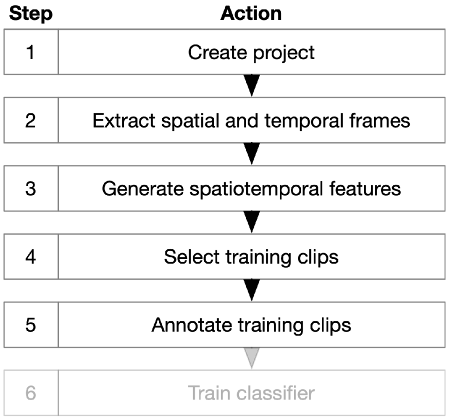
DeepAction Example Script: Creating a New Project

This example covers the basics of creating a new DeepAction project.

Table of Contents



Step	Action
1	Create project
2	Extract spatial and temporal frames
3	Generate spatiotemporal features
4	Select training clips
5	Annotate training clips
6	Train classifier

Setting up the toolbox

The only step required to set up the toolbox is adding the toolbox to your MATLAB path. To do so, change the variable TOOLBOX_PATH below to the location of the toolbox on your system (here, I assume the toolbox is located in the present working directory). The command SAVEPATH saves the current search path so the toolbox doesn't need to be added to the path each time a new MATLAB instance is opened. Comment it out if this is not desirable.

```
toolbox_path = pwd;
addpath(genpath(toolbox_path))
savepath;
```

Downloading the demonstration data

Next, download the demonstration project(s) if you have not done so already. Here, we'll be importing data from ./example_datasets/demo_data, so you only need to download the demo_data folder to run this example. The demonstration data for the examples is included via Google Drive link here.

Creating the project

Next, we define an instance of our DeepAction project by specifying the path to the project folder. When we create a new project, DeepAction will create the specified folder and initialize a configuration file `config.txt`. Here, for example, if we want to create a project named `DeepActionDemo` in the current folder, the path to our project folder is

```
projectName = 'project_1'
projectFolder = fullfile('/Users/harriscaw/Documents/Behavior classification/Projects'
```

We then initialize a DeepActionProject using the specified folder projectFolder as follows:

```
project = DeepActionProject(projectFolder);
```

Then, to create the project, we simply run the `CreateProject()` method, which creates the project folder and populates it with a configuration file.

```
project.CreateProject()
```

The configuration file, `config.txt`, contains the project parameters and is used to run the pipeline. Detailed for the configuration file can be found in `Config.md`. In this example, we will highlight the relevant parameters for steps 1-5 of the DeepAction workflow.

Importing videos

After creating an empty project, we need to import the videos we'd like to label. To do so, we run the `ImportVideos` method on our DeepAction project with a table of the names and paths to the videos we'd like to add.

Here, we use the example dataset provided. We define the names and paths to the videos as follows:

```
videoFolder = '/Users/harriscaw/Documents/Behavior classification/Example datasets/pro
VideoName = {'20080321162447', '20080324115556', '20080324143707A', '20080324143707B',
VideoPath = fullfile(videoFolder, strcat(VideoName, '.mp4'));
```

We then store the video names and paths as a table:

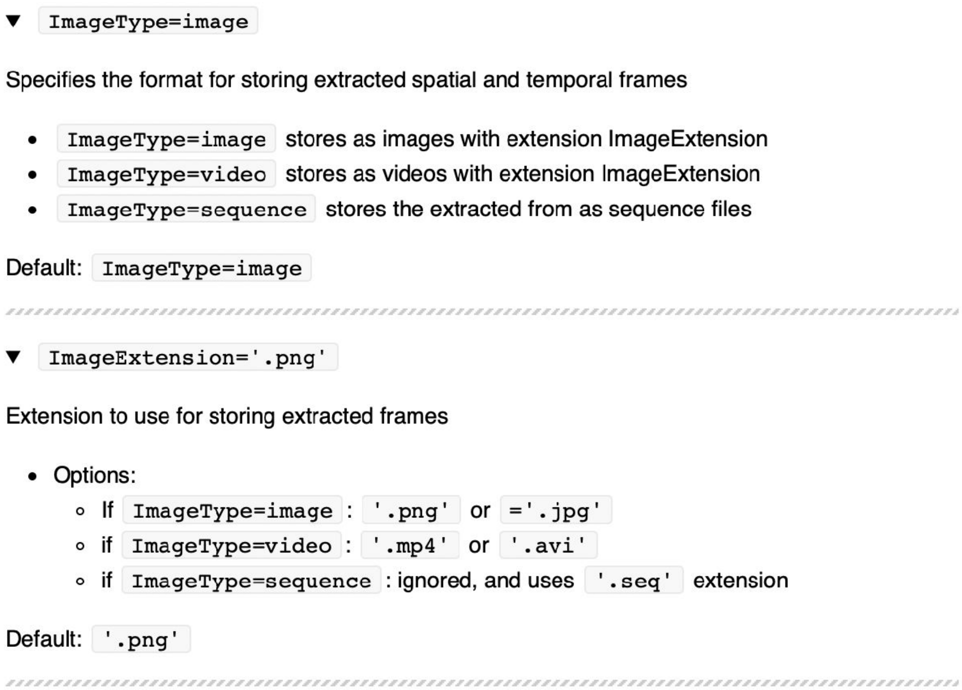
```
importTable = table(VideoName, VideoPath)
disp(importTable)
```

Now, we import the video into the project by calling the `ImportVideos` method. This method copies all the videos specified by `importTable.VideoPath` into the project's `./videos` folder, with names specified by `importTable.VideoName`. It also initializes as set of empty annotations for each video in the `./annotations` folder.

```
project.ImportVideos(importTable)
```

Generating spatial and temporal frames

We then generate the spatial and temporal frames for the project (step 2). To do so, we first specify the format with which to store the extracted frame data. In the configuration file, we do this via the `ImageType` parameter. Here we have three options:



▼ `ImageType=image`

Specifies the format for storing extracted spatial and temporal frames

- `ImageType=image` stores as images with extension ImageExtension
- `ImageType=video` stores as videos with extension ImageExtension
- `ImageType=sequence` stores the extracted from as sequence files

Default: `ImageType=image`

▼ `ImageExtension='.png'`

Extension to use for storing extracted frames

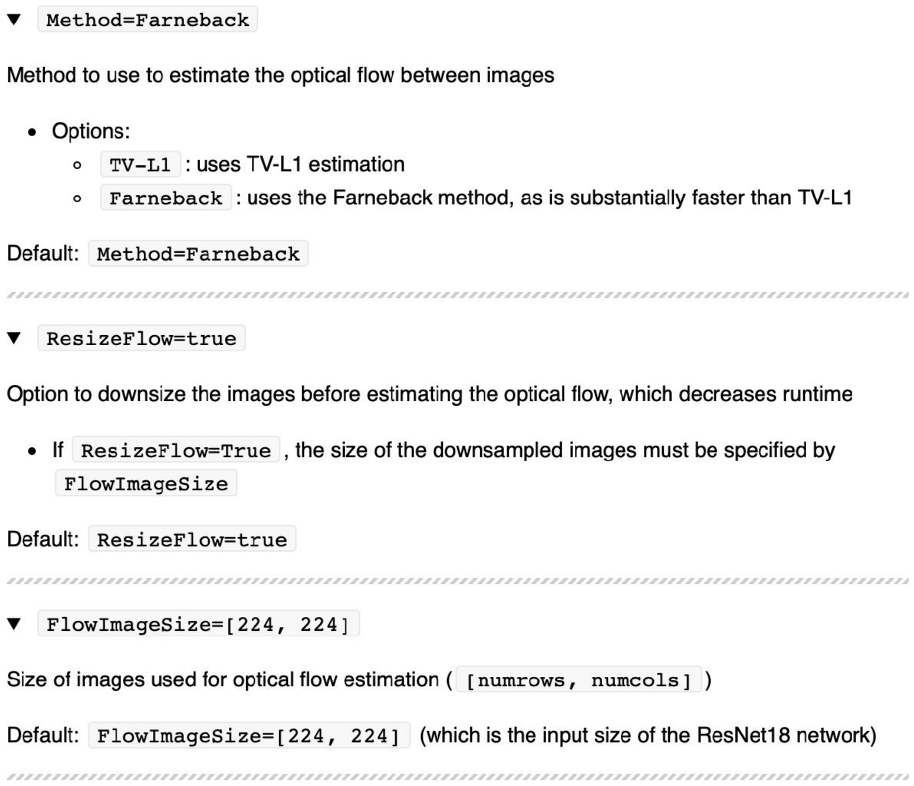
- Options:
    - If `ImageType=image` : `'.png'` or `='.jpg'`
    - if `ImageType=video` : `'.mp4'` or `'.avi'`
    - if `ImageType=sequence` : ignored, and uses `'.seq'` extension

Default: `'.png'`

By default, frames are stores as `.png` images.

We then extract the spatial and temporal frames using the `GenerateFrames`method. By default, both spatial and temporal frames are extracted. To extract only the spatial stream, specify `TemporalStream=false` in the configuration file.

There are a number of options for extracting the temporal stream, which can be computaitonally intensive. They are:



▼ `Method=Farneback`

Method to use to estimate the optical flow between images

- Options:
  - `TV-L1` : uses TV-L1 estimation
  - `Farneback` : uses the Farneback method, as is substantially faster than TV-L1

Default: `Method=Farneback`

▼ `ResizeFlow=true`

Option to downsize the images before estimating the optical flow, which decreases runtime

- If `ResizeFlow=True` , the size of the downsampled images must be specified by `FlowImageSize`

Default: `ResizeFlow=true`

▼ `FlowImageSize=[224, 224]`

Size of images used for optical flow estimation (`[numrows, numcols]`)

Default: `FlowImageSize=[224, 224]` (which is the input size of the ResNet18 network)

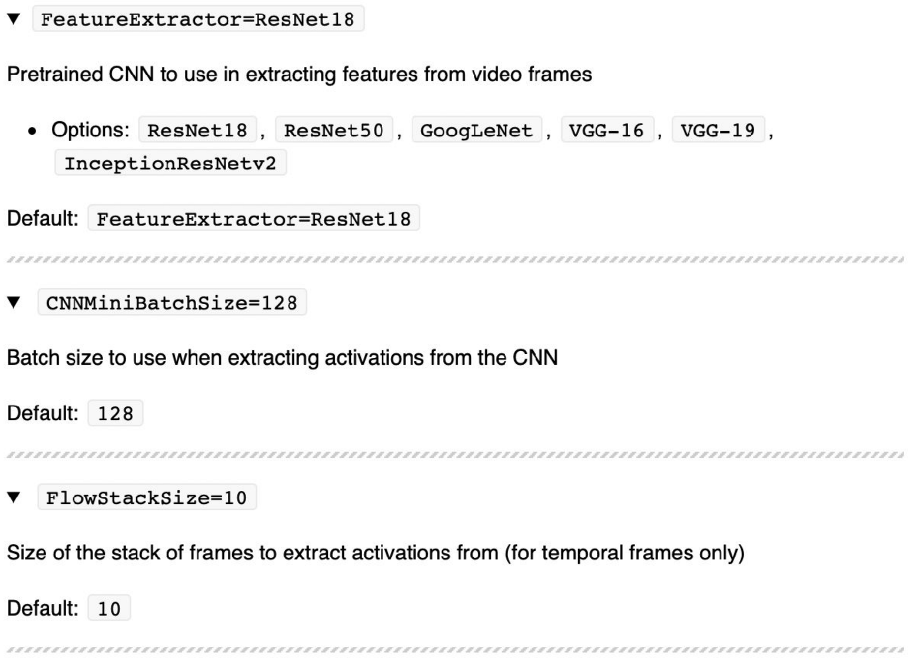
After specifing the desired parameters, we run the `GenerateFrames` method, which extracts frames from all the videos in the project. If the method is interrupted, videos for which all frames have already been extracted will be skipped.

There is an additional option to parallelize the `GenerateFrames` method, which is often faster if sufficient computational resources are available. To do so, specify the parameter `Parallelize` as `true` when running the generate frames command (i.e., `project.GenerateFrames('Parallelize', true)`). By default, the method is not parallelized.

```
project.GenerateFrames();
```

Extracting CNN features

After generating the spatial (and if desired, temporal) frames, we extract features using pretrained convolutional neural networks (CNN) via the `ExtractFeatures` method. The relevant configuration file parameters are described below:



▼ `FeatureExtractor=ResNet18`

Pretrained CNN to use in extracting features from video frames

- Options: `ResNet18` , `ResNet50` , `GoogLeNet` , `VGG-16` , `VGG-19` , `InceptionResNetv2`

Default: `FeatureExtractor=ResNet18`

▼ `CNNMiniBatchSize=128`

Batch size to use when extracting activations from the CNN

Default: `128`

▼ `FlowStackSize=10`

Size of the stack of frames to extract activations from (for temporal frames only)

Default: `10`

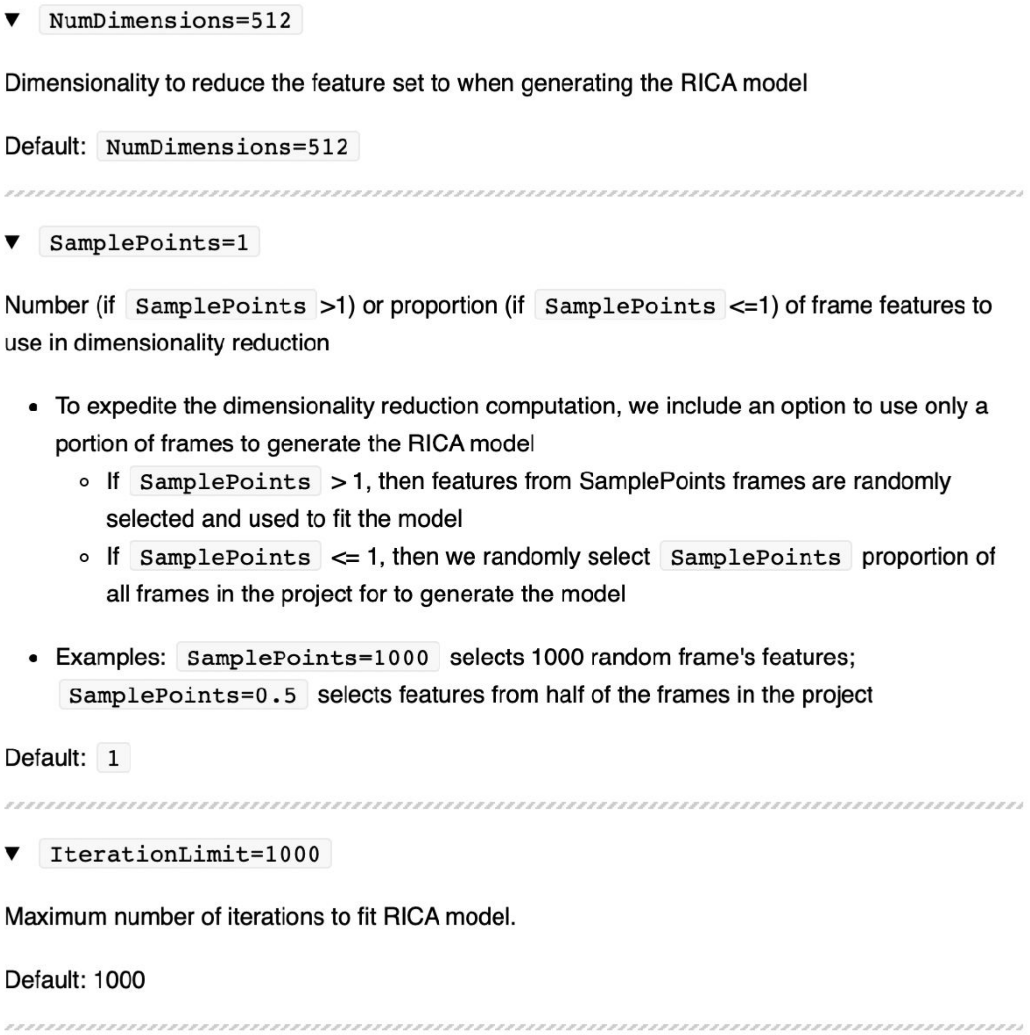
As with frame generation, we again have the option to parallelize this process. To do so, specify `project.ExtractFeatures('Parallelize', true)`. By default the method does not use parallelization.

```
project.ExtractFeatures();
```

Reducing feature dimensionality

After spatial and temporal features have been extracted from video frames, there is an option to reduce the size of the input features to the classifier via dimensionality reduction. This is useful for decreasing training time and memory usage, and we have found it is significantly improved classifier performance. In this example we use reconstruction independent component analysis (RICA) to create reduce the dimensionality of the features from 1024 to 512.

The method GenerateRICAModel is used to create a RICA model object, which is stored in the ./rica_models subfolder of the project folder. After the model is generated, it can be reused, so this process only needs to be completed once (for a given set of input streams and reduced dimensionality). The parameters used in generating the RICA model are as follows:



▼ `NumDimensions=512`

Dimensionality to reduce the feature set to when generating the RICA model

Default: `NumDimensions=512`

▼ `SamplePoints=1`

Number (if `SamplePoints` >1) or proportion (if `SamplePoints` <=1) of frame features to use in dimensionality reduction

- To expedite the dimensionality reduction computation, we include an option to use only a portion of frames to generate the RICA model
  - If `SamplePoints` > 1, then features from SamplePoints frames are randomly selected and used to fit the model
  - If `SamplePoints` <= 1, then we randomly select `SamplePoints` proportion of all frames in the project for to generate the model
- Examples: `SamplePoints=1000` selects 1000 random frame's features; `SamplePoints=0.5` selects features from half of the frames in the project

Default: `1`

▼ `IterationLimit=1000`

Maximum number of iterations to fit RICA model.

Default: 1000

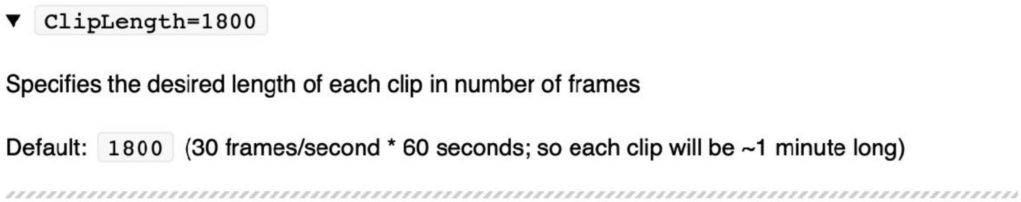
```
project.GenerateRICAModel()
```

Next, we define using the set of behaviors of interest in our project using the *Behavior Table Application* included in this toolbox. In addition to the names of the behaviors, we also require uses provide a keyboard key corresponding to that behavior of interest. This will be used in the manual annotation app to denote the start/stop of behaviors by keypress. Behaviors can be added and deleted via the corresponding buttons. Behavior labels and keys can be edited by clicking on the corresponding cells.

```
BehaviorTable(project)
```

Manual annotation of training data

After features have been extracted, we select frames to manually label for use in training the classifier. To do so, we divide each video into short clips and then select a random subset of those clips to annotate. To relevant parameter for dividing the project videos into clips is:



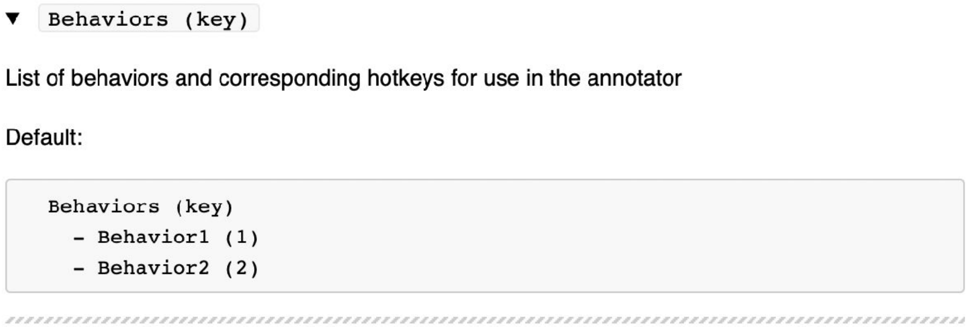
▼ `ClipLength=1800`

Specifies the desired length of each clip in number of frames

Default: `1800` (30 frames/second * 60 seconds; so each clip will be ~1 minute long)

```
project = project.GetAnnotatorData()
```

We can then annotate the clips using the annotator. We have the option to specify the set of behaviors used in the annotations prior to launching the annotator via the `Behaviors` parameter, or we can add, delete, and edit behavior (and their corresponding hotkeys) within the app.



▼ `Behaviors (key)`

List of behaviors and corresponding hotkeys for use in the annotator

Default:

```
Behaviors (key)
  - Behavior1 (1)
  - Behavior2 (2)
```

```
project = project.LaunchAnnotator();
```

Recap

In this segment we demonstrated the process for creating a project, generating spatial and temporal frames and features, and launching the manual annotator.