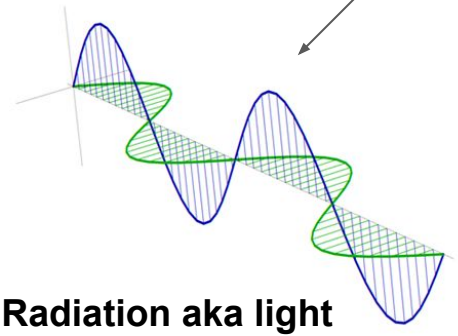
Classifying Galaxy Morphologies

Andrei Ignat, Ankit Gupta, Keshav Mathur, Ryan Hausen

Astrophysics as Big Data



Radiation aka light

Travelling all the way

to our planet

Telescopes

Astrophysics + Machine Learning + HPC = 

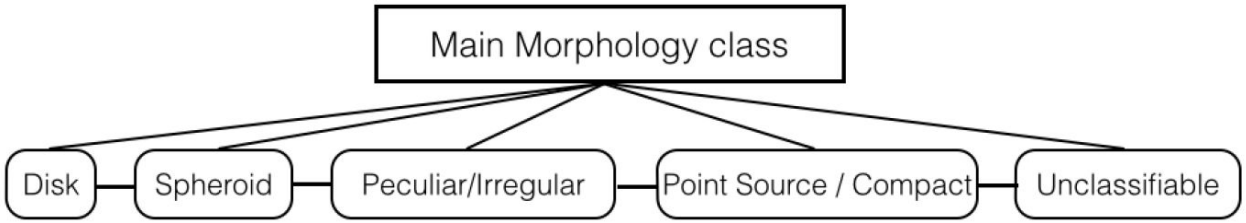
Huge quantity of astrophysical data requires

a. ML techniques for analysis
   - basically all the different branches of machine learning can be applied to astro data
   - software implementations are generally open source(e.g. http://www.astroml.org/)
   - we will focus on https://www.tensorflow.org/

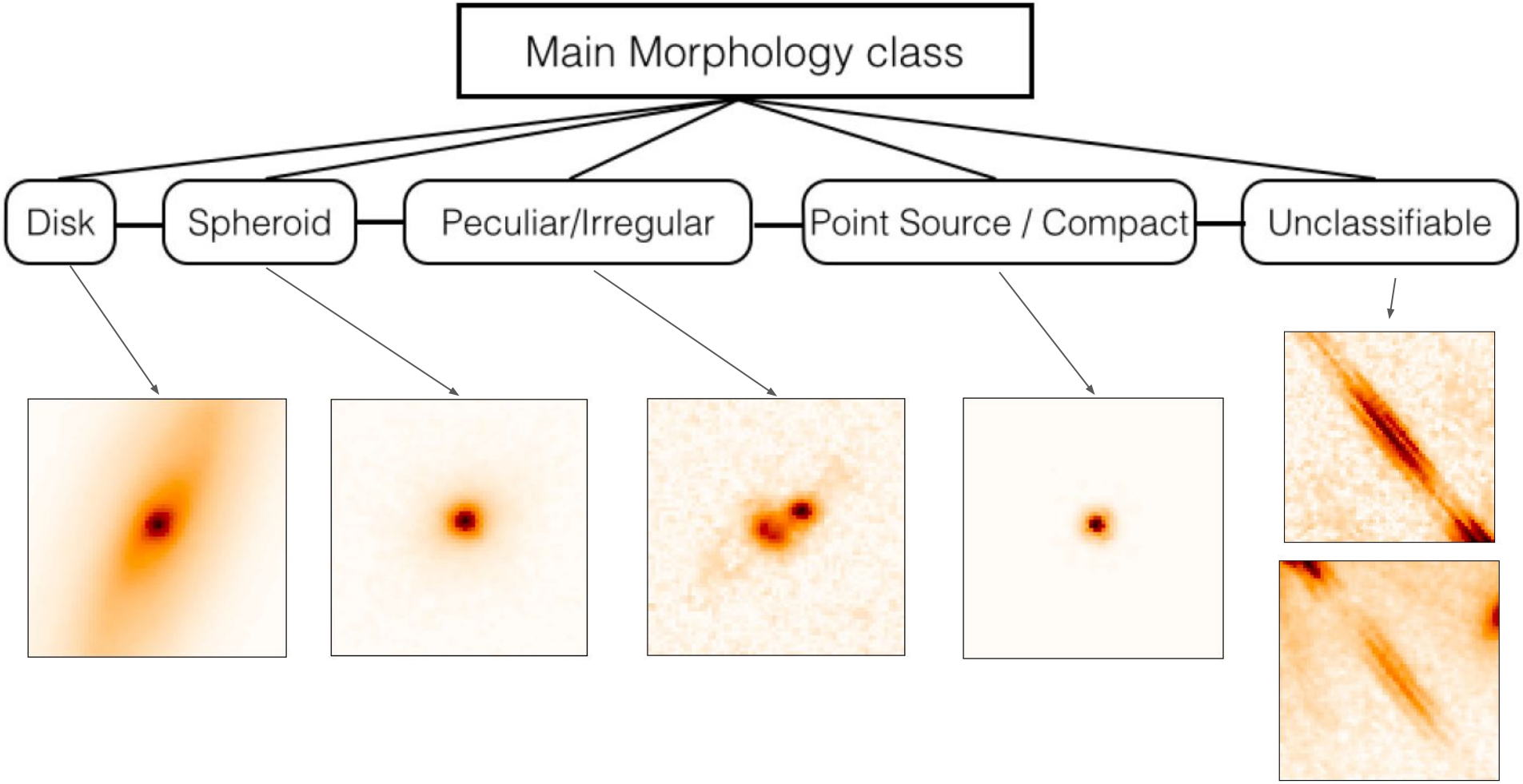
b. High performance clusters to process and analyze the data
   - ML software packages are (usually) distributed
   - some of the largest computing clusters are dedicated to astrophysical data
   - we will use Hyades

The problem(Huertas-Company et. al 2015)

- Galaxies have different morphologies(shapes)
- Abundance of data makes it impossible to classify everything by hand



IDEA: What if we classified the shape of some galaxies by hand, and trained a computer model to do the rest for us?


Main Morphology class
Disk
Spheroid
Peculiar/Irregular
Point Source / Compact
Unclassifiable

The CANDELS fields aka The Dataset

GOODS-S

- contains ~8000 galaxies
- all the galaxies in the dataset were visually classified by humans
- great for **training** the model

GOODS-N, UDS, EGS, COSMOS

- contains ~42000 galaxies
- none of them have been classified
- great motivation for training a **classifier**

Computational requirements

Original paper uses a **Tesla M2090**

- 512 CUDA cores / 6GB DDR5
- double floating point performance: 665 Gigaflops(peak)
- single floating point performance: 1331 Gigaflops(peak)

We have a more expensive toy **Tesla K20**

- 2496 CUDA cores / 5GB DDR5
- double floating point performance: 1170 Gigaflops(peak)
- single floating point performance: 3520 Gigaflops(peak)

Partial dataset

As a starting point, we will use the following subset of the data:

- the original ~8000 galaxies yield a total number of ~50000 images
- these images are obtained using “data multiplication” techniques
- use the images in the H wavelength band => ~23000/50000 images

- train, train, train!

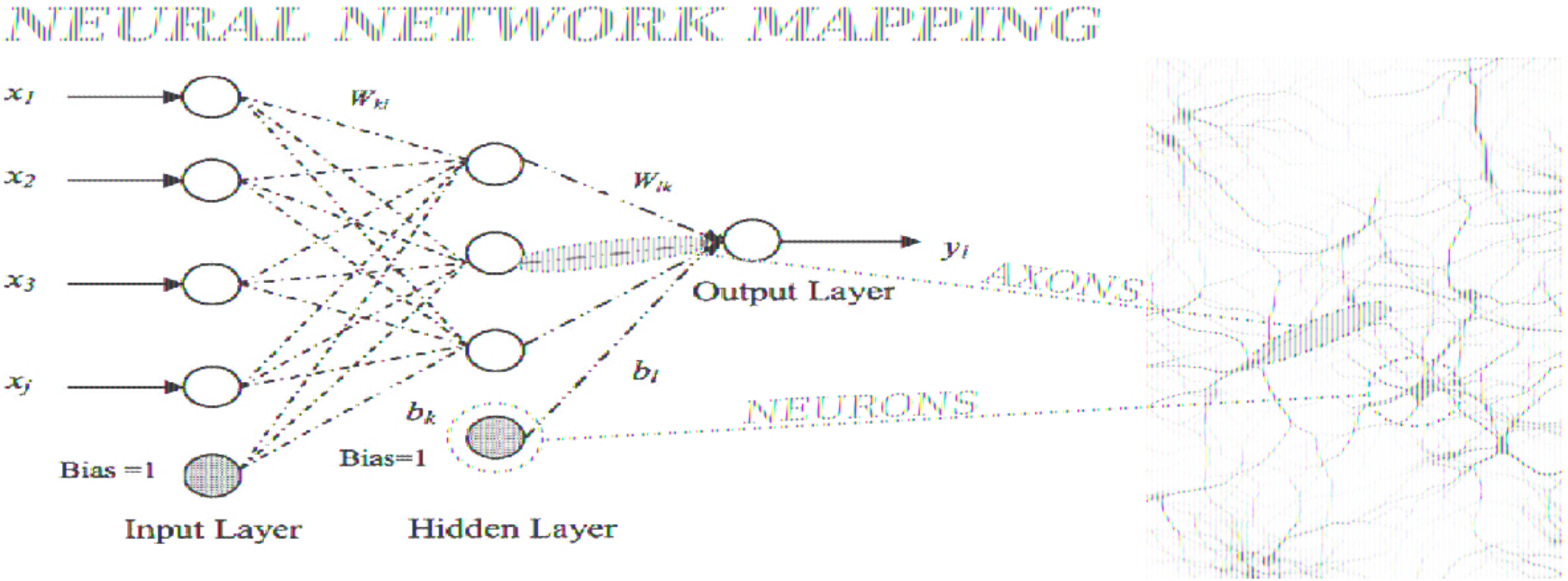
How to approach this problem?

- The problem of morphology classification or pattern recognition is not a new one. We (humans) do it all the time.
- So, all we need to do is model the neural network in our brain artificially on a machine!
- But how does brain work?

Biological Neural Networks

- Human brain has evolved over millions of years and has gotten better at doing many things one of them being pattern recognition.
- Neurons are the computational building blocks of our brain. They connect with each other in complex ways and enable us to “learn”.
- Each neuron inside our brain can be thought of as a **function**. It takes some input and gives some output (depending on its properties).
- “We learn when we make mistakes.”
- We **model** the human brain on a machine using artificial neurons like perceptron or sigmoid.

Biological Neural Network to ANN

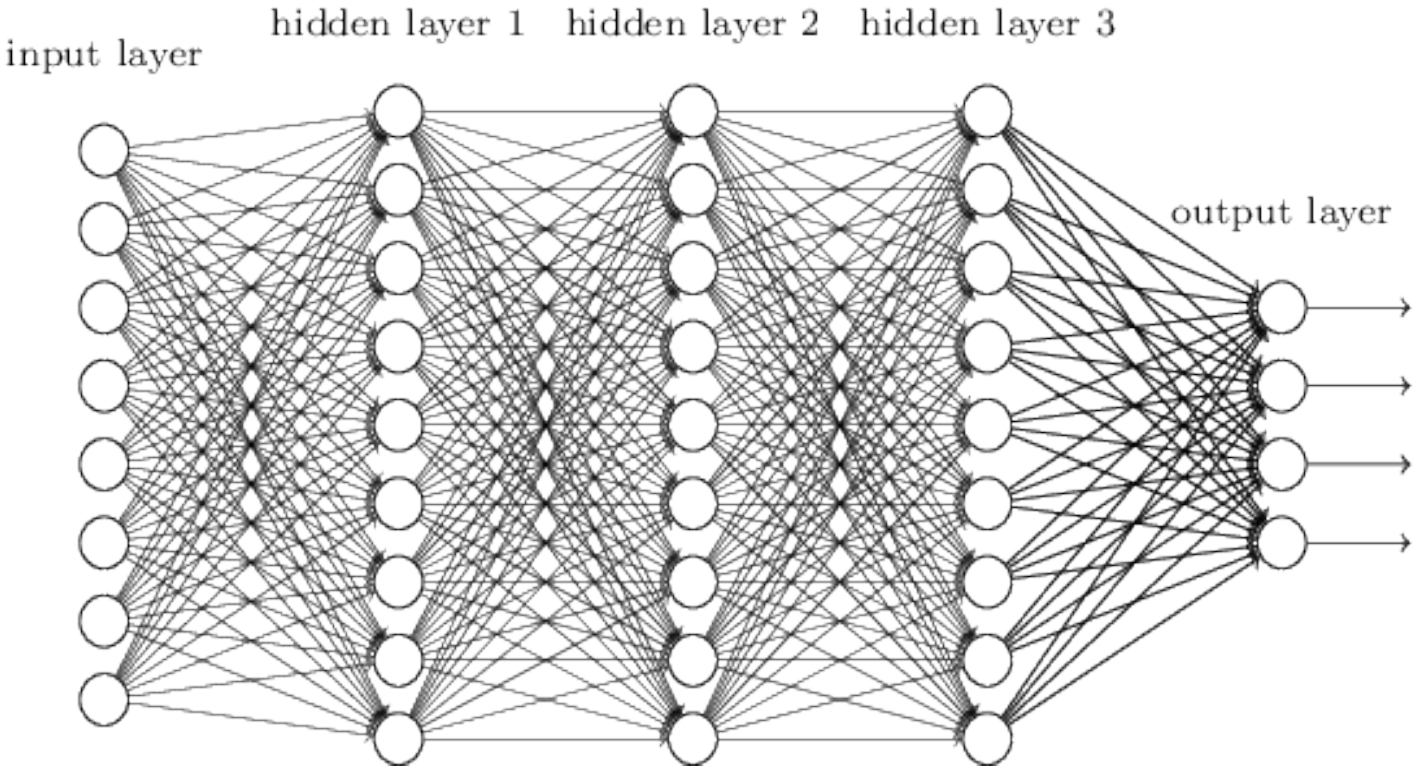


Artificial Neural Networks (ANN)

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." - Dr. Robert Hecht-Nielsen (inventor of the first neurocomputer)

- A network formed by interconnection between layers of artificial neurons.
- A perceptron or a sigmoid combines all the **weighted** inputs (and biases) and applies the activation function on it.
- Each neuron has its own weight vector and bias (offset).
- The behavior of artificial neurons varies across different layers of ANN depending on their weights and biases.
- We feed input to the network from input layer, each layer computes its output and feeds to the next one till we reach the output layer.

Artificial Neural Network (ANN) - Visualization



Artificial Neural Network (ANN) : Terminologies

- Weights and bias
  - Each neuron in every layer of the network has weights and biases which are used to calculate its output.
  - They are also known as network parameters.
  - We learn them during training.
- Cost/Loss function
  - During training, we calculate how different is the network output from the real output based on a cost function. We have used RMSE.
- Backpropagation
  - This is the learning algorithm in which based on the cost function, the network parameters are changed so as to perform better in future.
  - Weights are updated during this step

ANN: Training and Testing

- Divide the total available data into training and test data.
- Learning is done on the training data only.
- Test data is kept unseen from the network during training or learning phase so we can calculate the performance of our network on unseen data.
- Also, during training we need to control hyper-parameters like learning rate in order to make sure that our network does not overfit.
- An overfit network performs well on the training data but poorly on test.
- Ideally, the test and training data should have equal proportion of all the classes.

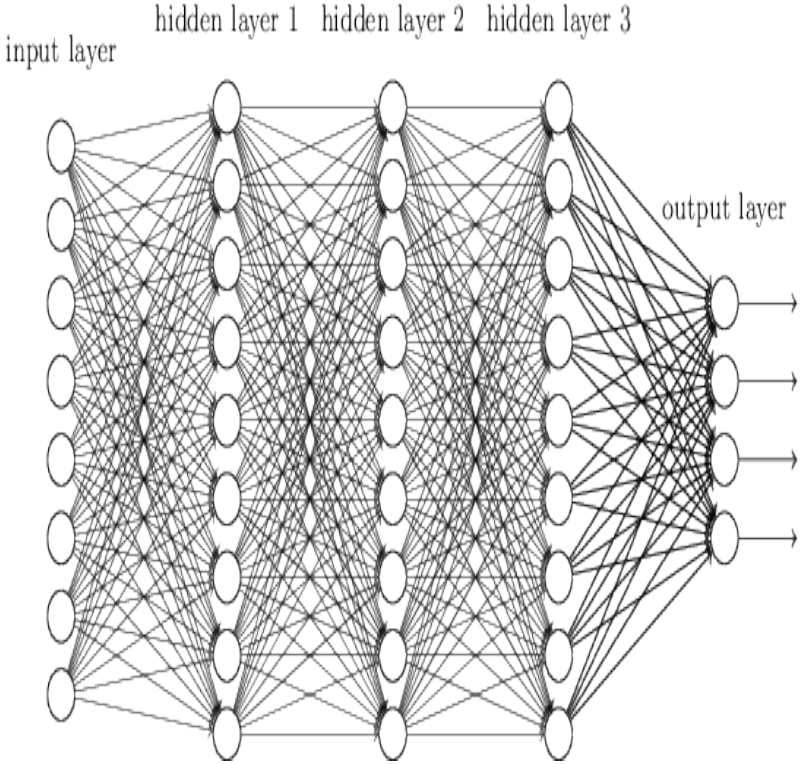
Artificial Neural Networks: Limitations

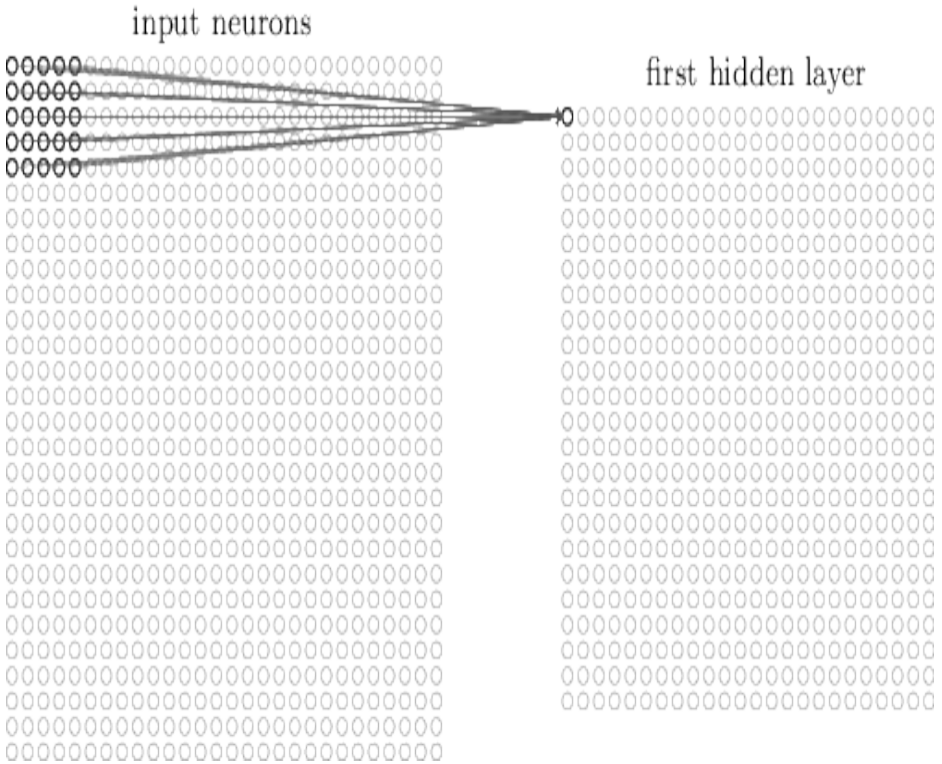
- The ANN architecture doesn’t provide optimum results for vision problems.
- Since every pair of neuron between two layers have their own weights and biases, learning to recognize a shape in one part of the image doesn’t transfer to the same object in some different part of the image!
- In ANN, the number of parameters to be learnt is ridiculously high.
- Also, there is very less modeling a programmer can do. The only thing we can change is the size and number of hidden layers.
- Convolutional Neural Networks provide a much better architecture for vision problems.

Convolutional Neural Networks

- In CNN, we use a local receptive field or filter and apply it on the input image.
- All the pixels inside this field are “tied” together and fed to **one** neuron of the next layer.
- This is unlike ANN, where we connected one neuron from previous layer to each neuron in the next one.
- We then slide this local receptive filter across the whole image and form the next hidden layer.
- The learning part of CNN is similar to ANN. Only the way we treat input is different.

ANN vs CNN


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer


input neurons
first hidden layer

Implementing a Deep Neural Network

- There are several open source frameworks available to implement a deep neural network. Some of them are:
  - Theano
  - Tensorflow
  - Keras
  - Caffe
  - Mocha
- For our implementation we have used Tensorflow.

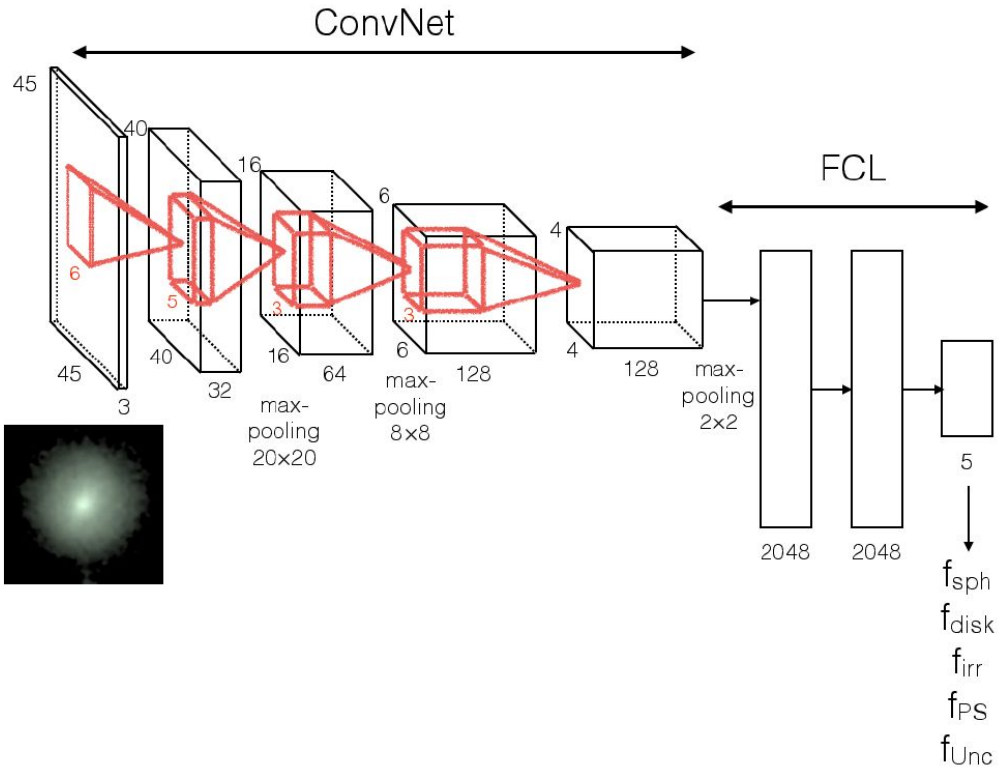
Tensorflow

- Tensorflow is an open source software library for numerical computation using data flow graphs.
- In a data flow graph, the nodes are the mathematical computational units while the edges are multi-dimensional data arrays or tensors.
- Its flexible architecture allows deploying computation on one or more CPUs or GPUs.
- Provides easy to use Python interface.
- Developed at Google.

Tensorflow: How to use it?

- In order to use Tensorflow, we need to visualize our network as a dataflow graph.
- In a deep neural network, input data passes through series of computational layers and gives output.
- A programmer needs to specify the computational architecture in tensorflow and pass data.

The network [Huertas-Company, M., et al]



Parameters

	type	# features	filter size	non-linearity	initial biases	initial weights
1	convolutional	32	6×6	ReLU	0.1	$\mathcal{N}(0, 0.01)$
2	convolutional	64	5×5	ReLU	0.1	$\mathcal{N}(0, 0.01)$
3	convolutional	128	3×3	ReLU	0.1	$\mathcal{N}(0, 0.01)$
4	convolutional	128	3×3	ReLU	0.1	$\mathcal{N}(0, 0.1)$
5	dense	2048	–	maxout (2)	0.01	$\mathcal{N}(0, 0.001)$
6	dense	2048	–	maxout (2)	0.01	$\mathcal{N}(0, 0.001)$
7	dense	37	–	constraints	0.1	$\mathcal{N}(0, 0.01)$

Avoiding overfitting

- Data augmentation: adding noise, rotations, translation, scaling and flipping.
- Regularization: We use dropout in the dense layers to penalize model complexity and promote learning on their own amongst the neurons.
- The model will start to overfit after a certain number of epochs when the test RMSE starts increasing even though the train RMSE will be decreasing. Good point to stop.

Missing from our implementation

- Maxout
- Model averaging
- Brightness adjustment

Performance

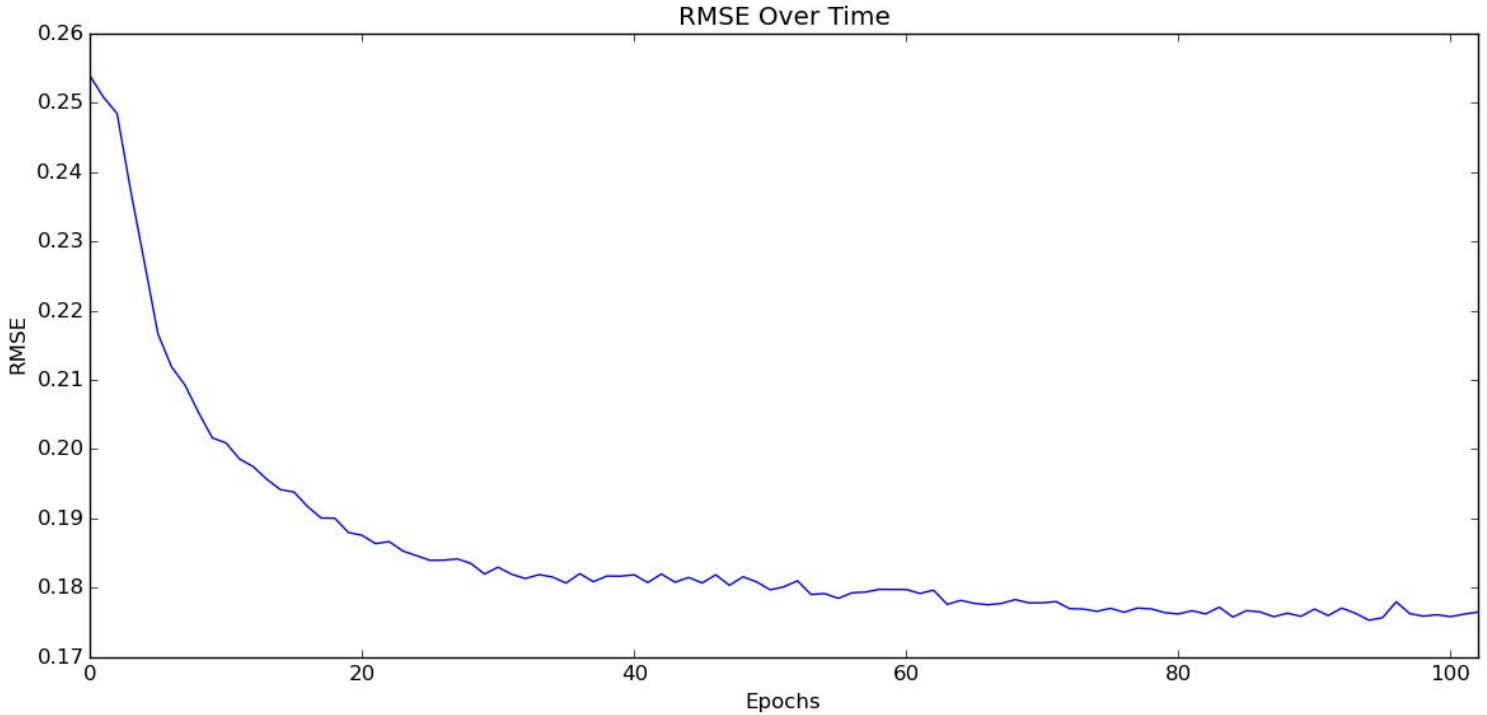
- We trained the network on Hyades' GPU node (Tesla K20) and a PC (GeForce GTX 650).
- At Hyades it takes an average of < 50 seconds per epoch (without real time augmentation). Each epoch involves learning from 19,000 images.
- Before doing pre processing (rescaling to 45x45) it took 8 minutes per epoch on Hyades compared to just just 108 seconds on the PC. IO latency? Needs inspection.
- But Hyades has the benefit of higher memory and can be trained on bigger batches and can do predictions in bulk instead of batches.

Results

The network was evaluated using Root Mean Square Error (RMSE)

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Results


RMSE Over Time
RMSE
0.26
0.25
0.24
0.23
0.22
0.21
0.20
0.19
0.18
0.17
0
20
40
60
80
100
Epochs

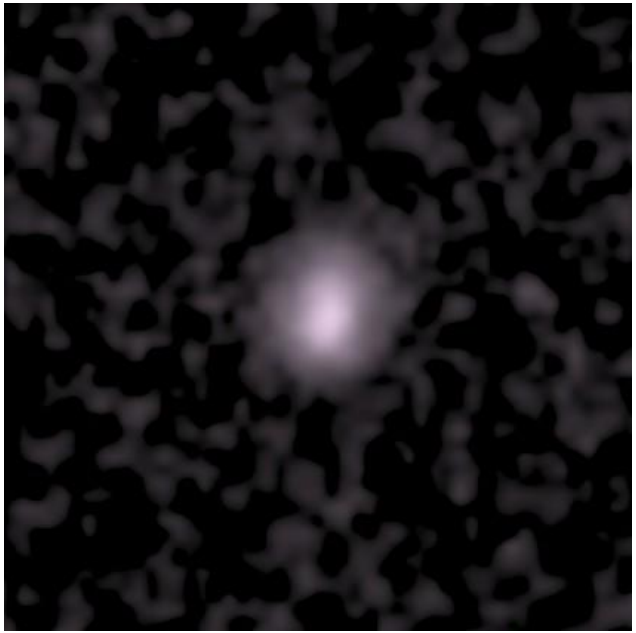
Results

Although we were able to achieve reasonable results ~.17, we didn’t match the best results from the paper. This could be for a couple reasons:

- Preprocessing differences
- Real time data augmentation differences
- Model averaging
- Maxout layers

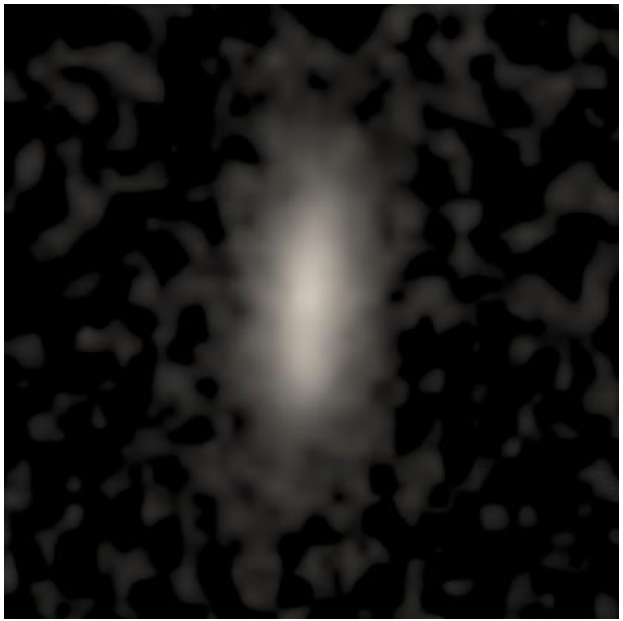
Classifications from the model

Spheroid	.64
Disk	.32
Irregular	.09
Point Source	.17
Unclassifiable	.04



Classifications from the model

Spheroid	.03
Disk	.94
Irregular	.07
Point Source	.01
Unclassifiable	.04



Future Work

- Continue to improve the model to match paper, and get better results
- Apply deep learning techniques to other astronomical problems
- Explore other ideas in image preprocessing
- Comparison between performance of TensorFlow and other deep learning frameworks