M-PACT: An Open Source Platform for Repeatable Activity Classification Research

Eric Hofesmann  
CSE, University of Michigan  
erichof@umich.edu

Madan Ravi Ganesh  
ECE, University of Michigan  
madantrg@umich.edu

Jason J. Corso  
ECE, University of Michigan  
jjcorso@umich.edu

Abstract

There are many hurdles that prevent the replication of existing work which hinders the development of new activity classification models. These hurdles include switching between multiple deep learning libraries and the development of boilerplate experimental pipelines. We present M-PACT to overcome existing issues by removing the need to develop boilerplate code which allows users to quickly prototype action classification models while leveraging existing state-of-the-art (SOTA) models available in the platform. M-PACT is the first to offer four SOTA activity classification models, I3D, C3D, ResNet50+LSTM, and TSN, under a single platform with reproducible competitive results. This platform allows for the generation of models and results over activity recognition datasets through the use of modular code, various preprocessing and neural network layers, and seamless data flow. In this paper, we present the system architecture, detail the functions of various modules, and describe the basic tools to develop a new model in M-PACT.

1. Introduction

Empirical fields like activity classification in computer vision demand repeatable and measurable software systems to support benchmarking and research progress. The availability of large-scale computational resources as well as open-source deep learning software, including Tensorflow [11], Pytorch [14], and Caffe [8], have taken steps toward providing such software systems. Shared, open-source deep learning library code reduces the expected time to develop new models. However, these libraries fall short of meeting the full demands of repeatable and measurable software systems: models developed in one library are not easily integrated with models from another library; reported results may not use the same practices or even be reproducible; the necessity to switch between different libraries greatly slows down the research cycle. The delays in model integration across libraries are, in part, due to differences in compilation procedures and general coding structures wherein even keywords can significantly vary in meaning. For example, “dropout” can indicate the ratio of nodes to keep (Tensorflow) or to drop (PyTorch), depending on the library being used.

In this work, we introduce the Michigan Platform for Activity Classification in Tensorflow, a platform that allows the user to focus solely on the creation or fine-tuning of models while providing simple abstractions for functions like data input, metric computation, and feature extraction. Users do not need to build their experimental pipeline from scratch and can quickly prototype models to run experiments. Furthermore, M-PACT provides implementations of several SOTA models allowing users to leverage the potential of existing models. Since M-PACT is developed using a single language and library, Python and Tensorflow, and a consistent coding style throughout, it avoids the complication of interpreting code written in different libraries or following different practices. Thus, M-PACT removes the necessity of re-implementing existing code from other libraries and the time to set up and run experiments all while keeping the user focused on developing new activity classification models.

The main contributions of M-PACT are:
• In terms of usage
  – Availability of SOTA activity classification models under a common platform.
  – A multitude of metrics that allow for quick and efficient model testing which standardizes comparisons among various models.
  – Ability to extract features from existing SOTA activity classification models.
  – Single node multi-GPU support to train and fine-tune models.
• In terms of design
  – Efficient and clear coding style, supplemented with simple comments, that allows users to assimilate and develop models quickly.
  – Modular code design that offers a vast array of customizable options for deep neural network layers and video preprocessing functions.
  – Customized input pipeline that condenses the loading and processing of videos into a set of flexible and easy-to-use options for the user.

2. Related Work

Most deep learning libraries, like Caffe, have a built-in model zoo containing standard image feature extraction models like VGG16 [17], AlexNet [10], and ResNet [6] to help bootstrap model development. To the best of our knowledge, these libraries do not offer models pre-trained on video-based tasks. Within most standard libraries, videos are accessed and handled explicitly as a collection of image files under a specific video directory. This introduces large processing overheads with respect to the file access-read-process functionality. When combined with the lack of video- and clip-specific preprocessing functions, this forces video-based models to perform suboptimally.

To the best of our knowledge we are the first to provide a compilation of SOTA activity classification algorithms under a single library. This is a difficult task because of two reasons, 1) cross-compatibility issues between different versions of a library and 2) differences in function implementations between libraries. For example, in Caffe, researchers modify the base code causing cross-compatibility issues when combining multiple models. In Tensorflow, variants like TF-slim [15] and Keras [3] form unreliable bases for M-PACT because of subtle differences in function implementations and varying abstraction levels.

Among the many possibilities of contemporary libraries, we chose Tensorflow since it is well-adapted and known to produce fast systems. Through Tensorflow, M-PACT can provide video- and clip-level preprocessing functions along with existing frame-wise preprocessing. By explicitly dealing with data as videos and clips, offering a vast collection of clip-level functions to the user, and combining some of the best features from existing libraries we provide a large platform for the quick development of video-based models.

3. Breakdown of M-PACT

M-PACT consists of three main components, 1) Input Data Block, 2) Model Definition Block, and 3) Execution Block, as shown in Fig. 1. Aside from custom tasks, the Model Definition Block is the only portion of M-PACT in which the user is required to write code.

M-PACT follows a strict file structure convention, shown
in Fig. 2 that must be followed. The main M-PACT directory contains the python files necessary to train, test, and create a model and the directories containing logs, results, models, scripts, and utils. Results and logs contain the model weights and tensorboard logs of trained and tested models. Scripts contains the shell script used to download the initial weights for the provided models in M-PACT. The utils and models directories store a host of useful files and directories with a fixed structure.

M-PACT provides extensive flexibility in loading data and quickly prototyping and experimenting on new models. The following sections will provide a deeper discussion about the Input Data Block, the Model Definition Block, and the Execution Block respectively.

3.1. Input Data Block

The Input Data Block is the entry point for data within M-PACT. Its overall functionality can be divided into two broad stages: 1) reading video data, from TFRecords, into a format compatible with the platform, and 2) extracting clips from videos and preprocessing them. Fig. 3 illustrates the structure and flow of data as it passes through this block.

3.1.1 Read video data

All the components of the Input Data Block are constructed in Tensorflow to facilitate efficient and parallel data loading. The names of the video files for a selected dataset are loaded into a file queue. Each dequeued file name is then read and broken down into a user-specified number of clips which get processed by a model-specific preprocessing function. To allow the extraction of multiple clips from a given video, a separate clips queue is used to store all of the clips. The file queue and clips queue work together to ensure exactly one mini-batch worth of clips is passed into a model during each iteration. If a video contains more than one mini-batch of clips, then the excess clips will be provided to the next mini-batch. When there are too few clips in a video to form a mini-batch, then the missing clips are generated from a new video.

3.1.2 Extract clips

In typical video understanding work, it is sometimes more useful to divide a video into multiple clips and process them individually instead of operating on the entire video at once. To this end, M-PACT offers multiple options to break down a video into any desired number of clips. In order to do so, a number of clip specifications such as the clip length, total number of clips per video, offset from beginning of video to select a clip, and the stride of the sliding window from which clips are extracted, can be provided in any permutation. These specifications are provided in the form of command line arguments. Fig. 4 lists a number of possible ways to construct a clip(s) from a given video.

Once clips have been extracted, they are automatically
Figure 5. The Model Definition Block is comprised of the 1) input video preprocessing steps, 2) network layer definitions, 3) loss specification, and 4) initial weights to load. This block contains the aspects of M-PACT that the user develops when creating a new model. The leftmost “Inputs” section of this figure represents the video data that is passed from the Input Data Block to the Model Definition Block.

processed using a selected model’s preprocessing pipeline which is defined within the Model Definition Block. Preprocessed clips can significantly vary in shape (height and width) as well as the number of frames when compared to the input video. However, the shape of the preprocessed clip is fixed throughout the remainder of the pipeline and each subsequent block regardless of the shape of the input video.

3.2. Model Definition Block

The Model Definition Block, shown in Fig. 5, includes the files and procedures that form a model in M-PACT and it is the only portion which requires mandatory coding. This block contains user-defined input video preprocessing steps, network architecture and layer definitions, losses associated with a model, and pre-trained initialization weights. To ease model development, M-PACT contains pre-defined preprocessing and network layer definitions, as well as an automated template generation system to ensure compatibility of a new model with the rest of the platform. Additionally, a multitude of command line arguments, for example the sequence length of an LSTM, are also accessible within the model class.

3.2.1 Preprocessing

The preprocessing pipeline, a crucial component of the Input Data Block, is defined with the Model Definition Block. There is a host of video and image processing steps which are commonly used for activity classification. These steps exist to either standardize the frames between videos, so that the input to the network is consistent, or to augment input data to alter the way the network learns. M-PACT offers a collection of common preprocessing functions, e.g. cropping and resizing, to be used in developing a custom preprocessing pipeline. More importantly, M-PACT is the first, to our knowledge, to offer video- and clip-level preprocessing functionality like temporal resampling, clip-based cropping or shuffling, and more.

With the wide range of preprocessing options that are available, it is important to try different combinations to determine which help a model perform well. To facilitate this, M-PACT supports the implementation of multiple preprocessing pipelines for a single model which can be selected using a command line argument during training or testing time. Each of these preprocessing pipelines must exist in independent files wherein each file contains a preprocess function which returns a processed video. This preprocess function takes a clip from the Input Data Block and returns the clip processed both spatially and temporally. The shape of this processed clip must match that expected by the model.

3.2.2 Model Architecture

Once the shape and characteristics of the processed videos have been fixed, they are passed in to the inference function of the model. The inference function contains the entire sequential definition of the layers that make up the model. M-PACT offers a collection of pre-defined neural network layer definitions, similar to the collection of functions available for video preprocessing, including 2D and 3D convolutions, fully connected layers, and more. These layers also occasionally add new functionality that is not available in Tensorflow; e.g., the grouping of convolutional filters from Caffe and custom height and width padding.

Since M-PACT is built atop Tensorflow, any basic Tensorflow functions are also applicable between layers.

The model file itself contains the definition of the model class and relevent methods including inference, loss, and initial weight loading, all of which are unique to each model. As previously mentioned, the preprocessing functions are defined in separate files and called inside of the model file while the inference, loss, and loading of initial weights are defined directly in the model file. The loss function is internally defined within each model in order to offer the flexibility of customized losses. Similar to preprocessing-
ing, the user may want to test various losses to determine which is most useful for a given model. M-PACT contains a command line argument that allows the user to switch between different user-defined loss types. Finally, the model class also contains an option to load initial weights for a model that has been pre-trained within the platform or otherwise.

### 3.3. Execution Block

The training and testing of activity classification models requires boilerplate code that every researcher must write before being able to run experiments. The **Execution Block** abstracts away this boilerplate code to a set of flexible command line arguments while at the same time offering the added functionality of checkpoint and metrics modules. Fig. 6 illustrates the general outline of the **Execution Block** and highlights the key modules used in training and testing phases. The following sections detail the relevant functionality of the **Execution Block** and describe the checkpoint and metric calculation systems of M-PACT.

#### 3.3.1 Training

The training pipeline used in M-PACT is similar to most standard Tensorflow-based training pipelines. It includes the definition of a model and an optimizer, loading of weights, application of gradients, execution of a Tensorflow session that runs the training operation, and the saving of a model’s state and metrics.

Two important points to note in the training process of M-PACT are the utilization of GPUs and the definition of epochs. Once the setup is complete, the model is replicated across a selected number of GPUs in the compute node and processed clip data from the **Input Data Block** are interfaced with the model. Currently, M-PACT only supports the extension of a model to multiple GPUs within a single compute node.

During the execution of the training phase, models are trained for a user-specified number of epochs. The number of iterations in each epoch is based on the number of videos in a given dataset, the mini-batch size, and the number of clips per video. Within a user-specified frequency of epochs, M-PACT saves a model’s state using the **Checkpoint Module**’s `save()` function. It is important to note that during the setup of an experiment, M-PACT offers adaptive learning rate control which steps down the learning rate when the training loss plateaus, alongside a variety of native optimizers.

#### 3.3.2 Testing

The testing pipeline loads the most recently saved state of a selected model and can be used to extract features or to calculate a model’s classification accuracy using the **Metrics Module**. When extracting information from a model, any layer within the specified model can be chosen using a command line argument. Additional command line arguments are available during the testing phase which add functionality like averaging predictions across clips of a video instead of classifying each clip individually or applying a softmax function to model outputs for classification.

#### 3.3.3 Checkpoint Module

An important functionality in any experimental setup is the ability to save and load the state of a model in the form of checkpoints. The **Checkpoint Module** contains functions that handle all of the essential operations with respect to checkpoints. The ability to save and recover checkpoints is crucial to postprocess data, fine-tune a model from any desired state, and restart training in case of a critical hardware failure. Each checkpoint is saved in a unique directory corresponding to the chosen dataset, preprocessing method, experiment name, and metric.

It is often difficult to convert checkpoints between various formats. For example, when weights from a Caffe model need to be loaded into a Tensorflow model, the `.caffemodel` must be loaded in Caffe and saved, possibly as a `numpy` file, before starting a Tensorflow session and saving the weights as a `.ckpt` file. To avoid this two-step process, the **Checkpoint Module** automatically saves and loads weights in a `numpy` format. This allows weights generated using M-PACT to be loaded more easily into other deep learning libraries. However, loading weights trained in other deep learning libraries into M-PACT is more complicated due to the variability in how model weights are stored and their dimensionality between libraries. Since there is no one-size-fits-all approach to converting weights into M-PACT, we provide a detailed code example in `utils\convert_checkpoint.py` to help
guide the user in converting model weights to an M-PACT-compatible format.

### 3.3.4 Metrics Module

Evaluating the performance of a trained model is a key part of the testing pipeline. M-PACT offers the Metrics Module for testing and logging the activity classification performance of a given model as well as extracting features from it. During testing, the chosen metric and any other specified variable is logged automatically using tensorflow.

Available performance metrics include average pooling and last frame prediction with the option of training a linear SVM classifier on extracted features. Average pooling is a common technique which uses the softmax per-class predictions of a model and averages them across all frames of an input video. From these averaged predictions, the label with the maximum value is used as the overall class prediction for that video. Last frame prediction simply uses the softmax per-class prediction of the last frame in the video to generate the overall class prediction. Last frame classification is referenced originally in the ConvNet+LSTM model [5]. In addition to these metrics, which directly classify the output of a model, M-PACT also provides the option of training a linear SVM classifier based off of features extracted from any user-specified layer within the model. This SVM is tested within the Metrics Module using the features extracted from each testing video.

The method used to calculate the performance of a model can vary depending on the output of the model. For example, C3D classifies multiple clips in a video individually and then averages these predictions to produce a single video-level prediction. On the other hand, TSN produces a single prediction for each video. The Metrics Module is robust enough to handle both such cases.

### 4. Using M-PACT

M-PACT is developed around two main use cases: 1) to extract features from activity classification models implemented in M-PACT, either as video representations or for classification, and 2) to serve as a development platform for new activity classification models. The only time that the user is required to write any code is when setting up a new model’s layer architecture and preprocessing pipeline. However, in order to successfully use the platform, the user must first convert their video dataset into the format expected by M-PACT.

#### 4.1. Dataset Conversion

M-PACT uses TFRecords to store videos and Tensorflow queues to parse the data. The efficiency of the M-PACT pipeline comes from the fact that the reading and processing of data is built into the Tensorflow graph and no external user input is required.

Any video dataset needs to be converted to TFRecords in order to be compatible with the M-PACT data pipeline. This can be accomplished by executing `generate_tfrecords_dataset.py`, found under the `utils` directory as shown in Fig. 2. This conversion file expects data to be stored in a video format, e.g. avi or mp4, within subdirectories named according to the video’s action class. The input to the conversion file is a command line argument indicating the path to the parent directory of the dataset. The final TFRecord, one for each video, stores eight attributes: data, number of frames, height, width, number of channels, true label, and the name of the video file as shown on the left side of Fig. 1.

### 4.2. Existing Models

Once a dataset is converted to the required format, users can readily apply the existing SOTA activity classification models, I3D, C3D, ResNet50+LSTM, and TSN, to any custom dataset. Since the SOTA models are available within M-PACT, users can quickly compare with or improve upon the generated results. The complete model definition and relevant weights, including a script to retrieve the weights from an online server, is included within the platform. Classification results or features can be extracted from these models by running `test.py` along with relevant command line arguments. If a dataset other than HMDB51 [11] or UCF101 [13] is used to generate results on existing models, then these models will need to be fine-tuned by running `train.py` with relevant command line arguments.

### 4.3. Model Development

M-PACT streamlines the training and testing pipeline to run from simple command line arguments and also requires only a minimal amount of coding to develop and use a new model. Model development begins by using the `create_model.py` program to generate a template for the model and preprocessing files, as shown in Fig. 2. This python file takes the model name as an input argument and creates a new model directory containing model and preprocessing files using that name. The structure of this new model directory will match those of pre-existing models like I3D, C3D, ResNet50+LSTM, and TSN, as seen in the center of Fig. 2.

The majority of code development occurs within the model and preprocessing files. The model file contains `TODO` statements in the methods “inference” and “loss” which must be implemented by the user with the available layer definitions and any Tensorflow loss function. Other optional functions, for example `preprocess_tfrecords` and `load_default_weights`, are available and have
| Base Model  | Pretraining     | HMDB51 Acc. (%) | UCF101 Acc. (%) |
|------------|----------------|-----------------|-----------------|
|            |                | Ours | Authors | Δ    | Ours | Authors | Δ    |
| C3D        | Sports-1M      | 52.94| 50.30 * | +2.64| 79.14| 82.30 * | +2.16|
| I3D        | Kinetics       | 66.54| 74.80 * | -8.26| 93.18| 95.60 * | -2.42|
| TSN        | ImageNet       | 51.70| 54.40   | -2.70| 85.25| 85.50   | -0.25|
| ResNet50+LSTM | ImageNet     | 45.36| 43.90   | +1.46| 79.25| 84.30   | -5.05|

Table 1. Mean RGB classification accuracies are shown for various SOTA action classification models across split 1 of HMDB51 and UCF101. Values marked with a (*) indicate averaged results across all three splits of a given model. Accuracies are reported for each baseline model and pretraining dataset combination. The weights used were pretrained on the video recognition datasets Sports-1M [9] and Kinetics [6] as well as the image recognition dataset ImageNet [4].

5. Benchmarks

A main contribution of M-PACT is the access it provides to activity classification results on SOTA models under a common platform. In this section, we provide numerical evaluations of M-PACT in terms of the activity classification performance of available SOTA models as well as speed and memory capacity.

5.1. SOTA Performance

Activity classification models can be roughly divided into two categories: 3D convolution-based and 2D convolution-based that apply a temporal consensus. Of the four SOTA models that have been implemented in M-PACT, two are 3D convolution-based models, I3D [2] and C3D [20], and two are 2D convolution-based models, TSN [22] and ResNet50+LSTM [6]. The focus of M-PACT is currently on RGB-based models only, thus two stream networks [16] have not been implemented.

HMDB51 [11] and UCF101 [13] are the activity classification datasets that have been used to replicate the performance of the SOTA models. These datasets were selected because the original authors of all four of the SOTA models provide results on both datasets. Newer and larger activity classification datasets, for example Kinetics [6], or even event classification datasets, for example Moments in Time [12], can be converted to TFRecords format and used in M-PACT. The deviation in performances between the original and M-PACT models can be attributed to a number of reasons including the lack of public model weights, missing details in the preprocessing and training procedures, unspecified hyperparameters during fine-tuning, and a conversion from the model’s native deep learning library to Tensorflow. The following subsections will detail the implementation of each model.

5.1.1 I3D

I3D [2] has the highest published performance out of all models implemented in M-PACT. It uses the Inception-V1 [19] network as a backbone and replicates each 2D convolutional filter to form 3D convolutions. While the code released for I3D is written in Tensorflow, only model weights trained on the Kinetics dataset [6] and a testing script are provided. Since no training code has been released, the I3D training procedure in M-PACT is reproduced from the written descriptions of the training process in the code repository and the original paper itself. Due to the uncertainty in the training procedure, there exists a drop in performance between the M-PACT and the authors results on HMDB51 and UCF101. However, even with the drop in performance I3D continues to be the best performing model in M-PACT.

5.1.2 C3D

C3D [20] is built around 3D convolutions, like I3D, and is commonly used for feature extraction and fine-tuning on video-based tasks. The original implementation of C3D is publicly available along with the fine-tuning procedure on UCF101. However, all code is written using Caffe and only the weights for C3D trained on Sports-1M [9] are given. The M-PACT implementation of the model and its training and testing pipelines follow the original Caffe code, used to fine-tune the model on UCF101, as closely as possible.

In the work introducing C3D [20], the model trained on Sports-1M is tested on UCF101 by extracting features and training an SVM. However, in subsequent work [21] C3D has been fine-tuned on both UCF101 and HMDB51. We
fine-tune and test C3D, rather than train an SVM, to maintain consistency with the other models in M-PACT.

5.1.3 TSN

The original TSN [22] model is written using Caffe, though the authors also provide an implementation in Pytorch. TSN is the only model of those in M-PACT where the original authors have provided both HMDB51 and UCF101 fine-tuned weights. In an effort to replicate the published performance as closely as possible, we use the provided weights in the presented results. However, the training pipeline for TSN has also been reimplemented in M-PACT so that the model can be retrained if desired.

5.1.4 ResNet50 + LSTM

The original concept of training an LSTM on the features from a 2D convolutional neural network (CNN) was conceived in the LRCN model [5]. ResNet50 + LSTM is a baseline presented in conjunction with the Kinetics [6] dataset and is used in M-PACT because it achieves a higher performance than LRCN. The model was constructed, trained, and tested according to the descriptions given in the original paper [6] since no code or weights are available for it. While this model may not perform as well as other models, LSTM [7]-based models are common in video tasks and should exist in a video classification platform.

5.2. Memory and Speed Tests

The pace of growth in hardware memory size is smaller than the growth in network dimensionality making it crucial that M-PACT maintains as small a memory footprint as possible when training and testing models. M-PACT is designed to be compatible with a CUDA-ready Nvidia GPU with sufficient VRAM to load a model with at least a single mini-batch of data into memory.

Table 2 shows the amount of GPU memory that each model within M-PACT uses when the mini-batch size is set to one. I3D and C3D consume the most memory due to their use of 3D convolutional filters. ResNet50+LSTM uses significantly less since it only consists of a 2D CNN feeding into an LSTM. TSN is much smaller than any other model since it consists of only a 2D CNN classifying frames individually and averaging the predictions. Because of this, the mini-batch size during training can be much larger for TSN than for any other model.

Tensorflow was selected as the backbone of M-PACT, in part, due to the performance and minimal overhead that can be achieved during runtime. Table 2 enumerates the results from performance tests and quantifies the time taken to train networks and calculate classification metrics within M-PACT. For most models, the difference between training and testing speed is less than 0.25 seconds. However, the testing speed of TSN is nearly five times its training speed. This is likely due to the use of oversampling during the preprocessing of testing videos for TSN. Oversampling a video creates ten copies of each frame where each copy is either a crop, either from the center or a corner of the frame, or the mirrored versions of a crop.

The data shown in Table 2 should only be taken as a reference of the performance achieved in M-PACT. The speed and memory consumption of M-PACT will likely vary between users due to differing hardware and environments.

6. Conclusion

In field of activity classification, SOTA models are spread across languages and research pipelines making it difficult to compare and build upon their results. For example, the initial weights of C3D and TSN were trained in Caffe, ResNet50 in Keras, and I3D in Tensorflow. Each was written and trained using different hyperparameters which are often not available to the public. In these cases, experimentation is required to replicate the performance of a model. We present M-PACT, an open-source Tensorflow-based platform which provides a unified pipeline with access to SOTA activity classification models and an environment to develop new models.

M-PACT will serve as a basis for future models and remove the burden of replicating results and reimplementing networks from the community. M-PACT houses a variety of blocks and modules which make it quick and easy for a user to generate results on a model or develop their own model. The training and testing of these models has been streamlined down to command line calls with multiple arguments which can be used to alter the flow of data. Through GitHub, M-PACT will accept pull requests of new activity classification methods that are developed in the platform. The goal of M-PACT is to serve as the all encompassing and continually updated benchmarking and development plat-

| Model       | Memory | Train Speed | Test Speed |
|-------------|--------|-------------|------------|
| I3D         | 8.46 Gb| 0.91 s      | 1.15 s     |
| C3D         | 7.44 Gb| 0.17 s      | 0.14 s     |
| TSN         | 0.91 Gb| 0.18 s      | 0.92 s     |
| ResNet50+LSTM| 4.36 Gb| 0.84 s      | 0.93 s     |

Table 2. The table shows the GPU memory, shown in gigabytes, used by four SOTA activity classification models in M-PACT. These values were obtained when training on HMDB51 on a NVIDIA GTX 1080ti video card using a mini-batch size of one. The table also shows the speed, in seconds, to train and test these models. The training and testing times, on HMDB51 with a mini-batch size of one, are averaged over 1000 iterations.
form for SOTA activity classification.

7. Acknowledgements

Supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DOI/IBC) contract number D17PC00341. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DOI/IBC, or the U.S. Government.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In OSDI, volume 16, pages 265–283, 2016.
[2] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 4724–4733. IEEE, 2017.
[3] F. Chollet et al. Keras, 2015.
[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 248–255. Ieee, 2009.
[5] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2625–2634, 2015.
[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.
[8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014.
[9] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In CVPR, 2014.
[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.
[11] H. Kuehne, H. Jhuang, R. Stiefelhagen, and T. Serre. Hmdb51: A large video database for human motion recognition. In High Performance Computing in Science and Engineering J2, pages 571–582. Springer, 2013.
[12] M. Monfort, B. Zhou, S. A. Bargal, T. Yan, A. Andonian, K. Ramakrishnan, L. Brown, Q. Fan, D. Gutfruend, C. Vondrick, et al. Moments in time dataset: one million videos for event understanding.
[13] T. E. Oliphant. A guide to NumPy, volume 1. 2006.
[14] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.
[15] N. Silberman and S. Guadarrama. Tf-slim: A high level library to define complex models in tensorflow.
[16] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In Advances in neural information processing systems, pages 568–576, 2014.
[17] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.
[18] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. arXiv preprint arXiv:1212.0402, 2012.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9, 2015.

[20] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In Proceedings of the IEEE international conference on computer vision, pages 4489–4497, 2015.

[21] D. Tran, J. Ray, Z. Shou, S.-F. Chang, and M. Paluri. Convnet architecture search for spatiotemporal feature learning. arXiv preprint arXiv:1708.05038, 2017.

[22] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In European Conference on Computer Vision, pages 20–36. Springer, 2016.