INTRODUCTION TO RIVANNA
Rivanna in More Detail
Allocations

• Rivanna is allocated:
  At the most basic level, an allocation refers to a chunk of CPU time that you receive and can use to run your computation.

• Allocations are measured in service units (SUs), where
  \[1 \text{ SU} = 1 \text{ core-hour}\]

• All accounts on a given allocation share the service units.
CONNECTING & LOGGING ONTO RIVANNA
How to connect to Rivanna

• There are three ways to connect to Rivanna:

1. ssh client
   • Instructions for installing and using an ssh client are provided in the appendix of these slides.

2. FastX
   • Using your web browser, go to URL https://rivanna-desktop.hpc.virginia.edu and log in.
   • Click on “Launch Session”; Select “MATE” and click on “Launch”

3. Open-on-Demand -- Coming Soon!
   • Using your web browser, go to URL https://rivanna-portal.hpc.virginia.edu
   • You will need to “Netbadge” in.

Regardless of how you connect, you must use the UVa Anywhere VPN when off-grounds.

See http://its.virginia.edu/vpn/ for details.
We will use FastX today:

- In your web browser, go to URL:
  
  https://rivanna-desktop.hpc.virginia.edu
Starting up FastX

• Click “Launch Session”; Select MATE; Click Launch
FastX Environment

- A desktop for working on Rivanna
CLUSTER ENVIRONMENT
After you have logged in . . .

- You will be in your home directory.

- How you navigate will depend on how you connected to Rivanna.
  - `ssh` client:
    - A terminal window will appear. To navigate within your directory, you will need to use Unix/Linux commands.
    - See [https://arcs.virginia.edu/UNIX-tutorials-for-beginners](https://arcs.virginia.edu/UNIX-tutorials-for-beginners) to learn more about Unix/Linux commands.

- FastX:
  - A desktop environment will appear. You can use your mouse to navigate or open a terminal window to use Unix/Linux commands or start interactive applications.

- Open-on-Demand:
  - A dashboard will appear. You can click on the menu items across the top to access different tools, like a file manager, a job composer, or interactive applications.
Your Home Directory

• The default home directory on Rivanna has 50GB of storage capacity

  • This directory is distinct from the 4GB home directory provided by ITS.

  • The ITS home directory is available as /tiny/$USER
Checking your Home Storage

• To see how much disk space you have used in your home directory, open a Terminal window and type `hdquota` at the command-line prompt:

```
$ hdquota

Filesystem  |  Used  |  Avail  |  Limit  |  Percent Used
qhome       |  39G   |  12G    |  51G    |  77%
```
Checking your Allocation

• To see how many SUs you have available for running jobs, type `allocations` at the command-line prompt:

```
$ allocations

Allocations available to Misty S. Theatre(mst3k):

* robot_build: less than 6,917 service-units remaining.
* gizmonic-testing: less than 5,000 service-units remaining.
* servo: less than 59,759 service-units remaining, allocation will expire on 2017-01-01.
* crow-lab: less than 2,978 service-units remaining.
* gypsy: no service-units remaining
```
Your /scratch Directory

- Each user will have access to 10 TB of temporary storage.
  - It is located in a subdirectory under /scratch, and named with your userID
  - e.g., /scratch/mst3k
  - You are limited to 350,000 files in your scratch directory.

---

**Important:**
/scratch is **NOT permanent** storage and files older than **90 days** will be marked for deletion.
Running Jobs from Scratch

• We recommend that you run your jobs out of your /scratch directory for two reasons:
  • /scratch is on a Lustre filesystem (a storage system designed specifically for parallel access).
  • /scratch is connected to the compute nodes with Infiniband (a very fast network connection).

We also recommend that

• You keep copies of your programs and data in more permanent locations (e.g., your home directory or leased storage).
• After your jobs finish, you copy the results to more permanent storage).
Checking your /scratch Storage

• To see the amount of scratch space that is available to you, type `sfsq` at the command line prompt.

```
$ sfsq

'scratch' usage status for 'mst3k', last updated: 2016-09-08 16:26:12

- ~28/10,000 GBs allocated disk space
- 153/350,000 files created
- 151/153 files marked for deletion due to age limits

To view a list of all files marked for deletion, please run 'sfsq -l'
```
Moving data onto Rivanna

- You have several options for transferring data onto your home or /scratch directories.
  1. Use the `scp` command in a terminal window.
  2. Use a drag-and-drop option with MobaXterm (Windows) or Fugu (Mac OS).
  3. Use the web browser in the FastX desktop to download data from UVA Box.
  4. Set up a Globus endpoint on your laptop and use the Globus web interface to transfer files.

(See [https://arcs.virginia.edu/globus](https://arcs.virginia.edu/globus) for details)
MODULES
Modules

• Any application software that you want to use will need to be loaded with the **module load** command.

• For example:
  - module load matlab
  - module load anaconda/5.2.0-py3.6
  - module load gcc R/3.5.1

• You will need to load the module any time that you create a new shell
  - Every time that you log out and back in
  - Every time that you run a batch job on a compute node
Module Details

- **module avail** – Lists all available modules and versions.
- **module spider** – Shows all available modules
- **module key keyword** – Shows modules with the keyword in the description
- **module list** – Lists modules loaded in your environment.
- **module load mymod** – Loads the default module to set up the environment for some software.
  - **module load mymod/N.M** – Loads a specific version N.M of software mymod.
  - **module load compiler mpi mymod** – For compiler- and MPI- specific modules, loads the modules in the appropriate order and, optionally, the version.
- **module purge** – Clears all modules.
Learning more about a Module

• To locate a python module, try the following:

$ module avail python

$ module spider python

$ module key python

• To find bioinformatics software packages, try this:

$ module key bio

• The available software is also listed on our website:

https://arcs.virginia.edu/software-list
PARTITIONS (QUEUES)
Partitions (Queues)

- Rivanna has several partitions (or queues) for job submissions.
  - You will need to specify a partition when you submit a job.
  - To see the partitions that are available to you, type `queues` at the command-line prompt.

<table>
<thead>
<tr>
<th>Queue (partition)</th>
<th>Availability (idle%)</th>
<th>Time Limit</th>
<th>Queue Limit</th>
<th>Maximum Cores/Job</th>
<th>Maximum Mem/Core</th>
<th>Idle Nodes</th>
<th>SU Rate</th>
<th>Usable Accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard</td>
<td>43 13(72.2%)</td>
<td>7-days</td>
<td>none</td>
<td>20</td>
<td>64-GB</td>
<td>195</td>
<td>1.00</td>
<td>robot-build, gypsy</td>
</tr>
<tr>
<td>dev</td>
<td>1833(65.2%)</td>
<td>1 hours</td>
<td>none</td>
<td>4</td>
<td>254GB</td>
<td>59</td>
<td>0.00</td>
<td>robot-build, gypsy</td>
</tr>
<tr>
<td>parallel</td>
<td>3528(73.5%)</td>
<td>3-days</td>
<td>none</td>
<td>240</td>
<td>64-GB</td>
<td>176</td>
<td>1.00</td>
<td>robot-build, gypsy</td>
</tr>
<tr>
<td>largemem</td>
<td>48(60.0%)</td>
<td>7-days</td>
<td>none</td>
<td>16</td>
<td>500-GB</td>
<td>3</td>
<td>1.00</td>
<td>robot-build, gypsy</td>
</tr>
<tr>
<td>gpu</td>
<td>334(85.0%)</td>
<td>3-days</td>
<td>none</td>
<td>8</td>
<td>128-GB</td>
<td>10</td>
<td>1.00</td>
<td>robot-build, gypsy</td>
</tr>
<tr>
<td>knl</td>
<td>2048(100.0%)</td>
<td>3-days</td>
<td>none</td>
<td>2048</td>
<td>1-GB</td>
<td>8</td>
<td>1.00</td>
<td>robot-build, gypsy</td>
</tr>
</tbody>
</table>
Compute Node Partitions (aka Queues)

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Purpose</th>
<th>Job Time Limit</th>
<th>Memory / Node</th>
<th>Cores / Node</th>
<th># of Available Nodes</th>
<th>SU / Core Hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard</td>
<td>For jobs on a single compute node</td>
<td>7 days</td>
<td>128 GB 256 GB</td>
<td>20 28</td>
<td>265 (20-core nodes shared w/ parallel queue)</td>
<td>1.0</td>
</tr>
<tr>
<td>gpu</td>
<td>For jobs that can use general purpose graphical processing units (GPGPUs) (K80 or P100)</td>
<td>3 days</td>
<td>256 GB</td>
<td>28</td>
<td>14 (max 4 nodes per job)</td>
<td>1.0</td>
</tr>
<tr>
<td>parallel</td>
<td>For large parallel jobs on up to 120 nodes (&lt;= 2400 CPU cores)</td>
<td>3 days</td>
<td>128 GB 256 GB</td>
<td>20</td>
<td>240 (shared w/ standard queue)</td>
<td>1.0</td>
</tr>
<tr>
<td>largemem</td>
<td>For memory intensive jobs (&lt;= 16 cores/node)</td>
<td>7 days</td>
<td>1 TB</td>
<td>16</td>
<td>5 (max 2 per user)</td>
<td>1.0</td>
</tr>
<tr>
<td>dev</td>
<td>To run jobs that are quick tests of code</td>
<td>1 hour</td>
<td>128 GB</td>
<td>4</td>
<td>2</td>
<td>0.0</td>
</tr>
</tbody>
</table>
SLURM SCRIPTS
SLURM

• SLURM is the Simple Linux Utility for Resource Management.
  • It manages the hardware resources on the cluster (e.g. compute nodes/cpu cores, compute memory, etc.).

• SLURM allows you to request resources within the cluster to run your code.
  • It is used for submitting jobs to compute nodes from an access point (generally called a frontend).
  • Frontends are intended for editing, compiling, and very short test runs.
  • Production jobs go to the compute nodes through the resources manager.

• SLURM documentation:
  https://arcs.virginia.edu/slurm
  http://slurm.schedmd.com/documentation.html
Basic SLURM Script

- A SLURM script is a bash script with
  - SLURM directives (#SBATCH) and
  - command-line instructions for running your program.

```bash
#!/bin/bash
#SBATCH --nodes=1  # total number of nodes for the job
#SBATCH --ntasks=1 # how many copies of code to run
#SBATCH --cpus-per-task=1 # number of cores to use
#SBATCH --time=1-12:00:00 # amount of time for the whole job
#SBATCH --partition=standard # the queue/partition to run on
#SBATCH --account=myGroupName # the account/allocation to use

module purge
module load anaconda # load modules that my job needs
python hello.py  # command-line execution of my job
```
Basic SLURM Job
(Shorthand notation)

• Most of the SLURM directives have a short hand notation for the options

```bash
#!/bin/bash
#SBATCH -N 1  # total number of nodes for the job
#SBATCH -n 1  # how many copies of code to run
#SBATCH -c 1  # number of cores to use
#SBATCH -t 12:00:00  # amount of time for the whole job
#SBATCH -p standard  # the queue/partition to run on
#SBATCH -A myGroupName  # the account/allocation to use

module purge
module load anaconda
python hello.py  # command-line execution of my job
```
Submitting a SLURM Job

• To submit the SLURM command file to the queue, use the `sbatch` command at the command line prompt.

• For example, if the script on the previous slide is in a file named `job_script.slurm`, we can submit it as follows:

```
-bash-4.1$ sbatch job_script.slurm
Submitted batch job 18316
```
Checking Job Status

• To display the status of only your **active** jobs, type:

```
squeue -u <your_user_id>
```

```
-bash-4.1$ squeue -u mst3k

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>18316</td>
<td>standard</td>
<td>job_sci</td>
<td>mst3k</td>
<td>R</td>
<td>1:45</td>
<td>1</td>
<td>udc-aw38-34-1</td>
</tr>
</tbody>
</table>
```

• The `squeue` command will show pending jobs and running jobs, but not failed, canceled or completed job.
Checking Job Status

• To display the status of all jobs, type:

```
sacct -S <start_date>
```

```
bash-4.1$ sacct -S 2019-01-29

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Job Name</th>
<th>Command</th>
<th>Status</th>
<th>Return Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>3104009</td>
<td>RAxML_NoC+</td>
<td>standard</td>
<td>hpc_build</td>
<td>20</td>
</tr>
<tr>
<td>3104009</td>
<td>batch</td>
<td></td>
<td>hpc_build</td>
<td>20</td>
</tr>
<tr>
<td>3108537</td>
<td>raxmlHPC+</td>
<td></td>
<td>hpc_build</td>
<td>1</td>
</tr>
<tr>
<td>3108537</td>
<td>sys/dashb+</td>
<td>gpu</td>
<td>hpc_build</td>
<td>1</td>
</tr>
<tr>
<td>3108562</td>
<td>sys/dashb+</td>
<td>gpu</td>
<td>hpc_build</td>
<td>1</td>
</tr>
<tr>
<td>3109392</td>
<td>sys/dashb+</td>
<td>gpu</td>
<td>hpc_build</td>
<td>1</td>
</tr>
<tr>
<td>3109392</td>
<td>batch</td>
<td></td>
<td>hpc_build</td>
<td>1</td>
</tr>
<tr>
<td>31112064</td>
<td>srun</td>
<td>gpu</td>
<td>hpc_build</td>
<td>1</td>
</tr>
<tr>
<td>31112064</td>
<td>batch</td>
<td></td>
<td>hpc_build</td>
<td>1</td>
</tr>
</tbody>
</table>

• The `sacct` command lists all jobs (pending, running, completed, canceled, failed, etc.) since the specified date.
Deleting a Job

• To delete a job from the queue, use the `scancel` command with the job ID number at the command line prompt:

  ```bash
  -bash-4.1$ scancel 18316
  ```
EXAMPLES
To follow along . . .

• Go ahead and log into Rivanna.
• If using FastX, open up a terminal window.

• First, we will copy a set of examples into your account. At the command line, type:

```
cd
scp -r /share/resources/source_code/CS6501_examples/ .
```
Hello World Job

- To see that the directory is there, type:
  ```
  ls
  ```

- Move to the first folder (i.e., 01_serial) by typing:
  ```
  cd CS6501_examples/01_simple_SLURM
  ls
  ```

- You will see 2 files: hello.py and hello.slurm

- To view the contents of files, type more followed by the filename:
  ```
  more hello.slurm
  ```
Simple SLURM Job

- If your program performs lots of computation, but uses only one processor, you should use the standard queue.

```bash
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=00:05:00
#SBATCH --partition=standard
#SBATCH --account=your_allocation  #Edit to class-cs6501-004-sp19

module purge
module load anaconda
python hello.py
```
Simple Job

• Your results will be placed in a file with the name `slurm_12345678.out`, where 12345678 is replaced with the job ID number from your job submission.

  • Type `ls` to see if the output file exists in your directory.

  • You can look at the results by typing `more` following by the filename. For example:

    `more slurm_12345678.out`
PyTorch Job

• PyTorch is an open source Python package to create deep learning networks.

• The latest PyTorch versions are provided as prebuilt Singularity containers (called tensorflow) on Rivanna.

• All of the tensorflow container images provided on Rivanna require access to a GPU node.
PyTorch Container

• Before you run PyTorch, you will need to move a copy of the tensorflow container into your /scratch directory.

• This step only needs to be done once.

module load singularity
module load tensorflow/1.12.0-py36
cp $CONTAINERDIR/tensorflow-1.12.0-py36.simg /scratch/$USER
Using GPUs

- Certain applications can utilize for general purpose graphics processing units (GPGPUs) to accelerate computations.

- GPGPUs on Rivanna:
  - K80: dual GPUs per board, can do double precision
  - P100: single GPUs per board, double precision is software (slow)

- You must first request the gpu queue. Then with the gres option, type the architecture (if you care) and the number of GPUs.
  
  #SBATCH -p gpu
  #SBATCH --gres=gpu:k80:2
Caution: Limited # of GPUs

• There are only a handful of GPUs on Rivanna:
  • 10 K80s with 4 GPUs each
  • 4 P100s with 4 GPUs each

• You can check the status of the GPUs in two ways:
  • Type `queues` to see the percentage idle
  • Type `sinfo | grep gpu` to see if any GPU nodes are down.
Putting it all together in a Script

SLURM Script

```
#!/bin/bash
#SBATCH -o test.out
#SBATCH -e test.err
#SBATCH -p gpu
#SBATCH --gres=gpu:1
#SBATCH -c 2
#SBATCH -t 01:00:00
#SBATCH -A your_allocation

module purge
module load singularity
module load tensorflow

# Assuming that the container has been copied to /scratch/$USER
containerdir=/scratch/$USER
echo $containerdir
singularity exec --nv $containerdir/tensorflow-1.12.0-py36.simg \
python pytorch_mnist.py
```
NEED MORE HELP?

Office Hours
Tuesdays: 3 pm - 5 pm, PLSB 430
Thursdays: 10 am - noon, HSL, downstairs
Thursdays: 3 pm - 5 pm, PLSB 430

Website:
arcs.Virginia.edu
Or, for immediate help:
hpc-support@virginia.edu
APPENDICES

A: Using Jupyter Notebooks on Rivanna
B: Connecting to Rivanna with an ssh client
C: Connecting to Rivanna with MobaXterm
D: Neural Networks
APPENDIX A

Using Jupyter Notebooks on Rivanna
JupyterLab

• JupyterLab is a web-based tool that allows multiple users to run Jupyter notebooks on a remote system.

• ARCS now provides JupyterLab on Rivanna.
Accessing JupyterLab

• To access JupyterLab, type the following in your web browser:
  
  https://rivanna-portal.hpc.virginia.edu/

• After logging in via Netbadge in, you will be directed to the Open OnDemand main page.
Starting Jupyter Instance

• In the top, click on “Interactive Apps” and in the drop-down box, click on “Jupyter Lab”.
Starting a Jupyter Instance

- A form will appear that allows you to specify the resources for your Notebook.
  - Our example will be using TensorFlow; so, we need to make sure that we select the Rivanna Partition called “GPU”.
  - Also, don’t forget to put in your “MyGroup” name for the Allocation
  - Finally, click the blue “Launch” button at the bottom of the form (not shown here).
Starting a Jupyter Instance

- It may take a little bit of time for the resources to be allocated.

- Wait until a blue button with “Connect to Jupyter” appears.

- Click on the blue button.
JupyterLab Environment

You should see a list of folders and files in your home directory.

And, a set of tiles with empty notebooks or consoles.
Opening a Notebook

• If you have an existing notebook, you can use the left-pane to maneuver to the file and click on it to open it.

• Or, if you want to start a new notebook, you can click on the notebook tile, for the appropriate underlying system.
Classic Notebook

• If you feel more comfortable working with the former Jupyter interface, you can select:
  Help> Launch Classic Notebook

• But, for our example, we will stay with the Jupyter Lab format.
Copying our Notebook to your Directory

• We will open a terminal window to copy files into our home directory.
  • In the Launcher panel, scroll down until you see the “Other” category.
  • Click on the Terminal tile.
The Terminal Window

• A terminal window (or shell) will appear in a separate tab:
Copying our Notebook to your Directory

• Make sure that you are in your home directory by typing `cd`.

• Type:

```
cd
scp -r /share/resources/source_code/Notebooks/TensorFlow_Example .
```
Opening the Notebook

• Close the browser tab for the Terminal Window.

• You should be back on the page that shows your Home directory in Jupyter. (If not, click on the browser tab to get back to the Jupyter Home page.)

• In the file browser pane, click on the folders TensorFlow_Example and Notebooks to get to the file: Python_TensorFlow.ipynb

• Double-click on Python_TensorFlow.ipynb to open the notebook.
Running the Notebook

• To run a particular cell, click inside the cell and press Shift & Enter or Ctrl & Enter.
  • Shift & Enter will advance to the next cell
  • Ctrl & Enter will stay in the same cell

• To run the entire notebook, select
  • Run > Run All Cells
Cautions

• Any changes that you make to the notebook may be saved automatically.

• When the time for your session expires, the session will end without warning.

• Your Jupyter session will continue running until you delete it.
  • Go back to the “Interactive Sessions” tab.
  • Click on the red Delete button.
APPENDIX B

Connecting to Rivanna with an ssh client
SSH Clients

• You will need an ssh (secure shell) client on your computer.

  • On a Mac or Linux system, use ssh (Terminal application on Macs)
    
    \texttt{ssh -Y mst3k@rivanna.hpc.virginia.edu}

  • On a Windows system, use \texttt{MobaXterm}
    
    • To install MobaXterm use the URL: \url{http://mobaxterm.mobatek.net}
    
    • The free "home" version is fine for our purpose.

When you are Off-Grounds, you must use the UVa Anywhere VPN client.
Connecting to the Cluster

• The hostname for the Interactive frontends: rivanna.hpc.virginia.edu  
  (does load-balancing among the three front-ends)

• However, you also can log onto a specific front-end:  
  • rivanna1.hpc.virginia.edu  
  • rivanna2.hpc.virginia.edu  
  • rivanna3.hpc.virginia.edu  
  • rivanna-viz.hpc.virginia.edu
Connecting to the Cluster with ssh

• If you are on a Mac or Linux machine you can connect with ssh.
• Bring up a terminal window and type:
  
  `ssh -Y userID@rivanna.hpc.virginia.edu`

• When it prompts you for a password, use your Eservices password.
APPENDIX C

Connecting to Rivanna with MobaXterm (Windows)
Connecting to the Cluster with MobaXterm

- The first time that you start up MobaXterm, click on the Session icon.
Connecting to the Cluster with MobaXterm

- It will bring up a window asking for the type of session.
- Select SSH and click Okay.
Connecting to the Cluster with MobaXterm

- It will prompt you for remote host and username.
- You will have to click on the box next to “Specify username” before you can type in your username.
Connecting to the Cluster with MobaXterm

- It will prompt you for your password.
- Note: It will appear as if nothing is happening when you type in your password. It will not display circles or asterisks in place of the characters that you type.
Finally, a split screen will appear.

- The right pane is a terminal window.
- The left pane is a list of files in your remote folder that you can click, drag, and drop onto your local desktop.
Connecting to the Cluster with MobaXterm

• MobaXterm will save your session information.

• The next time that you open MobaXterm, you can double-click on the Session that you want.
APPENDIX D

Neural Networks
Neural Network

A computational model used in machine learning which is based on the biology of the human brain.
Neurons in the Brain

Neurons continuously receive signals, process the information, and fires out another signal.

The human brain has about 86 billion neurons, according to Dr. Suzana Herculano-Houzel.

Simulation of a Neuron

The “incoming signals” could be values from a data set(s).

A simple computation (like a weighted sum) is performed by the “nucleus”.

The result, $y$, is “fired out”.

$$ y = \sum_{i} w_i x_i $$
Simulation of a Neuron

The weights, $w_i$, are not known.

During training, the “best” set of weights are determined that will generate a value close to $y$ given a collection of inputs $x_i$. 

$$
\sum_i w_i x_i
$$
Simulation of a Neuron

A single neuron does not provide much information (often times, a 0/1 value)
A Network of Neurons

Different computations with different weights can be performed to produce different outputs.

This is called a feedforward network because all values progress from the input to the output.
A neural network has a single hidden layer. A network with two or more hidden layers is called a “deep neural network”.
TENSOR FLOW
What is TensorFlow?

An example of deep learning; a neural network that has many layers.

A software library, developed by the Google Brain Team
Deep Learning Neural Network

Image borrowed from:
http://www.kdnuggets.com/2017/05/deep-learning-big-deal.html
Terminology: Tensors

Tensor: A multi-dimensional array

Example: A sequence of images can be represented as a 4-D array: [image_num, row, col, color_channel]

Px_value[1, 1, 3, 2]=1
Terminology: Computational Graphs

• Computational graphs help to break down computations.
  • For example, the graph for $y=(x_1+x_2)*(x_2 - 5)$ is

```
x1 --> a = x1 + x2 --> y = a*b

x2 --> b = x2 - 5
```

The beauty of computational graphs is that they show where computations can be done in parallel.
CONVOLUTIONAL NEURAL NETWORKS
What are Convolutional Neural Networks?

Originally, convolutional neural networks (CNNs) were a technique for analyzing images.

CNNs apply multiple neural networks to subsets of a whole image in order to identify parts of the image.

Applications have expanded to include analysis of text, video, and audio.
The Idea behind CNN

Recall the old joke about the blindfolded scientists trying to identify an elephant.

A CNN works in a similar way. It breaks an image down into smaller parts and tests whether these parts match known parts.

It also needs to check if specific parts are within certain proximities.

For example, the tusks are near the trunk and not near the tail.

Image borrowed from https://tekrighter.wordpress.com/2014/03/13/metabolomics-elephants-and-blind-men/
Is the image on the left most like an X or an O?

Images borrowed from
What features are in common?
Building Blocks of CNN

• CNN performs a combination of layers
  • Convolution Layer
    • Compares a feature with all subsets of the image
    • Creates a map showing where the comparable features occur
  • Rectified Linear Units (ReLU) Layer
    • Goes through the features maps and replaces negative values with 0
  • Pooling Layer
    • Reduces the size of the rectified feature maps by taking the maximum value of a subset

• And, ends with a final layer
  • Classification (Fully-connected layer) layer
    • Combines the specific features to determine the classification of the image
Steps

• These layer can be repeated multiple times.
• The final layer converts the final feature map to the classification.
Example: MNIST Data

• The MNIST data set is a collection of handwritten digits (e.g., 0 – 9).

• Each digit is captured as an image with 28x28 pixels.

• The data set is already partitioned into a training set (60,000 images) and a test set (10,000 images).

• The tensorflow packages have tools for reading in the MNIST datasets.

• More details on the data are available at http://yann.lecun.com/exdb/mnist/

Image borrowed from *Getting Started with TensorFlow* by Giancarlo Zaccone
Coding CNN: General Steps

1. Load PyTorch Packages
2. Define How to Transform Data
3. Read in the Training Data
4. Read in the Test Data
5. Define the Model
6. Configure the Learning Process
7. Define the Training Process
8. Define the Testing Process
9. Train & Test the Model
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import os
Python

image_mean = 0.1307
image_std = 0.3081
batch_size = 64
test_batch_size = 1000
numCores = int(os.getenv('SLURM_CPUS_PER_TASK'))

transform = transforms.Compose([transforms.ToTensor(),
                                 transforms.Normalize((image_mean, ), (image_std, ))])
```python
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./data', train=True, download=True,
    transform=transform),
    batch_size = batch_size,
    shuffle = True, num_workers = numCores)
```
test_loader = torch.utils.data.DataLoader(  
datasets.MNIST('..../data', train=False,  
transform=transform), batch_size = test_batch_size,  
shuffle = True, num_workers = numCores)
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

**Python**

Where do the sizes come from in `nn.Linear`?

Initially, 1x28x28

\[ W_{out} = \text{floor}((W_{in} - \text{kernel} + 2 \times \text{padding})/2) + 1 \]

After first convolution: 20x12x12

After second convolution: 50x4x4
epochs = 10
lr = 0.01
momentum = 0.5
seed = 1
log_interval = 100

torch.manual_seed(seed)

device = torch.device("cuda")
model = Net().to(device)

optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)
def train(model, device, train_loader, optimizer, epoch, log_interval):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {}
        [{}/{} ({:.0f}%)]
        tLoss: {:.6f}'.format(epoch, batch_idx * len(data),
        len(train_loader.dataset), 100. * batch_idx /
        len(train_loader), loss.item()))
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()
test_loss /= len(test_loader.dataset)

print('Test set: Average loss: {:.4f}, Accuracy: {}/{},
{} {:.0f}%%'.format(test_loss, correct, len(test_loader.dataset), 100. * correct / len(test_loader.dataset)))
for epoch in range(1, epochs + 1):
    train(model, device, train_loader, optimizer, epoch, log_interval)

test(model, device, test_loader)