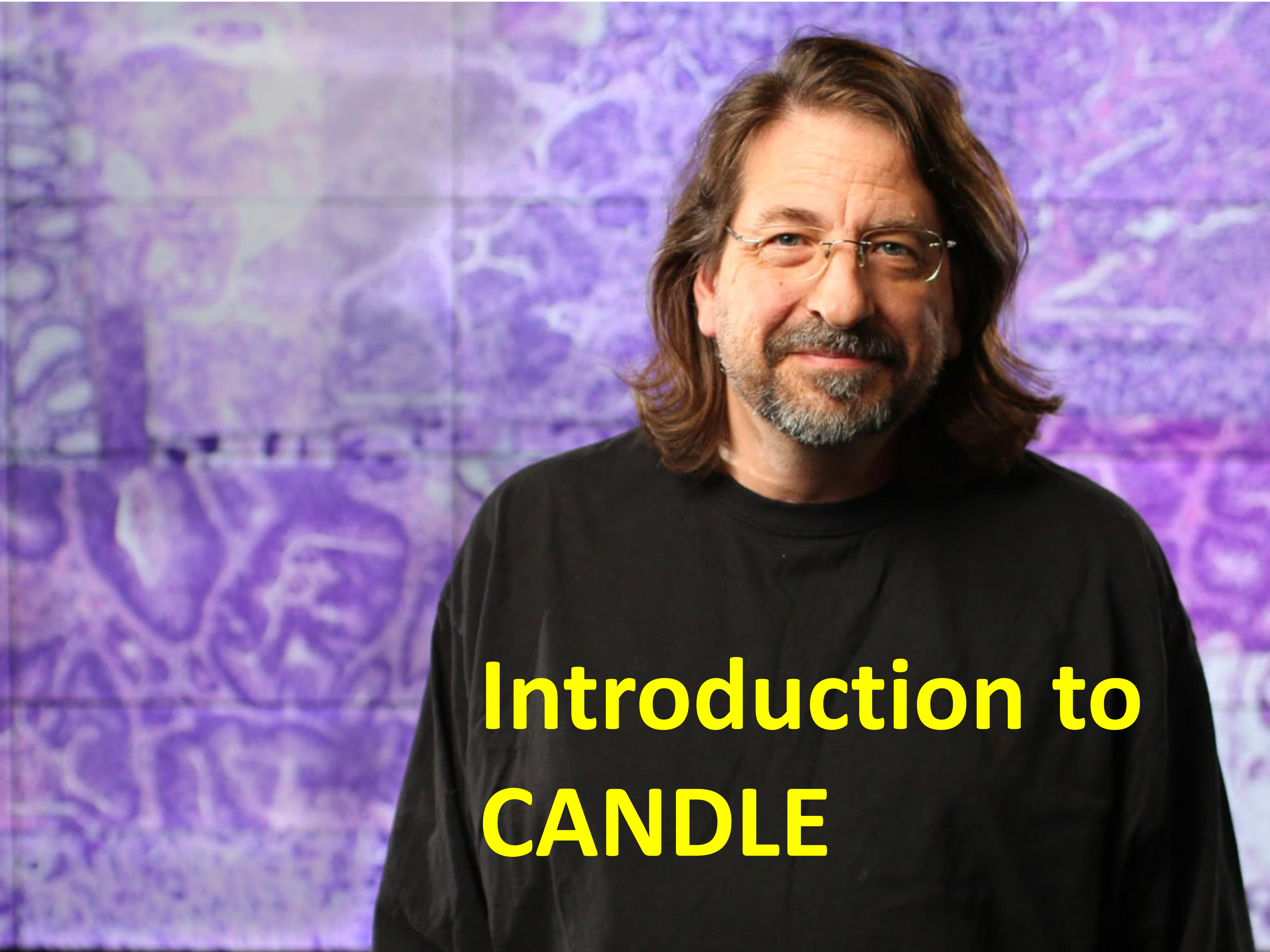
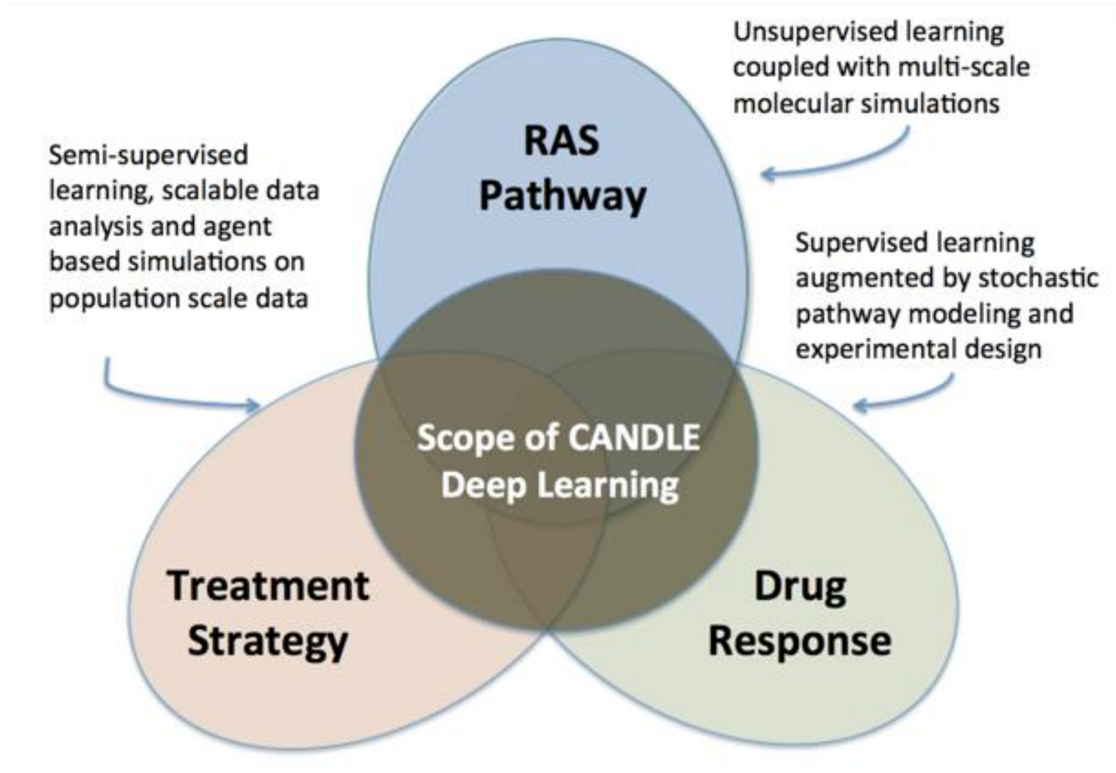
Introduction to CANDLE



ECP-CANDLE : CANcer Distributed Learning Environment



CANDLE Goals

Develop an exscale deep learning environment for cancer

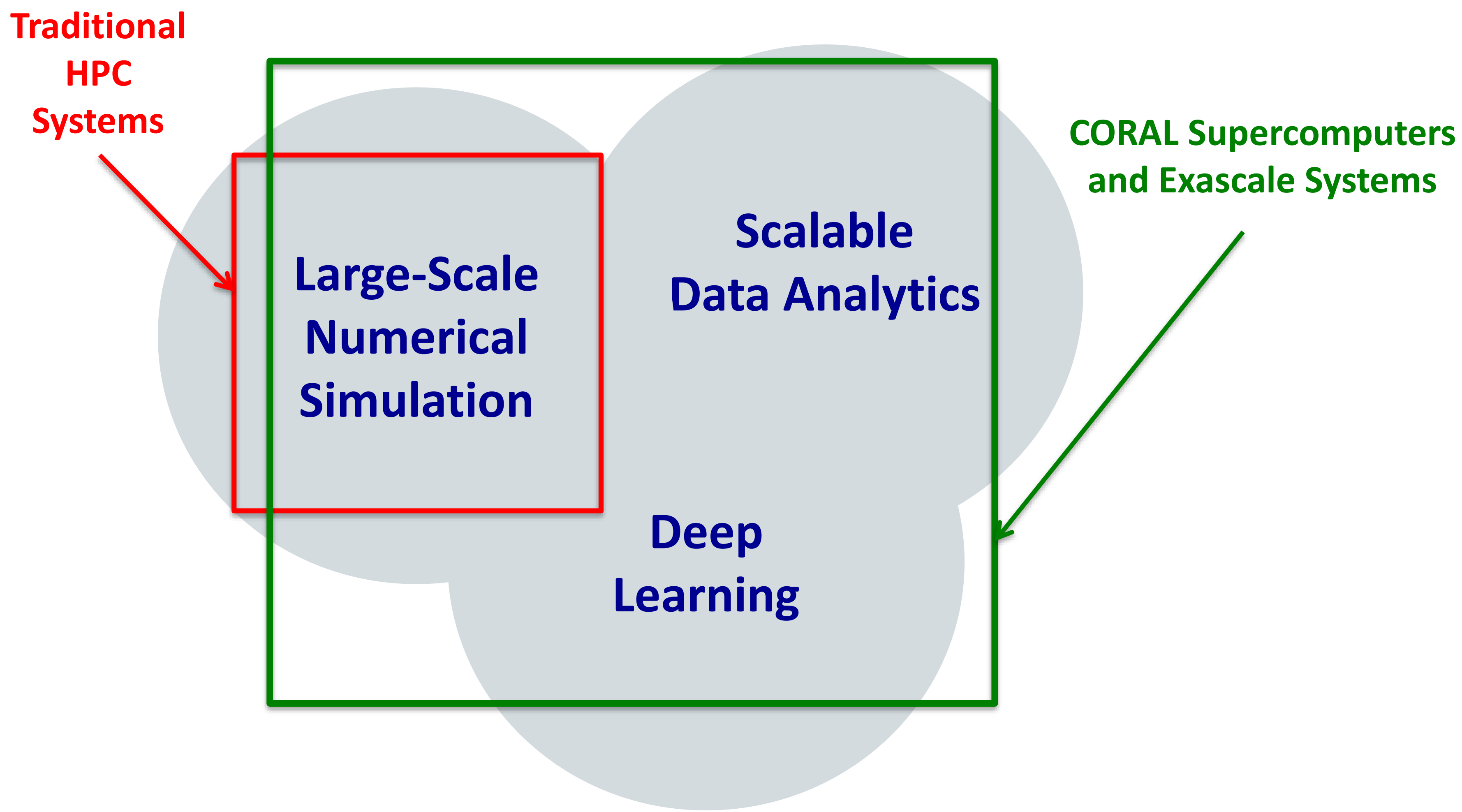
Building on open source Deep learning frameworks

Optimization for CORAL and exascale platforms

Support all three pilot project needs for deep dearning

Collaborate with DOE computing centers, HPC vendors and ECP co-design and software technology projects

DOE Objective: Dirve Integration of Simulation, Data Analytics and Machine Learning



Aurora 2021 (A21) Exascale System

Architectural support for three pillars

- **Large-scale Simulation (PDEs, traditional HPC)**
- **Data Intensive Applications (science pipelines)**
- **Deep Learning and Emerging Science AI**

CANDLE Challenge Problem Statement

Enable the most challenging deep learning problems in Cancer research to run on the most capable supercomputers in the DOE

Candle Functional Goals

- Enable high productivity for deep learning centric workflows
- Support Key DL frameworks on DOE supercomputers
- Support multiple paths to concurrency
- Manage training data, model search, scoring, optimization, production training and inference
- CANDLE runtime/supervisor (interface with batch schedulers)
- CANDLE library for improving model development (UQ, HPO, CV, MV)
- Well documented examples and tutorials
- Leverage as much open source as possible

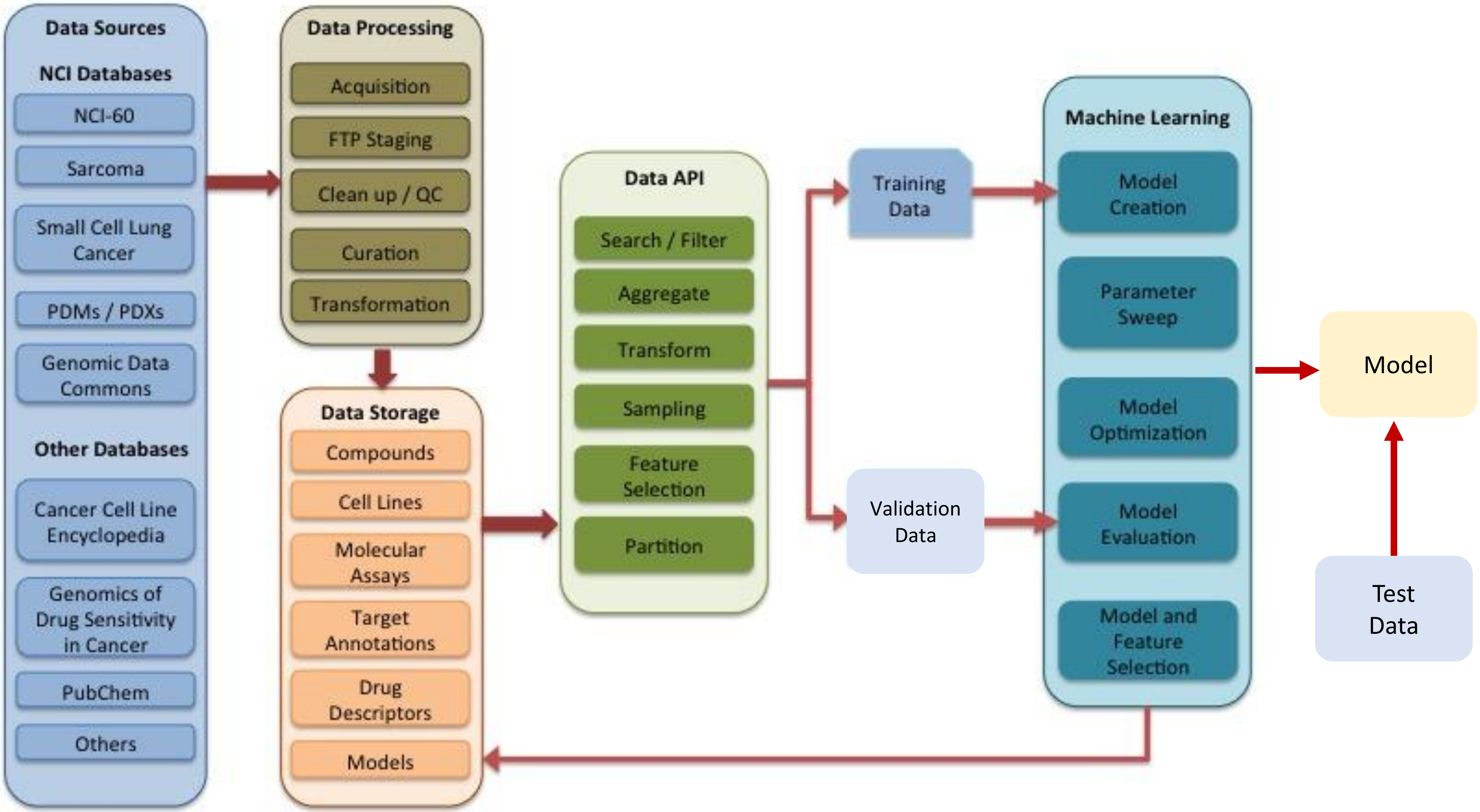
CANDLE Software Stack

Hyperparameter Sweeps, Data Management (e.g. DIGITS, Swift, etc.)	**Workflow**
Network description, Execution scripting API (e.g. Keras, Mocha)	**Scripting**
Tensor/Graph Execution Engine (e.g. Theano, TensorFlow, LBANN-LL, etc.)	**Engine**
Architecture Specific Optimization Layer (e.g. cuDNN, MKL-DNN, etc.)	**Optimization**

CANDLE Workflow Layer

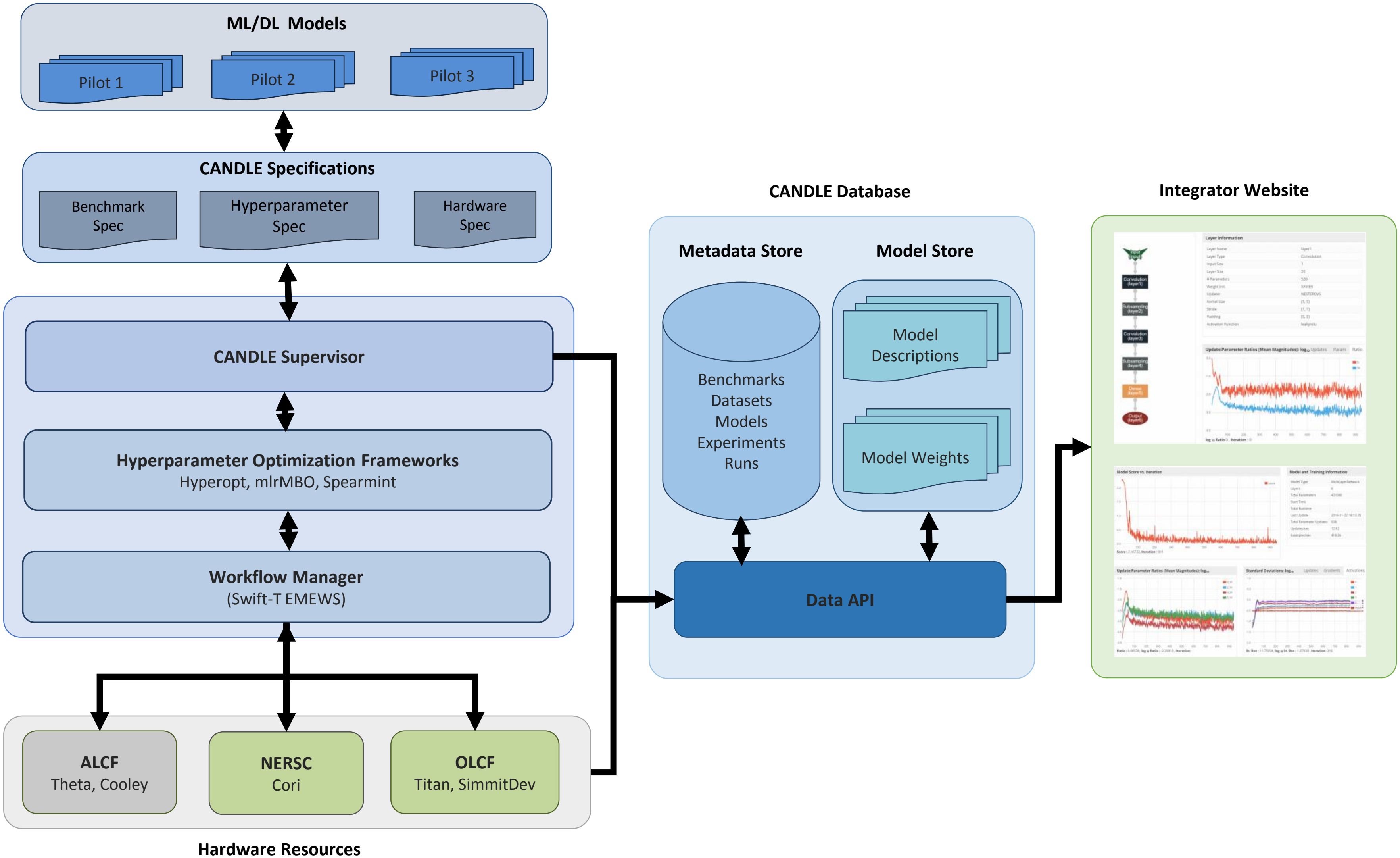
- "Convienence and Productivity" layer
- Used to manage large-scale training runs
  - Hyperparameter searches $O(10^4)$ jobs
  - Cross validation (5-fold, 10-fold, etc.)
  - Data encodings (log2, Z-score, percent, etc.)
  - Low-level optimizations (tensor backends)
- Locate and transform input data
- Manage caching on local NV store
  - Internal joins, batching management, epochs
- Each job could be 100's to 1000's of nodes
- Driver scripts manage runs of 1K >10M core/hrs

Pilot1 CANDLE General Workflow



Cancer Data Processing, Storage and Machine Learning Workflow

CANDLE System Architecture



Model Scripting Interface

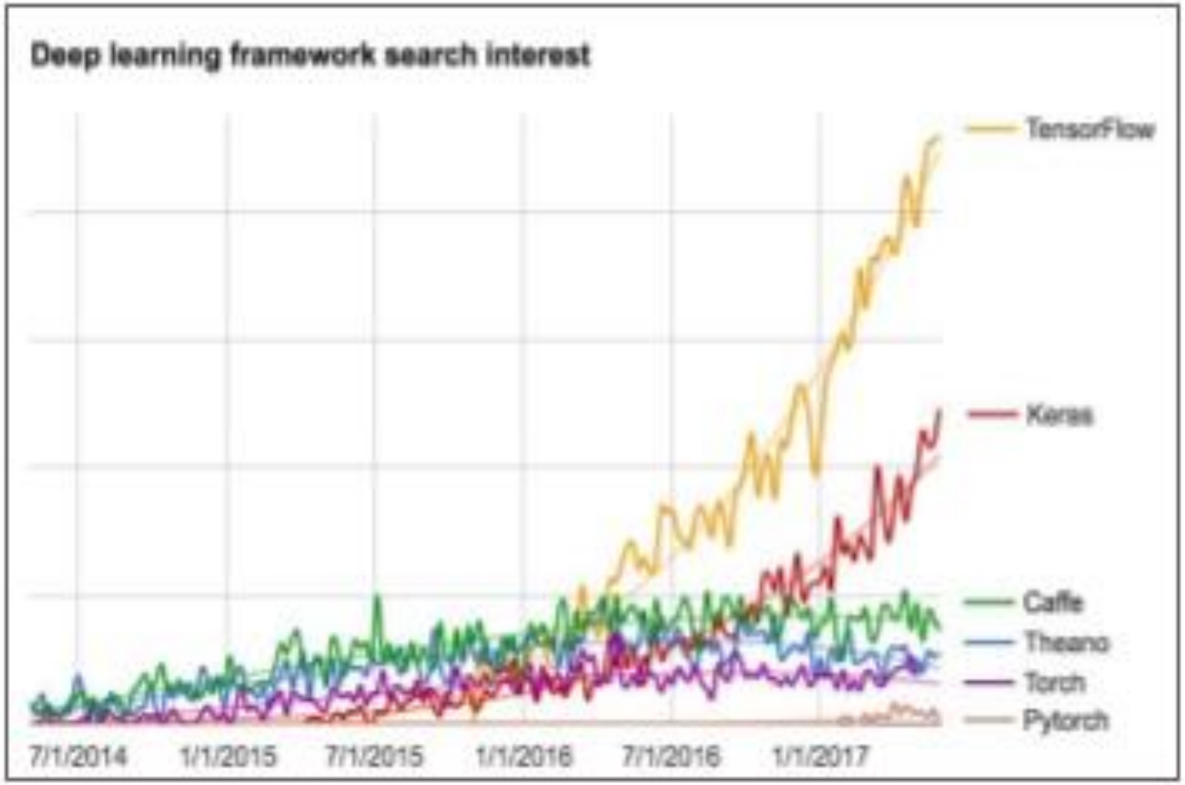
- Aimed at the user developing models.. Keras is our canonical example
- **Keras** – python interface
  - Theano and TensorFlow
  - target for LBANN
- **Mocha** – julia interface
  - Pure julia backend
  - cuDNN
- **Lasagne** – python interface
  - Theano

Keras

- https://keras.io/
- Minimalist, highly modular neural networks library
- Written in Python
- Capable of running on top of either TensorFlow/Theano and CNTK
- Developed with a focus on enabling fast experimentation

Keras

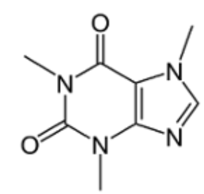
- Keras is the *de facto* deep learning frontend



Source: @fchollet, Jun 3 2017

DL Frameworks "Tensor Graph Engines"

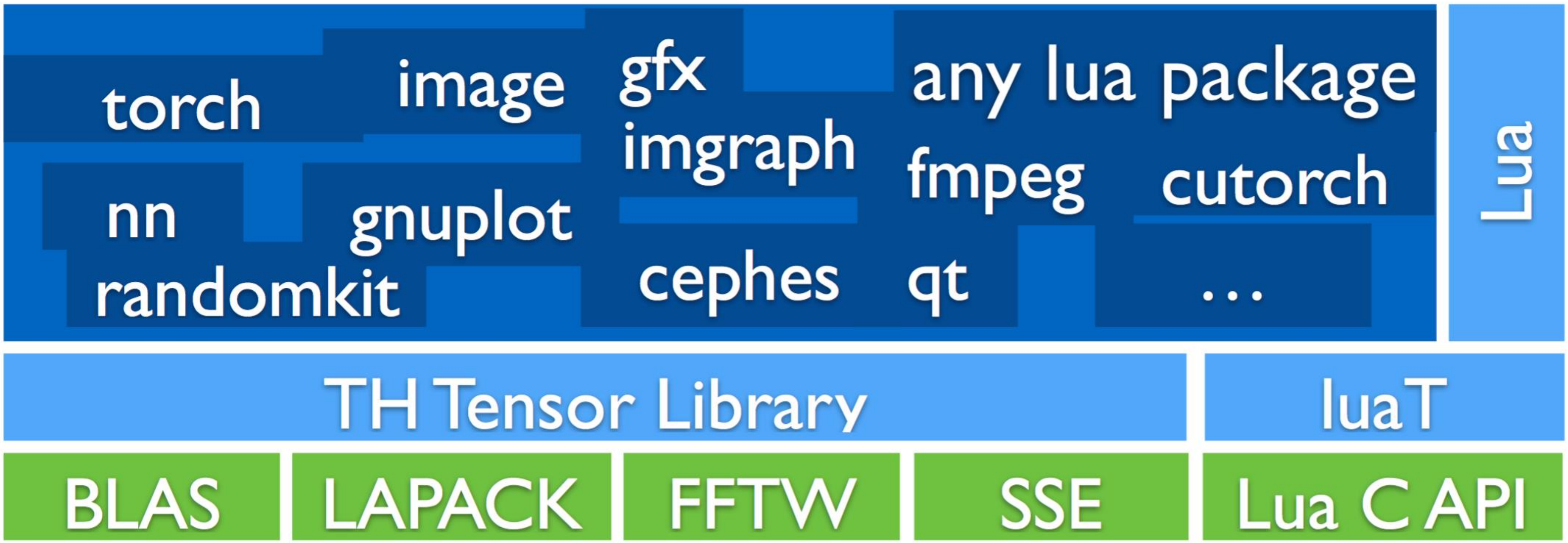
- **TensorFlow** (c++, symbolic diff+)
- **Theano** (c++, symbolic diff+)
- **Neon** (integrated) (python, symbolic diff+)
- **Torch7 TH Tensor** (c layer, symbolic diff-, pgks)
- **Mxnet** (integrated) (c++)
- **Caffe** (integrated) (c++, symbolic diff-)
- **Mocha** backend (julia + GPU)
- **LBANN** (c++, aimed at scalable hardware)
- **CNTK** backend (microsoft) (c++)
- **PaddlePaddle** (Baidu) (python, c++, GPU)



Open Source Framework Comparison

	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	+
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	+

Torch7 "Stack"



Hardware Optimization Layers

- **cuDNN** – NVIDIA low level library
  - Caffe, TensorFlow, Theano, Torch, CNTK
  - Supports many DL features, forwad and backward layer types for common topologies
  - Forward and backward convolution
- **MKL-DNN** – intel deep learning library
  - Convolution, pooling, ReLU, etc. C API
  - Cifar, *AlexNet*, VGG, *GoogleNet* and ResNet*.

Parallelism Options and I/O

- **Ensemble Parallelism** (replications for HPO, UQ or ensemble prediction)
- **Data Parallelism** (distributed training by partitioning training data)
- **Model parallelism** (parallel training by partitioning network)
- **Streaming** training data
- **Dashboard** reporting progress

Hyper Parameter Search

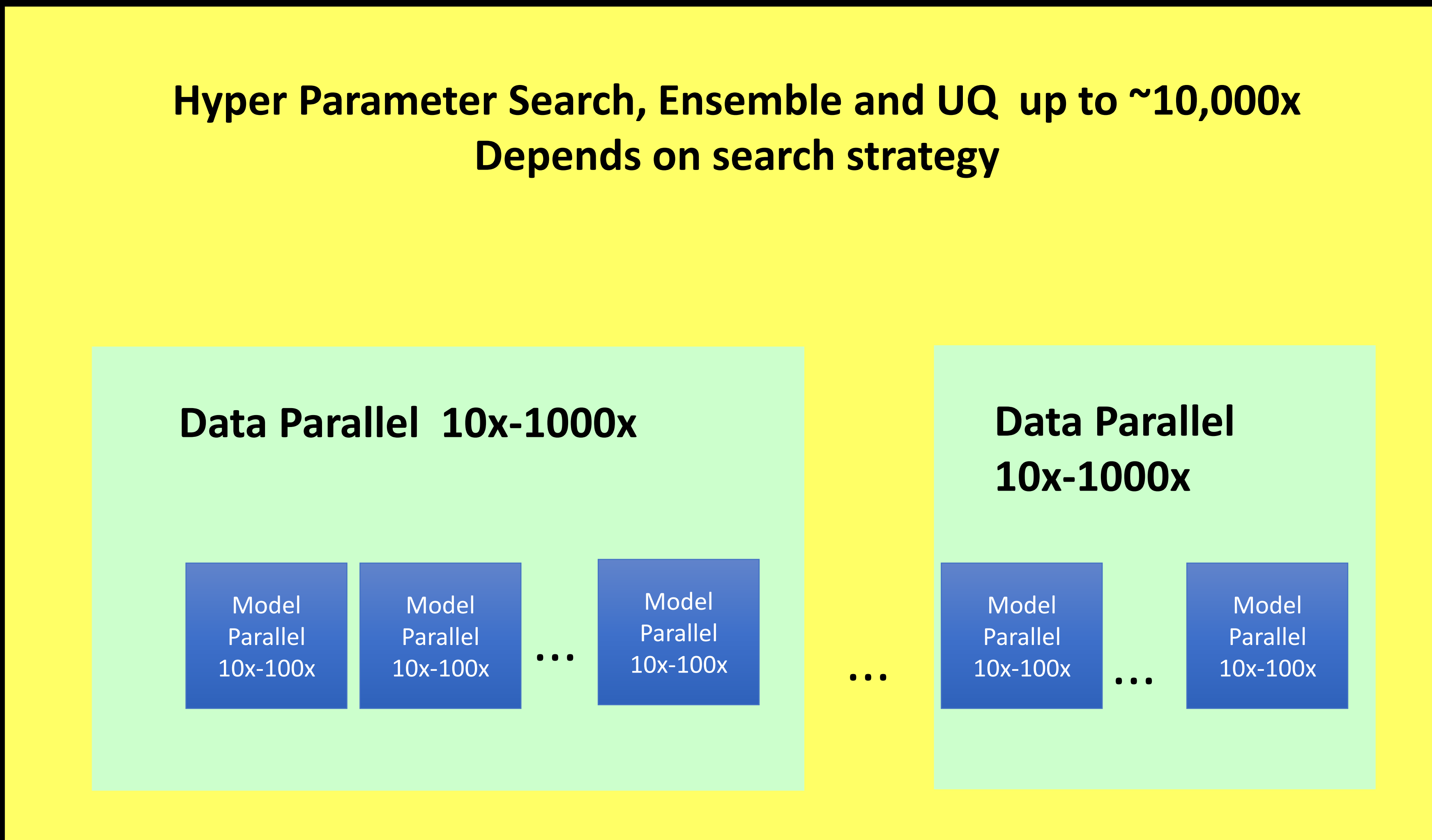
3 x 3 x 3 x 4 x 3 x 3 x 3 x 4 = 11,664 cases

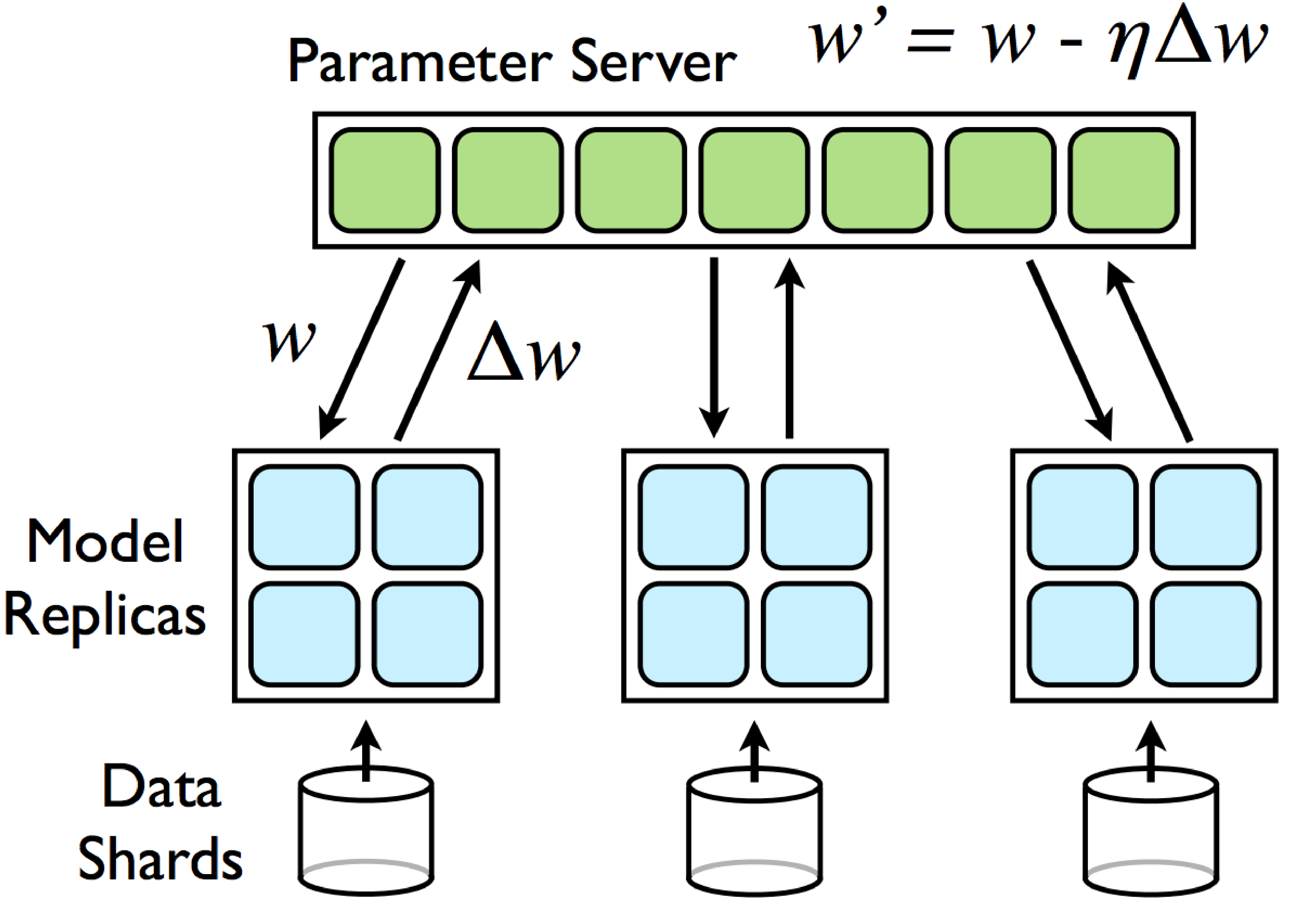
Hyperparameter	Considered values
Normalization	{standard-deviation, tanh, sqrt}
Feature type	{molecular-descriptors, tox-and-scaffold-similarities, ECFP4}
Fingerprint sparseness threshold	{5, 10, 20}
Number of Hidden Units	{1024, 4096, 8192, 16356}
Number of Layers	{1, 2, 3}
Learning Rate	{0.01, 0.05, 0.1}
Dropout	{no, yes (50% Hidden Dropout, 20% Input Dropout)}
L2 Weight Decay	$\{0, 10^{-6}, 10^{-5}, 10^{-4}\}$

Table 1. Hyperparameters considered for the neural networks. **Normalization:** Scaling of the predefined features. **Feature type:** Determines which of the features were used as input features. “molecular-descriptors” were the real-valued descriptors. “tox-and-scaffold-similarities” were the similarity scores to known toxicophores and scaffolds, “ECFP4” were the ECFP4 fingerprint features. We tested all possible combinations of these features. **Fingerprint sparseness threshold:** A feature was not used if it was only present in fewer compounds than the given number. **Number of hidden units:** The number of units in the hidden layer of the neural network. **Number of layers:** The number of layers of the neural network. **Learning rate:** The learning rate for the backpropagation algorithm. **Dropout:** Dropout rates. **L2 Weight Decay:** The weight decay hyperparameter.

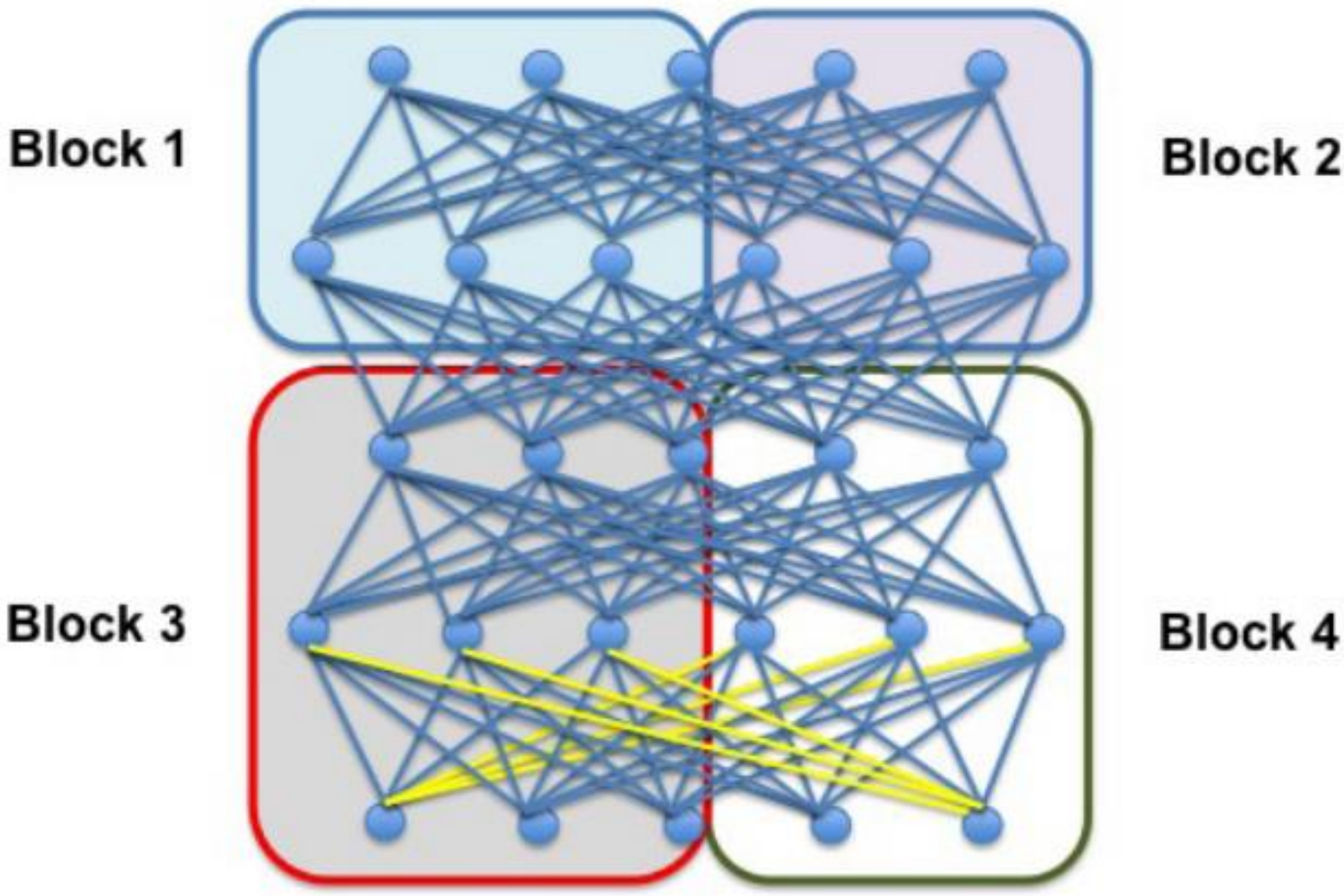
Parallelism Targets in CANDLE

10,000 x 10-1000 x 10-100 = 1M - 1000M "cores"

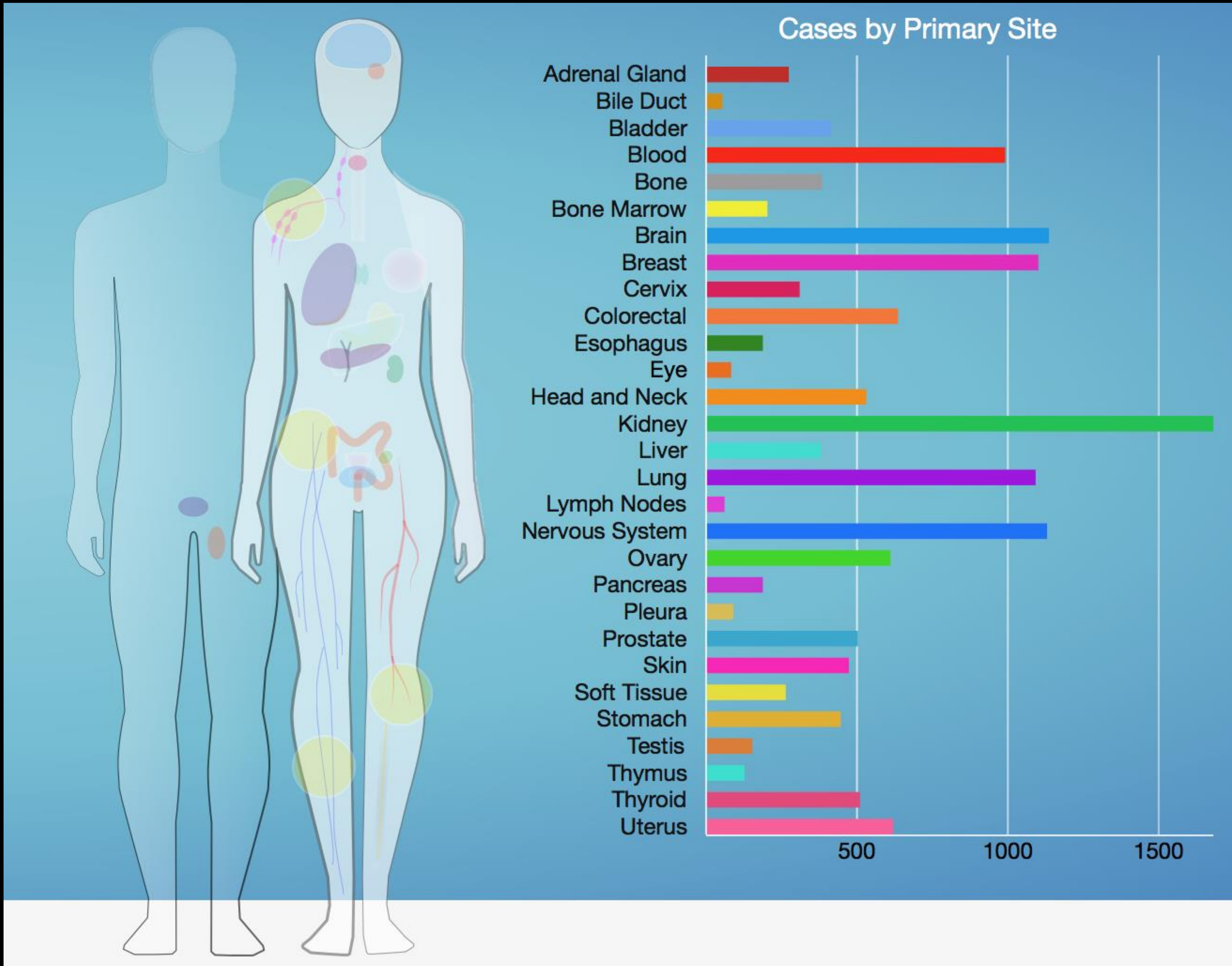


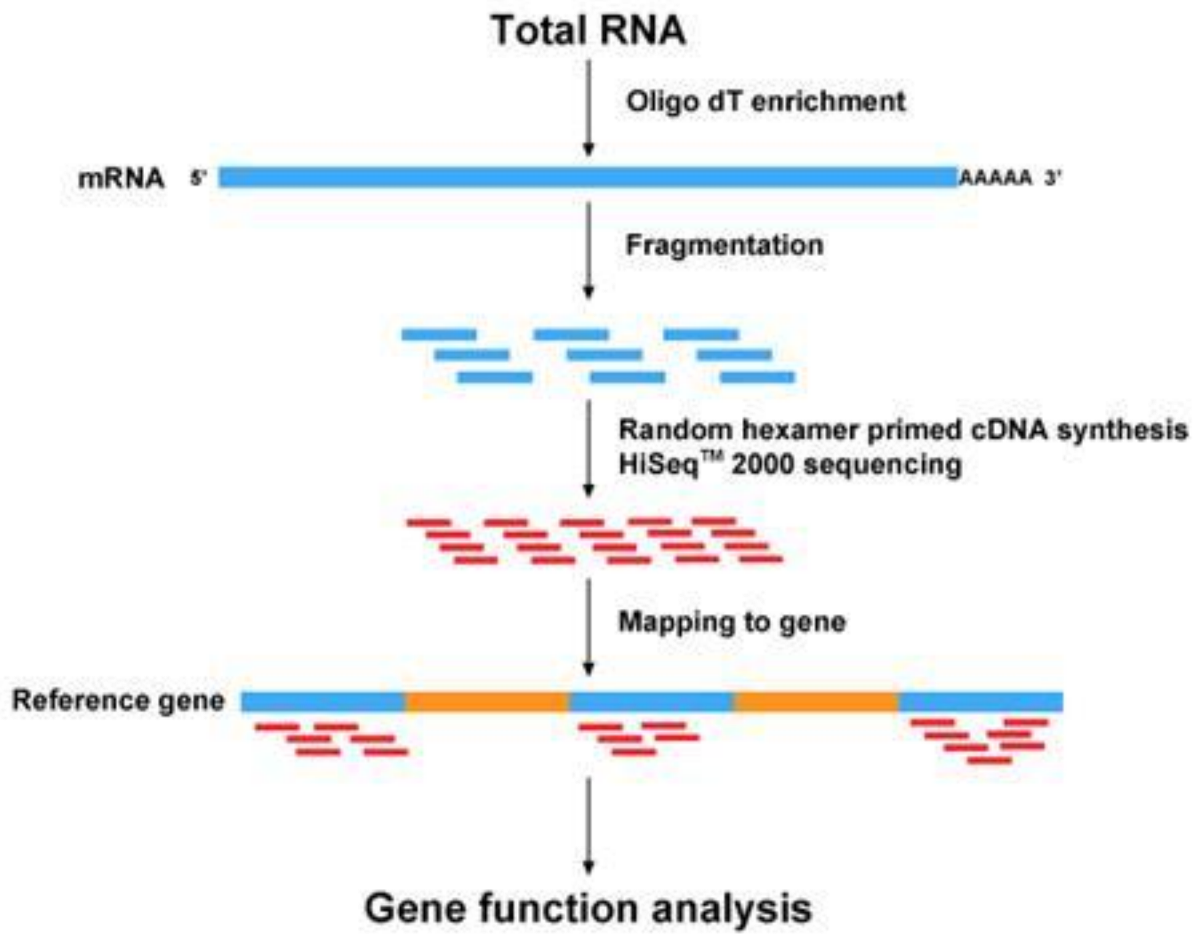

Parameter Server
w' = w - ηΔw
w
Δw
Model
Replicas
Data
Shards

Model Parallelism

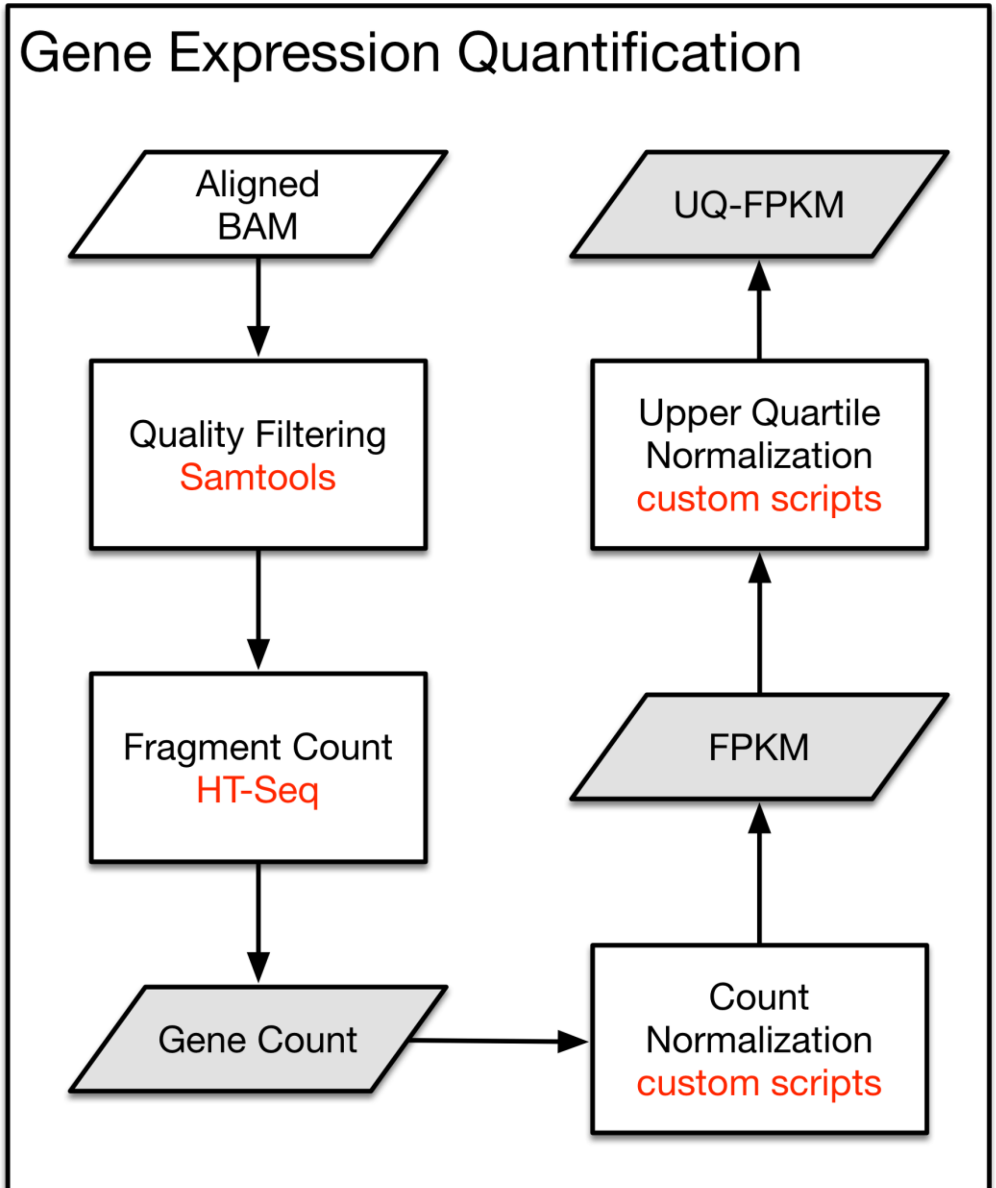


Example from Cancer: Type Classification


Cases by Primary Site
Adrenal Gland
Bile Duct
Bladder
Blood
Bone
Bone Marrow
Brain
Breast
Cervix
Colorectal
Esophagus
Eye
Head and Neck
Kidney
Liver
Lung
Lymph Nodes
Nervous System
Ovary
Pancreas
Pleura
Prostate
Skin
Soft Tissue
Stomach
Testis
Thymus
Thyroid
Uterus
500
1000
1500



Total RNA
Oligo dT enrichment
mRNA 5'
AAAAA 3'
Fragmentation
Random hexamer primed cDNA synthesis
HiSeq™ 2000 sequencing
Mapping to gene
Reference gene
Gene function analysis



RPKM (reads per kilobase per million mapped reads)
Upper Quantile (UQ)

FPKM

The Fragments per Kilobase of transcript per Million mapped reads (FPKM) calculation normalizes read count by dividing it by the gene length and the total number of reads mapped to protein-coding genes.

Upper Quartile FPKM

The upper quartile FPKM (FPKM-UQ) is a modified FPKM calculation in which the total protein-coding read count is replaced by the 75th percentile read count value for the sample.

Calculations

$$FPKM = \frac{RC_g * 10^9}{RC_{pc} * L} \qquad FPKM - UQ = \frac{RC_g * 10^9}{RC_{g75} * L}$$

- **RC_g**: Number of reads mapped to the gene
- **RC_{pc}**: Number of reads mapped to all protein-coding genes
- **RC_{g75}**: The 75th percentile read count value for genes in the sample
- **L**: Length of the gene in base pairs

Note: The read count is multiplied by a scalar (10^9) during normalization to account for the kilobase and 'million mapped reads' units.

Each Sample has > 60,000 columns

Pilot1 — root@45af2ce56389: /workspace/BMF-5/Pilot1/Combo — less -S combined_rnaseq_data — 225×61

Sample A1BG	A1CF	A2M	A2ML1	A3GALT2	A4GALT	A4GNT	AAAS	AACS	AADAC	AADACL2	AADACL3	AADACL4	AADAT	AAED1	AAGAB	AAK1	AAMDC	AAMP	AANAT	AAR2	AARD	AARS	AARS2	AARSD1	AASDH	AASDHPPT	
CCLE.22RV1	1.00	4.06	2.70	0.07	0.15	0.30	0.00	6.80	4.51	0.00	0.00	0.00	0.00	3.81	2.09	6.07	2.32	2.19	7.20	0.31	5.43	0.30	8.34	4.68	5.38	4.20	6
CCLE.2313287	0.03	3.06	0.03	0.07	0.00	1.74	0.03	5.69	3.00	0.06	0.00	0.00	0.00	0.08	3.15	6.25	1.81	3.25	6.86	0.06	5.60	0.00	8.97	4.07	5.41	3.63	6
CCLE.253J	0.82	0.00	0.08	0.01	0.00	3.35	0.53	6.44	2.68	0.06	0.00	0.00	0.04	0.06	2.74	6.80	2.17	3.22	7.05	0.29	4.86	0.00	7.60	3.75	5.53	2.80	5
CCLE.253JBV	0.36	0.00	0.16	0.01	0.06	4.14	0.48	6.33	3.32	0.04	0.01	0.00	0.00	0.01	3.10	6.61	1.55	2.95	7.52	1.12	4.98	0.06	7.82	4.19	5.78	3.30	5
CCLE.42MGBA	3.30	0.00	0.10	0.00	0.48	1.07	0.16	6.06	2.98	0.16	0.00	0.00	0.00	3.10	3.20	5.55	2.06	4.80	7.06	0.03	5.88	0.00	7.86	4.64	5.53	2.97	5
CCLE.5637	0.06	0.00	0.14	2.45	0.03	4.22	0.00	5.98	3.58	0.70	0.00	0.00	0.00	4.88	3.41	5.75	2.07	4.27	7.06	0.28	6.00	0.00	6.15	4.40	6.28	2.20	6
CCLE.59M	2.68	0.00	0.52	0.03	0.11	3.51	0.00	5.90	2.98	1.90	0.00	0.00	0.07	1.00	3.83	5.13	1.69	4.80	7.48	0.10	5.24	0.77	8.54	3.85	5.59	3.16	5
CCLE.639V	2.76	0.00	0.03	0.01	0.11	1.05	0.21	6.34	3.40	0.04	0.00	0.00	0.00	3.88	3.02	5.37	2.13	4.60	7.76	0.31	5.44	0.01	7.18	5.03	5.69	3.07	6
CCLE.647V	0.32	0.00	0.16	0.01	0.00	2.07	0.00	5.42	3.69	0.06	0.00	0.00	0.00	2.42	3.08	6.42	2.45	4.66	7.62	0.19	5.76	0.00	7.82	4.78	5.61	2.48	5
CCLE.697	3.32	0.00	0.39	0.03	0.19	0.04	0.03	6.24	1.07	0.00	0.00	0.00	0.00	0.03	0.33	5.09	2.07	4.21	6.95	0.06	5.24	0.03	8.03	4.50	5.88	3.36	5
CCLE.769P	0.03	1.53	0.32	0.01	0.10	2.77	0.37	6.02	3.26	0.14	0.00	0.00	0.00	2.43	3.76	5.65	1.71	4.17	7.21	0.29	5.09	0.07	7.90	4.33	5.72	2.69	4
CCLE.786O	0.00	0.06	0.03	0.00	0.04	2.64	0.12	5.58	3.35	0.00	0.01	0.00	0.00	3.03	4.86	6.00	1.69	3.84	6.92	0.63	5.64	0.03	8.15	3.74	4.37	3.21	6
CCLE.8305C	3.76	0.00	0.18	0.01	0.03	4.66	0.01	5.98	2.13	3.88	0.01	0.00	0.00	2.91	3.38	5.95	1.99	5.01	7.26	0.25	5.53	0.54	7.66	4.56	5.36	2.92	5
CCLE.8505C	1.48	0.04	0.20	0.03	0.00	2.69	0.14	6.19	2.72	4.70	0.00	0.00	0.00	2.28	4.03	5.54	1.97	4.76	7.05	0.23	5.88	0.00	7.19	4.77	5.70	2.67	6
CCLE.8MGBA	2.27	0.00	2.96	0.24	0.00	2.41	0.43	5.81	3.09	0.00	0.00	0.06	0.00	3.61	3.04	5.76	2.56	4.44	7.37	0.33	5.28	0.03	8.82	4.72	6.58	3.19	4
CCLE.A101D	3.01	0.00	4.92	0.06	0.34	3.31	0.07	6.10	3.87	0.01	0.01	0.00	0.01	4.08	3.45	5.20	2.11	3.36	7.82	0.06	5.83	2.32	7.55	5.72	5.27	3.09	5
CCLE.A1207	0.01	0.00	0.10	0.01	0.00	0.07	0.11	6.30	3.36	2.60	0.00	0.00	0.00	3.46	4.44	6.47	1.71	5.02	7.52	0.16	5.38	0.01	7.18	4.20	6.46	3.06	6
CCLE.A172	3.33	0.04	0.08	0.01	0.04	3.10	0.00	5.36	3.24	1.84	0.04	0.00	0.00	2.66	4.88	5.76	2.88	4.96	7.22	0.41	5.64	0.00	7.65	4.12	5.35	2.84	5
CCLE.A204	2.40	0.00	0.28	0.00	0.00	2.64	0.10	6.11	2.74	0.03	0.00	0.04	0.00	2.71	4.31	5.68	1.51	4.16	6.94	0.28	5.21	3.03	7.55	4.23	5.39	3.41	5
CCLE.A2058	2.33	0.00	2.22	0.03	0.25	0.37	0.03	6.13	2.68	0.18	0.00	0.01	0.00	5.07	2.92	5.73	1.78	3.58	7.18	0.24	5.43	0.00	7.90	4.61	5.78	3.70	6
CCLE.A253	1.02	0.01	0.07	1.78	0.38	5.37	0.00	5.67	3.25	0.60	0.97	0.10	0.00	2.78	3.33	5.87	1.96	3.53	7.08	0.46	5.38	0.00	7.32	3.73	5.44	3.43	5
CCLE.A2780	1.85	2.70	0.59	0.63	0.00	0.79	0.12	6.56	2.77	0.04	0.00	0.00	0.00	3.86	2.98	5.40	1.37	3.37	7.77	0.28	5.06	2.95	8.79	4.81	5.58	3.43	6
CCLE.A375	2.41	0.01	4.10	0.03	0.32	3.11	0.24	6.62	3.39	0.00	0.01	0.00	0.00	4.72	2.98	5.67	1.86	2.98	7.46	0.14	6.03	2.46	8.51	4.92	6.28	3.93	6
CCLE.A3KAW	2.21	0.08	0.20	0.00	0.08	0.10	0.00	6.21	3.55	0.04	0.10	0.00	0.00	3.44	0.01	4.97	3.03	2.83	6.87	0.48	4.98	0.07	8.88	4.15	5.57	3.58	5
CCLE.A427	2.47	0.00	0.15	0.01	0.00	2.71	0.43	6.89	2.50	0.19	0.00	0.00	0.00	3.12	3.92	5.38	1.46	4.61	7.19	0.11	5.10	0.36	7.21	3.88	5.85	2.93	4
CCLE.A498	0.45	2.42	0.10	0.01	0.00	4.51	0.19	5.40	2.79	2.50	0.01	0.00	0.26	2.65	3.88	5.16	1.83	3.66	7.44	0.28	5.34	1.24	8.31	3.64	5.09	2.87	4
CCLE.A4FUK	0.25	0.00	0.82	0.00	0.07	0.06	0.00	7.29	3.69	0.00	0.03	0.00	0.10	0.11	0.62	5.07	2.27	2.10	7.66	0.61	5.59	0.18	8.29	5.13	6.26	3.62	6
CCLE.A549	0.77	0.42	0.01	0.01	0.00	2.53	0.51	6.24	2.70	4.73	0.01	0.00	0.00	3.64	3.89	6.64	1.56	3.42	6.91	0.21	5.63	0.04	8.07	3.30	5.26	3.02	6
CCLE.A673	2.78	0.00	2.53	3.44	0.19	4.62	0.36	6.98	3.33	0.16	0.11	0.00	1.74	3.54	2.43	5.33	1.23	5.13	7.10	0.19	5.38	0.18	8.60	4.64	4.57	2.58	6
CCLE.A704	0.44	0.31	0.18	0.01	0.15	3.37	3.24	5.73	3.09	0.15	0.00	0.00	0.00	1.55	3.86	4.79	2.38	5.07	6.02	0.75	3.37	0.00	4.25	3.25	4.58	2.92	5
CCLE.ABC1	0.08	0.00	0.08	1.09	0.00	3.74	0.04	5.66	2.74	0.12	0.00	0.00	0.00	3.04	2.38	5.72	2.30	3.88	7.37	0.08	4.94	1.97	8.11	4.55	5.66	2.38	6
CCLE.ACCMESO1	3.92	0.01	1.38	0.01	0.03	4.07	0.06	5.01	2.29	2.47	0.00	0.00	0.00	2.19	4.90	5.90	1.53	5.08	6.82	0.19	4.97	0.00	5.97	3.58	5.26	2.68	5
CCLE.ACHN	0.37	0.04	0.07	0.04	0.00	3.55	1.96	6.10	3.74	0.04	0.00	0.00	0.06	3.55	4.19	6.14	2.24	2.75	7.28	0.18	4.38	0.00	9.43	3.67	5.47	3.27	5
CCLE.AGS	0.06	0.04	0.04	0.03	0.07	0.01	0.00	5.45	3.79	0.42	0.00	0.00	0.00	3.97	4.17	6.24	1.95	4.01	7.49	0.06	6.04	0.01	7.31	5.03	4.63	3.48	6
CCLE.ALLSIL	3.18	0.00	0.08	0.00	0.07	0.03	0.04	6.45	1.78	0.03	0.00	0.00	0.00	0.03	0.00	6.20	2.56	4.77	7.24	0.14	5.39	2.18	7.30	5.33	6.01	3.45	6
CCLE.AM38	2.35	0.00	0.08	0.07	0.08	0.01	0.06	6.64	3.86	4.17	0.00	0.01	0.00	4.01	4.10	5.54	2.13	4.85	7.47	0.12	5.09	0.04	7.89	4.58	6.15	3.35	7
CCLE.AML193	1.51	0.03	0.16	0.19	0.00	0.16	0.06	5.83	3.26	0.08	0.00	0.06	0.00	0.15	0.20	5.50	0.98	4.50	7.13	0.07	5.37	0.01	6.28	4.77	6.08	2.95	5
CCLE.AMO1	2.36	0.00	0.08	0.00	0.00	0.42	0.00	6.28	3.01	0.03	0.00	0.00	0.00	0.04	1.80	4.90	2.01	3.32	7.59	1.42	5.42	0.14	9.48	4.29	6.29	2.94	7
CCLE.AN3CA	2.73	0.06	0.12	0.04	0.16	0.14	0.03	6.20	3.25	0.04	0.00	0.00	0.00	0.10	3.25	6.04	2.31	3.78	7.29	0.14	5.39	0.00	7.84	5.48	5.67	3.08	5
CCLE.ASPC1	0.10	4.54	0.00	0.03	0.07	3.61	0.00	5.74	2.92	4.04	0.03	0.00	0.00	0.20	3.07	5.44	1.85	4.48	7.50	0.08	5.09	0.00	7.85	2.54	3.94	2.81	5
CCLE.AU565	1.28	0.01	1.88	2.98	0.06	0.88	0.00	6.32	4.35	0.50	0.00	0.00	0.00	2.24	2.47	5.03	1.88	2.74	8.04	0.14	6.06	3.47	6.51	4.36	4.70	3.46	5
CCLE.BC3C	0.38	0.00	0.11	0.00	0.00	4.67	0.04	6.22	3.24	3.72	0.18	0.00	0.00	2.44	4.05	5.40	2.07	3.93	6.87	0.41	5.75	0.03	8.25	4.36	5.55	2.94	5
CCLE.BCP1	0.23	0.11	0.06	0.03	0.00	0.04	0.00	6.18	2.58	0.08	0.00	0.00	0.00	0.01	3.18	6.22	1.68	2.54	8.15	1.72	5.42	0.20	8.30	4.62	5.36	3.89	6
CCLE.BCPAP	3.05	0.03	0.16	0.03	0.00	5.18	0.19	6.41	2.75	2.72	0.01	0.00	0.00	3.61	2.23	5.32	1.74	4.03	7.10	0.07	6.35	0.25	8.70	4.23	5.26	3.79	5
CCLE.BDCM	1.94	0.00	0.32	0.01	0.00	1.23	0.00	6.12	3.54	0.00	0.00	0.00	0.00	0.14	1.80	5.42	1.97	2.62	7.03	0.32	4.89	0.03	7.70	4.64	6.06	3.65	5
CCLE.BEN	0.04	3.62	0.19	0.03	0.08	2.08	0.03	5.82	4.80	0.61	0.10	0.00	0.00	2.91	3.13	5.60	1.11	3.90	7.28	0.10	5.16	0.21	8.57	3.75	4.40	3.10	5
CCLE.BFTC905	0.86	0.00	0.06	1.41	0.08	4.94	0.00	6.09	3.34	3.55	0.16	0.04	0.00	3.17	2.58	5.97	1.64	2.92	7.43	0.71	5.28	0.14	8.44	4.26	5.32	3.94	5
CCLE.BFTC909	2.26	0.00	0.11	0.03	0.07	3.63	0.41	6.25	3.43	0.21	0.00	0.00	0.00	2.68	3.41	5.44	1.63	4.76	6.90	0.18	4.87	0.10	8.70	3.87	5.43	2.79	3
CCLE.BHT101	1.69	4.39	0.08	0.03	0.03	3.34	0.91	5.77	3.48	3.62	0.01	0.00	0.00	1.22	3.43	4.84	1.60	3.00	7.24	1.44	4.72	0.06	8.78	3.97	5.23	3.82	5
CCLE.BHY	1.36	0.00	0.42	3.52	0.00	4.37	0.77	5.69	3.61	3.91	0.19	0.00	0.00	2.32	3.47	5.68	2.59	5.16	6.51	1.66	5.21	0.29	7.66	4.62	4.99	3.69	5
CCLE.BICR16	0.98	0.01	0.08	5.12	0.00	4.27	0.00	5.05	3.43	0.08	0.00	0.00	0.00	1.70	3.33	5.41	1.95	4.53	6.23	0.52	4.75	0.08	8.55	3.37	4.14	3.17	4
CCLE.BICR18	0.51	0.08	0.08	5.47	0.00	6.60	0.06	5.67	3.11	0.80	0.52	0.03	0.00	2.65	2.49	6.93	1.76	3.97	7.00	0.66	4.85	0.63	8.47	4.73	5.51	3.52	4
CCLE.BICR22	1.38	0.01	0.12	7.36	0.00	4.29	0.01	3.87	3.10	0.03	0.01	0.00	0.00	0.98	3.02	4.88	1.51	2.49	6.21	0.97	3.96	0.04	7.25	2.63	3.04	2.92	3
CCLE.BICR31	2.69	0.06	0.19	4.48	0.12	6.01	0.01	5.92	3.58	1.04	0.04	0.00	0.00	1.10	3.71	6.00	1.99	5.13	7.46	0.32	5.68	1.14	8.17	3.92	5.26	3.41	4
CCLE.BICR56	0.60	0.01	0.07	4.72	0.00	4.61	0.01	3.75	2.52	0.14	0.01	0.00	0.00	0.34	1.95	4.89	2.05	3.70	6.44	0.45	4.04	0.10	7.62	2.90	3.07	2.72	4
CCLE.BICR6	0.03	0.01	0.10	6.10	0.00	4.28	0.00	4.27	1.66	0.07	0.00	0.00	0.00	0.86	3.18	4.67	0.97	2.45	5.98	0.30	4.18	0.11	6.60	2.16	3.70	2.21	3
CCLE.BL41	1.79	0.04	0.07	0.08	0.00	2.34	0.00	6.82	2.68	0.03	0.00	0.00	0.00	0.04	0.11	6.30	0.24	2.58	6.94	1.58	5.29	0.06	8.07	4.51	6.37	2.17	5
CCLE.BL70	2.12	0.01	0.10	0.03	0.00	1.37	0.00	5.54	3.19	0.00	0.00	0.00	0.00	0.03	0.12	5.47	0.18	3.15	7.14	0.46	5.15	0.24	7.89	4.61	5.79	3.37	5
CCLE.BT-12	1.60	0.00	0.08	0.00	0.00	2.72	0.15	6.34	1.84	4.52	0.16	0.00	0.00	0.04	2.90	5.54	0.96	2.97	7.31	0.63	4.94	0.00	7.59	4.03	5.71	3.19	6

combined_rnaseq_data

One Hot Encoding of Categories

State	Binary	One-Hot	Hamming 2	Hamming 3
S0	000	00000001	0000	000000
S1	001	00000010	0011	000111
S2	010	00000100	0101	011001
S3	011	00001000	0110	011110
S4	100	00010000	1001	101010
S5	101	00100000	1010	101101
S6	110	01000000	1100	110011
S7	111	10000000	1111	110100

Open Source Framework Comparison

	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	+
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	+

Keras

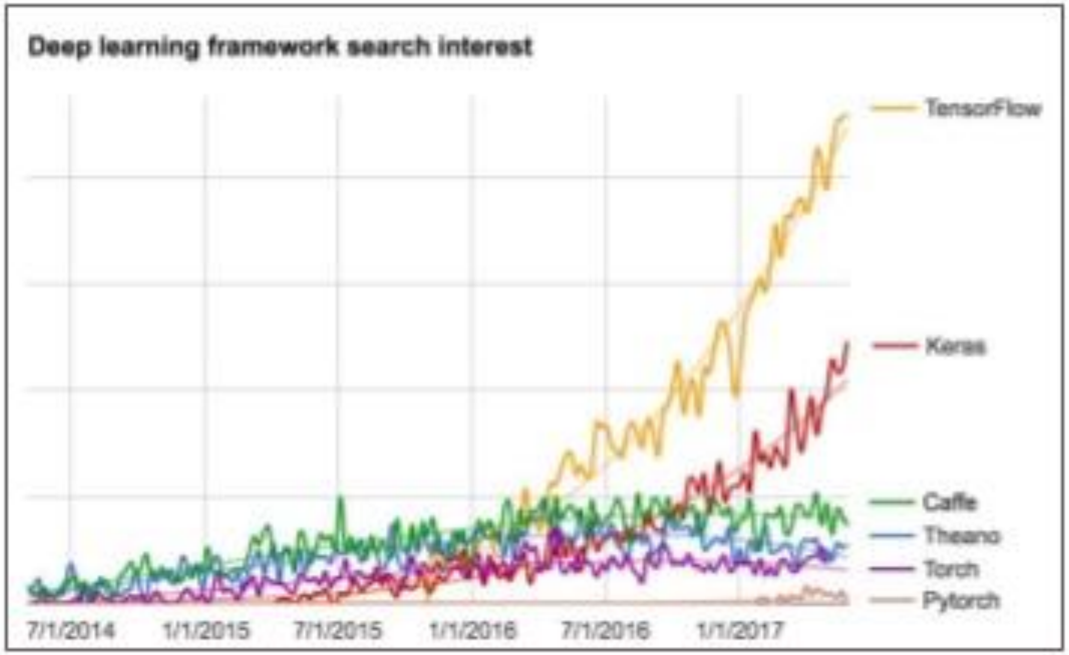
- https://keras.io/
- Minimalist, highly modular neural networks library
- Written in Python
- Capable of running on top of either TensorFlow/Theano and CNTK
- Developed with a focus on enabling fast experimentation

```python
from keras.layers import Input, Dense
from keras.models import Model

input_layer = Input(shape=(1000,))
fc_1 = Dense(512, activation='relu')(input_layer)
fc_2 = Dense(256, activation='relu')(fc_1)
output_layer = Dense(10, activation='softmax')(fc_2)

model = Model(input=input_layer, output=output_layer)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(bow, newsgroups.target)
predictions = model.predict(features).argmax(axis=1)
```

Keras

- Keras is the *de facto* deep learning frontend

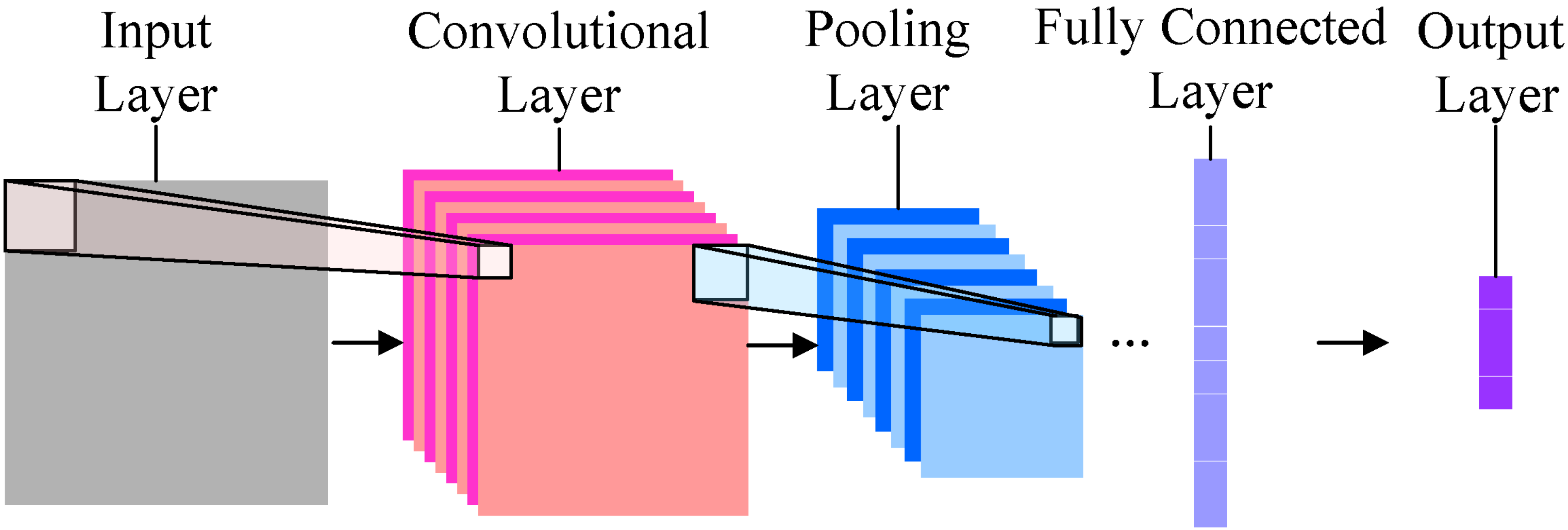
Source: @fchollet, Jun 3 2017

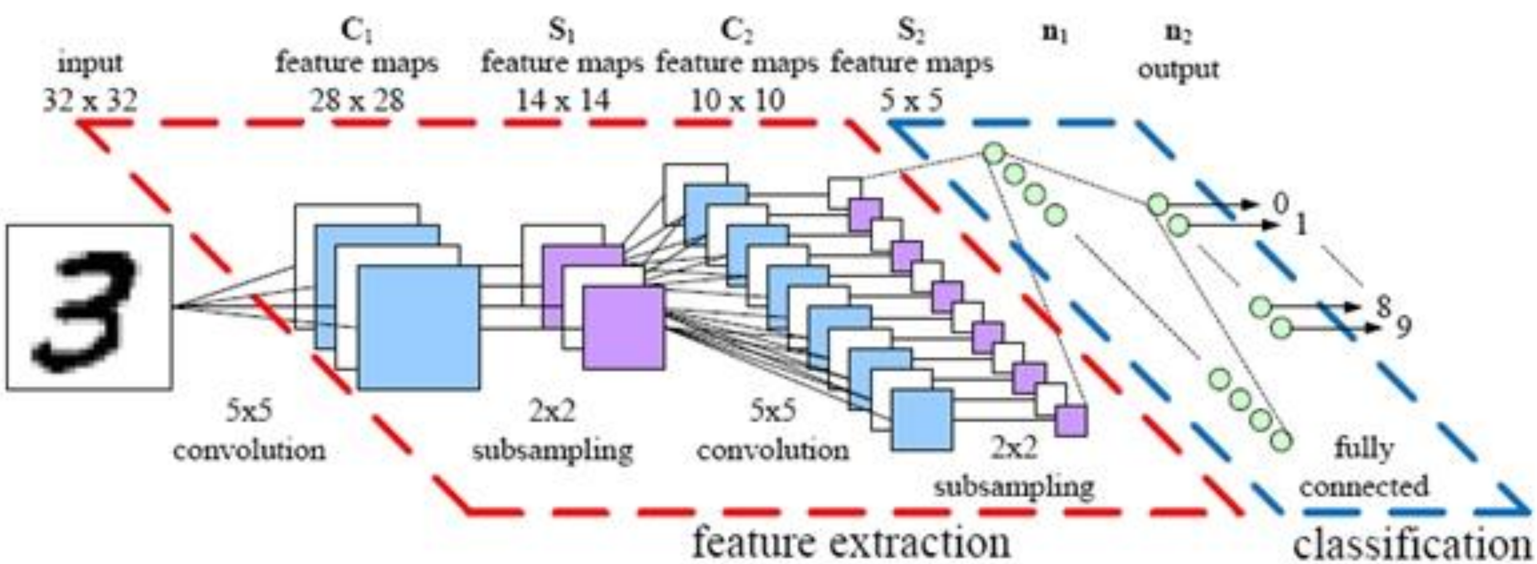


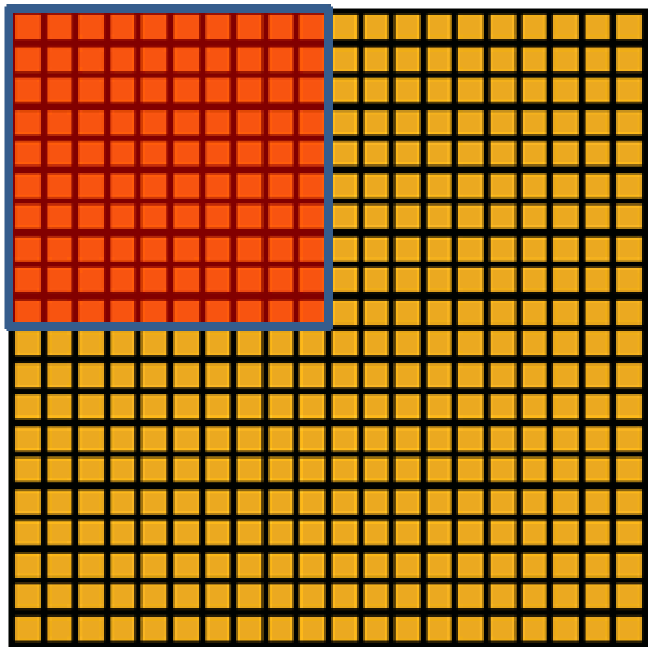
Neural Network for Classification (TC1)

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 60464, 128)	2688
activation_1 (Activation)	(None, 60464, 128)	0
max_pooling1d_1 (MaxPooling1	(None, 60464, 128)	0
conv1d_2 (Conv1D)	(None, 60455, 128)	163968
activation_2 (Activation)	(None, 60455, 128)	0
max_pooling1d_2 (MaxPooling1	(None, 6045, 128)	0
flatten_1 (Flatten)	(None, 773760)	0
dense_1 (Dense)	(None, 200)	154752200
activation_3 (Activation)	(None, 200)	0
dropout_1 (Dropout)	(None, 200)	0
dense_2 (Dense)	(None, 20)	4020
activation_4 (Activation)	(None, 20)	0
dropout_2 (Dropout)	(None, 20)	0
dense_3 (Dense)	(None, 36)	756
activation_5 (Activation)	(None, 36)	0

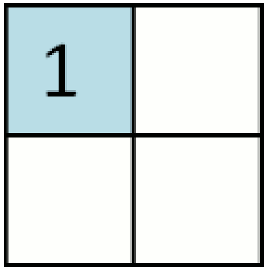
Total params: 154,923,632
Trainable params: 154,923,632
Non-trainable params: 0


Input Layer
Convolutional Layer
Pooling Layer
Fully Connected Layer
Output Layer
...


input
32 x 32
C_1
feature maps
28 x 28
S_1
feature maps
14 x 14
C_2
feature maps
10 x 10
S_2
feature maps
5 x 5
n_1
n_2
output
0
1
8
9
5x5
convolution
2x2
subsampling
5x5
convolution
2x2
subsampling
fully
connected
feature extraction
classification



Convolved feature



Pooled feature

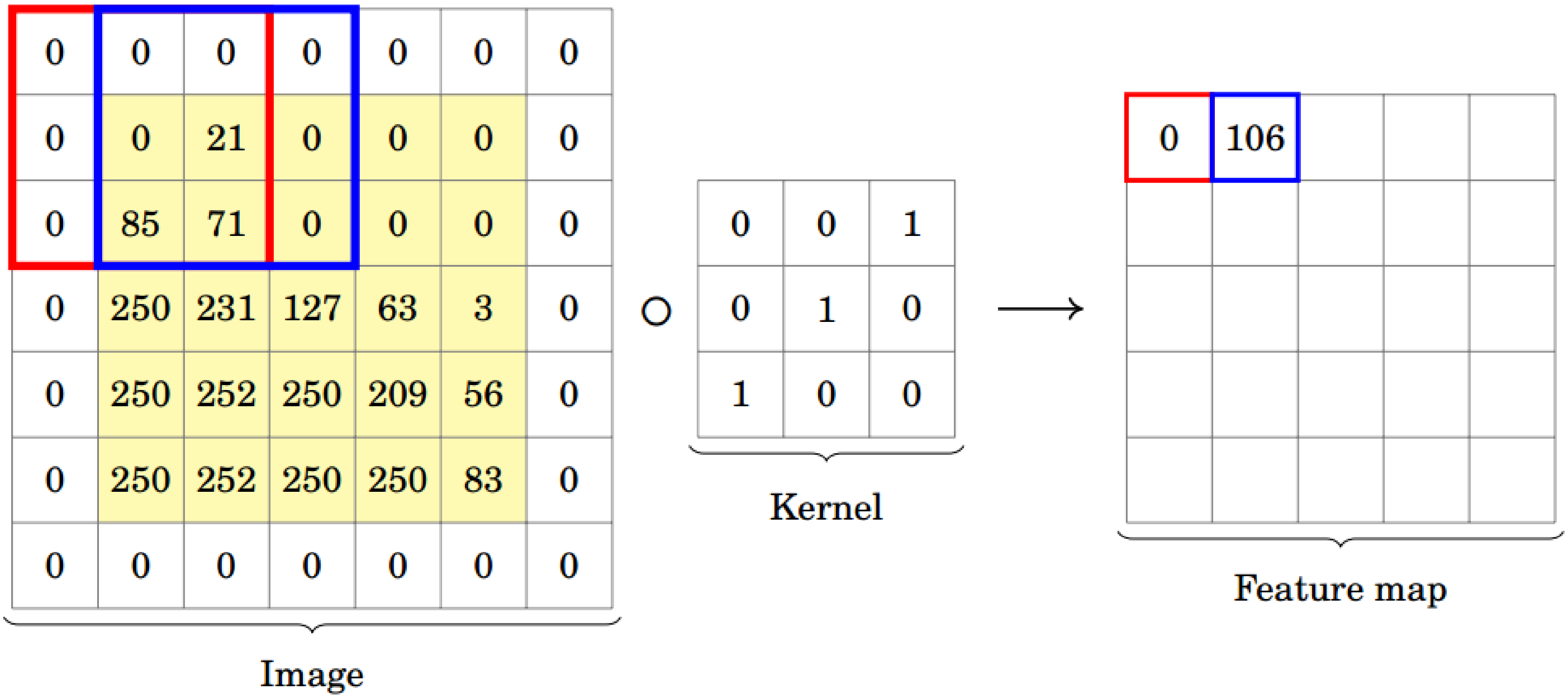
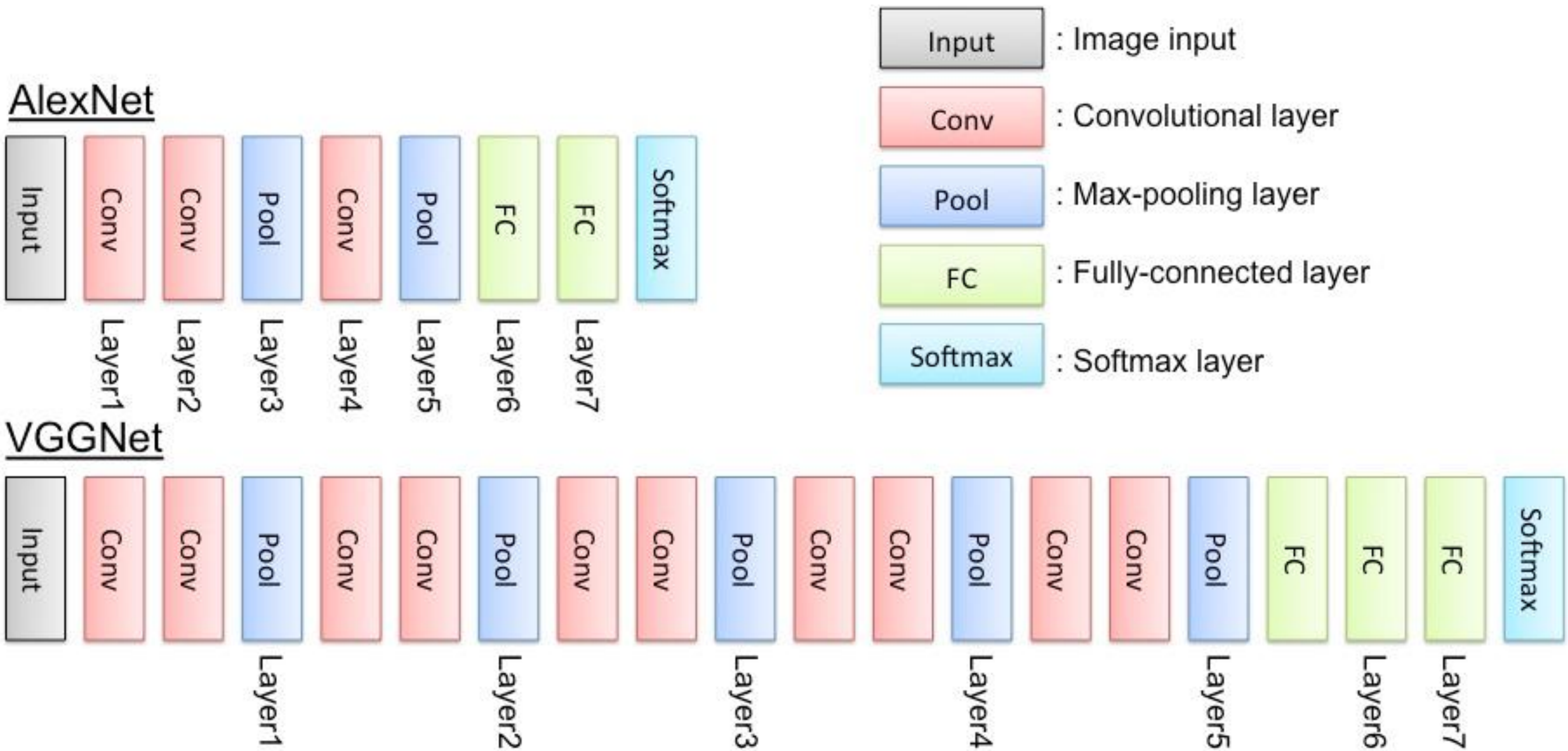
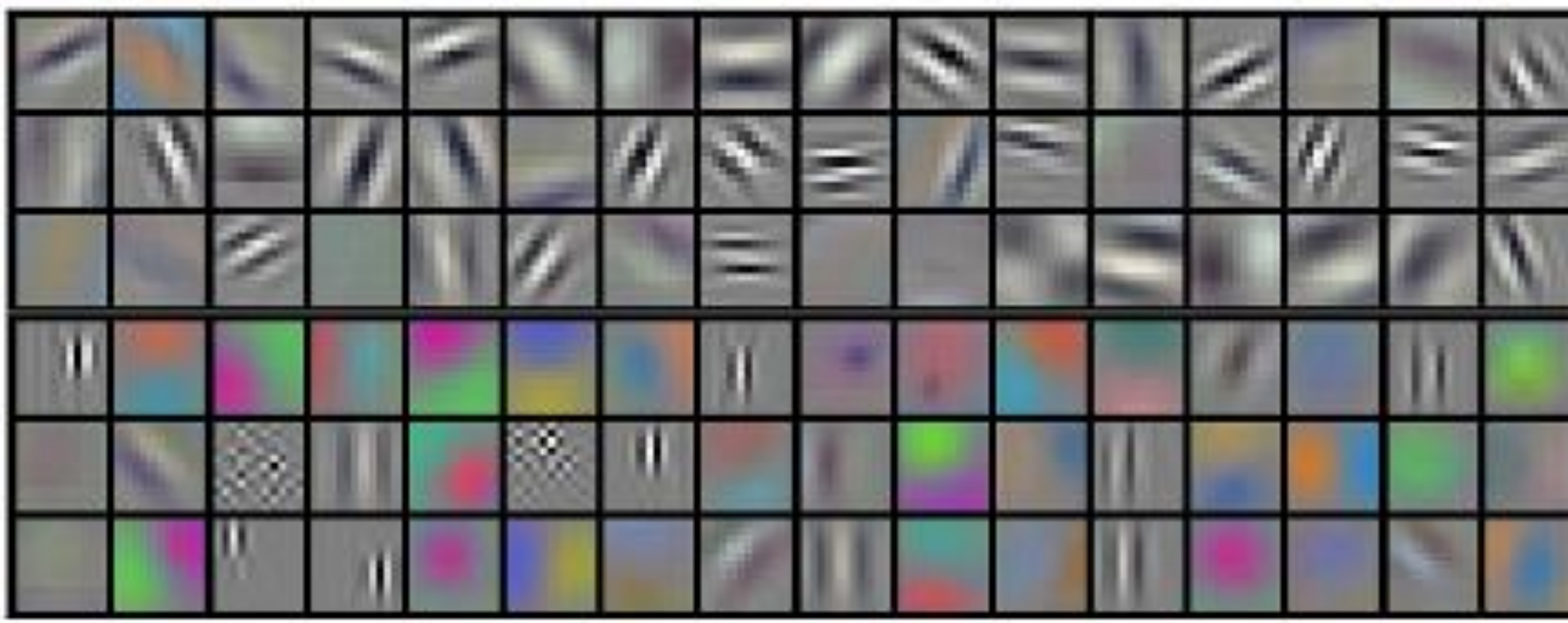


Image
Kernel
Feature map
0
106

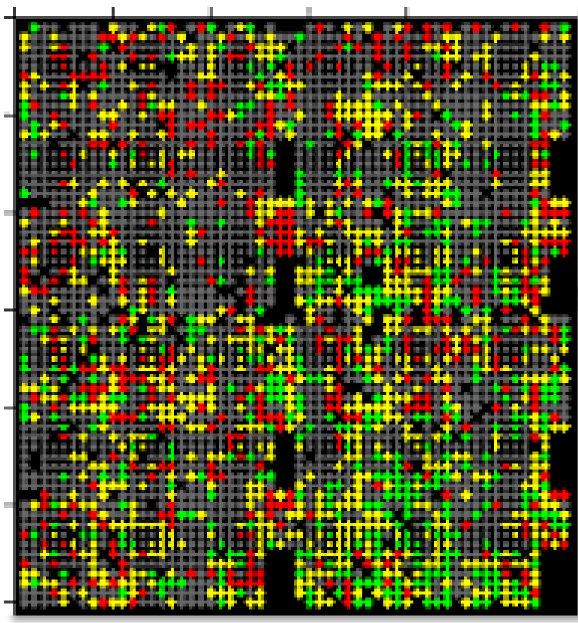
Networks Used to Classify Images

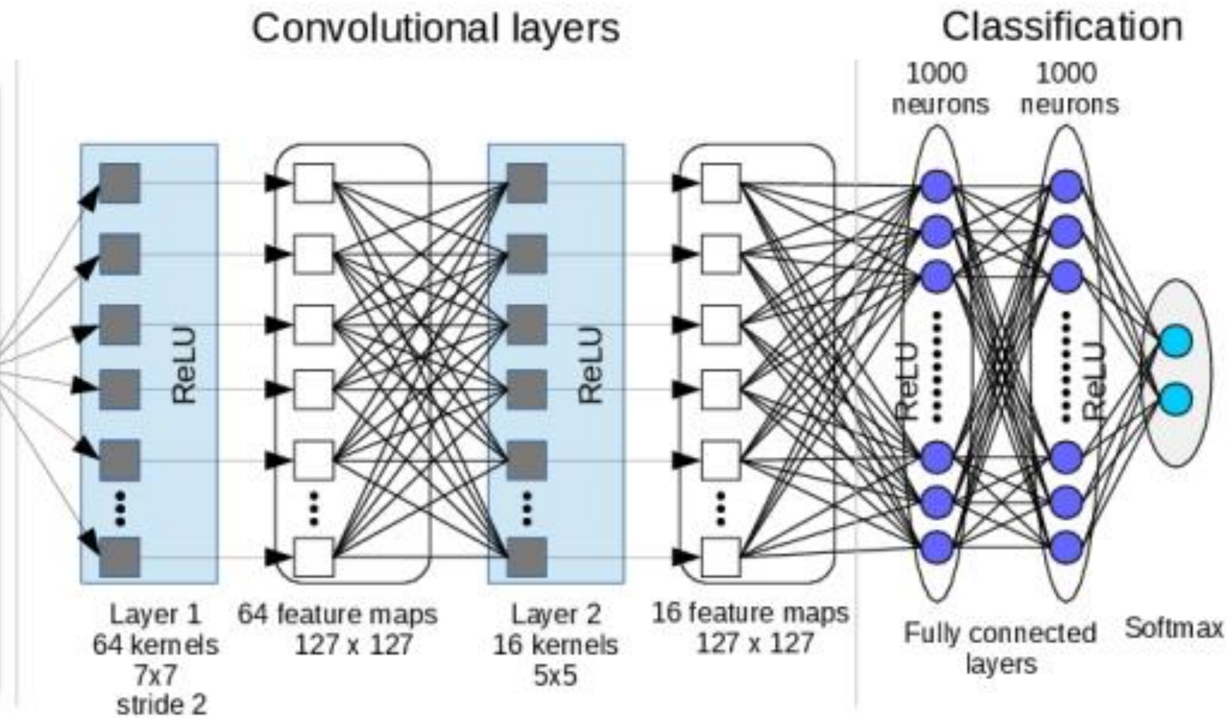


Example Features Learned in 2D Convolution Lower Layers



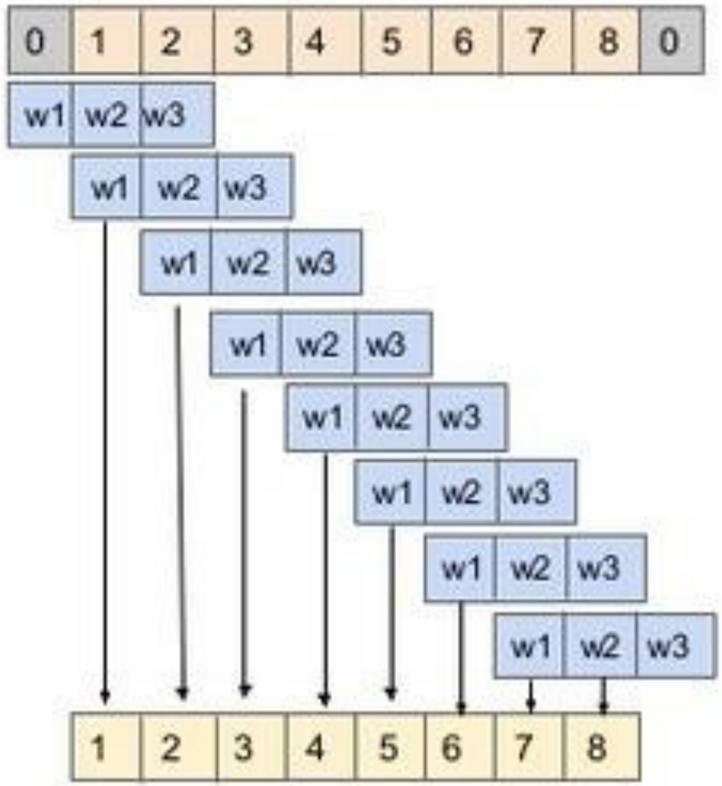
Convolution vs
Fullly (Dense) Connected Layers





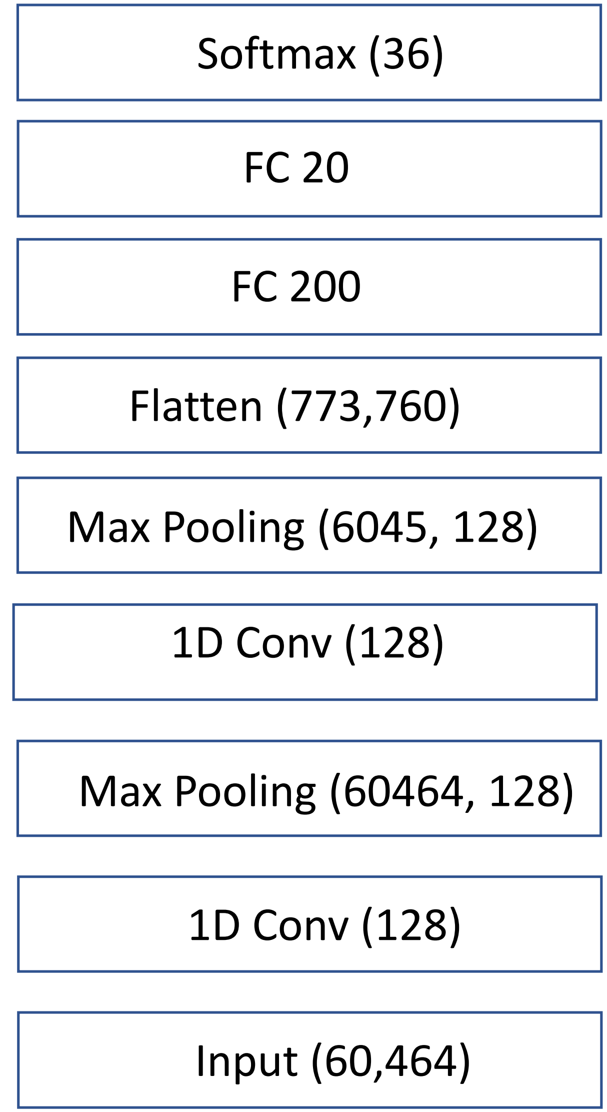
1D Convolutions

When we add zero padding, we normally do so on both sides of the sequence (as in image padding)



Neural Network for Classification

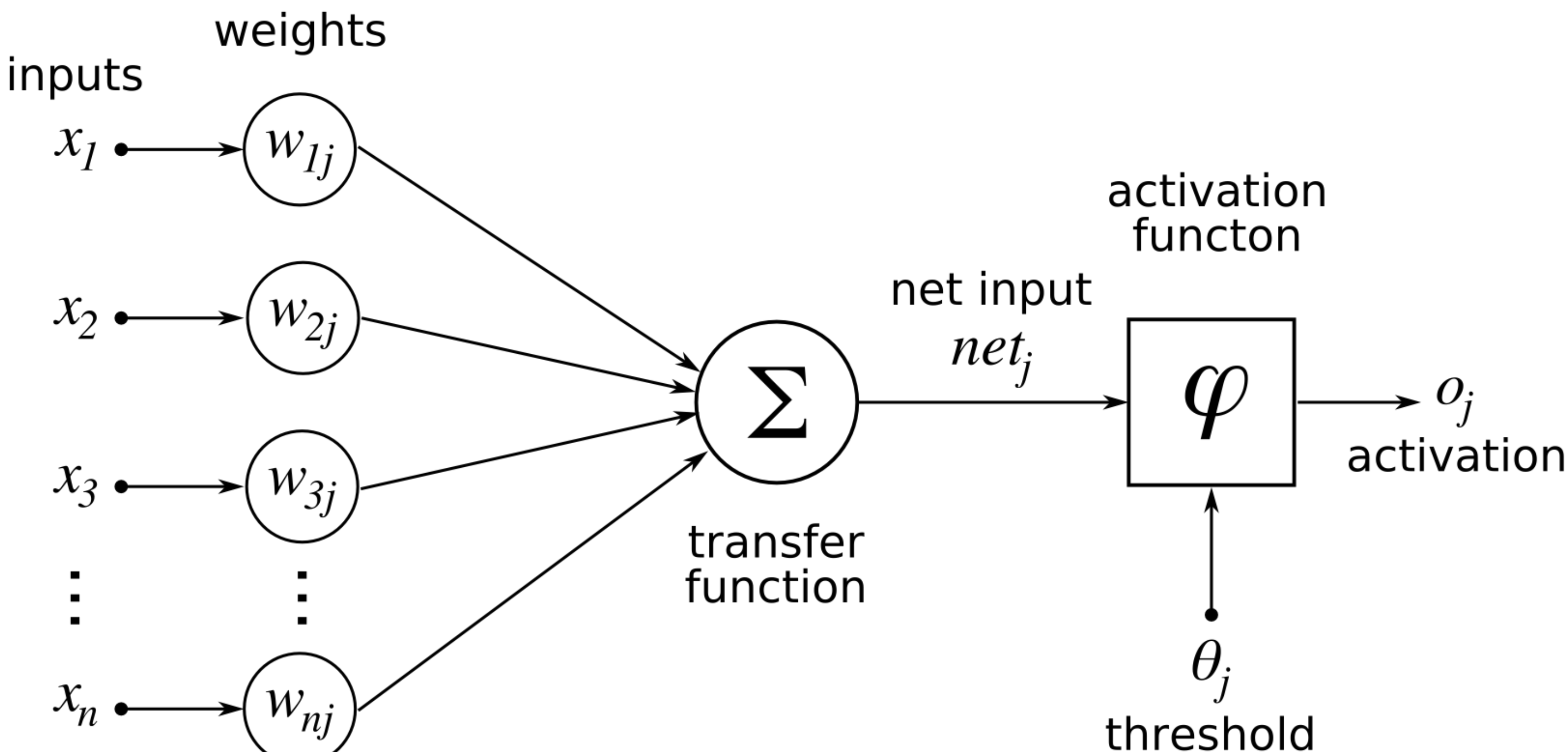
```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 60464, 128)        2688
_________________________________________________________________
activation_1 (Activation)    (None, 60464, 128)        0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 60464, 128)        0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 60455, 128)        163968
_________________________________________________________________
activation_2 (Activation)    (None, 60455, 128)        0
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 6045, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 773760)            0
_________________________________________________________________
dense_1 (Dense)              (None, 200)               154752200
_________________________________________________________________
activation_3 (Activation)    (None, 200)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_2 (Dense)              (None, 20)                4020
_________________________________________________________________
activation_4 (Activation)    (None, 20)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_3 (Dense)              (None, 36)                756
_________________________________________________________________
activation_5 (Activation)    (None, 36)                0
=================================================================
Total params: 154,923,632
Trainable params: 154,923,632
Non-trainable params: 0
_________________________________________________________________
```

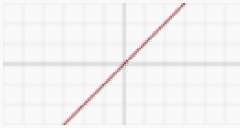
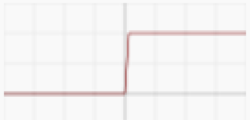
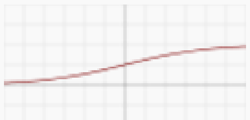
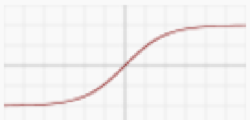
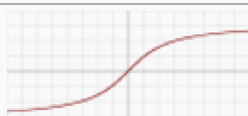
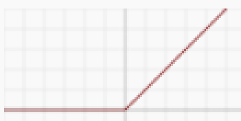
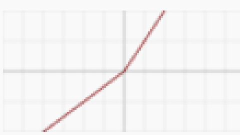
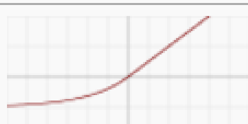
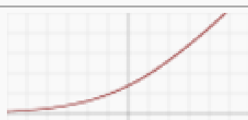


Setting up the Graph Structure

```
model = Sequential()
model.add(Conv1D(filters=128, kernel_size=20, strides=1, padding='valid', input_shape=(P, 1)))
model.add(Activation('relu'))
model.add(MaxPooling1D(pool_size=1))
model.add(Conv1D(filters=128, kernel_size=10, strides=1, padding='valid'))
model.add(Activation('relu'))
model.add(MaxPooling1D(pool_size=10))
model.add(Flatten())
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(Dense(20))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(Dense(CLASSES))
model.add(Activation('softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=SGD(),
              metrics=['accuracy'])
```

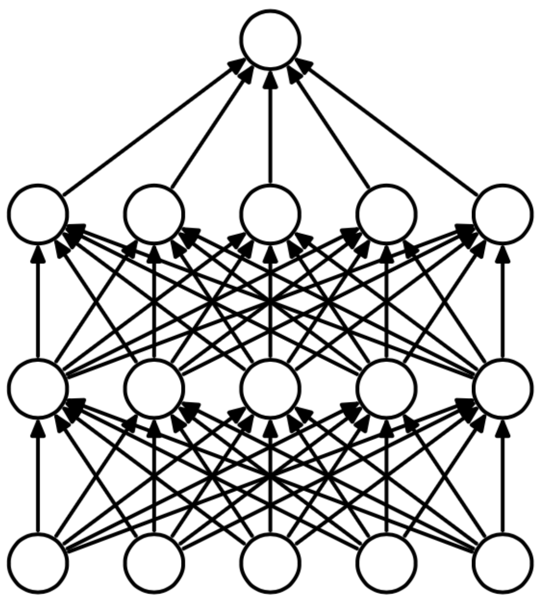
What Activation Function to Use?



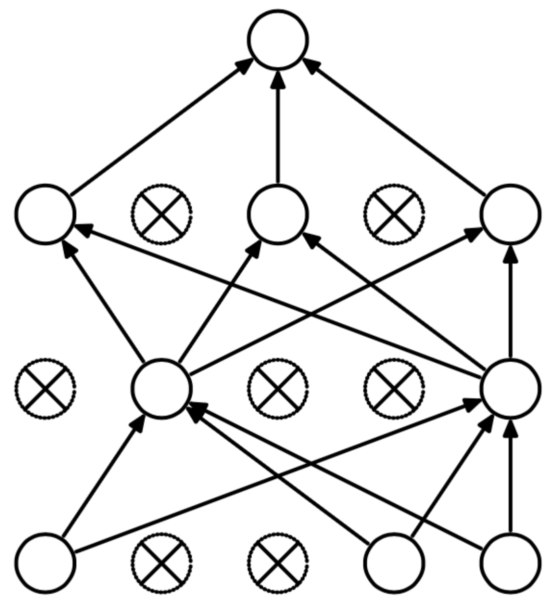
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \ne 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Dropout!

Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov



(a) Standard Neural Net



(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

What Loss Function to use?
What Optimizer to use?

```
model.compile(loss='categorical_crossentropy',
              optimizer=SGD(),
              metrics=['accuracy'])


# set up a bunch of callbacks to do work during model training..

checkpointer = ModelCheckpoint(filepath='nt3.autosave.model.h5', verbose=1, save_weights_only=False, save_best_only=True)
csv_logger = CSVLogger('training.log')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=10, verbose=1, mode='auto', epsilon=0.0001, cooldown=0, min_lr=0)

history = model.fit(X_train, Y_train,
                    batch_size=BATCH,
                    epochs=EPOCH,
                    verbose=1,
                    validation_data=(X_test, Y_test),
                    callbacks = [checkpointer, csv_logger, reduce_lr])

score = model.evaluate(X_test, Y_test, verbose=0)
```

Example Loss functions

Regression:

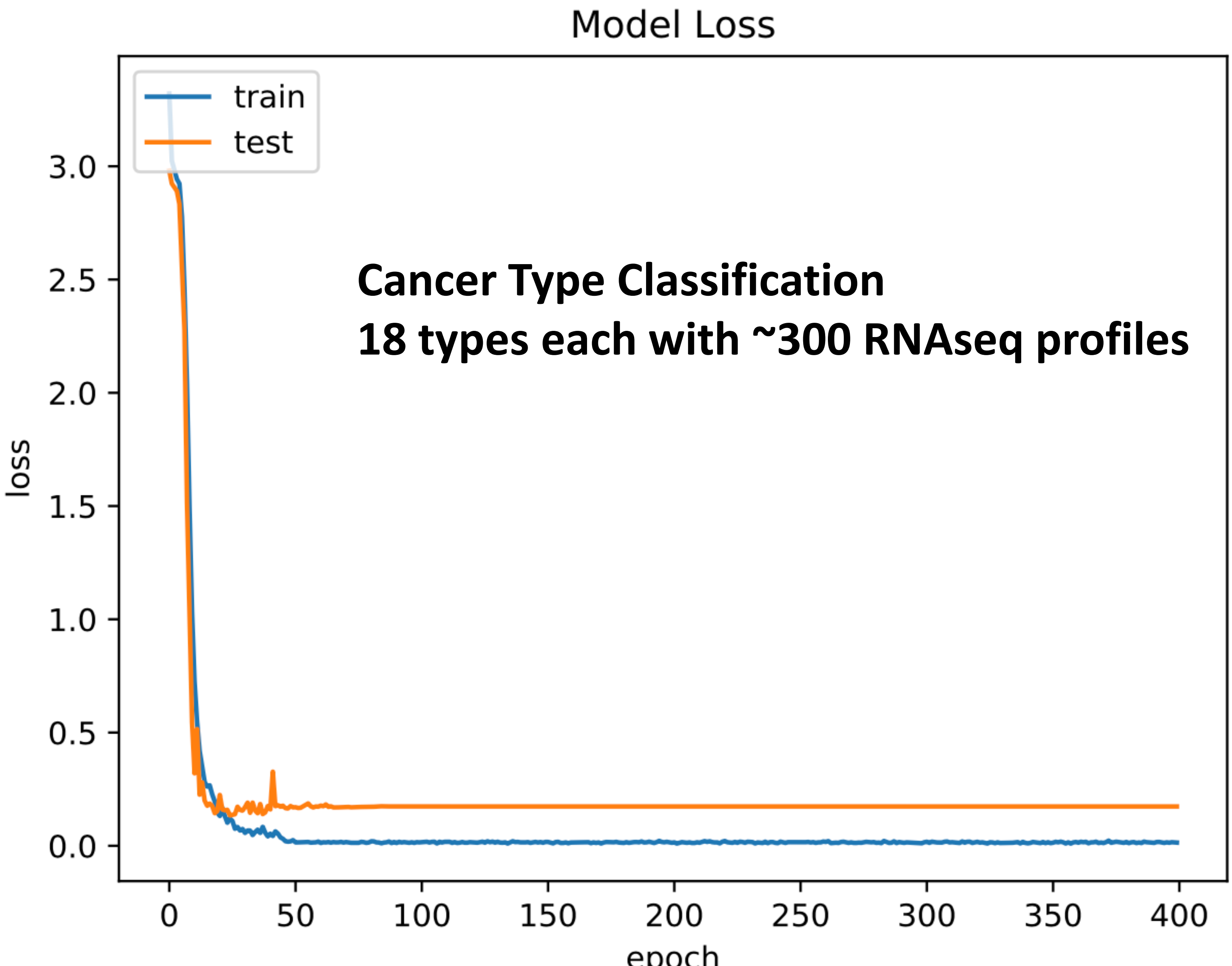
$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

Classification: cross-entropy (deviance)

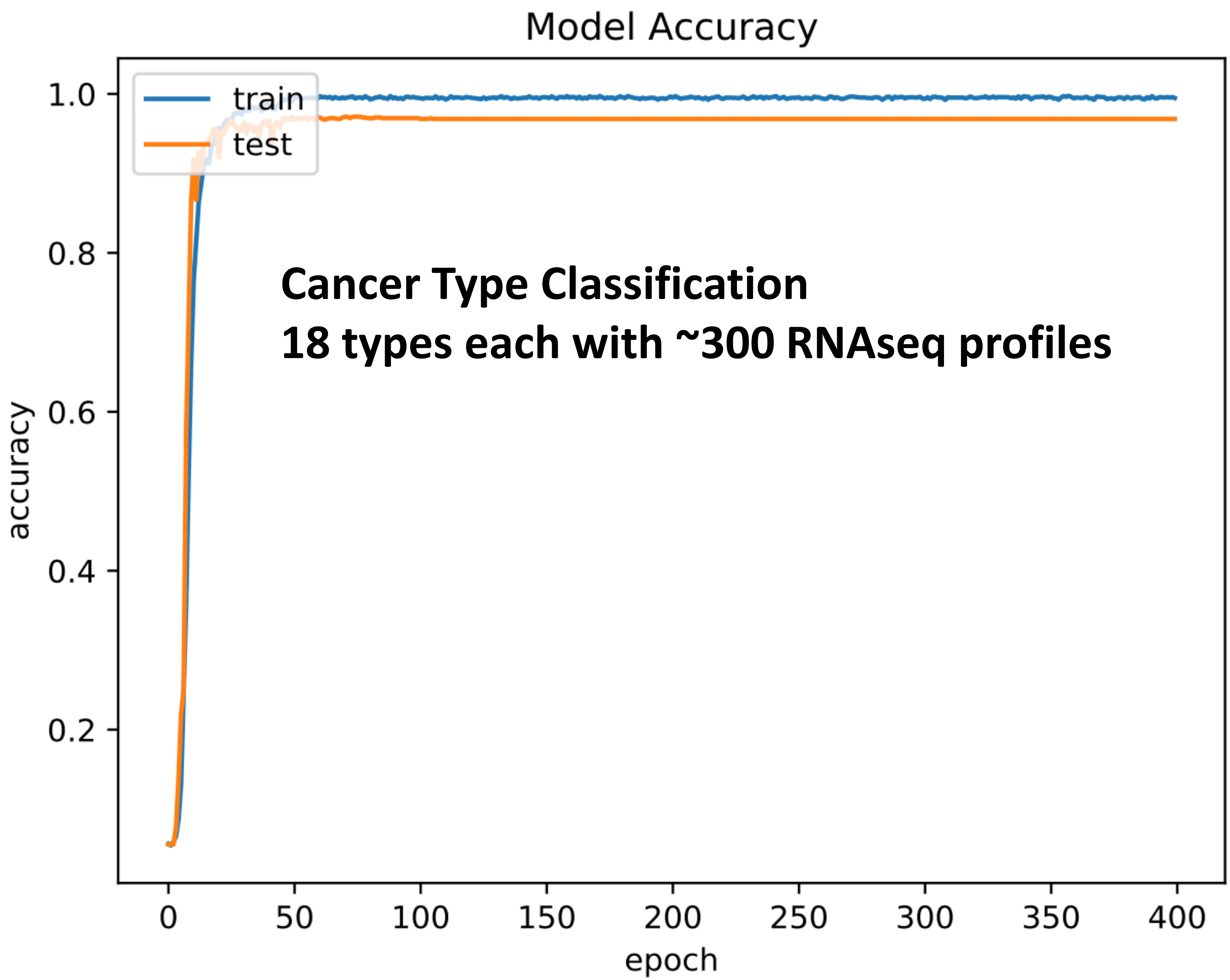
$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i).$$

Cancer Type Classification

```
4320/4320 [==============================] - 87s - loss: 3.2885 - acc: 0.0537 - val_loss: 2.9542 - val_acc: 0.0556
Epoch 2/400
4320/4320 [==============================] - 76s - loss: 2.9777 - acc: 0.0752 - val_loss: 2.8273 - val_acc: 0.1083
Epoch 3/400
4320/4320 [==============================] - 78s - loss: 2.8117 - acc: 0.1176 - val_loss: 2.5971 - val_acc: 0.2194
Epoch 4/400
4320/4320 [==============================] - 77s - loss: 2.5094 - acc: 0.2060 - val_loss: 2.1191 - val_acc: 0.3306
Epoch 5/400
4320/4320 [==============================] - 78s - loss: 2.0385 - acc: 0.3442 - val_loss: 1.6411 - val_acc: 0.4648
Epoch 6/400
4320/4320 [==============================] - 75s - loss: 1.4995 - acc: 0.5079 - val_loss: 0.9846 - val_acc: 0.7704
Epoch 7/400
4320/4320 [==============================] - 77s - loss: 1.0688 - acc: 0.6481 - val_loss: 0.5628 - val_acc: 0.8796
Epoch 8/400
4320/4320 [==============================] - 76s - loss: 0.7657 - acc: 0.7461 - val_loss: 0.4952 - val_acc: 0.8509
Epoch 9/400
4320/4320 [==============================] - 76s - loss: 0.5729 - acc: 0.8123 - val_loss: 0.2803 - val_acc: 0.9287
Epoch 10/400
4320/4320 [==============================] - 79s - loss: 0.4389 - acc: 0.8620 - val_loss: 0.1962 - val_acc: 0.9398
```

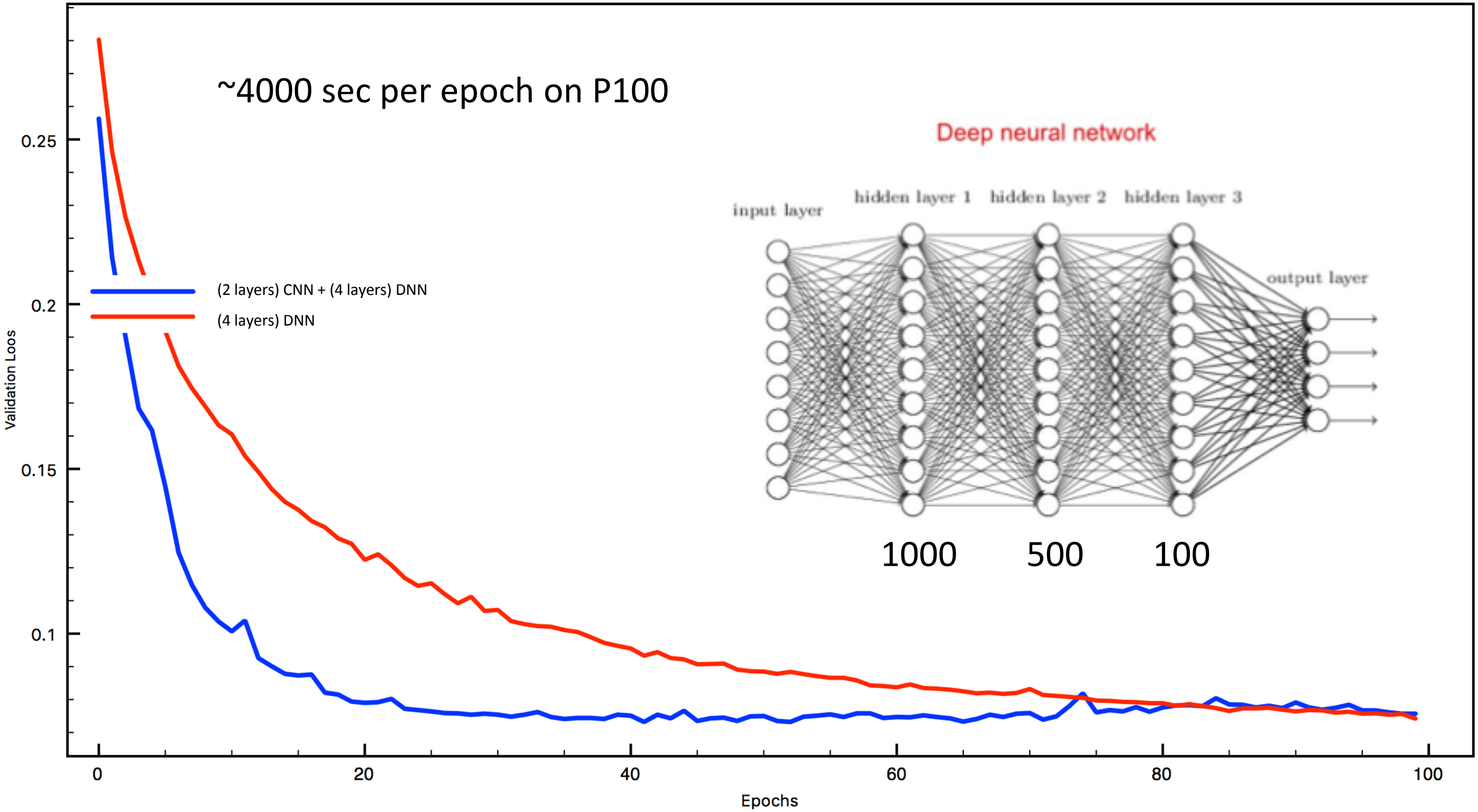


https://github.com/ECP-CANDLE/Benchmarks/tree/frameworks/Pilot1/TC1



https://github.com/ECP-CANDLE/Benchmarks/tree/frameworks/Pilot1/TC1

P1B3 Convergence ([C(100)xC(50)]x1000x500x100x50)

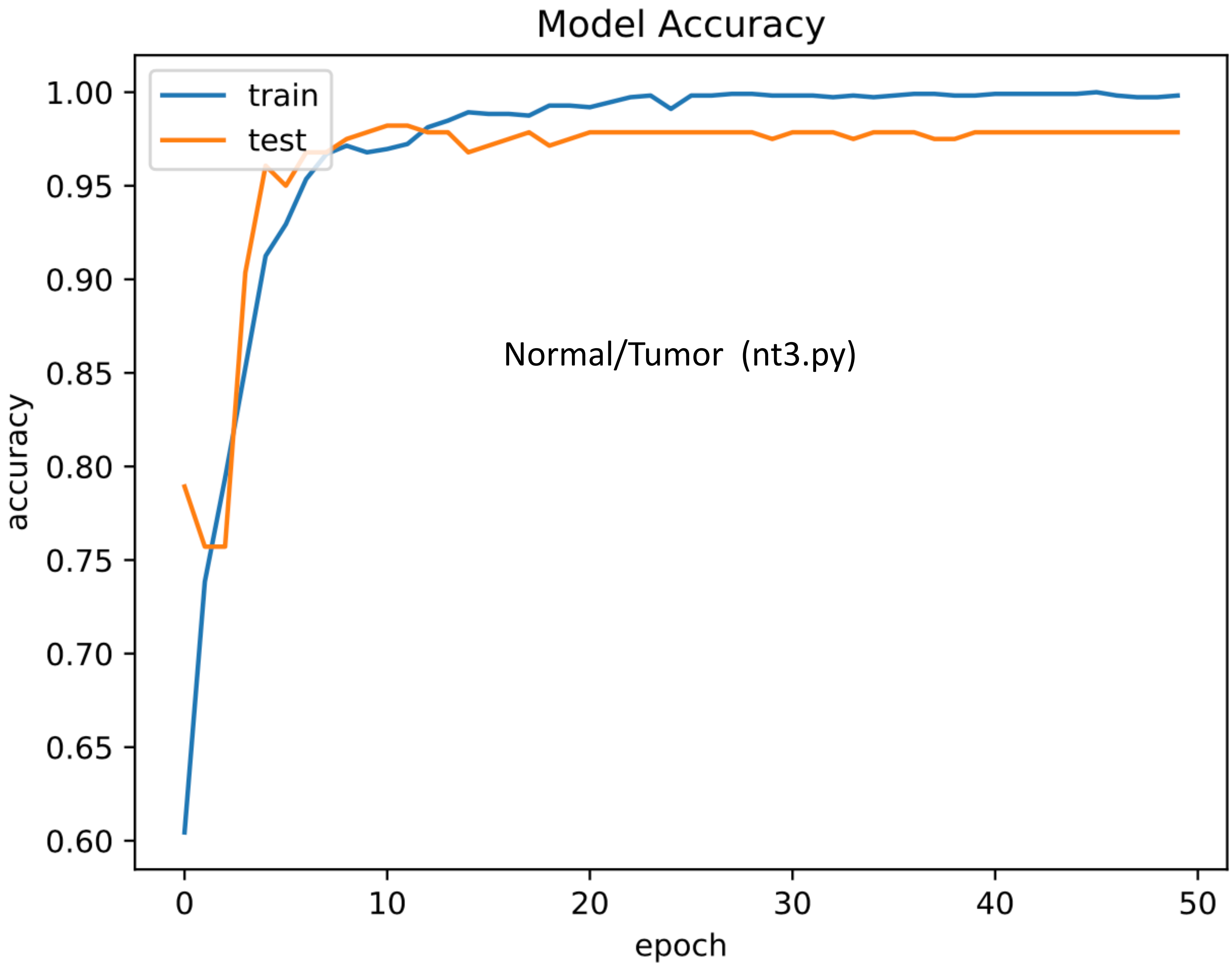


https://github.com/ECP-CANDLE/Benchmarks/tree/frameworks/Pilot1/P1B3

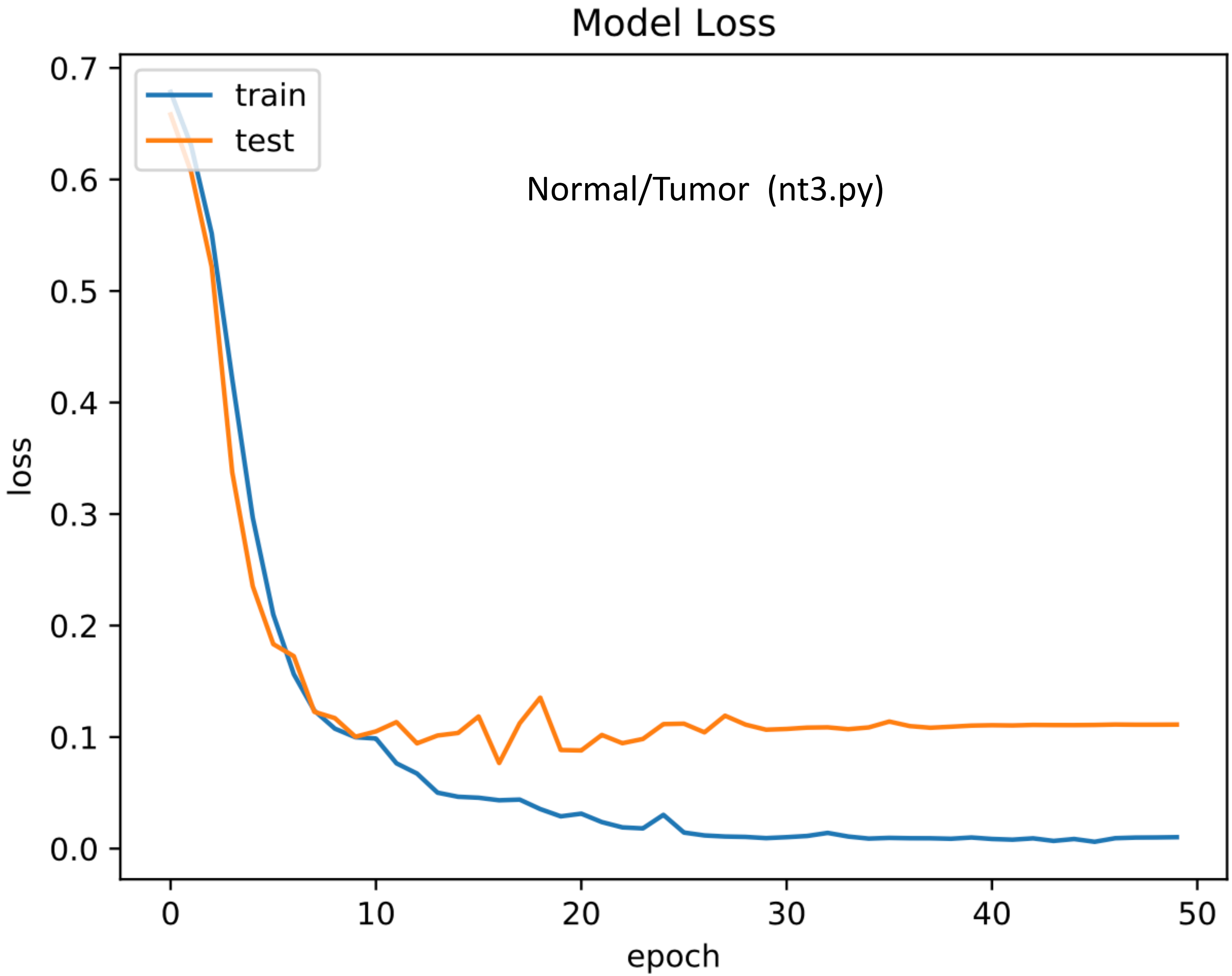
Tumor/Normal Classification

```
2017-10-29 20:44:44.570833: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1045] Creating TensorFlow device (/gpu:0) -> (device: 0, name: Tesla V100-DGXS-16GB, pci bus id:
1100/1120 [============================>.] - ETA: 0s - loss: 0.6787 - acc: 0.6027Epoch 00000: val_loss improved from inf to 0.65825, saving model to nt3.autosave.model.h5
1120/1120 [==============================] - 17s - loss: 0.6785 - acc: 0.6045 - val_loss: 0.6583 - val_acc: 0.7893
Epoch 2/50
1100/1120 [============================>.] - ETA: 0s - loss: 0.6310 - acc: 0.7364Epoch 00001: val_loss improved from 0.65825 to 0.60665, saving model to nt3.autosave.model.h5
1120/1120 [==============================] - 12s - loss: 0.6304 - acc: 0.7384 - val_loss: 0.6066 - val_acc: 0.7571
Epoch 3/50
1100/1120 [============================>.] - ETA: 0s - loss: 0.5529 - acc: 0.7927Epoch 00002: val_loss improved from 0.60665 to 0.52169, saving model to nt3.autosave.model.h5
1120/1120 [==============================] - 12s - loss: 0.5516 - acc: 0.7938 - val_loss: 0.5217 - val_acc: 0.7571
Epoch 4/50
1100/1120 [============================>.] - ETA: 0s - loss: 0.4216 - acc: 0.8518Epoch 00003: val_loss improved from 0.52169 to 0.33755, saving model to nt3.autosave.model.h5
1120/1120 [==============================] - 12s - loss: 0.4212 - acc: 0.8527 - val_loss: 0.3375 - val_acc: 0.9036
Epoch 5/50
1100/1120 [============================>.] - ETA: 0s - loss: 0.2969 - acc: 0.9127Epoch 00004: val_loss improved from 0.33755 to 0.23527, saving model to nt3.autosave.model.h5
1120/1120 [==============================] - 12s - loss: 0.2967 - acc: 0.9125 - val_loss: 0.2353 - val_acc: 0.9607
Epoch 6/50
1100/1120 [============================>.] - ETA: 0s - loss: 0.2105 - acc: 0.9291Epoch 00005: val_loss improved from 0.23527 to 0.18337, saving model to nt3.autosave.model.h5
1120/1120 [==============================] - 12s - loss: 0.2099 - acc: 0.9295 - val_loss: 0.1834 - val_acc: 0.9500
Epoch 7/50
```

```
1100/1120 [============================>.] - ETA: 0s - loss: 0.0080 - acc: 0.9991Epoch 00041: val_loss did not improve
1120/1120 [==============================] - 10s - loss: 0.0080 - acc: 0.9991 - val_loss: 0.1104 - val_acc: 0.9786
Epoch 43/50
1100/1120 [============================>.] - ETA: 0s - loss: 0.0093 - acc: 0.9991Epoch 00042: val_loss did not improve
1120/1120 [==============================] - 11s - loss: 0.0092 - acc: 0.9991 - val_loss: 0.1109 - val_acc: 0.9786
Epoch 44/50
1100/1120 [============================>.] - ETA: 0s - loss: 0.0070 - acc: 0.9991Epoch 00043: val_loss did not improve
1120/1120 [==============================] - 10s - loss: 0.0069 - acc: 0.9991 - val_loss: 0.1108 - val_acc: 0.9786
Epoch 45/50
1100/1120 [============================>.] - ETA: 0s - loss: 0.0085 - acc: 0.9991Epoch 00044: val_loss did not improve
1120/1120 [==============================] - 10s - loss: 0.0086 - acc: 0.9991 - val_loss: 0.1107 - val_acc: 0.9786
Epoch 46/50
1100/1120 [============================>.] - ETA: 0s - loss: 0.0062 - acc: 1.0000Epoch 00045: val_loss did not improve
1120/1120 [==============================] - 10s - loss: 0.0061 - acc: 1.0000 - val_loss: 0.1109 - val_acc: 0.9786
Epoch 47/50
1100/1120 [============================>.] - ETA: 0s - loss: 0.0094 - acc: 0.9982Epoch 00046: val_loss did not improve
1120/1120 [==============================] - 10s - loss: 0.0093 - acc: 0.9982 - val_loss: 0.1113 - val_acc: 0.9786
Epoch 48/50
1100/1120 [============================>.] - ETA: 0s - loss: 0.0098 - acc: 0.9973Epoch 00047: val_loss did not improve
```



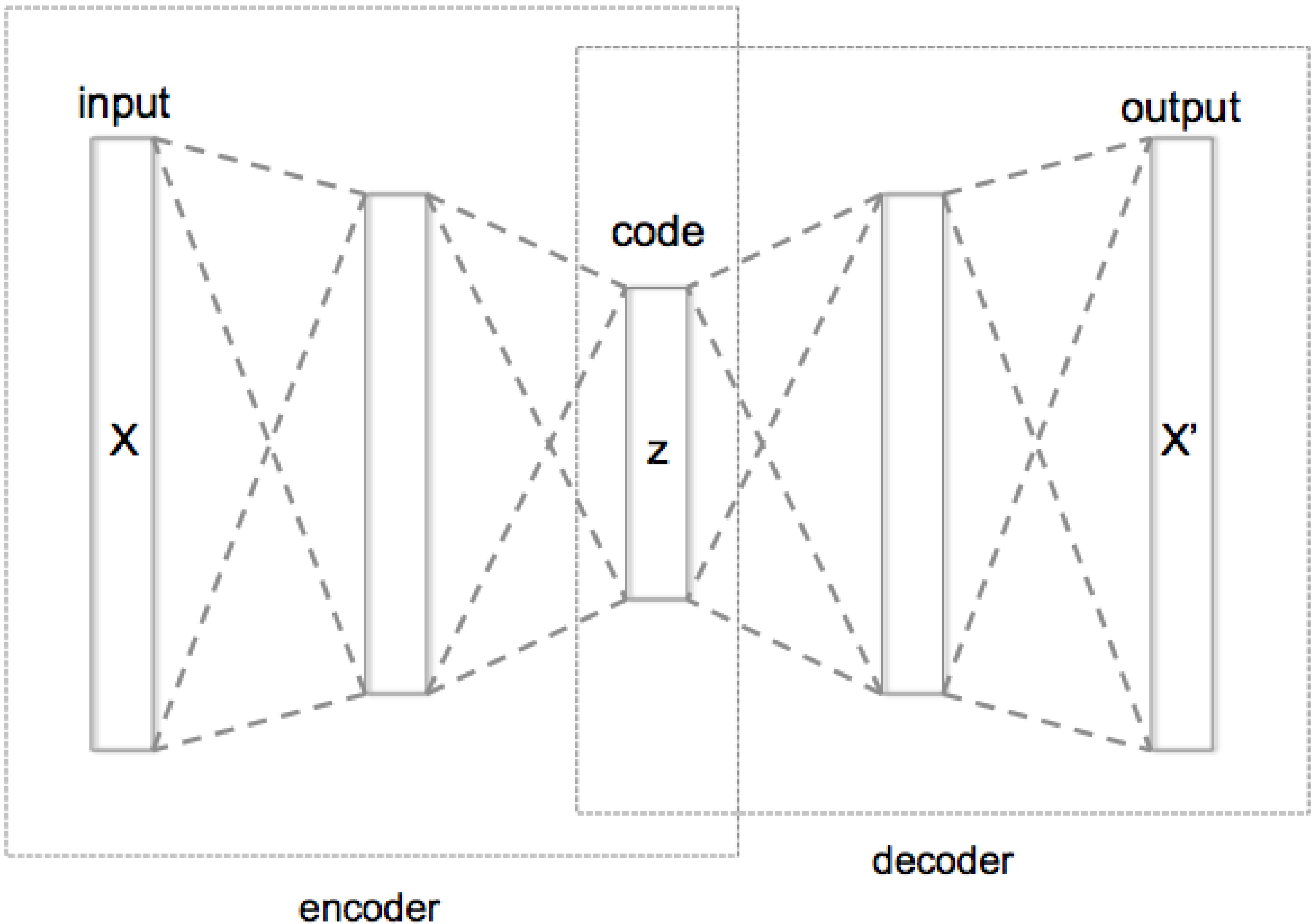
https://github.com/ECP-CANDLE/Benchmarks/tree/frameworks/Pilot1/NT3



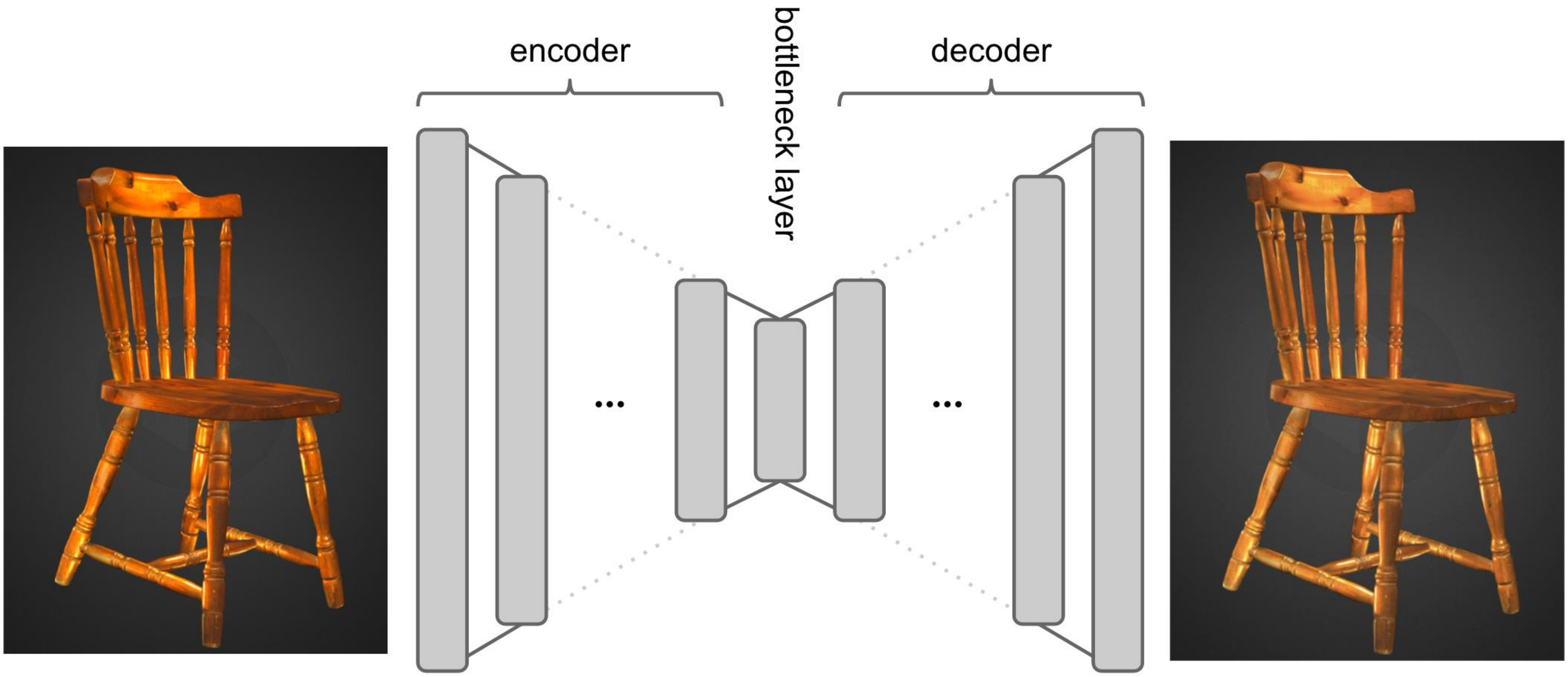
https://github.com/ECP-CANDLE/Benchmarks/tree/frameworks/Pilot1/NT3

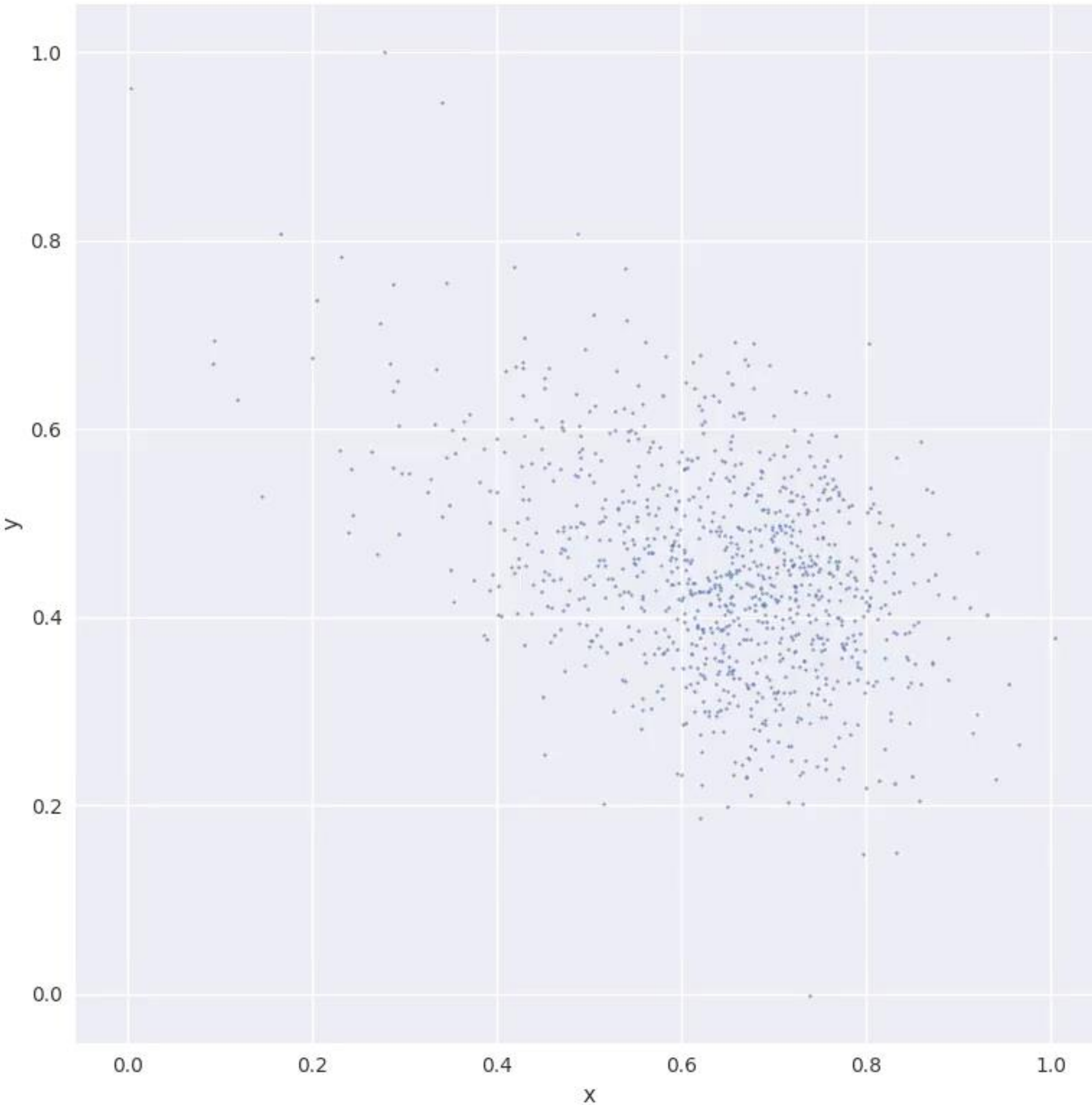
How did we know it might work?

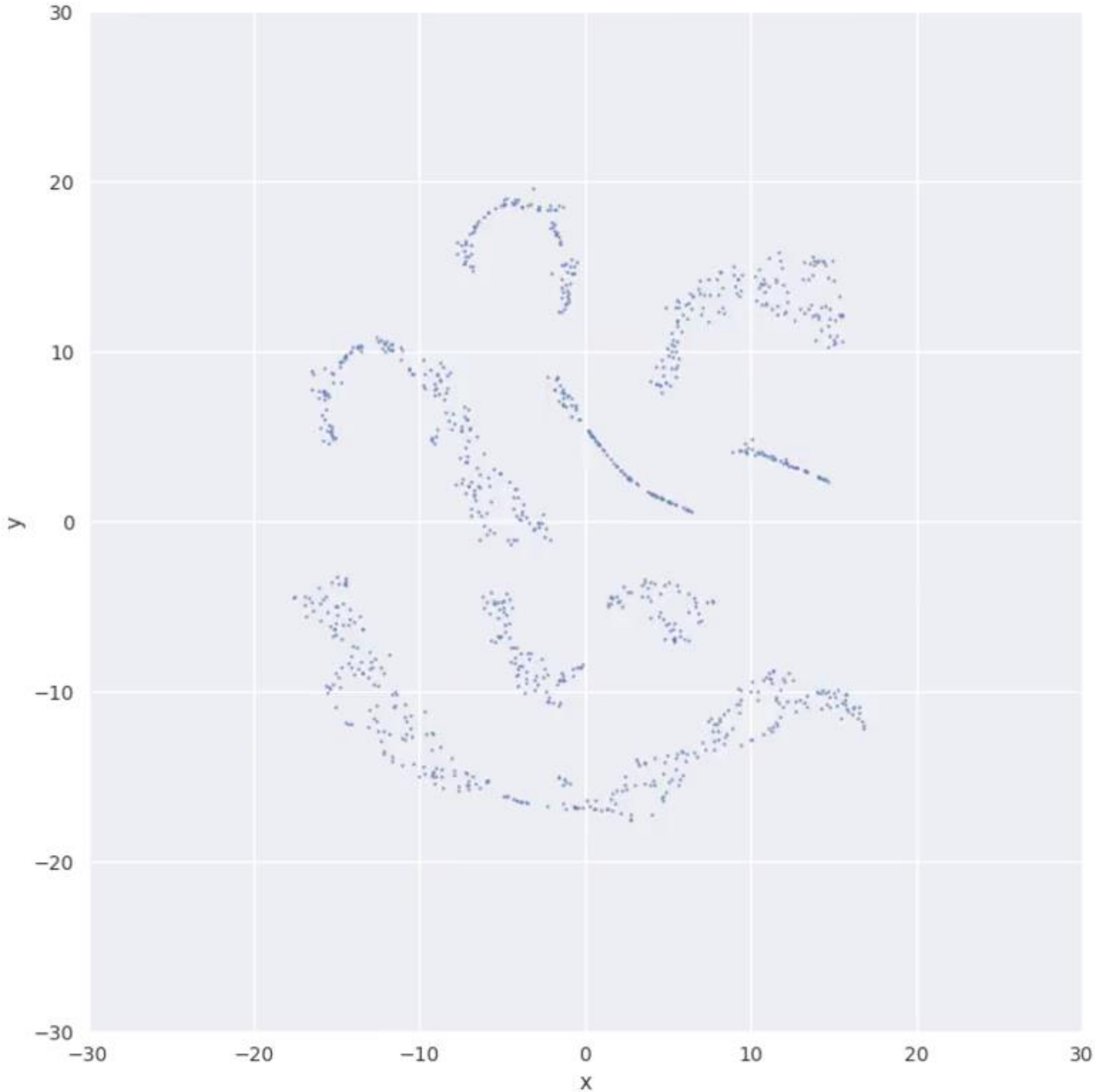
- Build autoencoders first with the features you are going to work with
- If you get reasonable reconstruction error then the model can learn a representation and that is a good sign
- Class balance seems to matter
- Number of training examples matters > 1000 is good > 10,000 better, > 100,000 much better
- Hyper parameter search is also important once you get something that basically works


input
X
code
z
output
X'
encoder
decoder

Autoencoder




1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x


30
20
10
0
−10
−20
−30
y
−30
−20
−10
0
10
20
30
x

Acknowledgements

Many thanks to DOE, NSF, NIH, DOD, ANL, UC, Moore Foundation, Sloan Foundation, Apple, Microsoft, Cray, Intel, NVIDIA and IBM for supporting our research group over the years