST) and is beginner-friendly. PyTorch’s CIFAR10 tutorial (in the official docs or on sites like Dive-into-DL) can guide through defining a CNN and training loop.
5. **OpenCV Integration**: Even though training models is usually done with high-level libraries, OpenCV can play a role:
   * Use OpenCV for **data augmentation** manually (rotate, blur images using cv2 functions) to show effects on training.
   * After training a model, save it and then **deploy with OpenCV**. For example, if a model is saved as ONNX, cv2.dnn.readNet(model.onnx) can load it and net.forward() can do inference on new images[[5][5]](https://colab.research.google.com/github/bigvisionai/upgrad_alumni_workshop_day2/blob/master/opencv_dnn_image_classification/OpenCV_DNN_Image_Classification.ipynb). This connects the dots between training and real-world usage in OpenCV applications.

**Dataset References:** Below is a summary of these datasets, including their sources, licenses, and ease of use:

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset** | **Source / Download** | **License** | **Ease of Use** |
| MNIST (Digits) | http://yann.lecun.com/exdb/mnist/ – also in Keras/PyTorch | MIT (open)[1](https://huggingface.co/datasets/p2pfl/MNIST) | Very High – small (70k 28×28 images)[1](https://huggingface.co/datasets/p2pfl/MNIST); built into most ML libraries for one-line loading. |
| CIFAR-10 | https://www.cs.toronto.edu/~kriz/cifar.html – also via TFDS/Torchvision | MIT (open)[3](https://huggingface.co/datasets/p2pfl/CIFAR10) | High – manageable size (60k 32×32 images)[3](https://huggingface.co/datasets/p2pfl/CIFAR10); standard in library loaders; numerous tutorials available. |
| Fashion-MNIST | https://github.com/zalandoresearch/fashion-mnist | MIT (open) | High – same format/size as MNIST; drop-in replacement for a slightly harder challenge (clothing images). |

*(Note: Fashion-MNIST can be an optional follow-up; it is as easy to use as MNIST but with 10 clothing item classes instead of digits.)*

A close-up of a sign

AI-generated content may be incorrect.

**2. Image Segmentation**

**Goal:** Partition images into meaningful regions or objects – e.g., classify every pixel in an image (semantic segmentation).\ **Recommended Datasets:** Use datasets that come with ground-truth segmentation masks:

* **Pascal VOC 2012:** A popular academic dataset with 20 object categories, each image having a corresponding segmentation mask marking each pixel’s class[[6]](https://huggingface.co/datasets/merve/pascal-voc). It’s relatively small (~1.7K segmented train images, 736 validation), so one can even train a model on a single GPU. The dataset is under CC BY 2.5 license (free to use with credit)[[6]](https://huggingface.co/datasets/merve/pascal-voc). Many beginner segmentation tutorials use VOC. For instance, the **U-Net** segmentation network was often demonstrated on a subset of VOC. PyTorch’s torchvision.datasets.VOCSegmentation can download it automatically.
* **MS COCO (Common Objects in Context):** A large-scale dataset with 80 object classes, each instance labeled with a segmentation polygon (used for instance and panoptic segmentation). COCO is significantly bigger, but it’s **completely free (CC BY 4.0) and very widely used**; pre-trained models are often trained on COCO[[7]](https://pyimagesearch.com/2018/09/03/semantic-segmentation-with-opencv-and-deep-learning/). Instead of training from scratch, the student can **use pre-trained models on COCO** for experiments – e.g., apply a pre-trained **Mask R-CNN** to images and visualize the predicted masks. This way they handle segmentation output without needing to train on COCO’s 120k images.

**Practice Approach:**

1. **Data Familiarization:** Start with Pascal VOC. Have the student inspect a few images and their annotation masks. An annotation in VOC is typically a PNG where each pixel’s value is an integer id for the class (e.g. 15 = person). They can use OpenCV to overlay masks on images with colors:
2. mask = cv2.imread('2007\_000033.png', cv2.IMREAD\_GRAYSCALE)
3. # mask pixel values correspond to class IDs; let's colorize:
4. color\_mask = color\_map[mask] # map each ID to a color
5. blended = cv2.addWeighted(image, 0.7, color\_mask, 0.3, 0)
6. cv2.imshow('Segmentation Overlay', blended)

This teaches how segmentation data is represented and how to manipulate it.

1. **Classical Segmentation Methods:** Before deep learning, segmentation used techniques like **thresholding, clustering, and edge detection**. The student can try:
   * **Thresholding and Contours:** Use simple color thresholding (with OpenCV cv2.inRange) to segment an object by color. For example, segment out the yellow pixels of a road sign from its background. Or apply k-means on pixels (OpenCV’s cv2.kmeans) to cluster an image into regions by color, an unsupervised segmentation. OpenCV’s documentation and examples cover techniques like these.
   * **GrabCut Algorithm:** OpenCV has an interactive segmentation method cv2.grabCut for foreground extraction given a bounding box. A fun exercise: take an image from VOC (say a person), ignore the provided mask, and instead try to segment the person with GrabCut. Compare the result to the ground truth mask to see how well a classical method can do. This provides intuition on the challenge of segmentation.
   * These classical approaches won’t achieve high accuracy on complex images, but they illustrate important concepts (thresholding, graph cuts, etc.). They also integrate well with OpenCV (which provides ready functions).
2. **Deep Learning Segmentation:** Next, introduce modern segmentation:
   * **Pre-trained Model Inference:** Use **OpenCV’s DNN** module or an external library to run a pre-trained segmentation network. For instance, load the ENet model (a lightweight segmentation CNN) pre-trained on Cityscapes (an urban scene dataset)[[7]](https://pyimagesearch.com/2018/09/03/semantic-segmentation-with-opencv-and-deep-learning/). PyImageSearch provides code for loading an ENet model and segmenting an image into classes (road, sky, person, etc.)[[7][7]](https://pyimagesearch.com/2018/09/03/semantic-segmentation-with-opencv-and-deep-learning/). The student can run this on sample images (perhaps from Cityscapes or even VOC/COCO images) to see pixel-wise predictions. OpenCV’s DNN can handle this in a few lines (readNet, blobFromImage, forward)[[5]](https://colab.research.google.com/github/bigvisionai/upgrad_alumni_workshop_day2/blob/master/opencv_dnn_image_classification/OpenCV_DNN_Image_Classification.ipynb).
   * **Training a Small Model:** If time permits, attempt to train a segmentation model on a subset of Pascal VOC. For example, train on 2-3 classes only (say person vs background) to simplify the task. One could use a smaller architecture or even repurpose a pre-trained classifier by converting it to a Fully Convolutional Network. However, training segmentation from scratch is computationally heavier than classification – ensure the student has GPU access if trying this. Alternatively, use a high-level library (like Keras with a TF segmentation model or fastai library) to train a U-Net on VOC. Fastai, for example, has concise code to train U-Net on segmentation data in a few lines.
   * Many open-source codebases exist for VOC segmentation. For instance, **Torchvision** provides VOCSegmentation loader and one can fine-tune a torchvision deeplabv3\_resnet50 (pre-trained on COCO) to VOC with minimal code. This teaches transfer learning in segmentation.
3. **Visualization and Evaluation:** Have the student quantify results. For example:
   * Compute **pixel accuracy** or **IoU (Intersection over Union)** for their predictions vs ground truth on VOC. Sklearn or simple numpy can compute IoU per class.
   * Visualize results using matplotlib or OpenCV: e.g., show side-by-side the input image, true mask, and predicted mask. This will mirror what they’ll need to do for worm videos (comparing algorithm output to truth or to visually expected shapes).
4. **Documentation & Resources:**
   * **Pascal VOC**: The dataset’s official site provides details on format and challenges. Additionally, the [HuggingFace dataset card for Pascal VOC][15] concisely summarizes its content (classes, annotation types)[[6][6]](https://huggingface.co/datasets/merve/pascal-voc). It’s a good reference for understanding the scope of VOC (it also notes VOC includes object detection and classification labels beyond segmentation).
   * **OpenCV and Segmentation**: Chapters in *Learning OpenCV* or online OpenCV tutorials on segmentation (thresholding, watershed, etc.) will help with classical techniques. For deep learning, OpenCV’s DNN docs plus blogs like PyImageSearch’s “Semantic segmentation with OpenCV”[[7][7]](https://pyimagesearch.com/2018/09/03/semantic-segmentation-with-opencv-and-deep-learning/) are valuable. They show step-by-step how to load a model and process results (including a color map for the segmentation output).
   * **GitHub Repos**: There are open implementations for training segmentation. For example, a GitHub project “Semantic-Segmentation-PyTorch for VOC2012” (MIT License, by DevikalyanDas) provides code to train DeepLab on VOC. Another resource is the fast.ai course which has a lesson on segmentation using the CAMVID dataset (a smaller road scene dataset) – that could be a gentler alternative dataset with fewer classes to experiment on.

**Dataset Summary (Segmentation):**

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset** | **Source / Download** | **License** | **Ease of Use** |
| Pascal VOC 2012 | http://host.robots.ox.ac.uk/pascal/VOC/ or via torchvision.datasets.VOCSegmentation | CC BY 2.5 (Attribution required)[6](https://huggingface.co/datasets/merve/pascal-voc) | Moderate – ~11k images, well-documented classes[6](https://huggingface.co/datasets/merve/pascal-voc); many pre-trained models and loaders available, making it student-friendly. |
| Cityscapes | https://www.cityscapes-dataset.com/ (registration needed for download) | Free for research (CC BY-NC 4.0) | Medium – 5k high-res street images with fine masks; excellent quality but requires sign-up. Use pre-trained models on it (e.g., ENet) to avoid heavy training. |
| MS COCO (2017) | https://cocodataset.org/ or via TFDS/HuggingFace | CC BY 4.0[8](https://www.innovatiana.com/en/datasets/ucf101) | Low (for training) – very large (118k images); however, \*\*High (for pre-trained use)\*\* – leverage ready models to experiment on a subset of images. |
| Oxford-IIIT Pets | https://www.robots.ox.ac.uk/~vgg/data/pets/ | Creative Commons (Attribution) | High – 7k images of cats & dogs with segmentation masks; small, fun dataset ideal for training a simple mask classifier (used in many Keras examples). |

*Note:* For segmentation, it’s perfectly fine to **use pre-trained networks** rather than train from scratch, given limited time. The focus should be on understanding how segmentation output is represented and how to use tools to get that output (whether via OpenCV, PyTorch, or TensorFlow). This experience will translate to understanding the worm body segmentation in CeleST.

A screenshot of a video tracking program

AI-generated content may be incorrect.

**3. Video-Based Object Tracking**

**Goal:** Track objects across video frames, maintaining their identities.\ **Recommended Datasets:** Start with **single-object tracking**, then explore **multi-object tracking**:

* **OTB (Online Tracking Benchmark):** A foundational dataset for single-object tracking, containing 50 (OTB-50) to 100 (OTB-100) short videos[[9]](https://service.tib.eu/ldmservice/dataset/otb-100). Each video has one target (person, animal, vehicle, etc.) with a ground-truth bounding box for each frame. The sequences feature challenges like occlusion, motion blur, scale change, etc. OTB is widely used to evaluate trackers; it’s freely available (academic use) and even includes an evaluation toolkit. Using OTB, the student can test an algorithm and quantitatively measure its accuracy (e.g., intersection-over-union of bounding boxes).
* **MOT Challenge (e.g., MOT15, MOT17):** A series of datasets for **Multi-Object Tracking**, especially for pedestrians in surveillance videos. For example, MOT17 has 7 training and 7 test sequences with many people per frame, each annotated with IDs through time[[10][10]](https://www.kaggle.com/datasets/mdraselsarker/mot15-challenge-dataset). The dataset focuses on crowded scenes (occlusions, people crossing)[[10]](https://www.kaggle.com/datasets/mdraselsarker/mot15-challenge-dataset). Ground truth includes bounding boxes + identity labels per frame. The MOT datasets are more complex (multiple targets), and are used with detection+tracking algorithms. MOT15/17 data (or subsets) are available on Kaggle under Apache 2.0 license[[10]](https://www.kaggle.com/datasets/mdraselsarker/mot15-challenge-dataset), meaning they’re easy to access and use in class projects.

**Practice Approach:**

1. **Single-Object Tracking with OpenCV:**
   * **Use OpenCV’s Built-in Trackers:** OpenCV provides multiple single-object tracker algorithms (BOOSTING, MIL, KCF, TLD, MedianFlow, GOTURN, MOSSE, CSRT) accessible via the cv2.Tracker API[[11][11]](https://learnopencv.com/multitracker-multiple-object-tracking-using-opencv-c-python/). Start by having the student **implement a simple tracker on an OTB video**. OTB videos come with an initial bounding box for frame 1. The student can initialize, for example, a KCF tracker:
   * tracker = cv2.TrackerKCF\_create()
   * tracker.init(frame1, init\_bbox)

then loop through frames updating ok, bbox = tracker.update(frame).

* + **Visualize Tracking:** Use OpenCV to draw the bounding box on each frame (cv2.rectangle) and maybe save the video with tracked boxes. This helps the student learn how to handle video frame by frame, something directly applicable to worm videos later.
  + **Experiment with Different Trackers:** Each algorithm has strengths/weaknesses. For instance, KCF is fast but might lose track on occlusion; CSRT is slower but more accurate for slight scale/rotation changes. The student can try MIL, MedianFlow, etc., by just changing one line (OpenCV’s multi-tracker tutorial shows how to create any tracker by name conveniently[[11][11]](https://learnopencv.com/multitracker-multiple-object-tracking-using-opencv-c-python/)). Using the same video and initial box across trackers, they can qualitatively compare performance (and quantitatively if using OTB ground truth data for error metrics).
  + **Measure Performance:** OTB defines metrics like **Center Location Error** and **Success Rate (IoU > threshold)**. The student can compute IoU between tracker output and ground truth for each frame. Simple Python code with coordinates can calculate these. Plotting the IoU over time gives insight into where the tracker fails. For example, they might see IoU drops during occlusion or fast motion – a discussion point about algorithm limits.
  + **Resources:** The LearnOpenCV blog on MultiTracker (and preceding single-object tracker posts) is a great guide[[11][11]](https://learnopencv.com/multitracker-multiple-object-tracking-using-opencv-c-python/). It provides code and explains each tracker’s basics. OpenCV’s official documentation and sample code (search for samples/python/tracking.py in OpenCV) are also useful for reference.

1. **Multi-Object Tracking (MOT) introduction:**
   * Once comfortable with single-object tracking, introduce the idea of tracking *multiple* objects (e.g., all people in a scene). Clarify that this typically involves two parts: object **detection** in each frame (to find what objects are present and where), and then a **data association** step to link detections frame-to-frame into trajectories.
   * For practice, the student can simulate a simple MOT on a short sequence from MOT17:
     + Use a pre-trained person detector (e.g., OpenCV’s HOG+SVM person detector, or a deep detector like YOLOv3 via DNN). Run it on each frame to get bounding boxes for people[[10]](https://www.kaggle.com/datasets/mdraselsarker/mot15-challenge-dataset).
     + Apply a basic tracking approach to link these detections. A straightforward method: compute the overlap or distance between detections in consecutive frames and match ones that are close as the “same person”. This can be done with a greedy algorithm or Hungarian Algorithm (available via scipy.optimize.linear\_sum\_assignment) for optimal matching.
     + Assign each person an ID and draw it on the frame. The MOT dataset’s ground truth can be used to evaluate ID switch errors or mostly to visually check if the tracking is sensible.
   * Note: full MOT algorithms (like SORT, DeepSORT, FairMOT) are quite complex; we don’t expect from-scratch implementation. But it’s educational to attempt a naive tracker and realize common issues (e.g., two very close pedestrians might swap IDs if association is naive).
   * **Open-Source Tools:** If interested, there are libraries like **Norfair** (open-source, MIT License) which make multi-object tracking simpler by handling the matching given detection inputs. It integrates with OpenCV and could be an easy way for the student to experiment with MOT without coding it all themselves.
   * If time is short, an alternative is to use **OpenCV’s MultiTracker** in a semi-automatic way: the student manually initializes a bounding box for each object in the first frame (e.g., draw boxes around 3 cars in a video), then let MultiTracker keep track of those multiple ROIs in subsequent frames[[11][11]](https://learnopencv.com/multitracker-multiple-object-tracking-using-opencv-c-python/). This doesn’t involve detection; it’s like running several single-object trackers in parallel (OpenCV’s MultiTracker simply aggregates them). It won’t handle new objects entering or re-identification if lost, but it’s a quick demo of tracking several objects and managing multiple ROIs.
   * **Visualization:** As MOT involves many bounding boxes and ID labels, it’s a great opportunity for the student to enhance their OpenCV drawing skills (different colors for different IDs, putting ID text using cv2.putText, etc.). They can also output results to a file (OpenCV VideoWriter) to share their tracking outcome.
2. **Documentation & Examples:**
   * The **OTB website/paper** explains how tracking is evaluated and provides the dataset. The LDM service snippet describes OTB-100’s content[[9]](https://service.tib.eu/ldmservice/dataset/otb-100). Reading the OTB benchmark paper (Wu et al. 2013/2015) introduction could give the student an understanding of common tracking challenges (they enumerate things like illumination variation, occlusion, etc.).
   * **OpenCV Tracker API**: Point to OpenCV’s online docs for the cv2.legacy.Tracker classes (since trackers are in the legacy module as of OpenCV 4.x). It lists what algorithms are available and basic usage. The LearnOpenCV tutorial we cited covers code in both C++ and Python for using these trackers[[11][11]](https://learnopencv.com/multitracker-multiple-object-tracking-using-opencv-c-python/).
   * **MOT Challenge**: The MOT challenge website (motchallenge.net) provides data, evaluation scripts, and even videos of results. The Kaggle data card for MOT15[[10][10]](https://www.kaggle.com/datasets/mdraselsarker/mot15-challenge-dataset) gives a good overview: multiple sequences, ground truth with IDs, and the kind of difficulties present (occlusions, crowds). This can contextualize why MOT is a step up in complexity.
   * If the student is interested in deep learning approaches for MOT, you could reference research like **DeepSORT** or **FairMOT** (both have GitHub implementations). These are advanced, but since our goal is training material, mentioning them can inspire the student about what state-of-the-art looks like beyond simple methods they tried.
   * **Practical tip:** If OTB or MOT seem too large to start, even a single video from elsewhere can be used (e.g., a short video of a ball being tossed to track the ball). The key is to practice with the OpenCV tracking API in a real scenario. The student might record a quick video with their phone and then write a script to track an object in it.

**Dataset Summary (Tracking):**

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset** | **Source / Download** | **License** | **Ease of Use** |
| OTB-50 / OTB-100 | http://cvlab.hanyang.ac.kr/tracker\_benchmark/index.html (or https://figshare.com/articles/dataset/OTB2015/7003617) | Academic free use (benchmark publication)[9](https://service.tib.eu/ldmservice/dataset/otb-100) | High – download videos and ground-truth (easy format). Well-established evaluation methods; can test OpenCV trackers directly on it. |
| MOT15, MOT17 (Multi-Object) | https://motchallenge.net/data (or [Kaggle](https://www.kaggle.com/datasets/mdraselsarker/mot15-challenge-dataset)) | CC BY-NC or Apache 2.0 via Kaggle[10](https://www.kaggle.com/datasets/mdraselsarker/mot15-challenge-dataset) | Medium – data is organized by sequence; evaluating requires more complex metrics. Good for advanced experimentation after single-object tracking. |
| UAV123 | https://cemse.kaust.edu.sa/ivul/uav123 (drone videos) | Free for research (citation required) | Medium – 123 aerial videos with single-object (a vehicle/pedestrian) to track. Adds challenge of camera motion. Can apply OpenCV trackers similarly to OTB. |
| Custom video + VATIC labels | Record a video and annotate with a tool like VATIC or CVAT | User’s own (license N/A) | Low – if creating your own dataset, annotation is time-consuming. Not needed unless practicing the annotation process; otherwise use provided datasets above. |

By working with tracking, the student will become adept at handling video streams, which is crucial. They will learn how to initialize processes using the first frame (just like initializing worm contours in CeleST) and how to update results over time. This experience mapping IDs to objects over time is directly relevant to tracking worms through frames and possibly identifying them.

A close-up of a few words

AI-generated content may be incorrect.

**4. Video-Based Movement Analysis**

**Goal:** Analyze and recognize more complex movement patterns in video – going beyond tracking *where* an object is, to understanding *what* the object is doing. This could include action recognition (classifying the type of motion) or extracting quantitative features of motion (speed, trajectory patterns, etc.). In context of worms, “movement analysis” means computing features like wave frequency; for training purposes with humans, it could mean classifying actions like walking vs running.

**Recommended Datasets:**

* **KTH Actions Dataset:** A classic controlled dataset for human action recognition[[12]](https://github.com/vkhoi/KTH-Action-Recognition). It has 6 action classes: walking, jogging, running, boxing, hand waving, hand clapping, performed by 25 subjects in 4 scenarios (outdoors, outdoors with variations, indoors)[[12]](https://github.com/vkhoi/KTH-Action-Recognition). Videos are small (160×120, ~4 seconds each, 25 fps) and clean (static camera, homogeneous background)[[13]](https://www.csc.kth.se/cvap/actions/). This simplicity makes KTH ideal for experimenting with classical approaches (non-deep-learning). The dataset is publicly available for **non-commercial use** with attribution[[13]](https://www.csc.kth.se/cvap/actions/).
* **UCF101:** A much larger and more diverse action dataset comprising 101 categories (like push-ups, salsa spin, playing violin, etc.), with about 100+ videos per class[[8]](https://www.innovatiana.com/en/datasets/ucf101). The clips are from YouTube, so they have varied background, camera motion, etc., representing “in the wild” scenarios[[8]](https://www.innovatiana.com/en/datasets/ucf101). UCF101 is **free for academic use (CC BY-NC 4.0)**[**[8]**](https://www.innovatiana.com/en/datasets/ucf101). It’s a go-to dataset for training and evaluating deep video models. For our training purpose, UCF101 might be overkill to train from scratch, but the student can use pre-trained models on it or subsets of it to explore advanced techniques.

**Practice Approach:**

1. **Understanding Motion Representation:** Introduce the concept of spatial-temporal features. Unlike static images, video adds the time dimension. Discuss basic ways to represent motion:
   * **Optical Flow:** A vector field that describes apparent pixel motion between frames. OpenCV implements algorithms like Farnebäck’s dense optical flow (cv2.calcOpticalFlowFarneback) and Lucas-Kanade sparse flow. The student can compute optical flow on a KTH video (say, a walking sequence) and visualize it (e.g., draw arrows or color-code flow). Optical flow provides a low-level description of movement—areas with movement, direction, and magnitude[[12]](https://github.com/vkhoi/KTH-Action-Recognition).
   * **Trajectory/Keypoint-based features:** For example, track a set of corner points through a video (using OpenCV’s goodFeaturesToTrack + Lucas-Kanade optical flow). The paths of these points can characterize the motion (straight line vs oscillatory etc.). This is somewhat analogous to tracking worm midpoints, but here on a human silhouette or a ball.
   * **Pose Estimation (optional):** A more explicit motion description is detecting the pose/joint positions in each frame (with a tool like OpenCV’s DNN pose models or MediaPipe). Then features like joint angle changes or velocity can be derived. This might be advanced, but available pre-trained models make it feasible as an experiment on, say, boxing vs waving (different limb motion patterns).
2. **Action Classification (KTH):**
   * Using **KTH dataset**, the student can attempt to build a simple action recognizer:
     + **Optical Flow + ML:** One approach (as done by researchers earlier) is to compute optical flow for each video and extract features from it. For instance, compute the **histogram of flow orientations and magnitudes** (HOF – histogram of optical flow) or use flow as input to a Bag-of-Words model. The cited KTH Action Recognition project on GitHub tried “Optical Flow + Bag of Visual Words + SVM” and got ~78% accuracy[[12]](https://github.com/vkhoi/KTH-Action-Recognition). The student can follow a simplified version: calculate flow for every pair of frames, maybe downsample in space (like a 4x4 grid of flow, averaging directions in each cell) to form a feature vector per frame, then average those for the video or train an SVM on all frames labeled by action. This pipeline uses many skills: OpenCV for flow, NumPy for feature aggregation, sklearn for SVM – tying together multiple tools.
     + **Keypoint Trajectories:** Another classical method is computing feature point trajectories and encoding them (there are known features called Motion Boundary Histograms, etc., but that might be too detailed). Still, even a heuristic like “running has faster motion than walking” can be verified by comparing optical flow magnitudes between a running video and a walking video (e.g., take the average flow magnitude as a proxy for speed).
     + **Simple Deep Learning:** Train a small convolutional network on the frames or a simple recurrent network on frame sequences. KTH is small enough that even a 3D CNN or an LSTM on frame features could be tried. For example, treat each frame’s CNN features (from a pre-trained ResNet on ImageNet) as a sequence and train an LSTM to classify the action. However, implementing this is more complex and might require using PyTorch or TensorFlow rather than pure OpenCV. If the student is up for it, it’s a great exposure to deep learning on video. If not, focusing on the classical approach is fine.
   * **Evaluation:** KTH has a standard split (as mentioned in the original paper, e.g., train on persons 1-16, test on 17-25). The student can simply split videos by person or do a random split. Evaluate accuracy of their classifier on test videos. With only 6 classes, accuracy is an intuitive metric. They can also make a confusion matrix to see which actions confuse the model (perhaps jogging vs running might be tricky, as they are similar).
   * **Leverage Existing Code:** The GitHub repo by **vkhoi/KTH-Action-Recognition** (MIT License) is a goldmine[[12][12]](https://github.com/vkhoi/KTH-Action-Recognition). It contains implementations of multiple approaches (SIFT + BoW, Optical Flow + BoW, CNN, etc.) along with results. The student can read the README to understand what worked best (they found CNN+optical flow achieved ~90%[[12]](https://github.com/vkhoi/KTH-Action-Recognition)). They could even run parts of that code or inspect how optical flow was used to form features. This saves time and lets them compare their own attempt to an existing solution.
3. **Movement Quantification:** Not all movement analysis is classification. Sometimes we want metrics (like CeleST does for worms). For a human example, you might have the student measure:
   * **Speed of a subject:** in KTH videos, how to measure jogging vs running speed? Since camera is static, they could track the person (using tracking from section 3) and then convert the bounding box movement into a speed (pixels/frame). Compare average pixel velocity for walking, jogging, running – it should increase from walking to running. This parallels measuring worm centroid speed or wave speed.
   * **Repetition rate:** If someone is clapping in a video, can we detect the periodic motion? The student could analyze the optical flow or pixel intensity at the hands to detect peaks (each clap) and compute a frequency. This is analogous to detecting worm body wave frequency. For example, count how many times the hand region goes from apart to together in the handclapping videos.
   * These kinds of analysis build intuition for deriving insights from raw tracking data, similar to how CeleST computes curvature, wave count, etc. from worm skeletons.
4. **Advanced (UCF101 or others):** If the student is enthusiastic and time allows:
   * **Pre-trained Action Recognition Models:** There are models like I3D (Inflated 3D ConvNet by Google) or C3D that are pre-trained on large video datasets (Kinetics, Sports1M). The student could use OpenCV’s DNN (if model is converted to a supported format) or more straightforward, use TensorFlow/PyTorch hub models, to classify a few UCF101 videos. For instance, take a “BasketballDunk” video from UCF101 and see if a pre-trained model correctly predicts “dunk” or a related action. This exposes them to the state-of-the-art in action recognition.
   * **Compare with KTH**: They will see UCF101 classes are much more detailed and the video backgrounds are messy. It emphasizes why simpler datasets (like KTH or our worm videos in a controlled environment) allow using simpler methods, whereas real-world videos require deep learning and big data. This can motivate them when transitioning to worm videos – worms are simpler shapes/background (like KTH’s simple backgrounds), so classical and simpler ML might suffice there, unlike for classifying YouTube videos.
   * **Other datasets**: Mention **HMDB-51** (51 actions, also CC BY-NC) or **Sports-1M** (a huge YouTube dataset by Google) if they are interested in more. Not necessary to use them, but good to know of. HMDB-51 is another academic set but somewhat noisy; Kinetics-600/700 by DeepMind is a very large modern dataset (also mostly non-commercial, via YouTube). This is just FYI for the student’s knowledge.
5. **Documentation & Resources:**
   * **KTH Dataset Info:** The KTH official site (Nada KTH) provides a description of how videos were recorded and a link to download zips per action[[13][13]](https://www.csc.kth.se/cvap/actions/). The details we cited give a clear picture: 600 videos total, 6 actions, etc. This can be included in the training material so the student knows the scope.
   * **Optical Flow Documentation:** OpenCV docs on calcOpticalFlowFarneback explain parameters (like pyramid scale, etc.). Additionally, tutorials (e.g., a blog “Optical Flow with Python/OpenCV” on LearnOpenCV or PyImageSearch) show how to visualize flow with color coding (commonly using the HSV color wheel for direction/magnitude).
   * **Action Recognition Literature:** A gentle intro is the survey “Recognizing Human Actions: a Local SVM Approach” by Schuldt et al. (the KTH creators)[[13]](https://www.csc.kth.se/cvap/actions/), which explains the early approach with local features + SVM. For a modern perspective, blogs or medium articles on “Video Classification with CNN + LSTM” could be used if the student goes that route.
   * **GitHub Repos:** We already mentioned one for KTH. For UCF101, there are many (search “UCF101 PyTorch” on GitHub – you’ll find code for training 3D CNNs or two-stream networks). These can be overwhelming, but if the student is curious how a research-level project looks, it’s worth browsing one.
   * **Tools:** If they want to label or play with their own movement data (like record themselves doing an action), tools like Labelbox or CVAT can annotate videos, or simpler, even writing code to label frames by action if recording in separate takes. But given we have ready datasets, it’s probably unnecessary.

**Dataset Summary (Movement/Action):**

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset** | **Source / Download** | **License** | **Ease of Use** |
| KTH Actions | [KTH CVAP](https://www.csc.kth.se/cvap/actions/) (downloadable as zip files per action) | CC BY-NC (research only)[13](https://www.csc.kth.se/cvap/actions/) | High – small dataset, uniform videos; easy to apply classical algorithms. Ground truth is just the labels per video (no complex annotations required). |
| UCF101 | https://www.crcv.ucf.edu/data/UCF101.php (official download) or https://huggingface.co/datasets/quchenyuan/UCF101-ZIP | CC BY-NC 4.0[8](https://www.innovatiana.com/en/datasets/ucf101) | Medium – large number of videos, requires more compute for training. Great for pre-trained model testing or transfer learning experiments. |
| Weizmann Actions | http://www.wisdom.weizmann.ac.il/~vision/SpaceTimeActions.html | Free for research (citation) | High – very small (10 people, 10 actions), simple backgrounds. Good for an initial toy experiment, though KTH already covers similar ground with more data. |
| Sports Videos (YouTube-8M, Sports1M) | Google Research, YouTube (via TensorFlow Datasets) | Varies (YouTube-8M is CC BY 4.0) | Low – extremely large-scale and only for advanced deep learning exploration. Mentioned for awareness, not for actual use in a short training module. |

By engaging with movement analysis on humans, the student will learn techniques to extract meaningful signals from motion. This directly parallels what they will do for worms (e.g., using optical flow or differences to detect worm bends, counting occurrences of a motion pattern, etc.). It also reinforces earlier skills: they might combine tracking (section 3) with movement metrics (section 4) – for example, track a person and measure their speed; similarly, track a worm and measure its wave speed.

**Conclusion and Next Steps**

Through these progressive exercises in classification, segmentation, tracking, and movement analytics, the student will build a strong toolkit:

* **Programming skills:** loading datasets, using OpenCV for image/video I/O and visualization, and using ML libraries for model training.
* **Conceptual understanding:** from pixels to features to models, and from frames to motion to actions. They will understand how to evaluate results (accuracy, IoU, etc.) which is crucial for scientific work.
* **Open-source familiarity:** They will navigate public datasets and implement reproducible experiments, which is great preparation for working with the open data in the CeleST project and potentially contributing improvements.

Throughout, emphasize an iterative, experimental approach:

* Encourage them to **plot graphs** (e.g., training curves for classification, IoU over time for tracking, flow histograms for actions) to interpret outcomes.
* Encourage use of **GitHub repositories and official examples** as starting points – standing on the shoulders of open-source projects accelerates learning (and all recommended projects/datasets here are free to use).
* Ensure they note any **licensing requirements** (like citing sources for KTH or VOC) in reports or code comments, instilling good practice for research compliance.

By mastering these fundamentals, the student will be well-equipped to tackle rewriting CeleST’s algorithms in Python. For instance, when faced with “multi-worm tracking and curvature analysis,” they can recall: *“This is similar to multi-object tracking plus movement feature extraction like I did with pedestrians and KTH actions”*. The analogies will help them confidently implement and debug the more complex worm-specific code. And with the provided open-source references, they have a rich resource pool to draw inspiration from at every step.

**References**

[1] [p2pfl/MNIST · Datasets at Hugging Face](https://huggingface.co/datasets/p2pfl/MNIST)

[2] [MNIST digits classification dataset - Keras](https://keras.io/api/datasets/mnist/)

[3] [p2pfl/CIFAR10 · Datasets at Hugging Face](https://huggingface.co/datasets/p2pfl/CIFAR10)

[4] [Your First Image Classifier: Using k-NN to Classify Images](https://pyimagesearch.com/2021/04/17/your-first-image-classifier-using-k-nn-to-classify-images/)

[5] [OpenCV\_DNN\_Image\_Classification.ipynb - Colab](https://colab.research.google.com/github/bigvisionai/upgrad_alumni_workshop_day2/blob/master/opencv_dnn_image_classification/OpenCV_DNN_Image_Classification.ipynb)

[6] [merve/pascal-voc · Datasets at Hugging Face](https://huggingface.co/datasets/merve/pascal-voc)

[7] [Semantic segmentation with OpenCV and deep learning](https://pyimagesearch.com/2018/09/03/semantic-segmentation-with-opencv-and-deep-learning/)

[8] [UCF101 Dataset: Annotated videos for action recognition | Innovatiana](https://www.innovatiana.com/en/datasets/ucf101)

[9] [OTB-100 - Dataset - LDM](https://service.tib.eu/ldmservice/dataset/otb-100)

[10] [MOT15 Challenge Dataset - Kaggle](https://www.kaggle.com/datasets/mdraselsarker/mot15-challenge-dataset)

[11] [MultiTracker : Multiple Object Tracking using OpenCV (C++/Python)](https://learnopencv.com/multitracker-multiple-object-tracking-using-opencv-c-python/)

[12] [Action Recognition on the KTH dataset - GitHub](https://github.com/vkhoi/KTH-Action-Recognition)

[13] [Recognition of human actions - KTH](https://www.csc.kth.se/cvap/actions/)