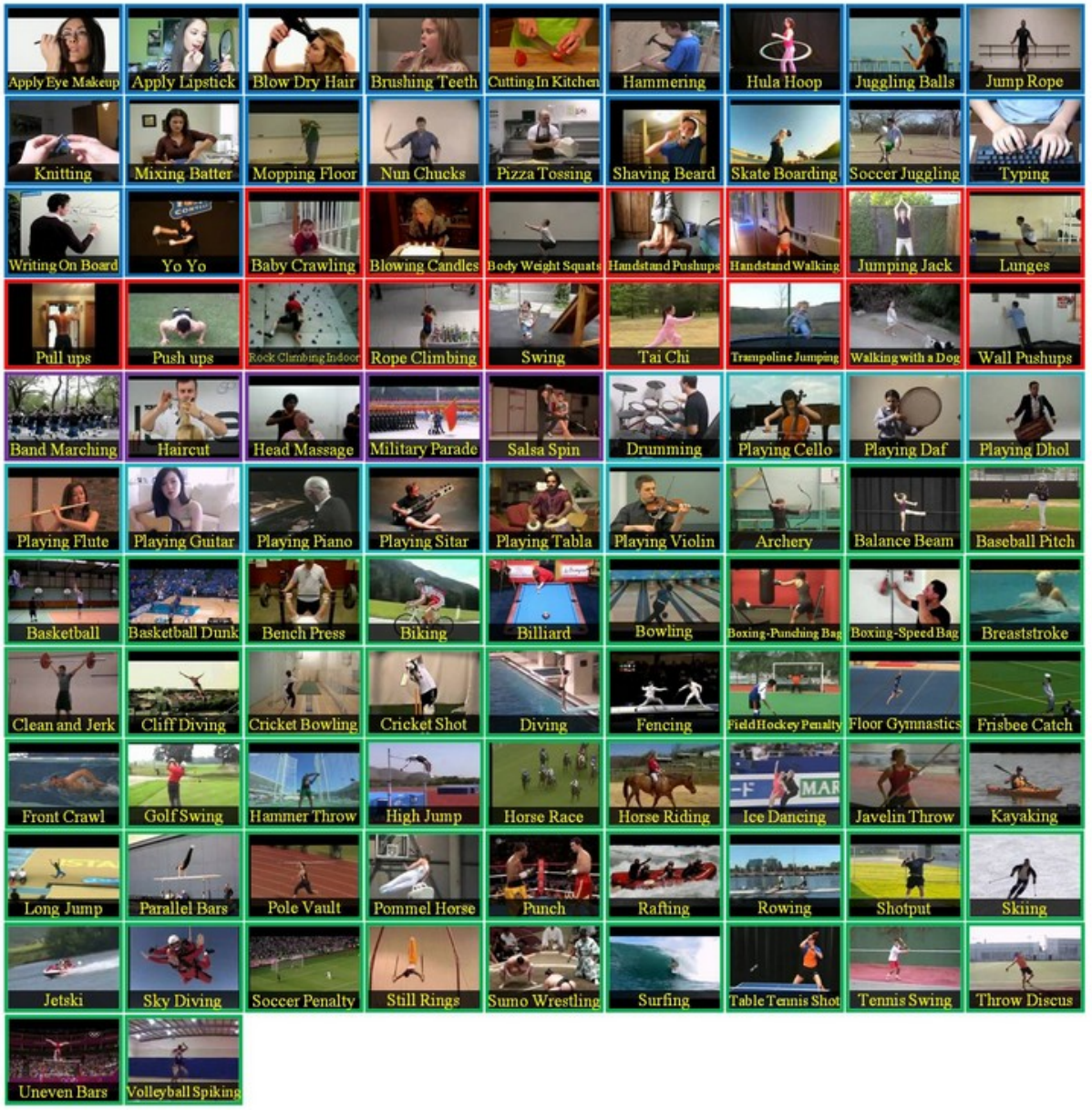
HW3 - Action Recognition In Short Video Clips

- Parts 1 and 2 due Wednesday, Nov 9. Code will be provided. Just need to run it yourself
- Part 3 due Monday, Nov 14
- Will discuss what needs to be turned in
- Will provide a document write up more detailed than this presentation to follow along with the assignment

UCF-101 Action Recognition Dataset



- 13,320 videos, 101 action categories
- 320x240 dimension frames
- ~2-10 seconds average clip duration (30 fps)
- Original 43.90% classification rate (2012) (has since improved greatly)

Assignment Overview

- 1.) Finetune a pre-trained convolutional neural network on single frames from video clips
- 2.) Save feature vector output from last CNN layer for all video clips
- 3.) Train an RNN on sequences of features

- I will provide the majority of the code for parts 1 and 2 which will use Chainer
- I will provide starter code for part 3 in Chainer. This part will be more open ended and can be implemented in Theano (or any other package) if desired.

Large-scale Video Classification with Convolutional Neural Networks (2014)

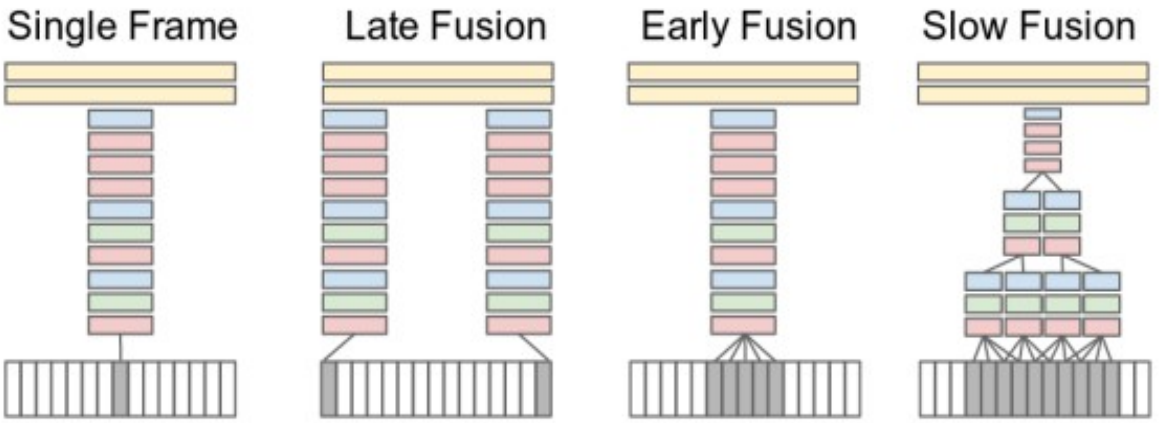


Figure 1: Explored approaches for fusing information over temporal dimension through the network. Red, green and blue boxes indicate convolutional, normalization and pooling layers respectively. In the Slow Fusion model, the depicted columns share parameters.

- Use features from CNN pre-trained on larger dataset (ImageNet)
- Combine inputs for multiple time steps for classification over video clip
- 65.4% accuracy

Beyond Short Snippets: Deep Networks for Video Classification (2015)

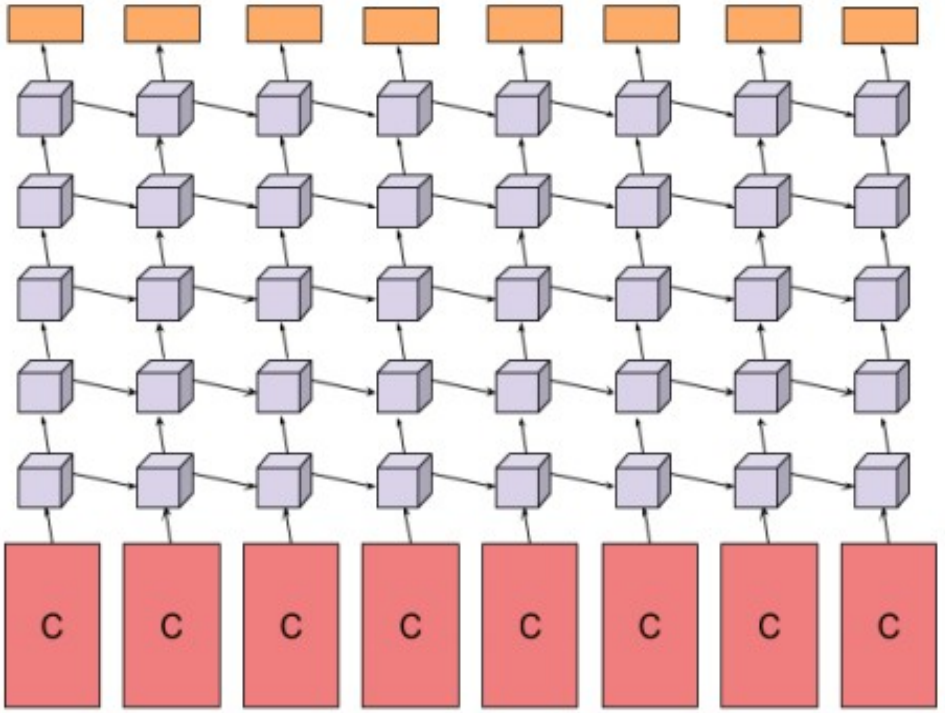


Figure 4: Deep Video LSTM takes input the output from the final CNN layer at each consecutive video frame. CNN outputs are processed forward through time and upwards through five layers of stacked LSTMs. A softmax layer predicts the class at each time step. The parameters of the convolutional networks (pink) and softmax classifier (orange) are shared across time steps.

- Feed CNN output into stacked LSTMs
- Two CNNs (image and optical flow)
- 88.60% accuracy with included optical flow images
- 73.0% single frame accuracy and 82.6% LSTM accuracy (no optical flow)

Part 1 – Finetune pre-trained CNN

- Not exactly enough data to train CNN from scratch, only train top layers
- Model takes up lots of memory (small batch size=slow training), split into two parts (CNN + RNN)
- Dataset size (saving convolutional features vs. fully connected features)
- Data augmentation limitation when saving features before RNN

Part 1 – Finetune pre-trained CNN

- VGGNet.py and VGG.model – Network pre-trained on ImageNet
- VGGNet_1024.py – Modified network to finetune
- frameBasedTrain.py – script for training VGGNet_1024
- helperFunctions.py – functions used by frameBasedTrain.py for loading in dataset
- pip install imageio – for reading .avi, first time running code will automatically download ffmpeg (cv2 video loading module doesn't work on BlueWaters for some reason)

Part 1 – Finetune pre-trained CNN

- frameBasedTrain.py can be run as is, no need to change parameters (weight decay, learning rate, batch size, number of epochs, etc.)
- helperFunctions.py has code commented out for data augmentation (random crop/rotation) but does includes flipping, haven't tested if data augmentation improves results (takes longer to train)
- Should take only a few hours depending on BlueWaters IO speed

Part 2 – Saving features from CNN

- After training, want to save features (dim 1024) for all frames in all videos. Save as .npy
- Save final output (prediction) for single frame evaluation
- saveFeatures_1024.py – Script for saving all features and predictions into new directories
- saveFeatures.sh – Splits dataset into 12 jobs to be submitted to BlueWaters. Walltime=02:30:00
- generate_pbs.py – Used by saveFeatures.sh to automatically generate and run .pbs file for submitting jobs
- No need to modify any code here after completing part 1. Simply run saveFeatures.sh

Part 2 – Saving features from CNN

- save_hdf5.py – creates train1024.hdf5, test1024.hdf5, and testPredictions1024.hdf5. BlueWaters can read data from .hdf5 files much faster than individual .npy files. Greatly speeds up RNN training later
- normalizeData.py – Calculates the mean and standard deviation on the train dataset to normalize the 1024 dimension feature vectors (for both train and test), better for training RNN
- singleFrameAccuracy.py – Calculates accuracy, top 5 accuracy, and top 10 accuracy based on the saved predictions

What to turn in – Parts 1 and 2

- Report test accuracy given from singleFrameAccuracy.py
- Modify singleFrameAccuracy.py and create a confusion matrix giving a breakdown of what classes are frequently misclassified
- This part is due first simply to make sure you do it ahead of time and have enough time to play around with training an RNN for part 3
- Feel free to perform this another time with the data augmentation from helperFunctions.py uncommented, might need to tune parameters (learning rate/weight decay), I'm curious to see the results
- My results: Accuracy(top 1, top 5, top 10): (0.626,0.853,0.915)

Part 3 – Training RNN

- rnn.py – Starter file for training RNN model (1 LSTM layer + 1 Linear/Sigmoid Layer + 1 Output Layer)
- Many changeable parameters – Number of layers, Batch Normalization, Gradient Clipping, Dropout, Learning Rate, Weight Decay, Length of Sequence, Data Augmentation, Batch Size, Truncated Back Propagation, Loss Scaling, Inference Technique

Things to think about

- Length of sequence and truncated back propagation – I chose 120 frames (4 seconds) with 30 frames for back propagation. How does this effect what the network is trying to “learn”?
- What to do if video is smaller/larger than 120 frames? Too small - repeat sequence in reverse direction to maintain smooth motion until it's long enough. Too big – select random segment of length 120. Could also simply train on variable length sequences (but how does this effect what the network can/should be “learning”?

Things to think about

- Inference Technique – Test videos on full length sequence or partial sequence? Which partial sequence(s) to choose? Average output over multiple sequences?
- Early in the sequence, the LSTM hidden layers are initialized poorly => poor output. Scale the prediction to bias towards end of sequence
- Bias the loss function to end of the sequence, can improve training

Things to think about

- Features are trained on single images, how much do they change through the video? Is there enough change for the RNN to “learn” how to discriminate between similar classes? (This is why optical flow greatly improves results)
- Loss of spatial relationships in fully connected layers of CNN. Limitation of 1024 feature vector for RNN

Things to think about

- Overtraining
  - 1 Layer: ~100% train accuracy, test=(0.671,0.886,0.939)
  - 2 Layer: ~100% train accuracy, test=(0.648,0.865,0.928)
  - 3 Layer: ~100% train accuracy, test=(0.620,0.857,0.916)
- Use dropout and other regularization techniques to prevent this problem. Deeper LSTM networks can perform equally as well if not better
- Use batch loss as opposed to train accuracy as an indicator of when to stop training. Train accuracy can be misleading. Too large of learning rate can lead to high train accuracy without decreasing loss (will explain in more detail)

What to turn in? - Part 3

- Train a variety of models and report training loss/accuracy and test accuracy. Also report method for calculating test accuracy (inference strategy, test multiple sequences from same video clip and average prediction, etc.)
- Should be able to get roughly 5% increase on test set versus single frame accuracy
- Check if CNN trained on augmented data performs better (not required)
- Confusion matrix giving a breakdown of what classes are frequently misclassified
- Find 5-10 classes with largest improvement from single frame method to RNN method. Which classes improved the most and is there a reason why? Which classes didn't improve at all or decreased in performance?
- *This part is a lot more open ended than parts 1 and 2. I will not grade harshly as long as you make an attempt at trying multiple things and you write down notes/observations/results*
- *There are tons of ways to approach this problem. I chose a method that I think has a nice balance between computational time, what you can learn about CNNs/RNNs, and ease of implementation. Feel free to try anything else or make suggestions on what you think can improve the results or make this a better assignment for you as a student*
- *Please post any questions/concerns on Piazza. I ran into multiple problems while creating this assignment (primarily with IO speed on BlueWaters)*