Accelerating deep neural network training for action recognition on a cluster of GPUs

Guojing Cong[1], **Giacomo Domeniconi**[1], Joshua Shapiro[1], Fan Zhou[2],Barry Chen[3]

[1]IBM TJ Watson Research Center, NY

[2]Georgia Tech Atlanta, GA

[3]National Laboratory Livermore, CA

Outline

- Introduction of Action Recognition in Videos
- Two-Streams CNNs for Action Recognition
- Adaptive-batchsize model averaging with sparse communication
    - The AAVG algorithm
    - Dynamically adapting batchsizes
    - Customizing the optimizer for AAVG
    - Transfer learning for training the flow stream
- Conclusions

Action Recognition in Videos

Action Recognition in Videos

- Input: video
- Output: the action label
- Why perform action recognition?
  - Surveillance footage
  - Automatic video organization / tagging
  - Search-by-video
  - ….
- What is an action?
  - **Action**: walking, pointing, putting, etc.
  - **Activity**: talking on the phone, drinking tea, etc.
  - **Event**: a soccer game, a birthday party, etc.

Level of semantics

Crowd Behavior/Event Analysis

credit: Bingbing Ni

Why Action Recognition is challenging

- Different scales
  - People may appear at different scales in different videos, yet perform the same action
- Movement of the camera
  - The camera may be a handheld camera, and the person holding it can cause it to shake.
- Occlusions
  - Action may not be fully visible
- Action variation
  - Different people perform different actions in different ways
- Action Recognition task is both very computing and memory intensive
  - The required neural networks to accomplish it are huge
    - Up to 100 or 200 layers
    - Up to 10 or 100M parameters
  - The size of the datasets and features is big
    - Up to 100 TB

CNN for Action Recognition

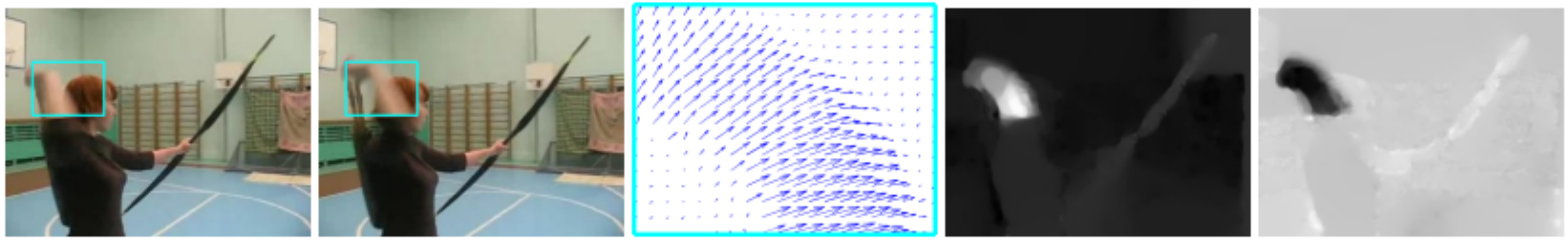
Two-Streams CNNs for Action Recognition

- Extend deep Convolution Networks to action recognition in video
- The current state-of-art approaches make use of **two streams of CNNs**
  - Proposed by Simonyan and Zisserman[1] in 2014
  - Inspired by the human visual cortex
  - Using 2D convolutions (images), or
  - Using 3D convolutions (streams of images, the third dimension is the time)
- Two separated recognition streams:
  - Spatial (or RGB) stream– image recognition CNN
  - Temporal (or flow) stream – motion recognition CNN
    - Based on Optical Flow

[1]Simonyan K and Zisserman A. *Two-stream convolutional networks for action recognition in videos*. CoRR, 2014

Optical flow

- Optical flow refers to the visible motion of an object in an image, and the **apparent 'flow' of pixels** in an image
- It is the result of 3D motion being projected on a 2D image plane
- The optical flow can be used as an estimation of object velocity and position of object in the next frame
- We used the OpenCV’s TV-L1 estimation

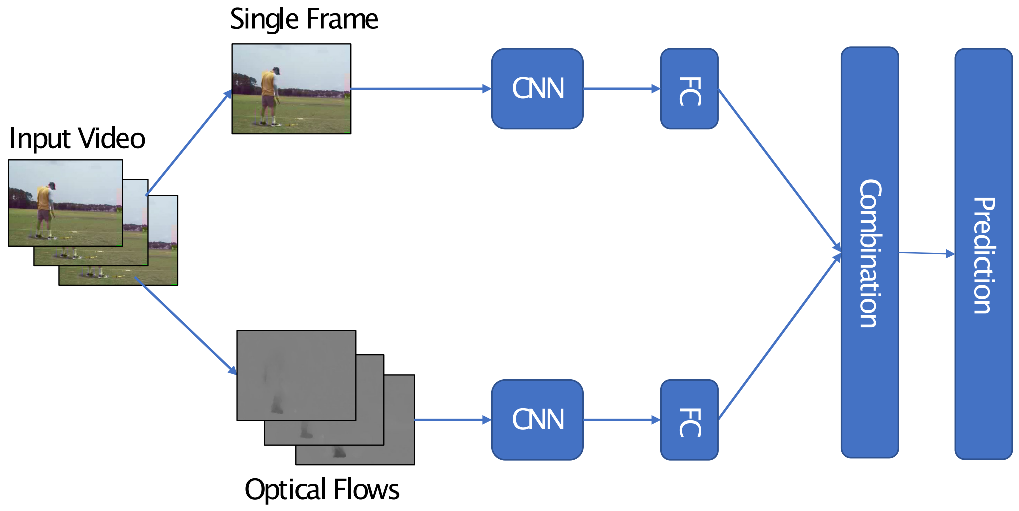


(a) (b) (c) (d) (e)

(a),(b): a pair of consecutive video frames with the area around a moving hand.
(c): a close-up of dense optical flow in the outlined area;
(d): horizontal component (higher intensity corresponds to positive values, lower intensity to negative values).
(e): vertical component.

Our two-streams 2 dimensional CNNs

- We follow the implementation of Simonyan and Zisserman[1]
- RGB stream: 1 random frame from an input video is sampled and fed into a CNN
- Flow stream: 10 consecutive flows are randomly sampled and fed to another CNN
- Both CNNs in are based on **ResNet152**
  - The weights in ResNet152 are pretrained on the **ImageNet** dataset
- Simple averaging is used to combine the predictions from the two streams
- The two streams are trained separately



Benchmark datasets and baseline results

- Our primary dataset is UCF-101
  - ~13,000 video in 101 action categories
  - ~9500 training and ~3700 validation videos
- Baseline single-GPU validation accuracy:
  - RGB stream = **85.04%**
  - Flow stream = **84.5%**
  - Two-streams combined = **91.3%**
- Training time on a single GPU
  - RGB stream takes around 12 hours
  - Flow stream takes more than two days
- Experiments performed on 4 IBM Minsky nodes
  - Each node has 2 Power8 CPUs with 10 cores each and 4 NVIDIA Tesla P100 GPUs
  - The interconnect between the nodes is Infiniband
- We also show results on the HMDB-51 dataset
  - ~6,800 video in 51 action categories

Adaptive-batchsize model averaging with sparse communication (AAVG)

AAVG

Algorithm 1 AAVG ($\mathcal{T}$, $\mathcal{V}$, K, B, m, γ, P, N, b_1, b_2)

initialize $\widetilde{\mathrm{w}}_1$
$a^* \leftarrow 0$, $B_0 \leftarrow B$, $\gamma_0 \leftarrow \gamma$
for $n = 1, ..., N$ **do**
  $B_n \leftarrow B_{n-1}$, $\gamma_n \leftarrow \gamma_{n-1}$
  for $j = 1, \dots, P$ in parallel **do**
    set $\mathrm{w}_n^j = \widetilde{\mathrm{w}}_n$
    for $k = 1, ..., K$ **do**
      randomly sample a mini-batch of size B_n from $\mathcal{T}$
      $\mathrm{w}_{n+k}^j \leftarrow \mathrm{w}_{n+k-1}^j - \frac{\gamma_n}{B_n} \sum_{s=1}^{B_n} \nabla F(\mathrm{w}_{n+k-1}^j; \xi_{k,s}^j)$
    end for
  end for
  $\widetilde{\mathrm{w}}_{n+1} \leftarrow \frac{1}{P} \sum_{j=1}^{P} \mathrm{w}_{n+K}^j$
  if $n\%m = 0$ **then**
    $a \leftarrow evaluate(\widetilde{\mathrm{w}}_{n+1}, \mathcal{V})$
    if $a < a^* \cdot b_1$ **then**
      $B_n \leftarrow B_n \cdot b_2$
    end if
    if $a > a^*$ **then**
      $a^* \leftarrow a$
    end if
  end if
end for

- **Adaptive-batchsize model averaging with sparse communication** (AAVG)
- Input parameters:
  - the training dataset $\mathcal{T}$
  - the validation dataset $\mathcal{V}$
  - the averaging interval K
  - the initial batchsize B
  - the validation interval m
  - the initial learning rate γ
  - the number of learners P
  - the number of training steps N
  - parameters b_1 and b_2 that are used to adapt batchsize

AAVG

Algorithm 1 AAVG ($\mathcal{T}$, $\mathcal{V}$, K, B, m, γ, P, N, b_1, b_2)

1: initialize $\widetilde{\mathrm{w}}_1$
2: $a^* \leftarrow 0$, $B_0 \leftarrow B$, $\gamma_0 \leftarrow \gamma$
3: **for** $n = 1, ..., N$ **do**
4:     $B_n \leftarrow B_{n-1}$, $\gamma_n \leftarrow \gamma_{n-1}$
5:     **for** $j = 1, \ldots, P$ in parallel **do**
6:         set $\mathrm{w}_n^j = \widetilde{\mathrm{w}}_n$
7:         **for** $k = 1, ..., K$ **do**
8:             randomly sample a mini-batch of size B_n from $\mathcal{T}$
9:             $\mathrm{w}_{n+k}^j \leftarrow \mathrm{w}_{n+k-1}^j - \frac{\gamma_n}{B_n} \sum_{s=1}^{B_n} \nabla F(\mathrm{w}_{n+k-1}^j; \xi_{k,s}^j)$
10:         **end for**
11:     **end for**
12:     $\widetilde{\mathrm{w}}_{n+1} \leftarrow \frac{1}{P} \sum_{j=1}^{P} \mathrm{w}_{n+K}^j$
13:     **if** $n \% m = 0$ **then**
14:         $a \leftarrow evaluate(\widetilde{\mathrm{w}}_{n+1}, \mathcal{V})$
15:         **if** $a < a^* \cdot b_1$ **then**
16:             $B_n \leftarrow B_n \cdot b_2$
17:         **end if**
18:         **if** $a > a^*$ **then**
19:             $a^* \leftarrow a$
20:         **end if**
21:     **end if**
22: **end for**

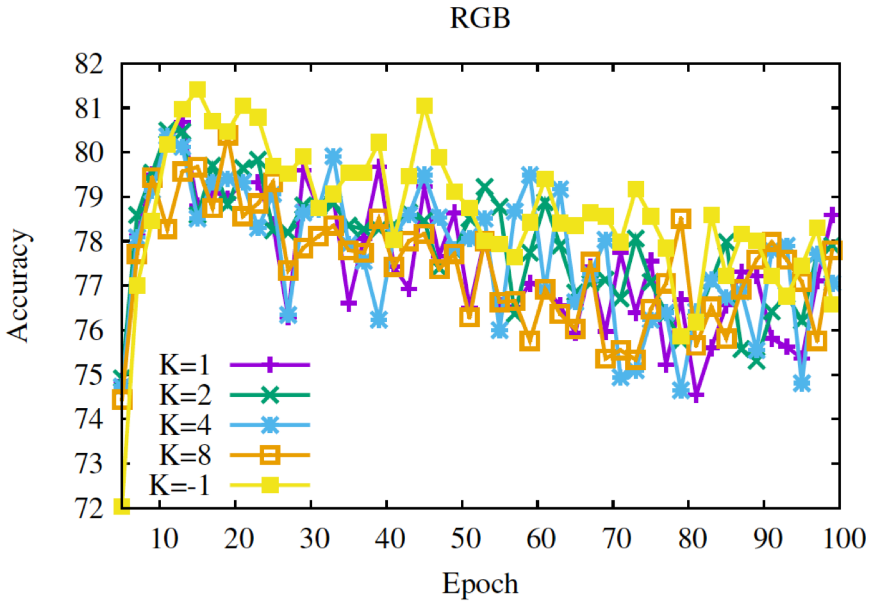
- P learners run stochastic gradient descent concurrently (lines 4 to 11) and average their parameters every K steps (line 12)
- When K=1, AAVG with constant batchsize is equivalent to hard-sync parallelization of SGD
  - Its convergence behavior is exactly the same as SGD with batchsize = PB
- Every m steps, the algorithm evaluates validation accuracy a on the validation dataset $\mathcal{V}$ and adapts batchsize according to the validation result (lines 13 to 21)

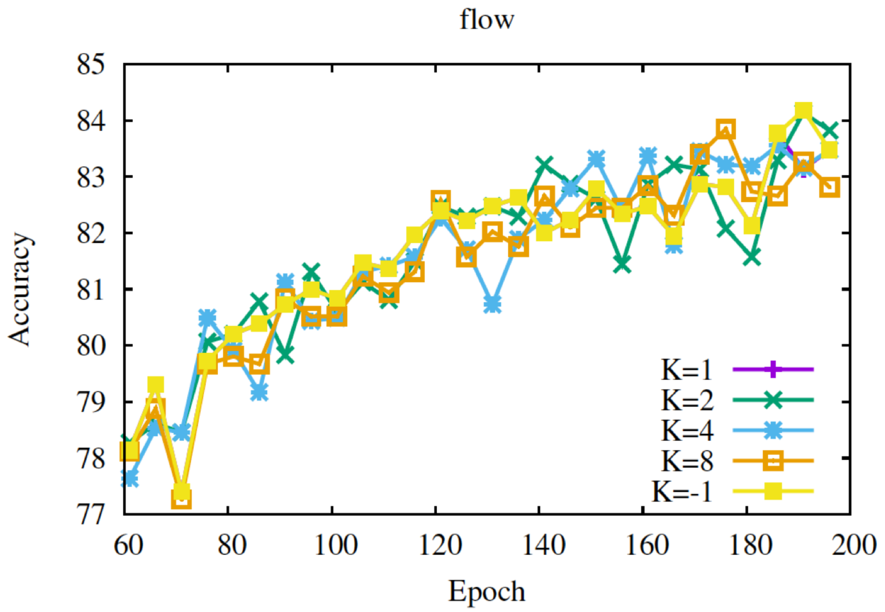
What is the best K?

- <u>The right *K* plays a critical role</u>
  - Too small *K* incurs high overhead due to frequent communication
  - Too high *K* incurs in slow or even non-convergence
- Intuitively, frequent averaging reduces the variance more frequently, thus one may think the smaller K the better.
- Surprisingly, it is not!

Results varying *K*

- We experiment with different *K* values and observe their impact on the validation accuracy





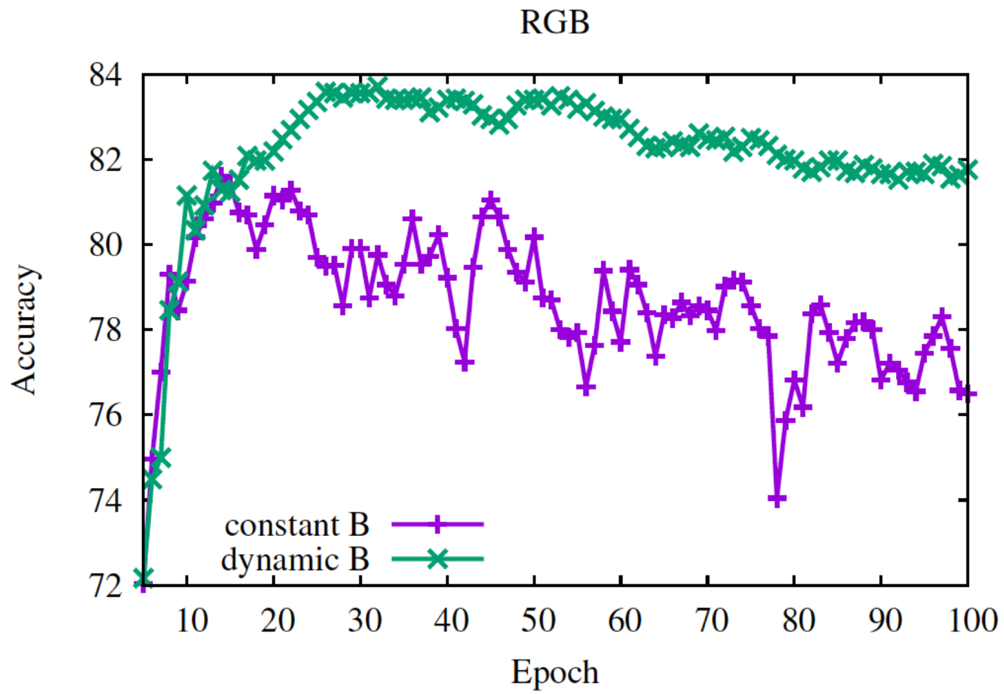
- High K provides good results and prevents communication overhead
- AAVG with K=-1 achieves near linear speedup
  - Ignoring I/O overhead (determined by the storage type and file system)

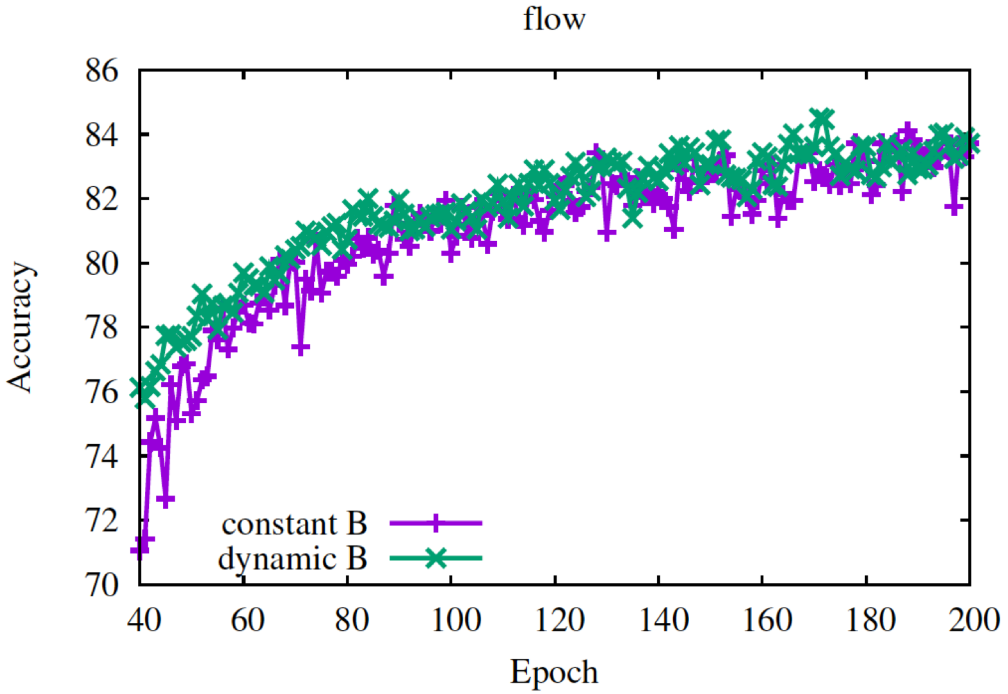
Dynamically Adapting Batchsizes

- The *adam* optimizer adaptively scales the learning rate for each individual gradient component
- Why not to consider the batchsize?
- Idea:
  → **higher B_n for higher n**
- Gradient estimates from small batches are sufficient at the beginning for rapid progress
- SGD with increasing batchsize should eventually start to resemble deterministic algorithms for strongly convex problems via larger B_n batch sampling

Dynamically Adapting Batchsizes

- AAVG increases the batchsize by a factor of b_2 whenever the validation accuracy does not improve by a margin of b_1
- In our implementation, we simply use $b_1=1$ and $b_2=2$
- We keep the maximum batchsize to 576



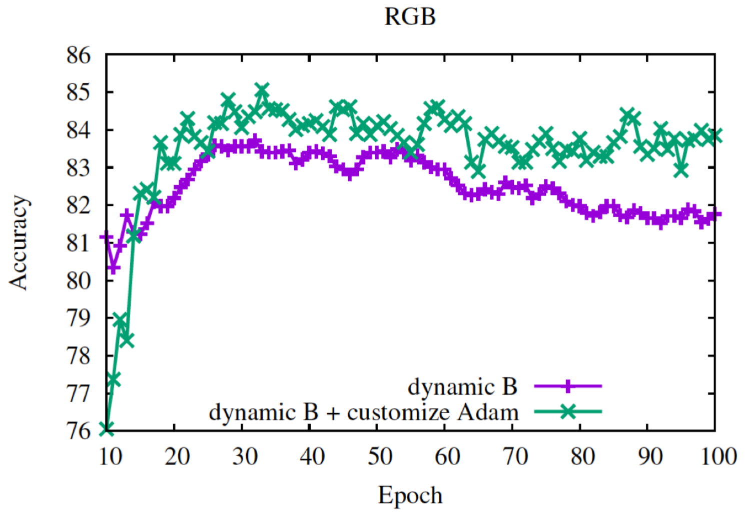


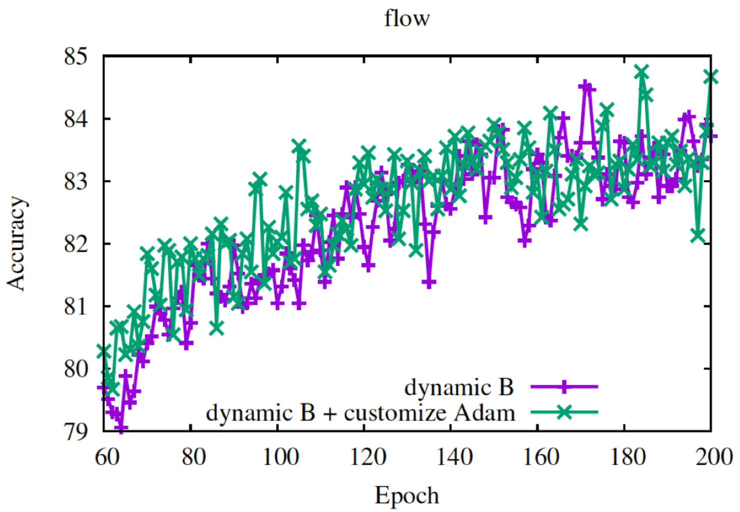
Customizing the optimizer for AAVG

- *Adam* optimizer uses two quantities to adapt the learning rate:
  - *m*, the weighted average of historical gradients
  - *v*, the weighted average of the historical squared gradients.
- Model averaging disrupts *adam*'s internal state
- We adjust *m* and *v* in the *adam* optimizer for AAVG.

Algorithm 2 Adjust-optimizer($\{m_j\}$, $\{v_j\}$, $\{m_j^b\}$, $\{v_j^b\}$, P)

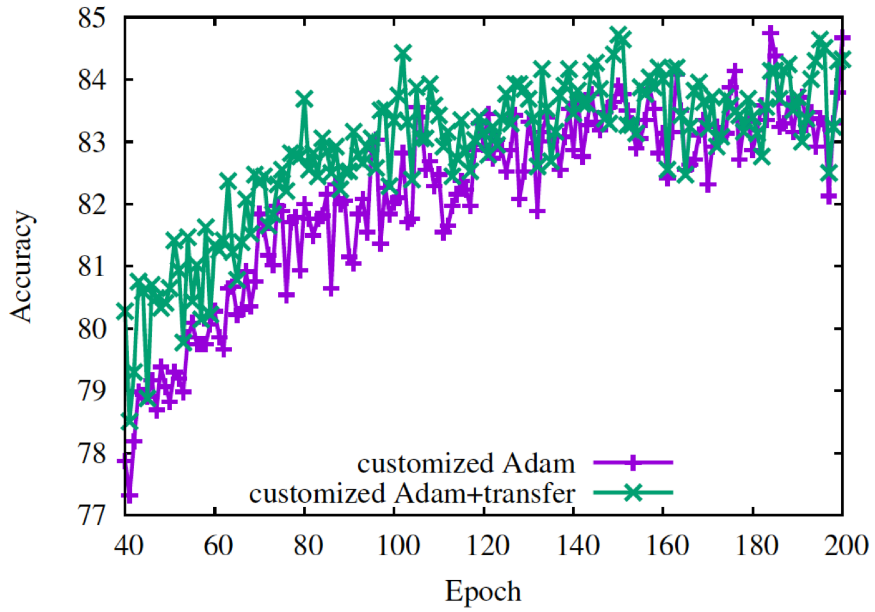
1: $z_j \leftarrow v_j^b + (m_j^b)^2$
2: set $m = \sum_j^P m_j$, $v = \sum_j^P v_j$
3: set $m^b = \sum_j^P m_j^b$, set $z = \sum_j^P z_j$
4: **for** $j = 1, \ldots, P$ in parallel **do**
5: $\quad m_j \leftarrow m/P$, $v_j \leftarrow v/P$
6: $\quad m_j^b \leftarrow m^b/P$, $v_j^b \leftarrow z/P - (m_j^b)^2$
7: **end for**





Transfer Learning in the Flow Stream

- Training the flow stream is significantly slower due to:
  - The pretrained ResNet model was trained with RGB images not flow inputs
  - The input layer has significantly more channels and thus more weights to train
- Proposal: Instead start the training with weights pre-trained on ImageNet, use the model trained for the RGB stream



- Similar highest validation accuracies
- Achieved faster with transfer learning (epoch 150 vs 185)

Results Recap

	UCF101			HMDB51		
	RGB	Flow	2stream	RGB	Flow	2stream
$AAVG_1$	81.6	84.1	88.3	-	-	-
$AAVG_2$	83.7	84.5	89.7	-	-	-
$AAVG_3$	85.6	84.7	91.4	61.4	55.9	67.0
$AAVG_4$	85.6	84.7	93.04	61.4	56.0	67.9

- AAVG_1 implements AAVG with constant batchsize
- AAVG_2 implements AAVG with adaptive batchsize
- AAVG_3 implements AAVG with both adaptive batchsize and tuned *adam* optimizer
- AAVG_4 is AAVG_3 with weights transferring from the RGB stream to the flow stream
- The RGB stream training on UCF101 with 16 GPUs takes 61 minutes using AAVG with customized *adam*, while the base-line single-GPU SGD implementation takes 2067 minutes to train to achieve similar validation accuracy

Conclusions

- AAVG is an efficient distributed training algorithm with adaptive batchsize that explicitly manages the impact of model averaging frequency on both convergence and communication overhead
- AAVG with very sparse synchronization (i.e. once per epoch), shows very good convergence behavior
  - As a happy coincidence, the communication overhead is very low
- AAVG shows up to super-linear speedups on 16 GPUs over the base-line single-GPU SGD implementation, while improving accuracy

- In our future work, we plan to evaluate our algorithm on larger datasets and on 3-Dimensional CNNs

Thank you!