1. **Transfer Learning for Video Classification**[1]

   It is highly recommended that you complete this project using Keras[2] and Python.

   (a) In this problem, we are trying to build a classifier that distinguishes videos of 5 different activities .

   (b) **Video Classification**

   Videos can be viewed as a sequences of individual images; therefore, one can deal with video classification as performing image classification L times, where L is the number of frames in a video. However, this setting has a problem: it does not capture the temporal nature of the video. That is why we use RNN-CNN for video classification. Training RNN-CNN's needs a lot of computational resources, and it might sometimes be excessive, so in this project we will take a simpler approach.

   (c) **Moving Averages over Predictions for Video Classification**

   In this approach, we train a CNN for image classification, and turn it into a video classifier using moving averages. If we rely on only one or a few frames of a video for classifying it, we will observe a phenomenon called flickering, which is assigning different labels to different frames of a video in the same class. However, if we use an average of predictions of probabilities for multiple frames, we will get a more reliable prediction for a video.

   (d) **Data Exploration and Pre-processing**

   i. Images in each class are given in separate folders in the folder `Sport Images`. We have several hundreds of images for five classes of sport activities, namely baseball, basketball, fencing, tennis, and volleyball.

   ii. Randomly select $\lceil 0.7n_i \rceil$ images from each folder as your training set, $\lceil 0.15n_i \rceil$ as validation set, and the rest as your test set, where n_i is the number of images in folder i and $\lceil x \rceil$ is the ceiling of x.

   iii. In order for all the images to have the same size, zero-pad or resize the images in your dataset. This can be done using various tools, including OpenCV.

   (e) **Transfer Learning for Image Classification**[3]

   i. When dealing with classification of relatively small image datasets, deep networks may not perform very well because of not having enough data to train them. In such cases, one usually uses *transfer learning*, which uses deep learning models that are trained on very large datasets such as `ImageNet` as feature extractors. The idea is that such deep networks have learned to extract meaningful features from an image using their layers, and those features can be used in learning other tasks. In order to do that, usually the last layer or the last few layers of the pre-trained network are removed, and the response of the layer before the removed layers to the images in the new dataset is used

[1] I acknowledge the contribution of my assistant Changxun Li in preparing this project.

[2] `https://keras.io`

[3] `https://builtin.com/data-science/transfer-learning`

as a feature vector to train one more multiple replacement layers. The dataset in this task has only several hundred images per class. Given that we have 5 classes, training a deep network with such a small dataset may not yield desirable results. In this project, you will use pre-trained models `ResNet50`, `EfficientNetB0`, and `VGG16`. For both pre-trained networks, you will only train the last fully connected layer, and will *freeze* all layers before them (i.e. we do not change their parameters during training) and use the outputs of the penultimate layer in the original pre-trained model as the features extracted from each image.

ii. To perform empirical regularization, crop, randomly zoom, rotate, flip, contrast, and translate images in your training set for image augmentation. You can use various tools to do this, including OpenCV.

iii. Use ReLU activation functions in the last layer and a softmax layer, along with batch normalization [4] and a dropout rate of 20% as well as ADAM optimizer. Use multinomial cross entropy loss. You can try any batch size, but a batch size of 5 seems reasonable.

iv. Train the networks `ResNet50`, `EfficientNetB0`, and `VGG16` for at least 50 epochs (preferably 100 epochs) and perform early stopping using the validation set. Keep the network parameters that have the lowest validation error. Plot the training and validation errors vs. epochs.

v. Report the Confusion Matrix, Precision, Recall, Accuracy, and F1 score for your model on both training and test sets. Remember that this is a multi-class classification problem.

(f) **Video Classification Using Moving Averages**

i. In order to have a better deep learner, reuse the validation and test data and train the network, without seriously overfitting it. In the lectures, we saw how this can be done.

ii. Apply at least L equally spaced frames of each video in the folder `Sport Videos`[5] to your model to obtain L vectors of probability predictions from the softmax in your model. You must choose L to be at least 100, but you are welcome to use all the frames in each video. Calculate the average $\bar{\mathbf{p}}$ of these probability vectors for each video.

iii. Select the class with maximum probability in the vector $\bar{\mathbf{p}}$ for each video and compare it to the actual label of the video.

iv. Report the Confusion Matrix, Precision, Recall, Accuracy, and F1 score for your model on the test data, i.e. videos. Remember that this is a multi-class classification problem.

[4] `https://en.wikipedia.org/wiki/Batch_normalization`

[5] Videos are in `.avi` format, that can be read by OpenCV. In case you encountered difficulty with this format, you can convert the videos to `avi`.