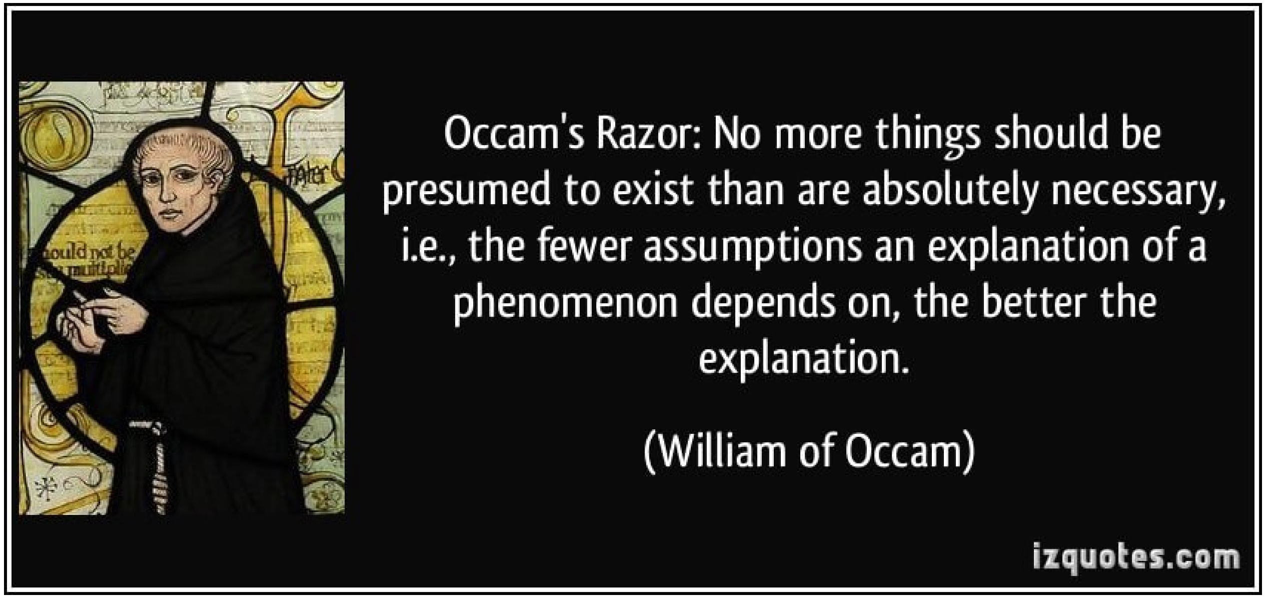
Deep Learning for Life Sciences: Bayesian Deep Learning

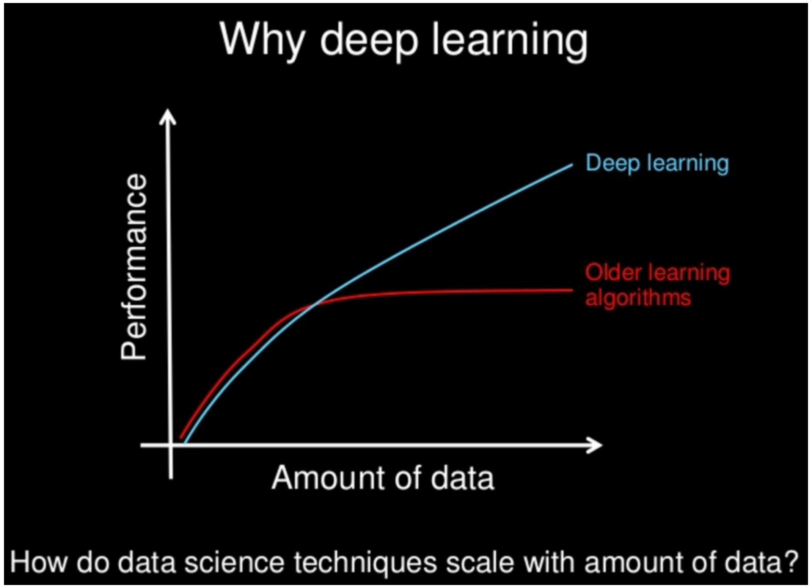
Nikolay Oskolkov, NBIS SciLifeLab

11.12.2020

Chellenges of Applying Deep Learning

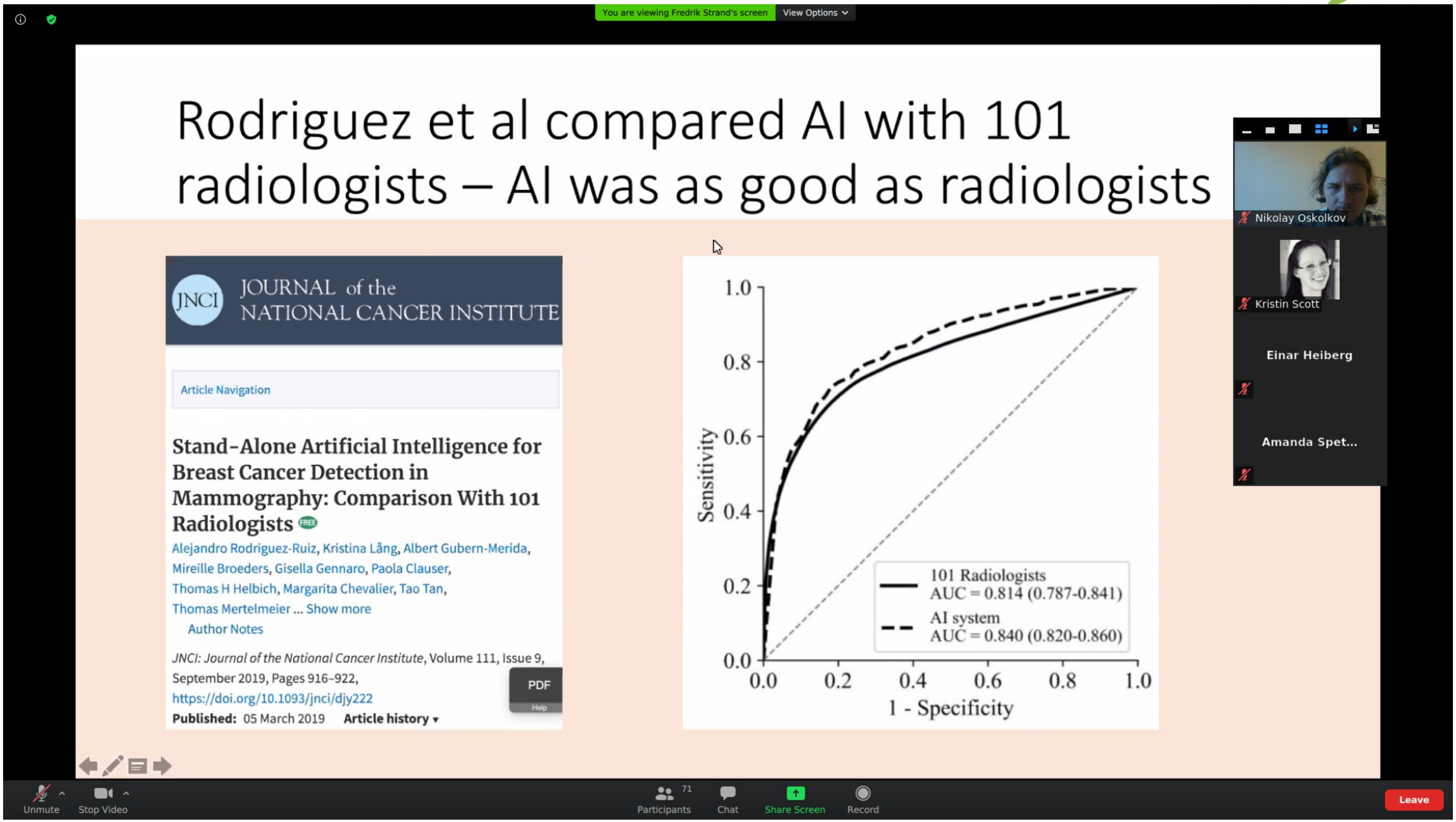
- Apply to real Life Science projects (NGS: tabular data)
- Apply only if Deep Learning better than simpler methods





Why don't neural networks always work?

Statistics vs. Machine Learning



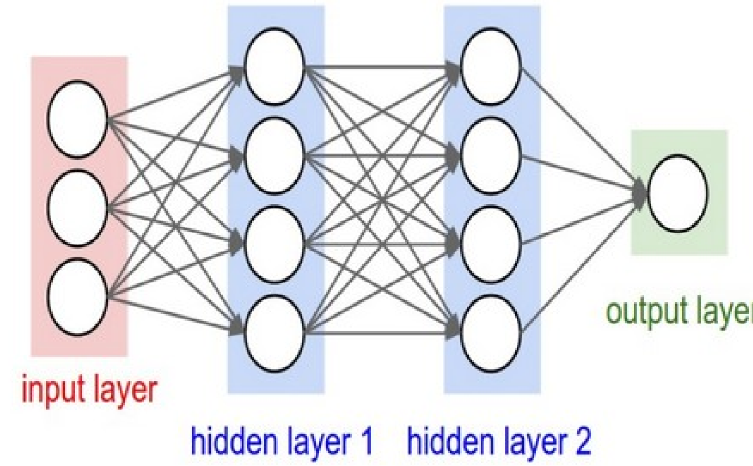
Why do you compare AI against radiologists?
You should compare it against simpler models

Deep Learning is not The Only Tool

Bayesianism

Frequentism

Deep Learning



P >> N

P ~ N

P << N

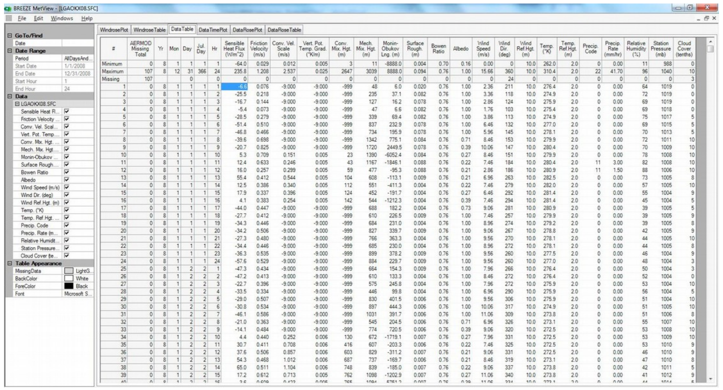
Amount of Data

Deep Learning is
a yet another tool

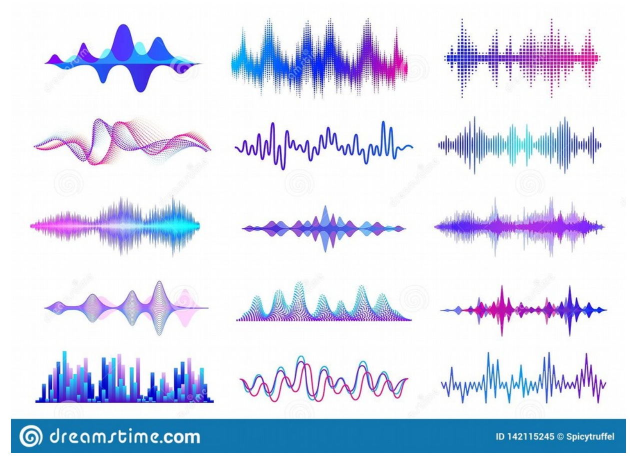
Comparison is important:
If you do not compare, your
neural network is the best

Various Types of Data

Tabular



Sound



Image

Text

Editing Wikipedia articles on

Medicine

Editing Wikipedia can be daunting for newbies, especially if you're contributing to Wikipedia for the first time as a class assignment. This guide is designed to assist students who have been assigned to contribute biomedical-related content to Wikipedia. Here's what other editors will expect you to know.

Be accurate

You're editing a resource millions of people use to make medical decisions, so it's vitally important to be accurate. Wikipedia is used more for medical information than the websites for WebMD, NIH, and the WHO. But with great power comes great responsibility!

Understand the guidelines

Wikipedia editors in the medicine area have developed additional guidelines to ensure that the content on Wikipedia is medically sound. Take extra time to read and understand these guidelines. When you edit an article, ensure your changes meet these special requirements. If not, your work is likely to be undone by other editors as they clean up after you. That takes valuable volunteer time away from creating content. If you're not comfortable working under these guidelines, talk to your instructor about an alternative off-wiki assignment.

Engage with editors

Part of the Wikipedia experience is receiving and responding to feedback from other editors. Do not submit your content on the last day, then leave Wikipedia! Real human volunteers from the Wikipedia community will likely read and respond to it, and it would be polite for you to acknowledge the time they volunteer to polish your work! Everything submitted to Wikipedia is reviewed by multiple, real humans! You may not get a comment, but if you do, please acknowledge it.

Watch out for close paraphrasing

Plagiarizing or close paraphrasing is never okay on Wikipedia and is a violation of your university's academic honor code. It's even worse on Wikipedia, as valuable volunteer time that could be used to create good content is instead used to clean up plagiarized work.

If you plagiarize or too closely paraphrase on Wikipedia, it is extremely likely that you'll be caught by other editors and there will be an online record of your plagiarism tied to your permanent online record.

Note that even educational materials from organizations like the WHO and abstracts of articles in PubMed are under copyright and cannot be copied. Write them in your own words whenever possible. If you aren't clear on what close paraphrasing is, visit your university's writing center.

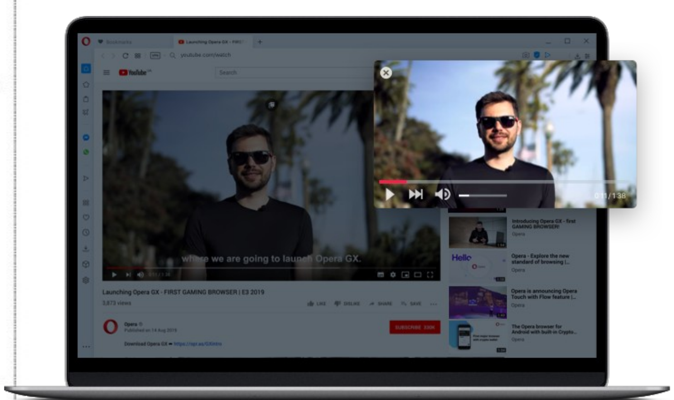
Scared? Don't be!

Everybody on Wikipedia wants to make the best encyclopedia they can. Take the time to understand the rules, and soon you'll be contributing to a valuable resource you use on a daily basis!

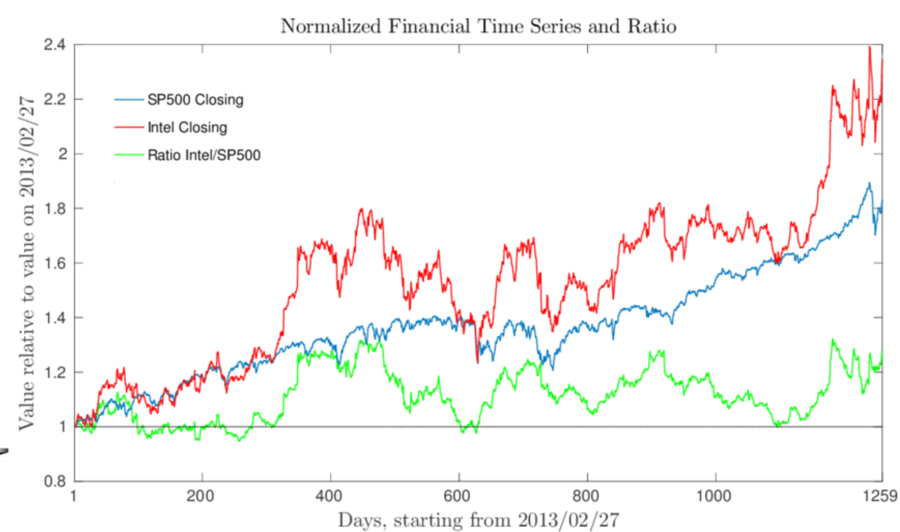
Wiki Edu

DATA

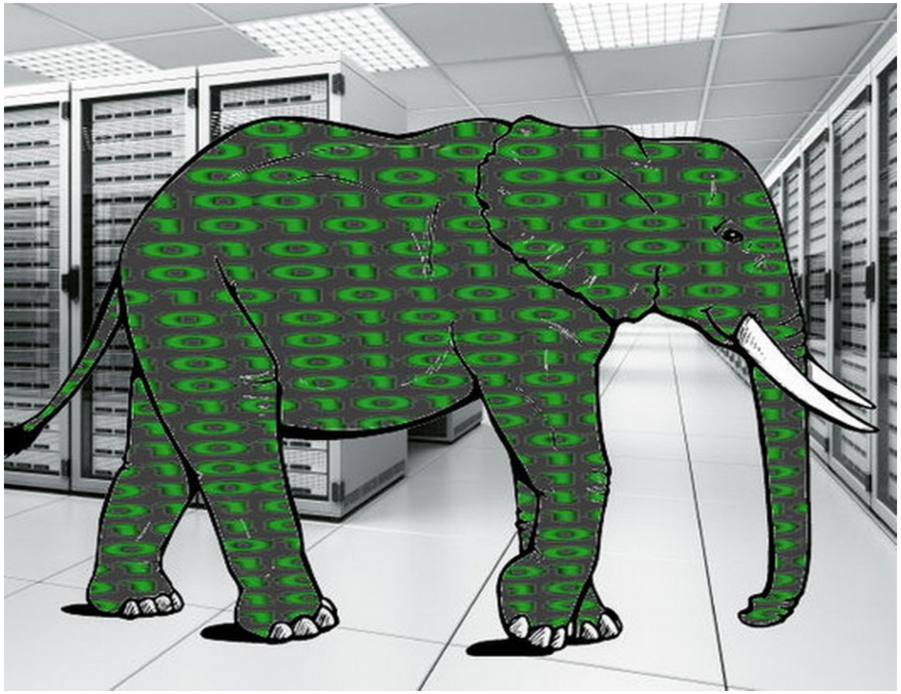
Video

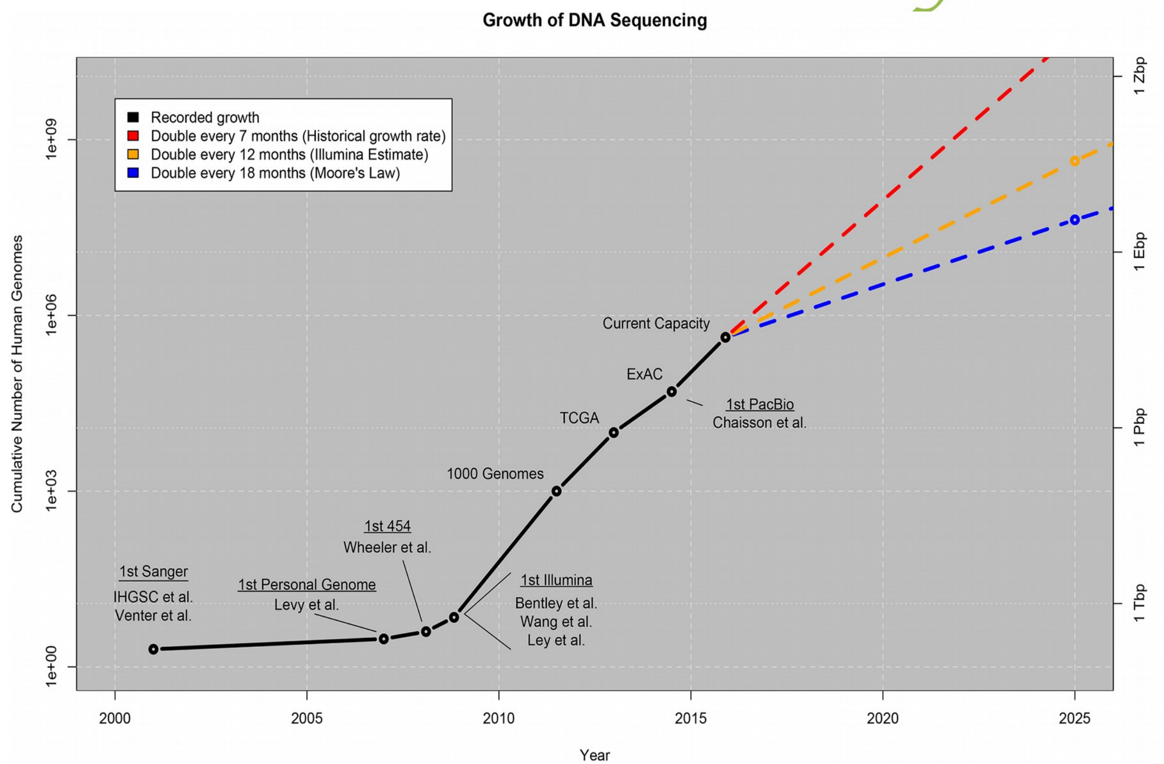


Time Series



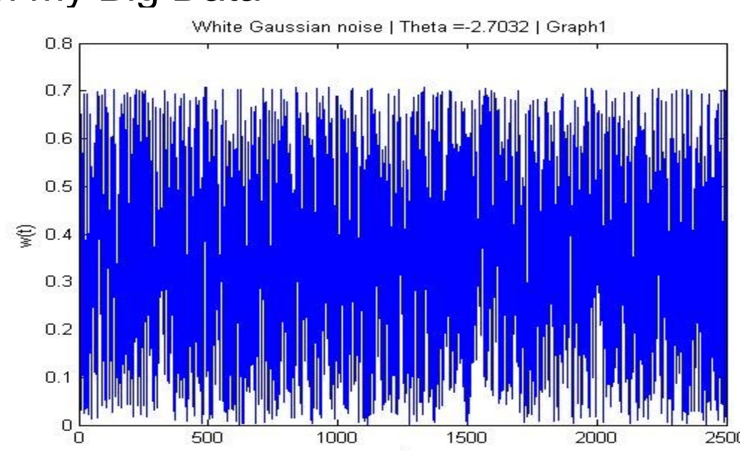
Do we have Big Data in Life Sciences?





I have 500 TB of data on my disk, this is big.

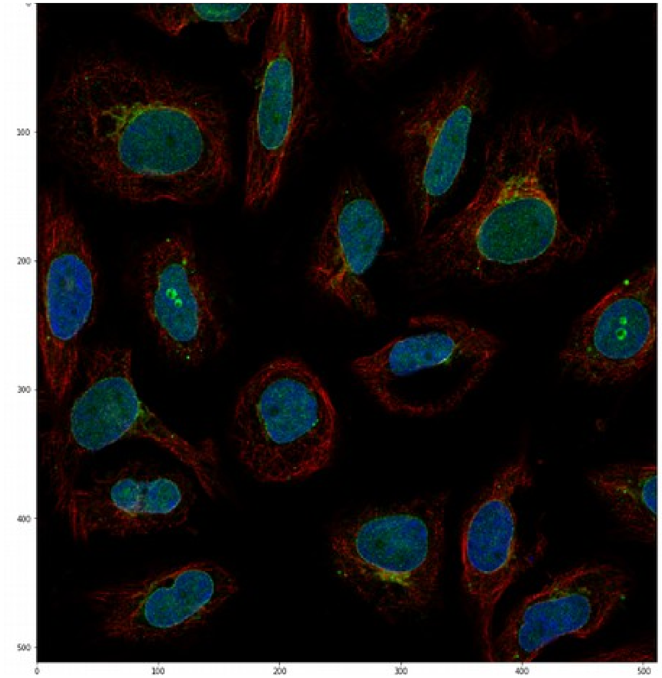
I have Big Data, I want to run Deep Learning on my Big Data

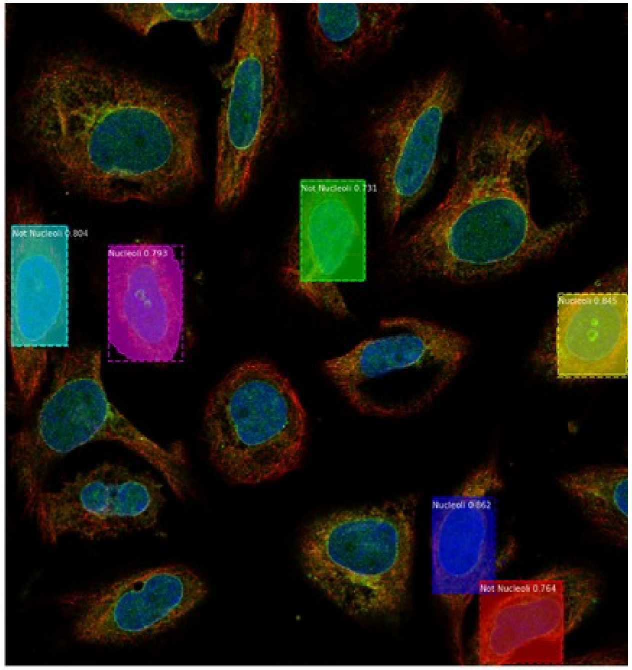


Possible Big Data in Life Sciences:

- Microscopy Imaging
- Single Cell Omics
- Metagenomics (possibly)
- Genomics (sequence is an observation)

Deep Learning on Microscopy Imaging



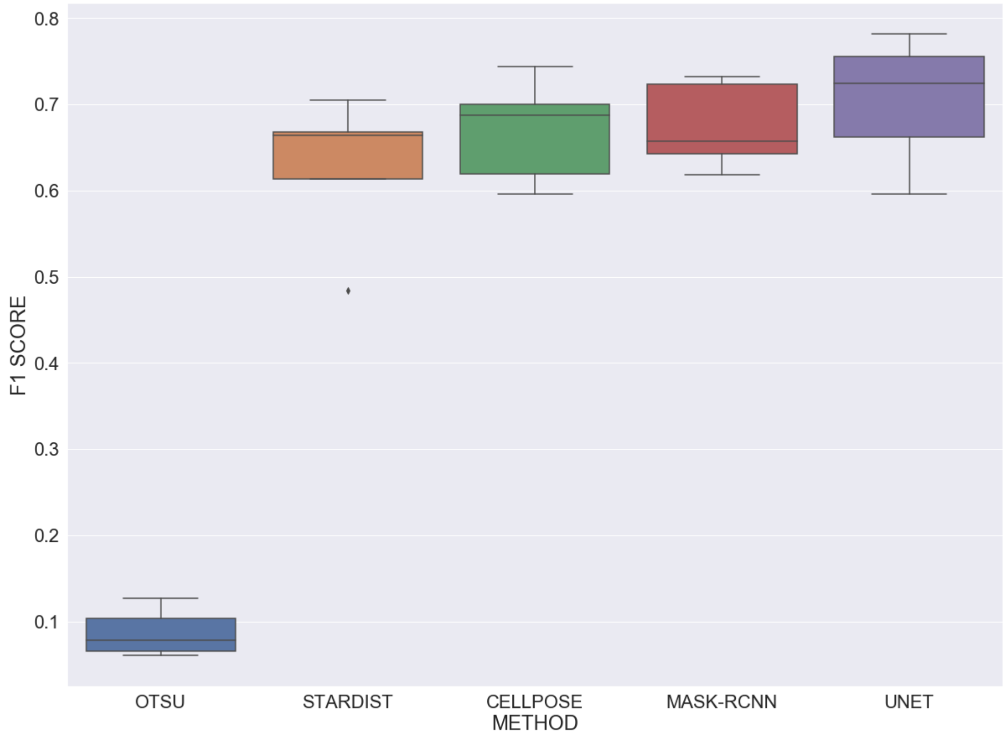


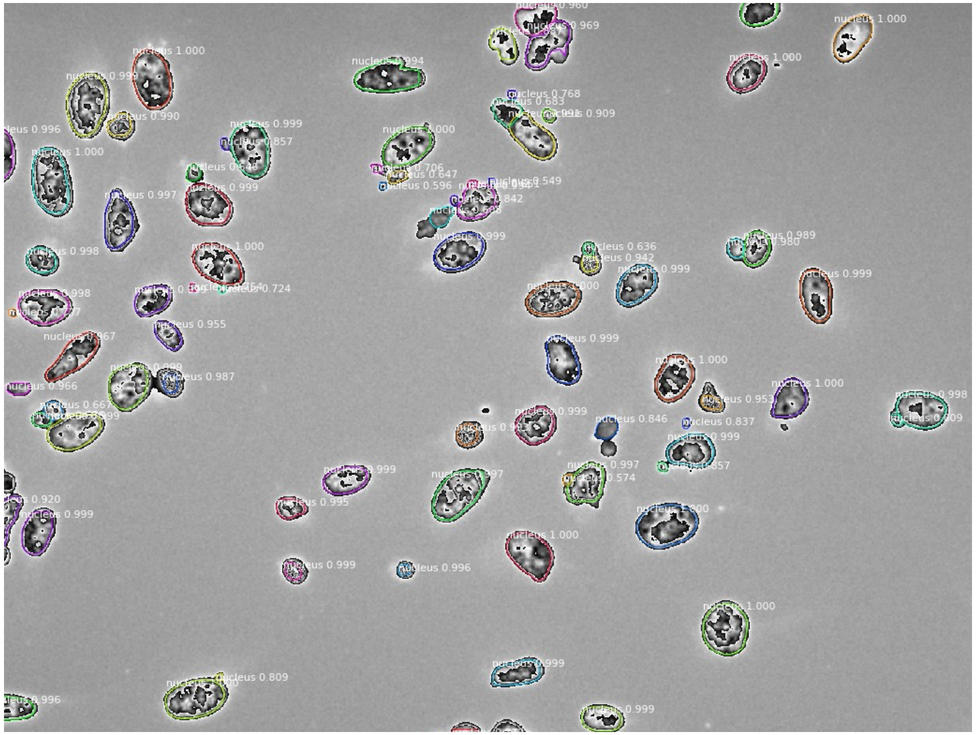
Object detection

Mask-RCNN

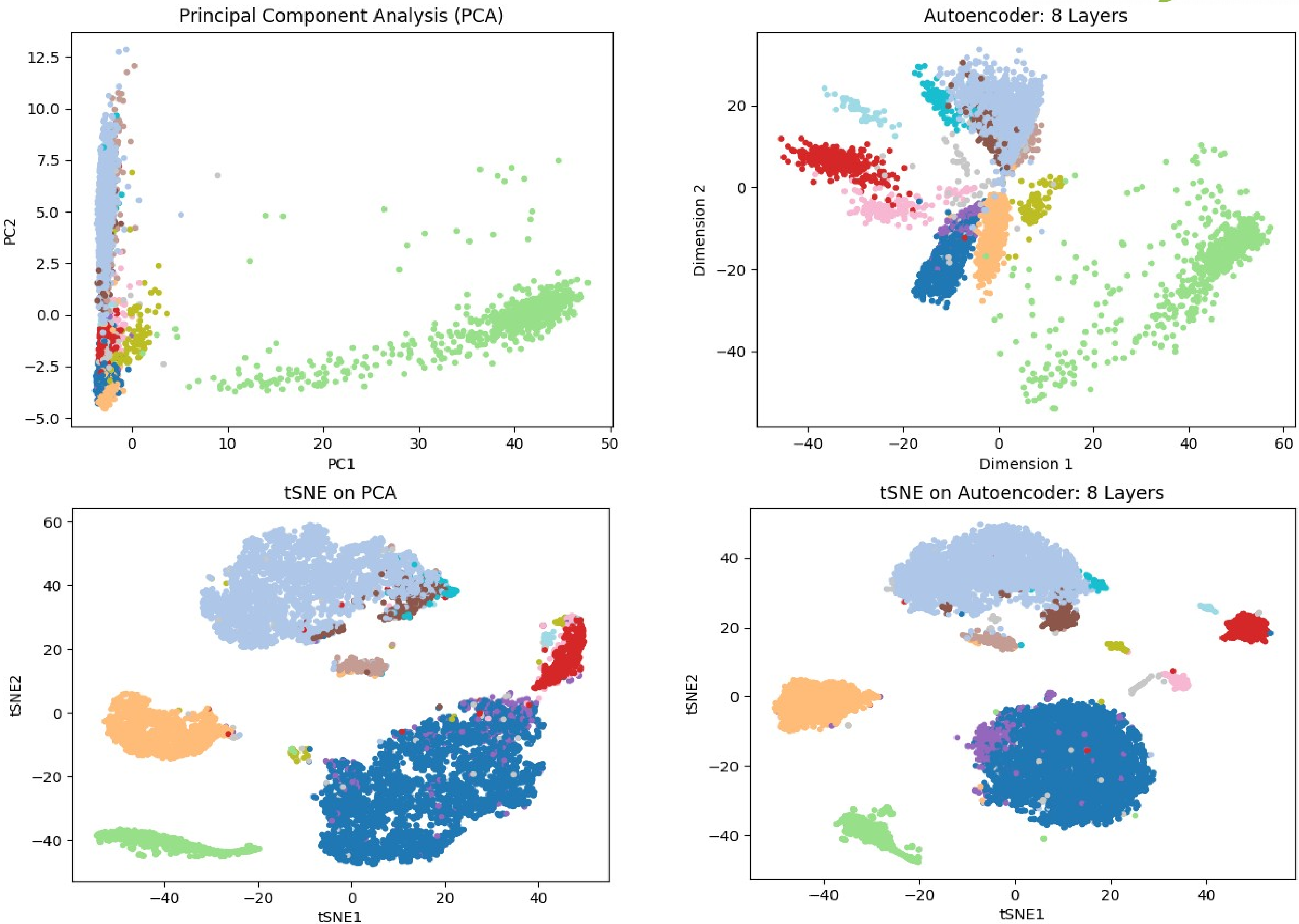
Nuclei segmentation

COMPARING DIFFERENT NUCLEI SEGMENTATION STRATEGIES FOR BBBC039 CELLS: IOU = 0.9

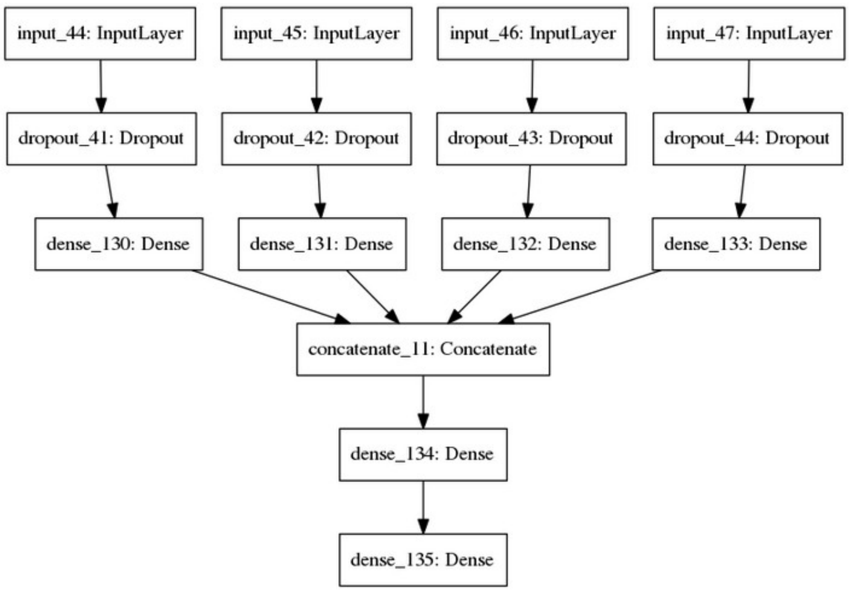


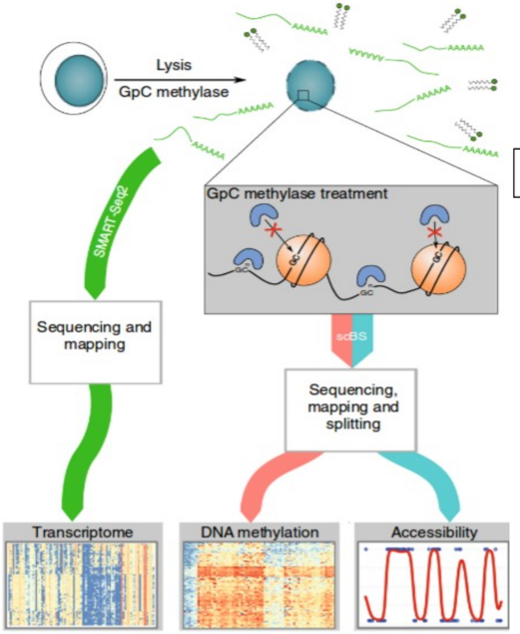


Deep Learning for Single Cell

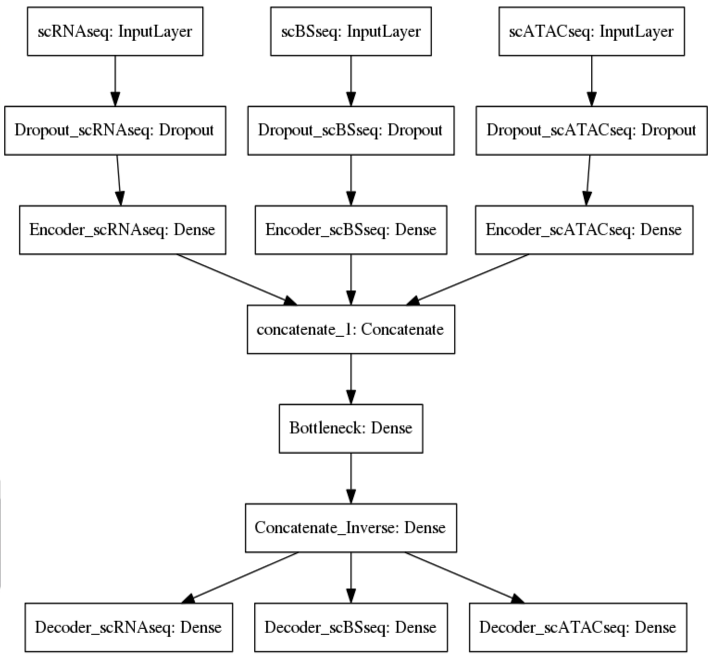


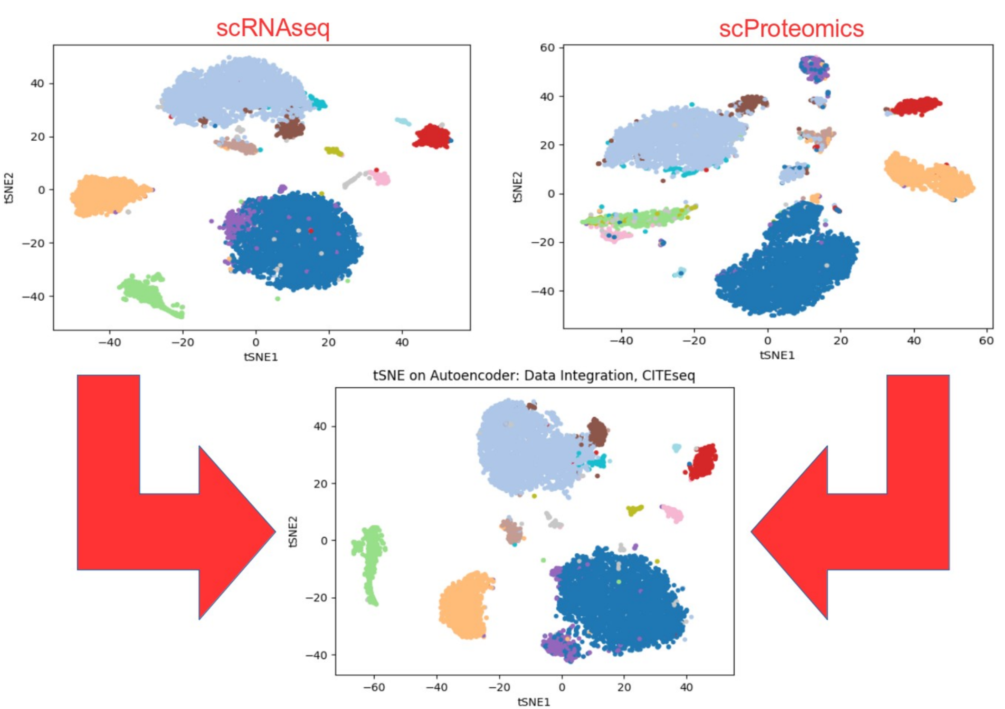
Deep Learning for Data Integration



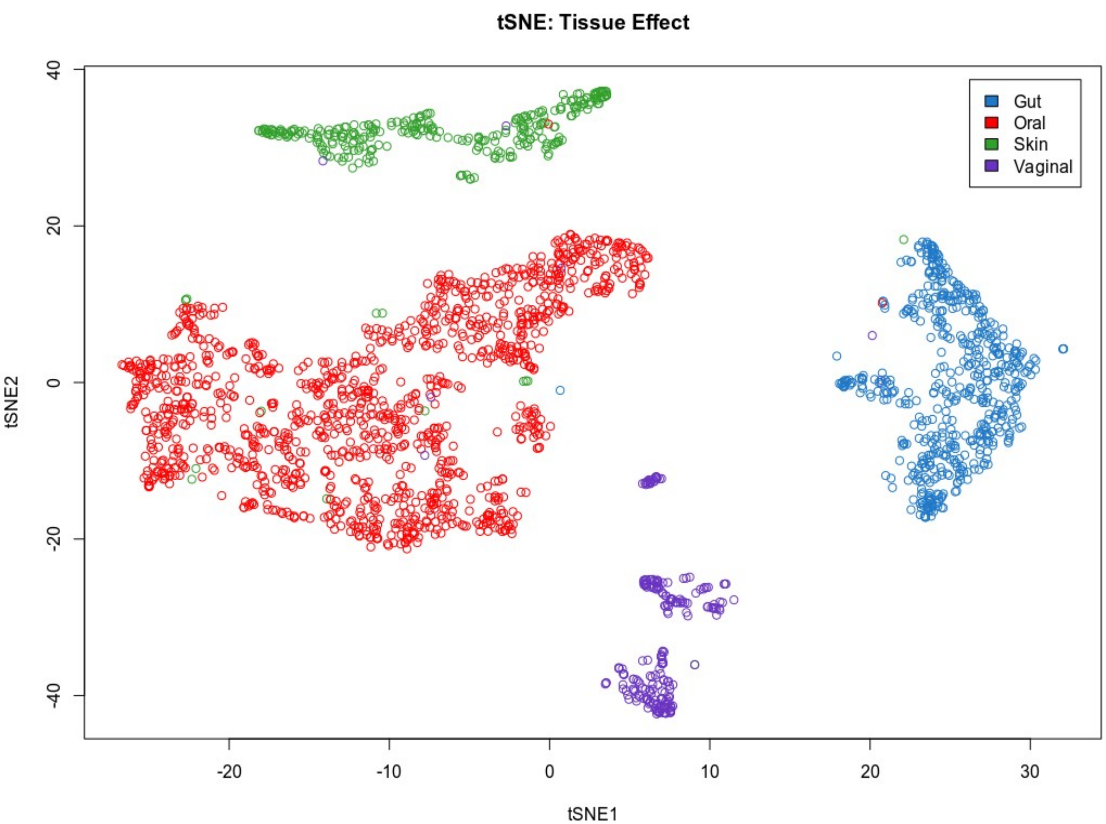


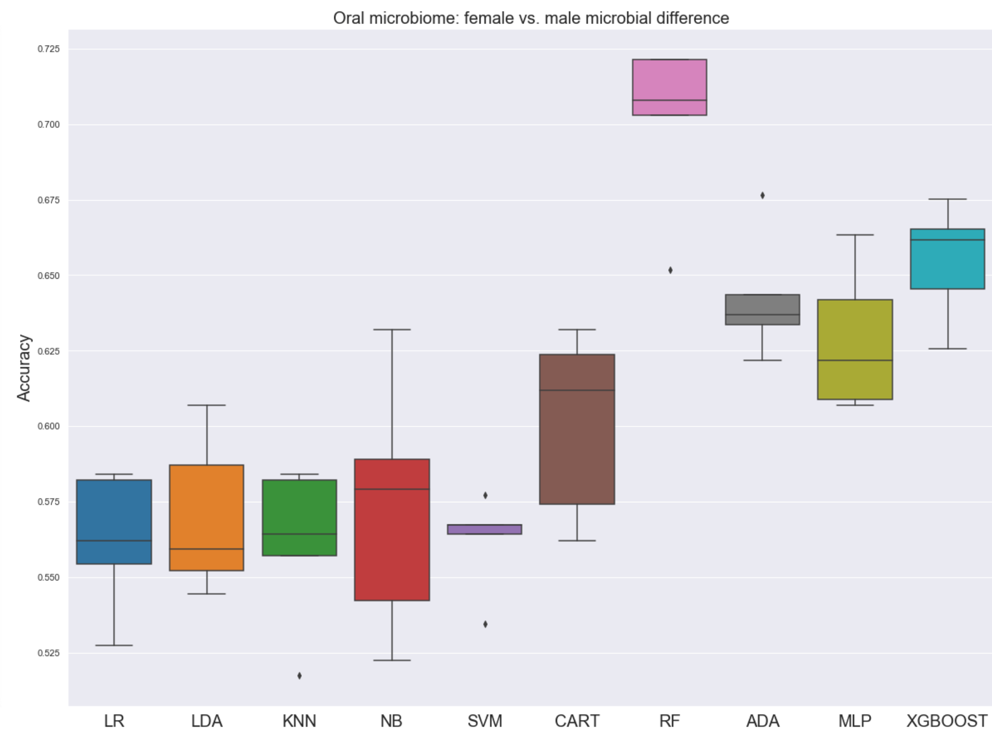
scNMTseq: Clark et al., 2018, Nature Communications 9, 781



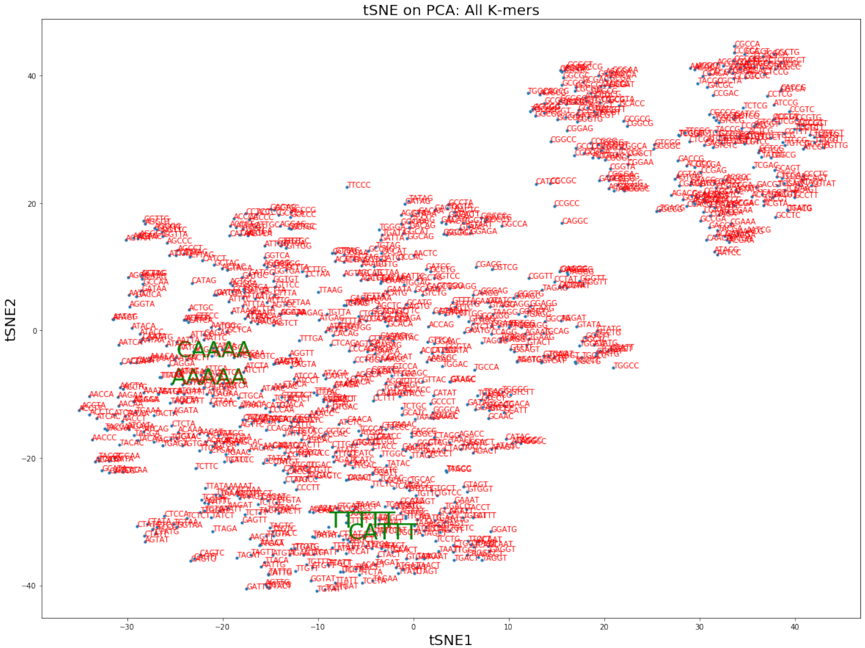


Deep Learning for Metagenomics



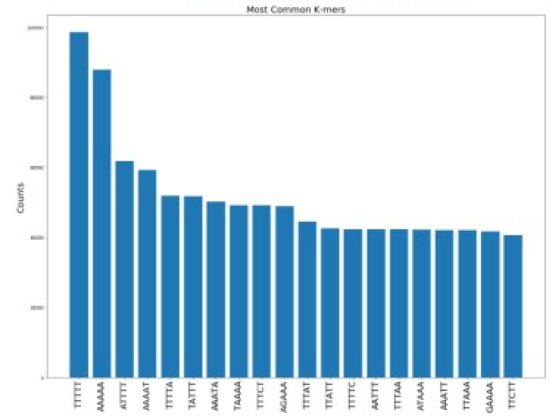


Deep Learning for Genomics



Neanderthal introgressed vs depleted

NLP: Bag of Words



Sequence

GATCGTAC

Kmers

GATCGTAC

GATCGTAC

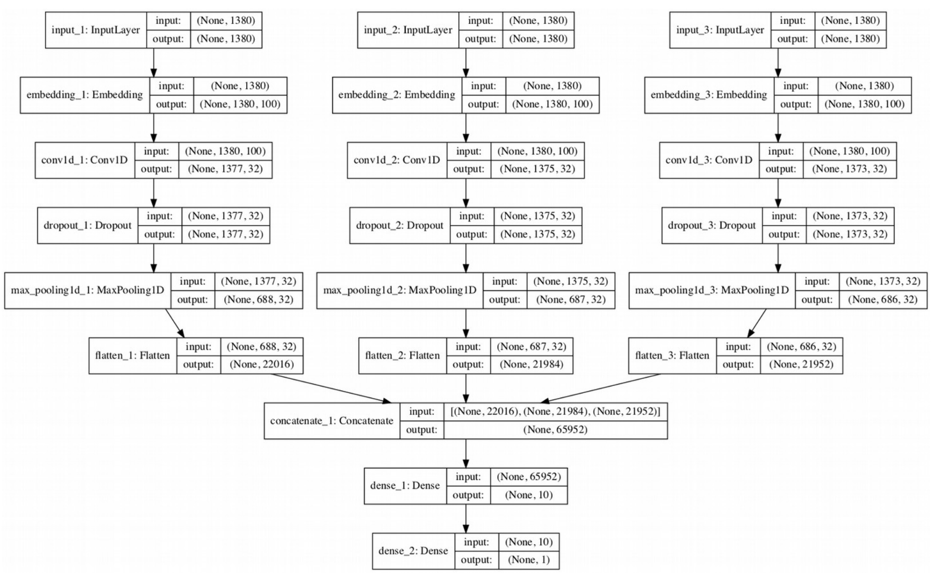
GATCGTAC

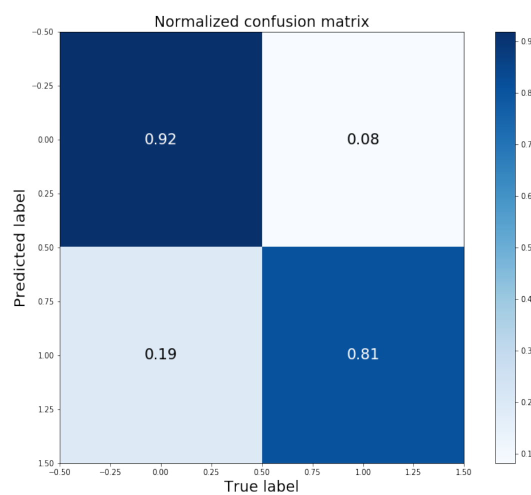
GATCGTAC

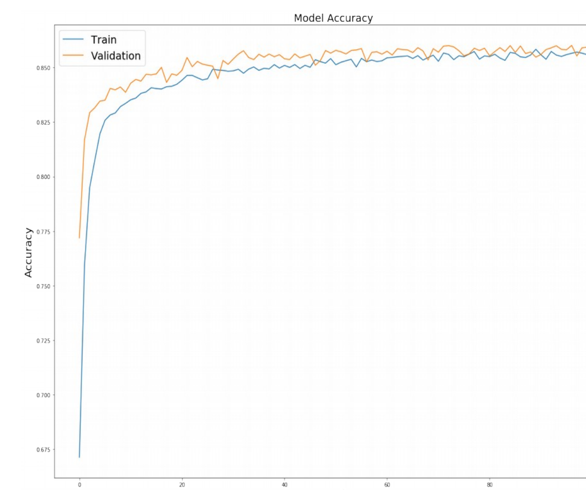
Sentence / Text

GATCG ATCGT TCGTA CGTAC

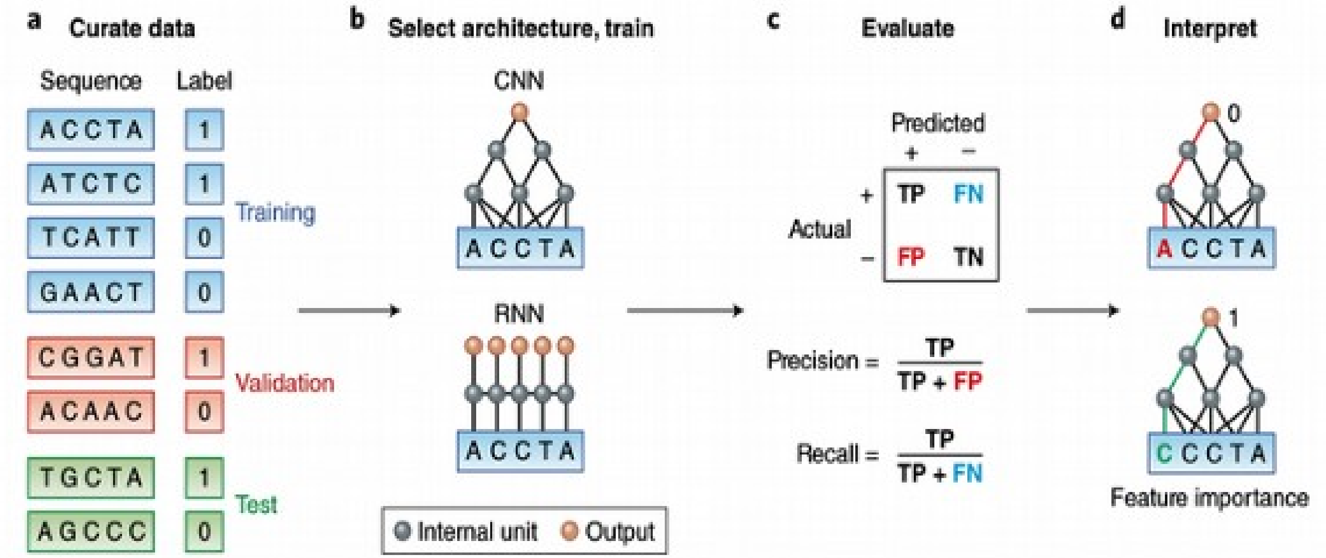
word word word word







Deep Learning on Ancient DNA (aDNA)

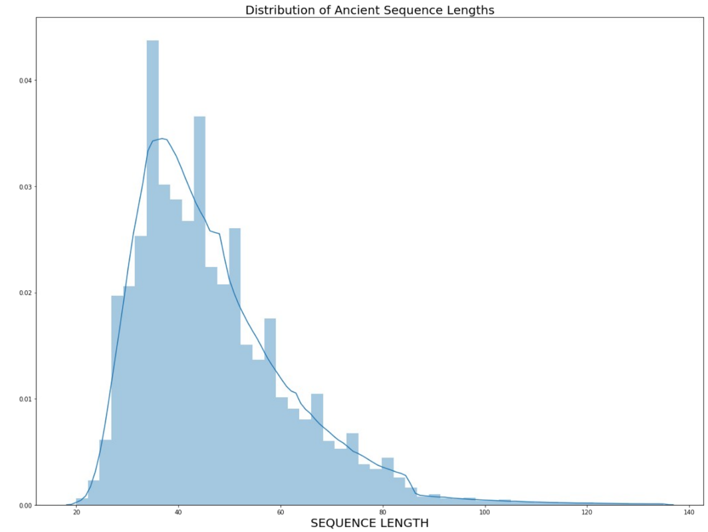


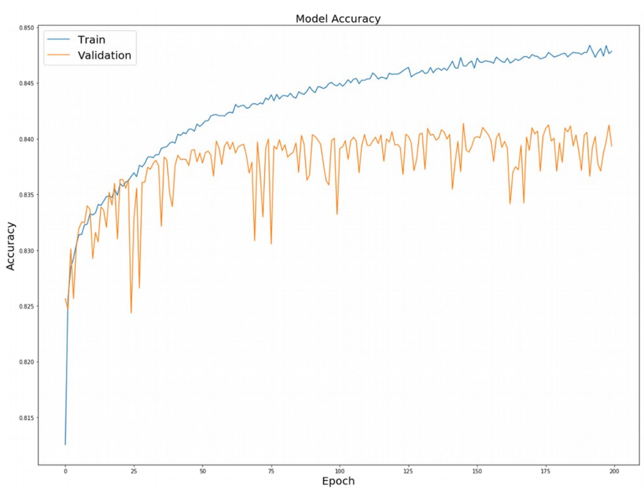
Zou et al., 2018, Nature Genetics 51, 12-18

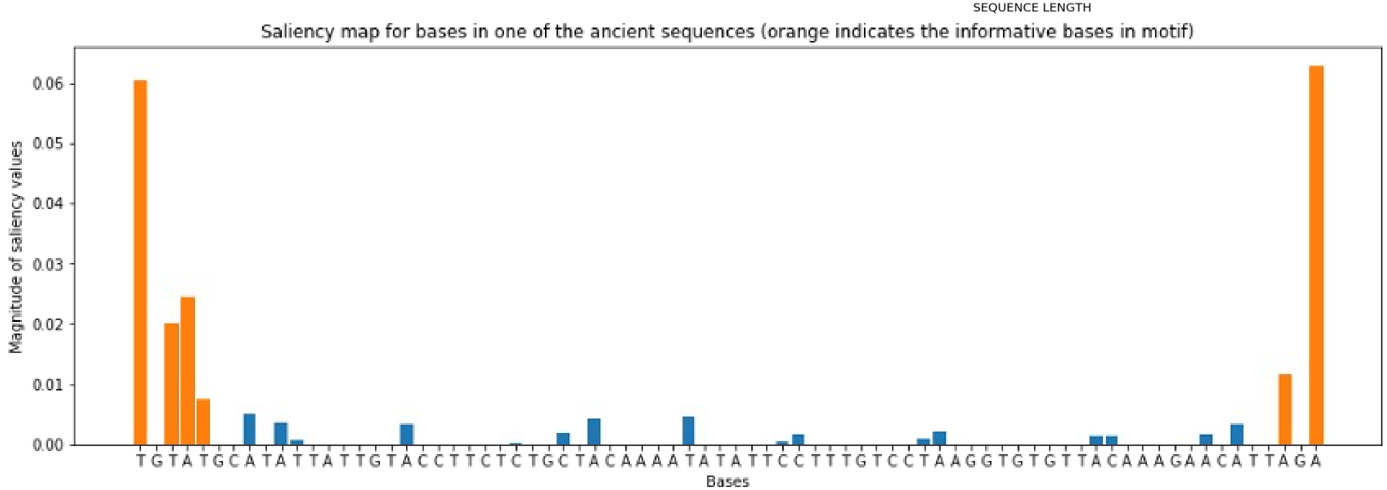
One sequence / read is one statistical observation

One genome (N=1) is a Big Data

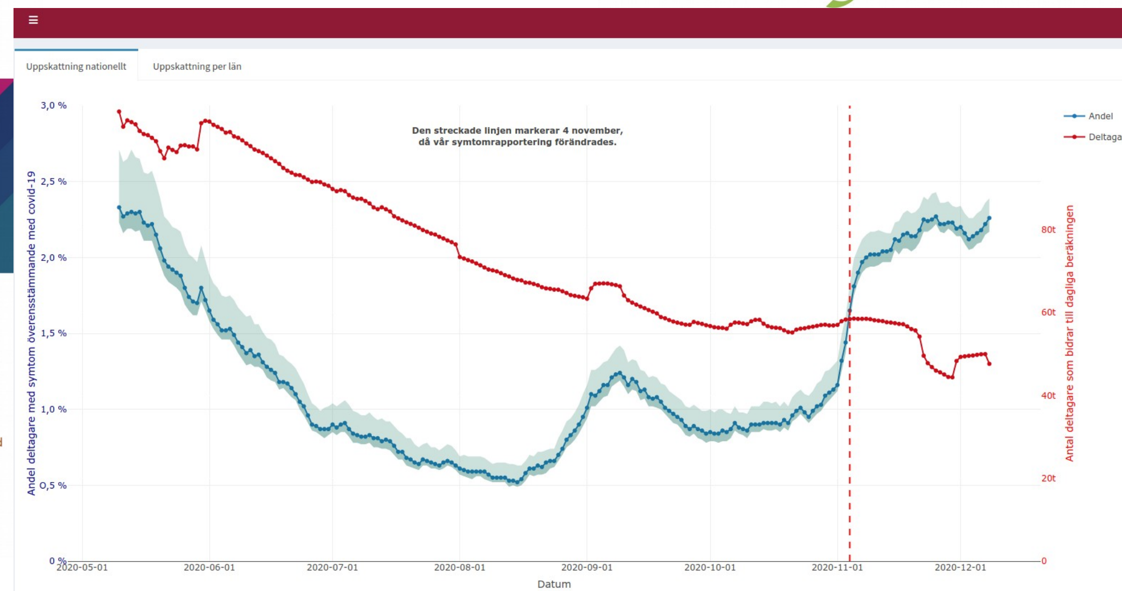
Modern contamination is a big problem for aDNA



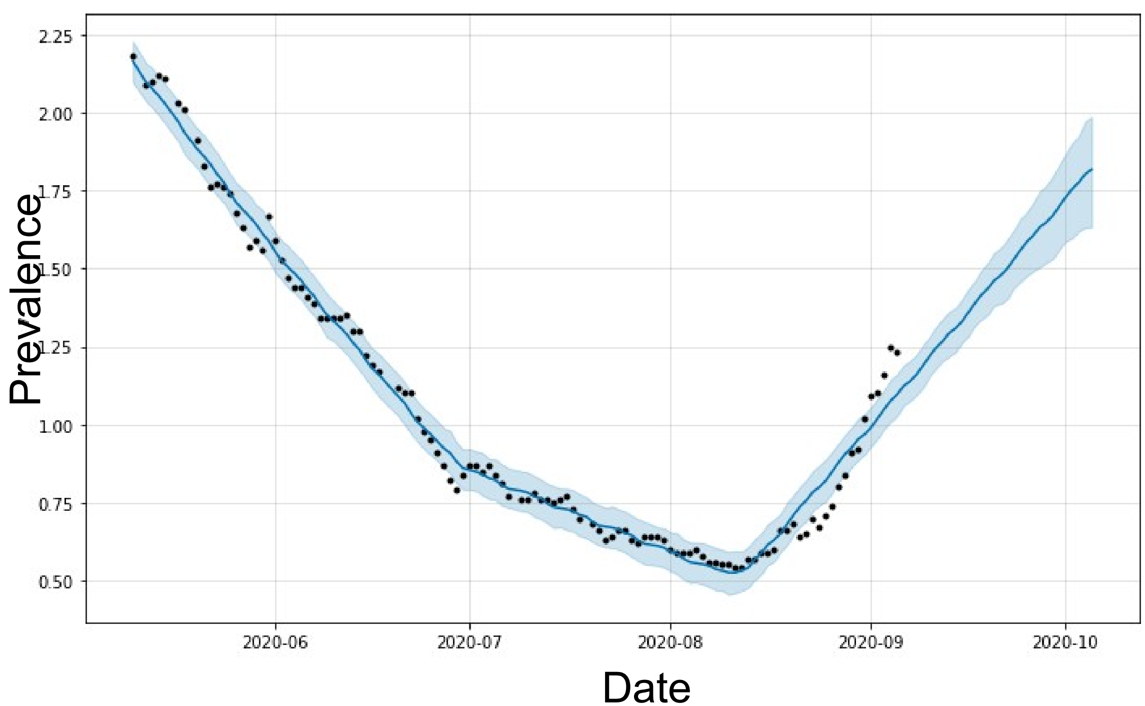


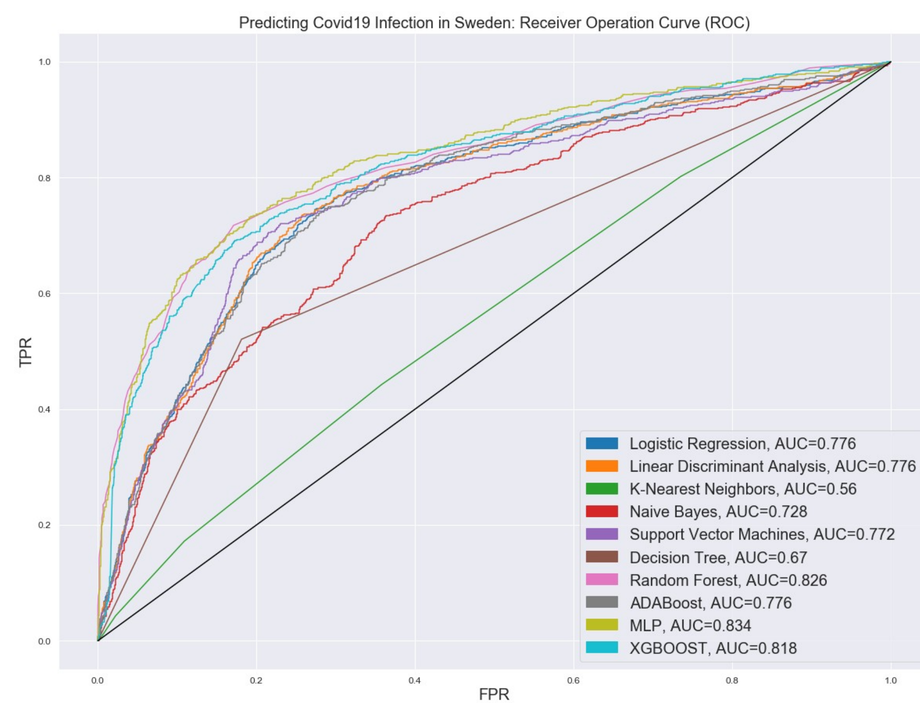


Deep Learning for Epidemiology

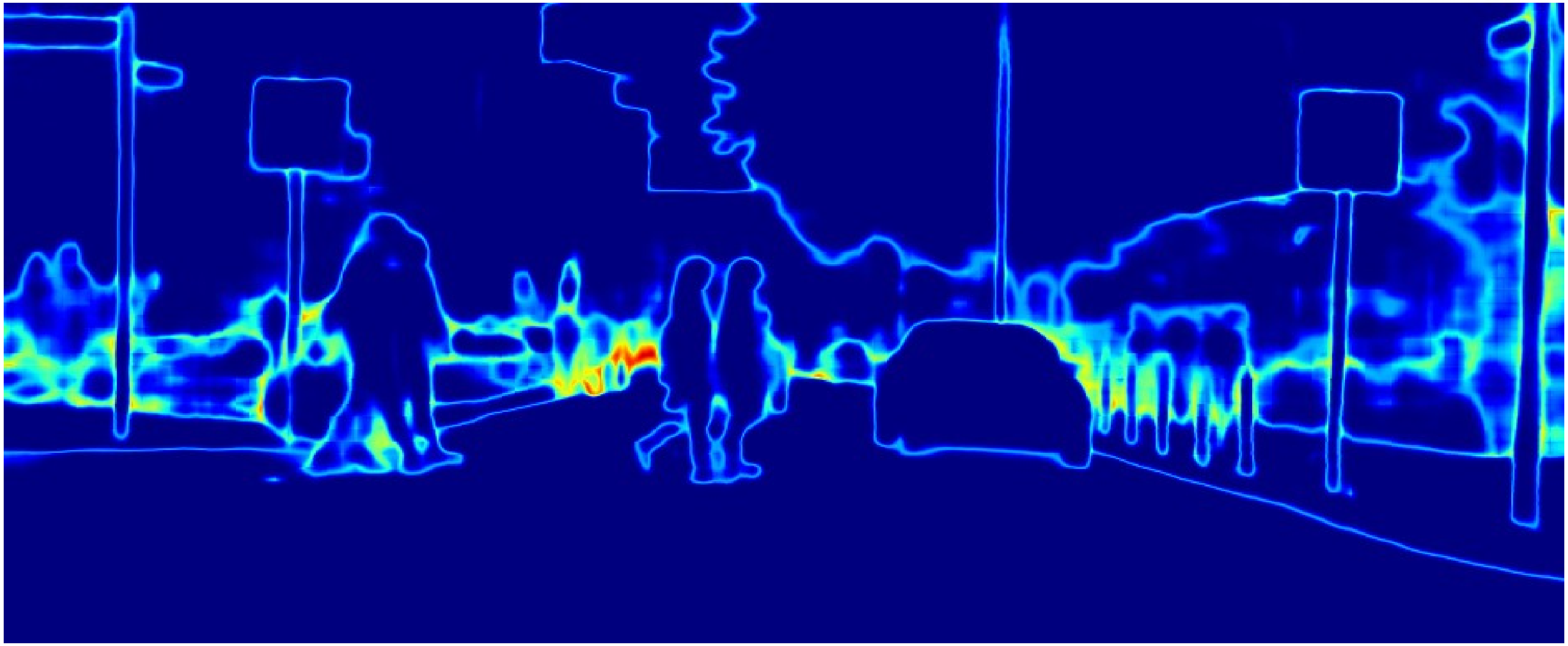


2.2 mln data points from 200 k individuals

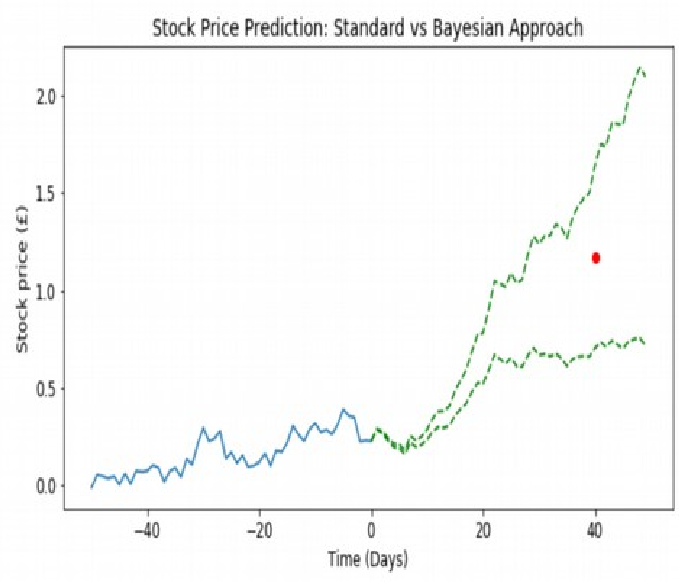


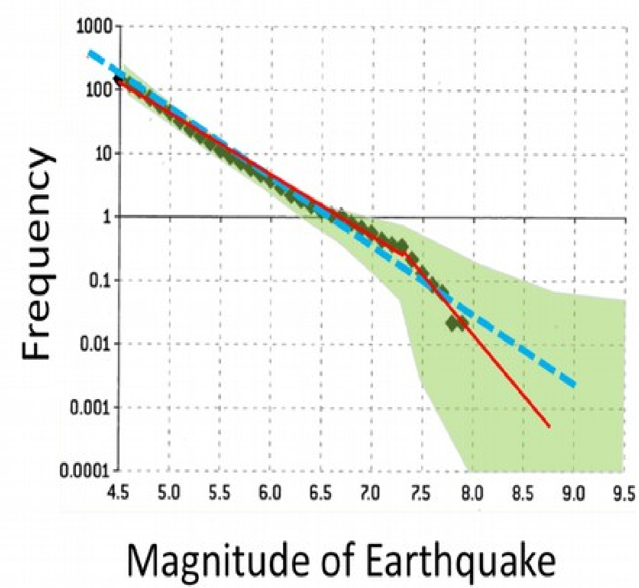


Deep Learning is not Good Enough

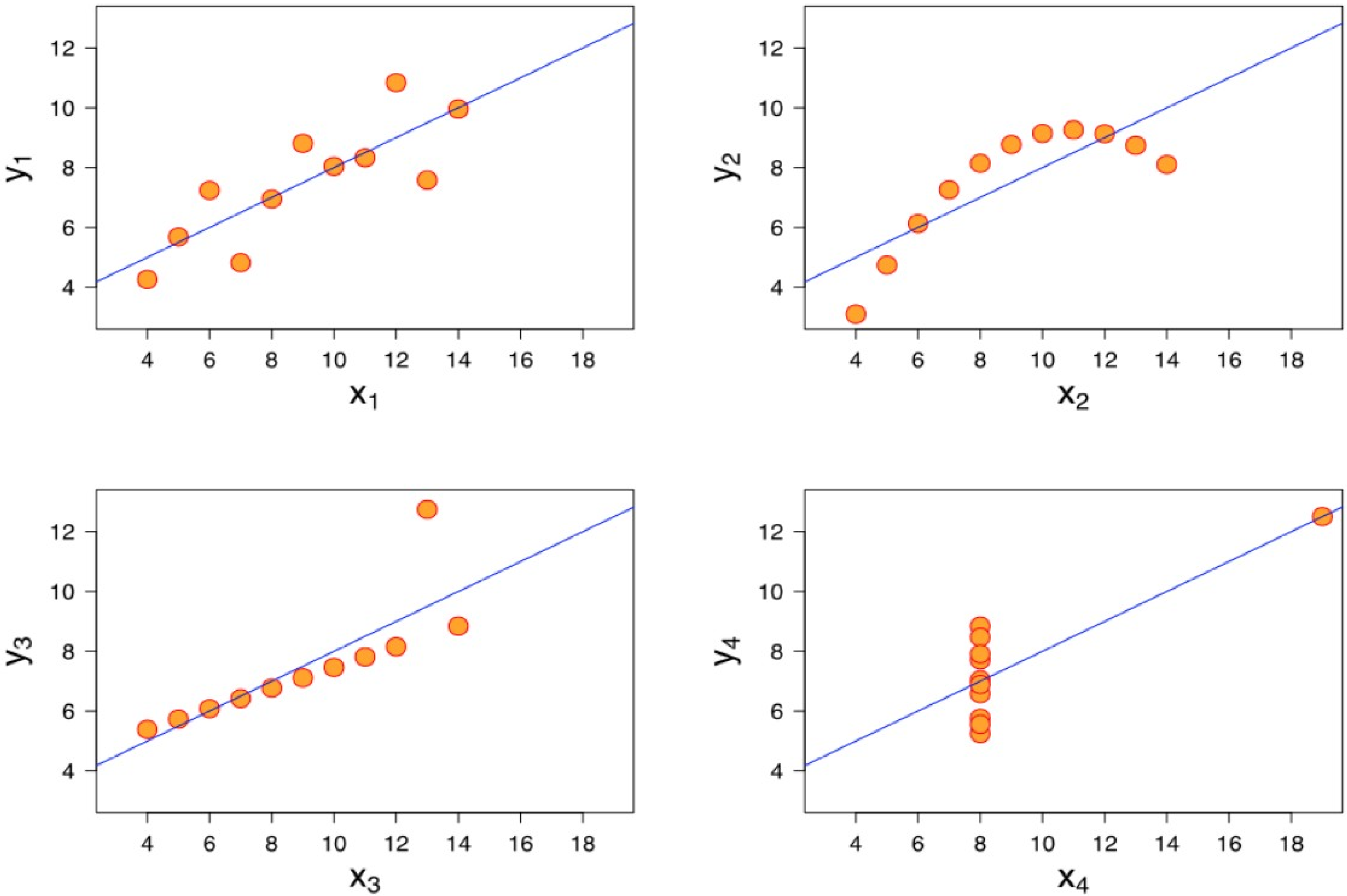


Intelligence is to know how much you do not know

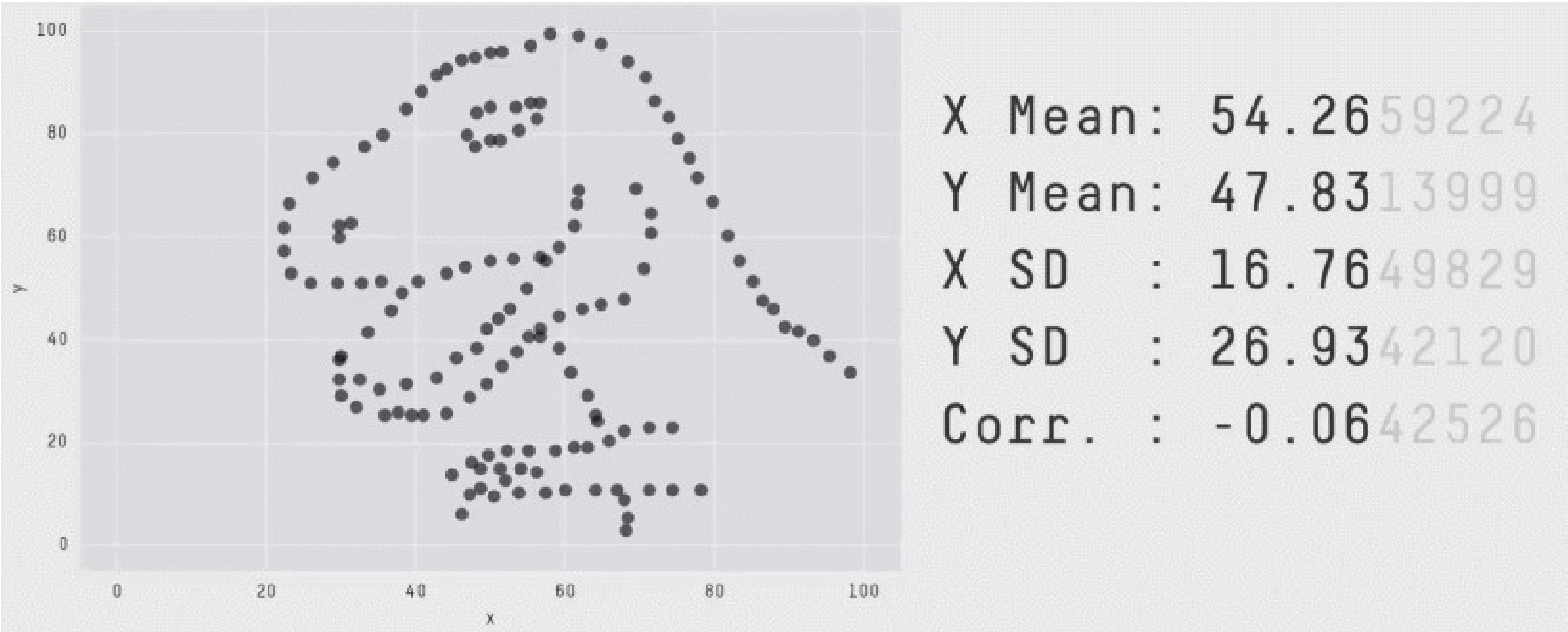


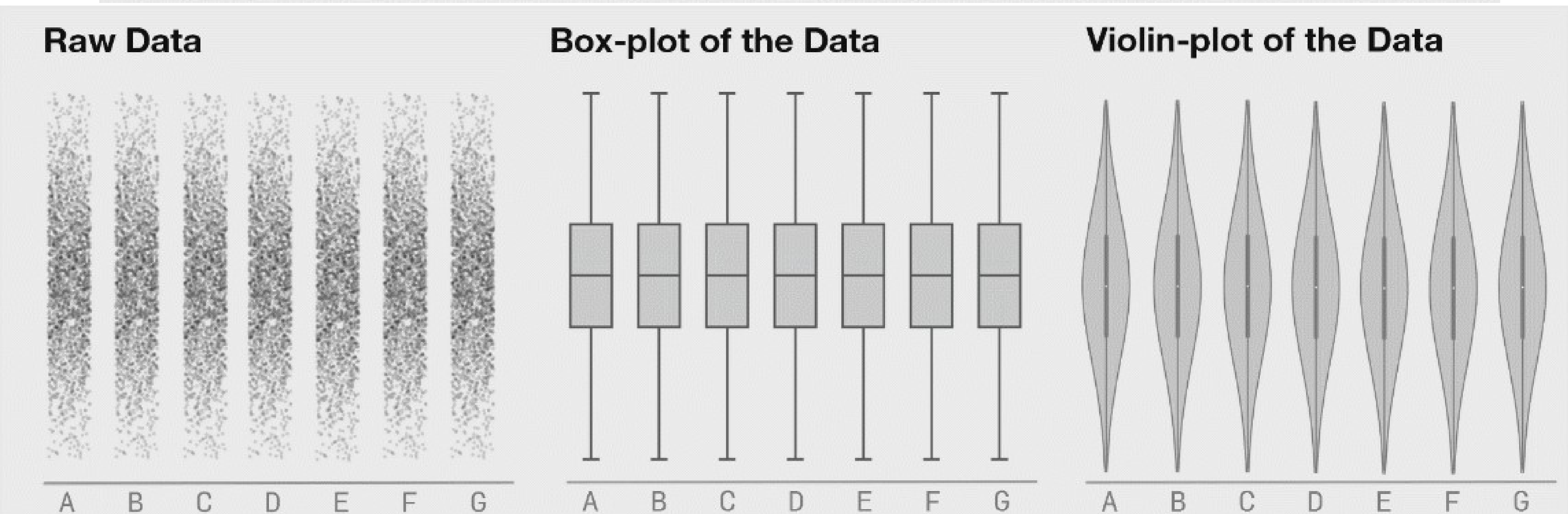


Frequentist Statistics Failure



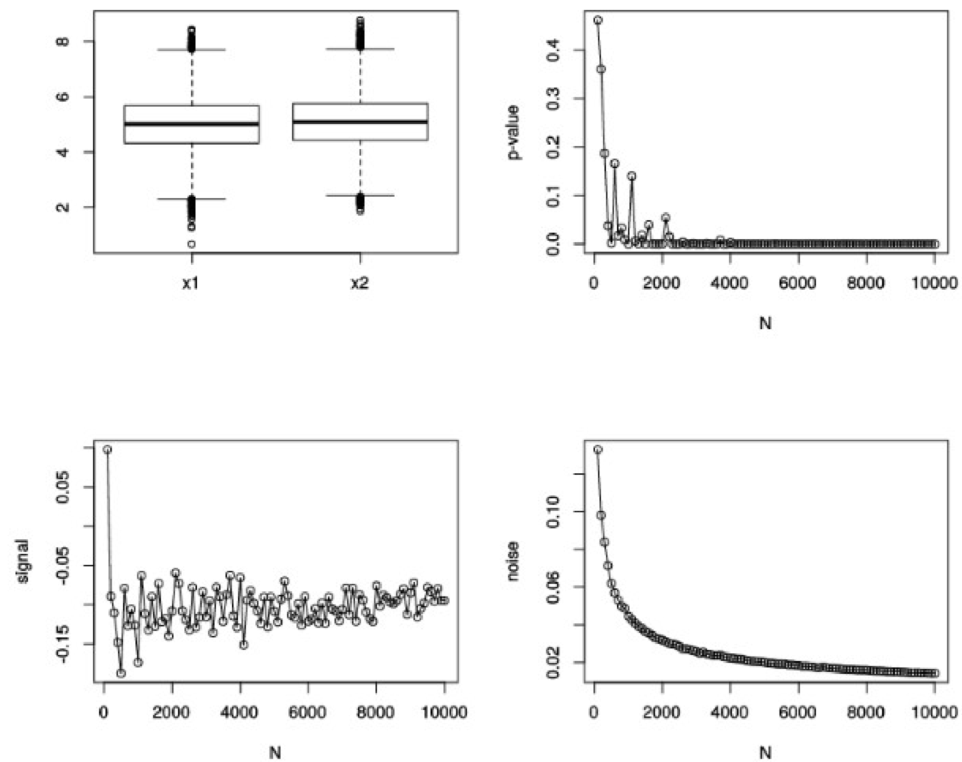
Why Frequentist Statistics is Brain Damaging



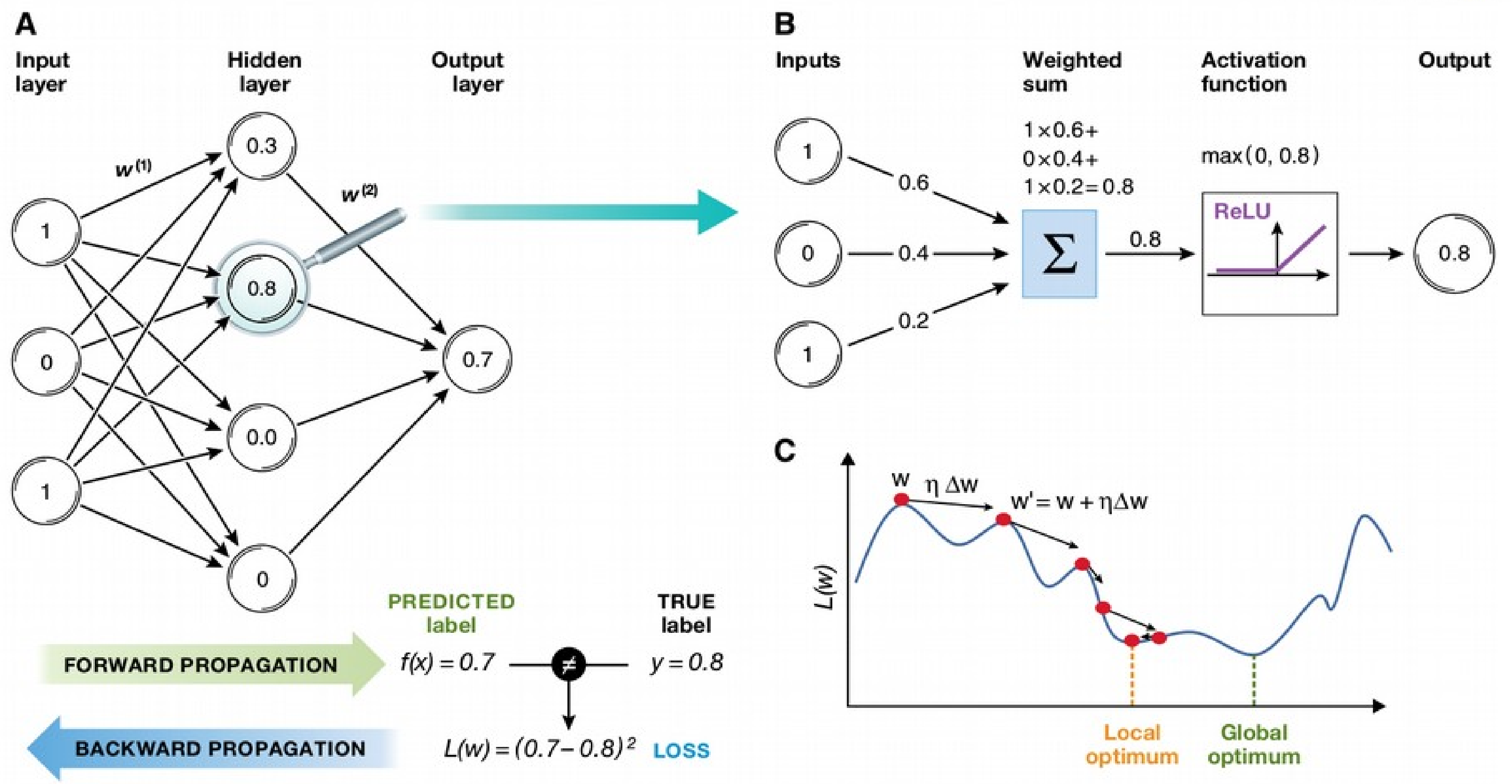


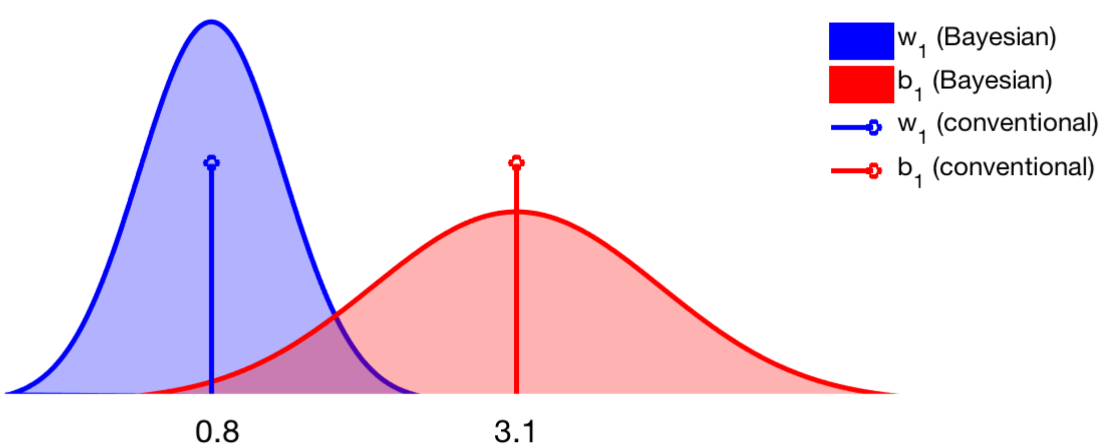
Pvalue is not good for ranking features


```r
FC <- 1.02
x_mean <- 5; x_sd <- 1
N_vector<-seq(from=100,to=10000,by=100)
x1 <- rnorm(N_vector, x_mean, x_sd)
x2 <- rnorm(N_vector, x_mean*FC, x_sd)
```

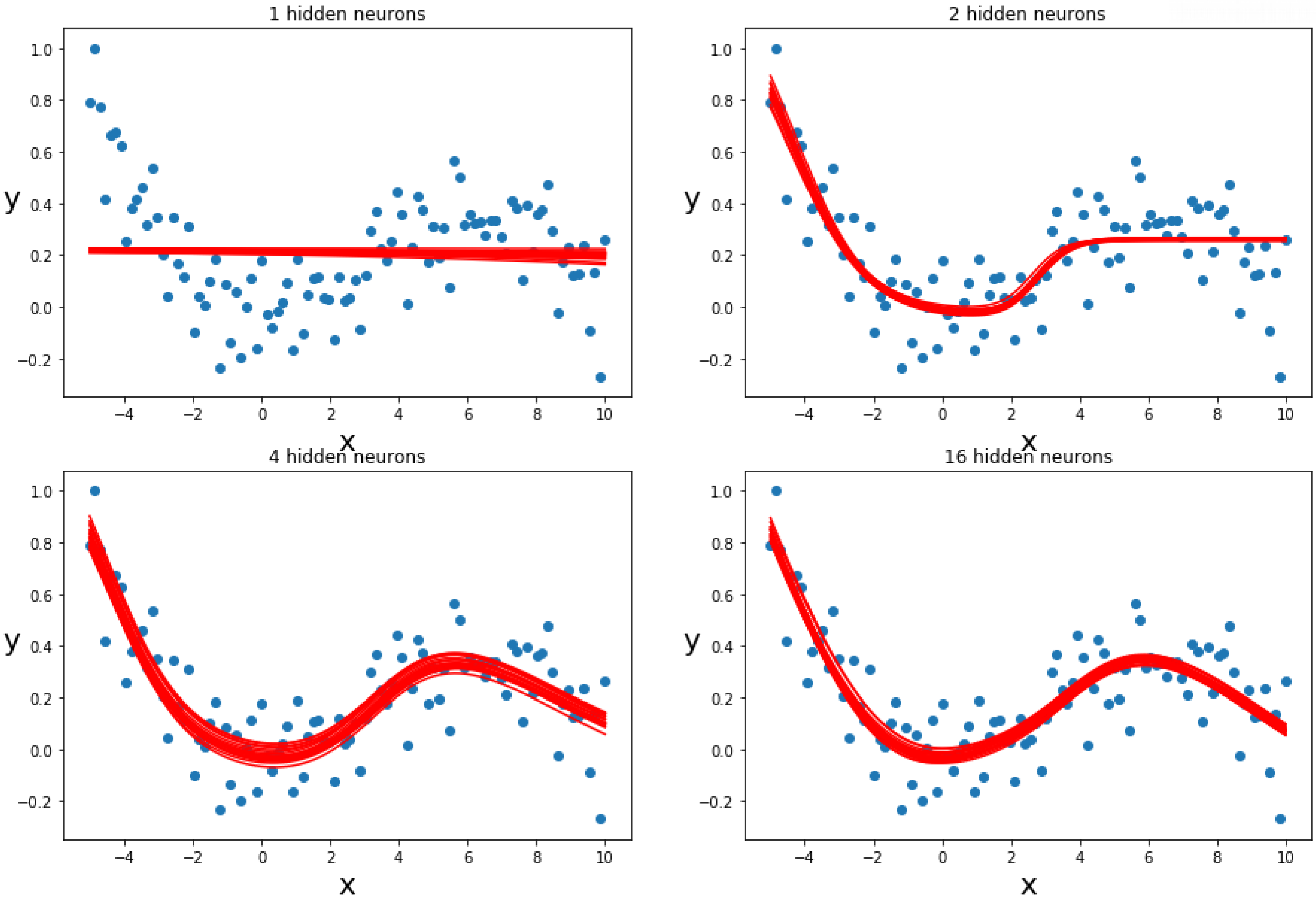


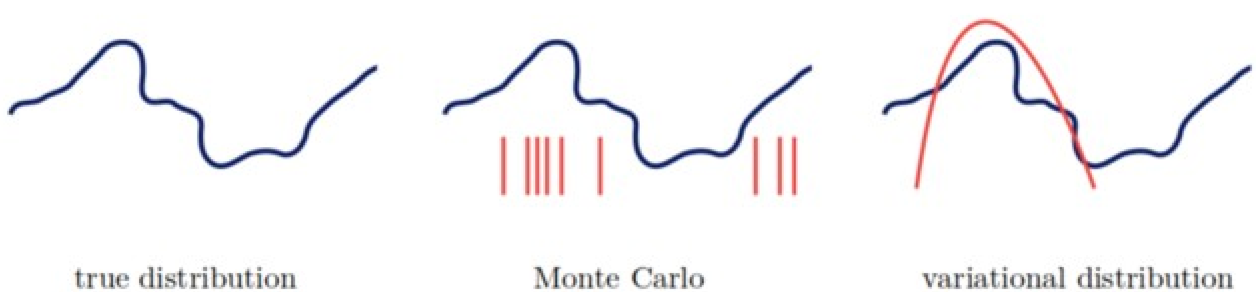
What is Bayesian Deep Learning?





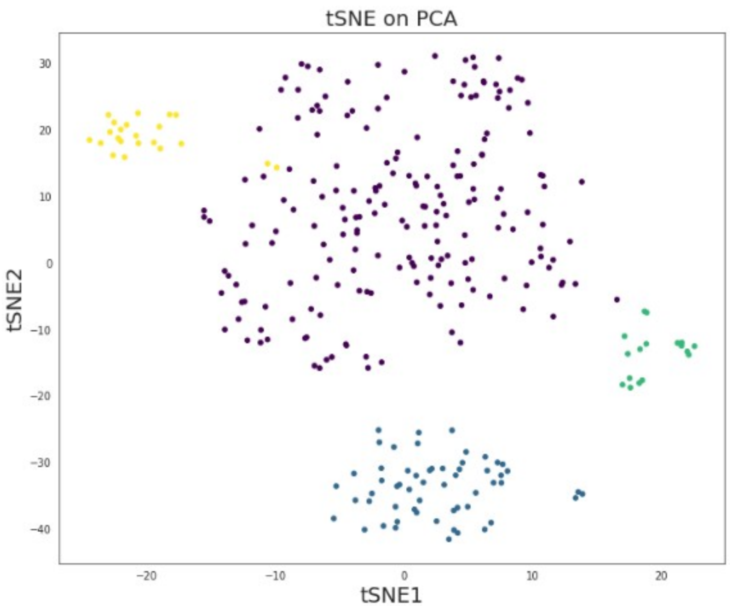
How Deep Learning Does Fitting

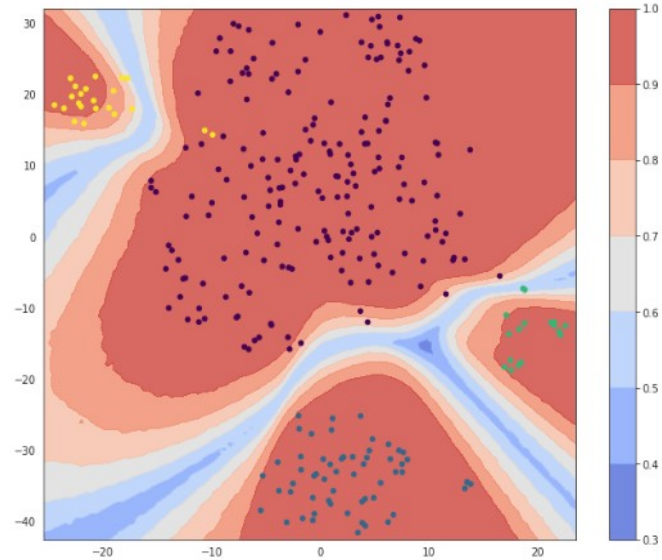


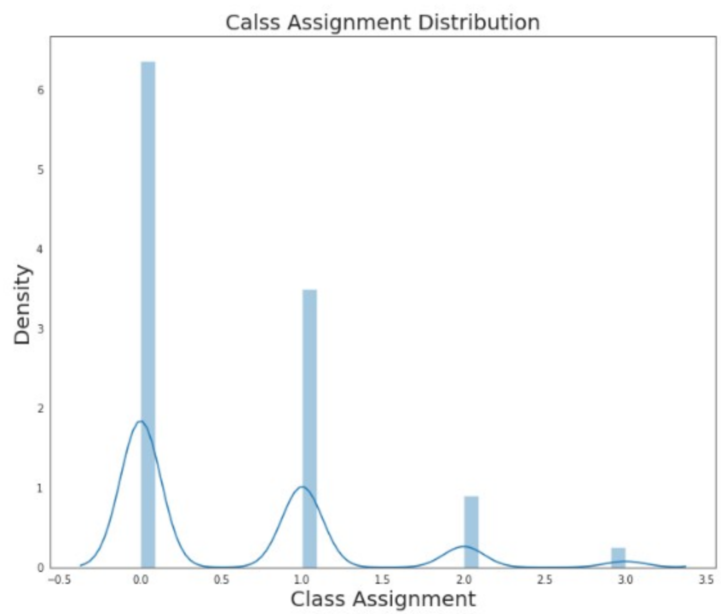


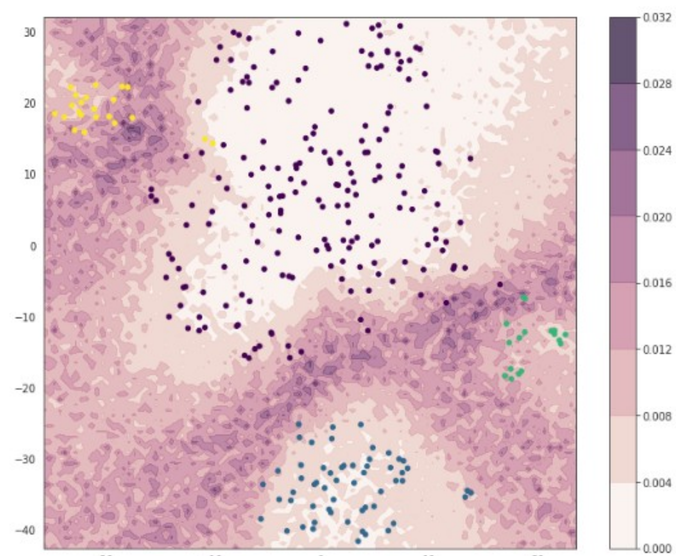
Bayesian Deep Learning

Superior for predictions on unseen data







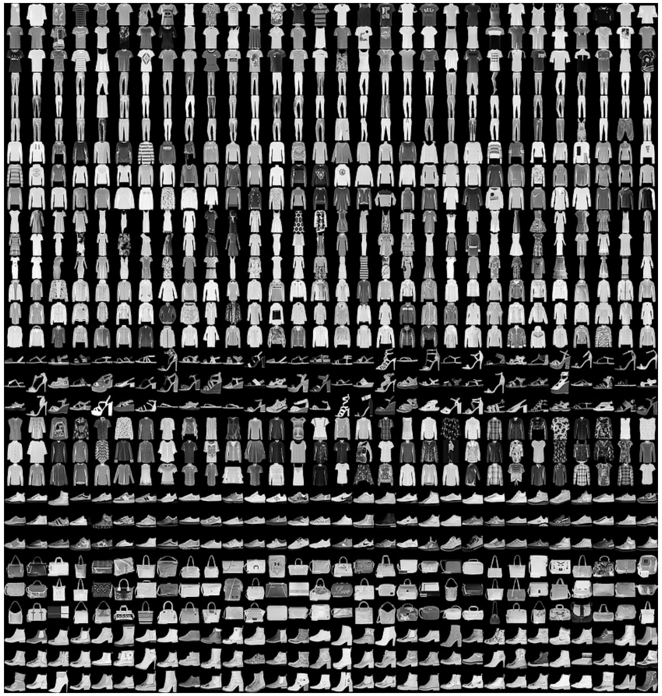


Bartoschek et al. 2018, Nature Communications, 9, 5150

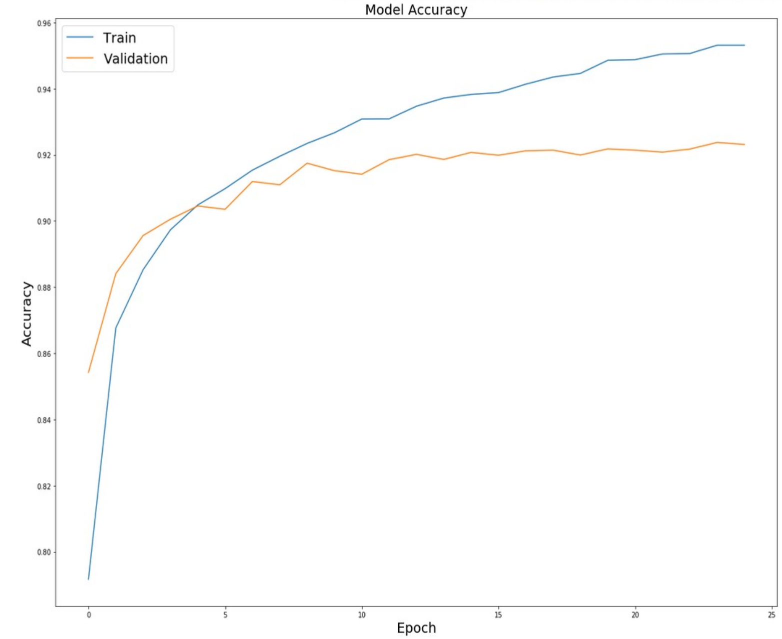
Uncertainties are crucial for Clinical Diagnostics

Frequentist Image Recognition

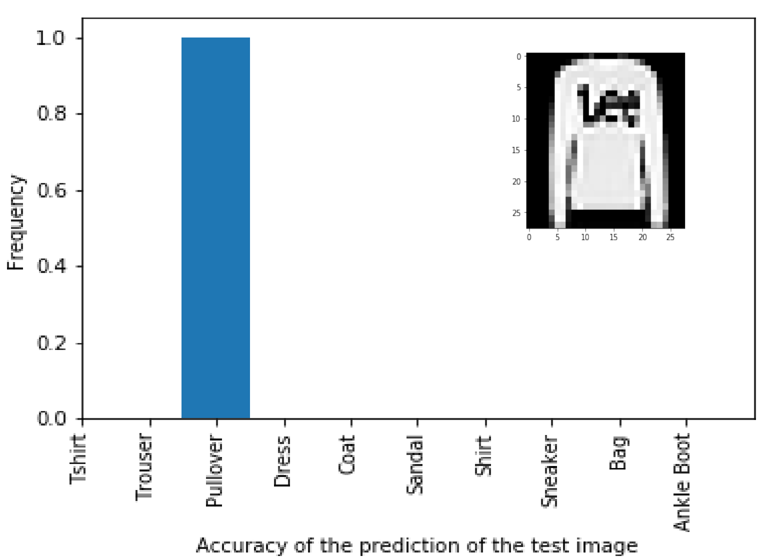
Fashion MNIST

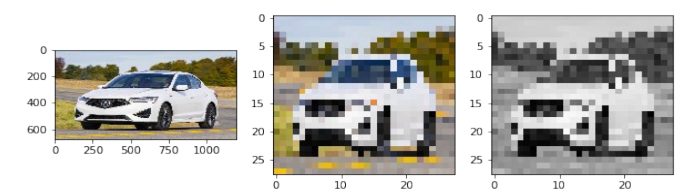


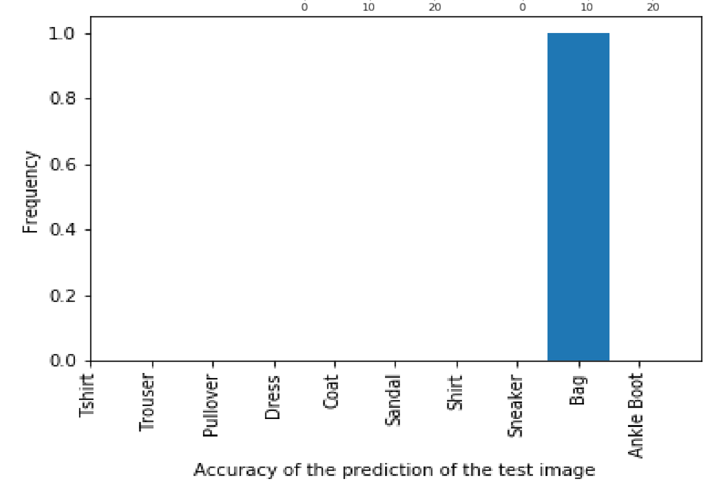
```
In [24]: # normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0

In [25]: # one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
print(num_classes)

10

In [27]: # Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(1, 28, 28), padding='same', activation='relu',
                 kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), padding='same', activation='relu',
                 kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
epochs = 25
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 32, 28, 28)        320
_________________________________________________________________
dropout_7 (Dropout)          (None, 32, 28, 28)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 28, 28)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 32, 14, 14)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_7 (Dense)              (None, 512)               3211776
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                5130
=================================================================
Total params: 3,226,474
Trainable params: 3,226,474
Non-trainable params: 0
_________________________________________________________________
None

In [28]: # Fit the model
#model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
history = model.fit(X_train, y_train, epochs = epochs, verbose = 1, validation_split = 0.25,
                    batch_size = 32, shuffle = True)

Train on 45000 samples, validate on 15000 samples
Epoch 1/25
45000/45000 [==============================] - 1158s 26ms/step - loss: 0.5762 - acc: 0.7917 - val_loss: 0.39
73 - val_acc: 0.8542
Epoch 2/25
45000/45000 [==============================] - 1124s 25ms/step - loss: 0.3643 - acc: 0.8676 - val_loss: 0.31
27 - val_acc: 0.8841
Epoch 3/25
45000/45000 [==============================] - 1158s 26ms/step - loss: 0.3129 - acc: 0.8853 - val_loss: 0.28
25 - val_acc: 0.8956
Epoch 4/25
45000/45000 [==============================] - 1609s 36ms/step - loss: 0.2813 - acc: 0.8973 - val_loss: 0.27
27 - val_acc: 0.9005
Epoch 5/25
45000/45000 [==============================] - 902s 20ms/step - loss: 0.2618 - acc: 0.9048 - val_loss: 0.258
8 - val_acc: 0.9045
Epoch 6/25
45000/45000 [==============================] - 936s 21ms/step - loss: 0.2451 - acc: 0.9098 - val_loss: 0.256
4 - val_acc: 0.9035
Epoch 7/25
```



Prediction





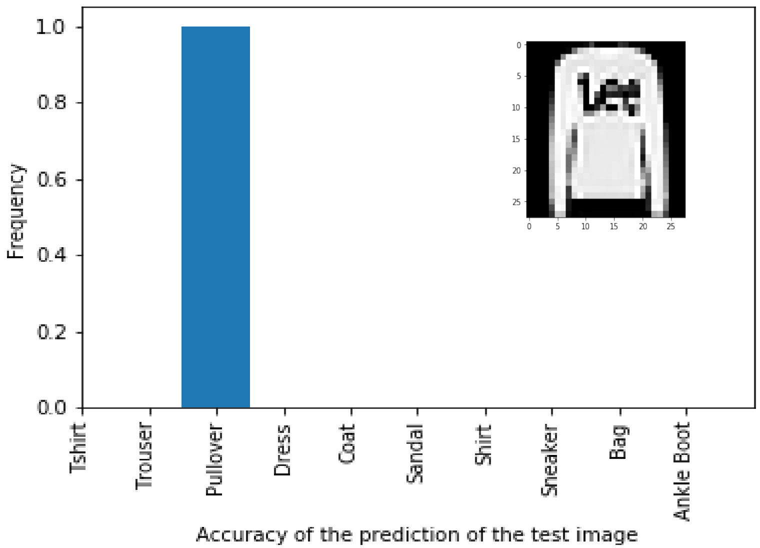


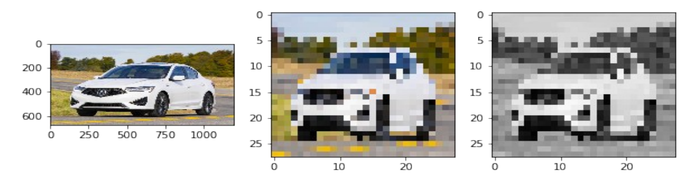
Bayesian Image Recognition

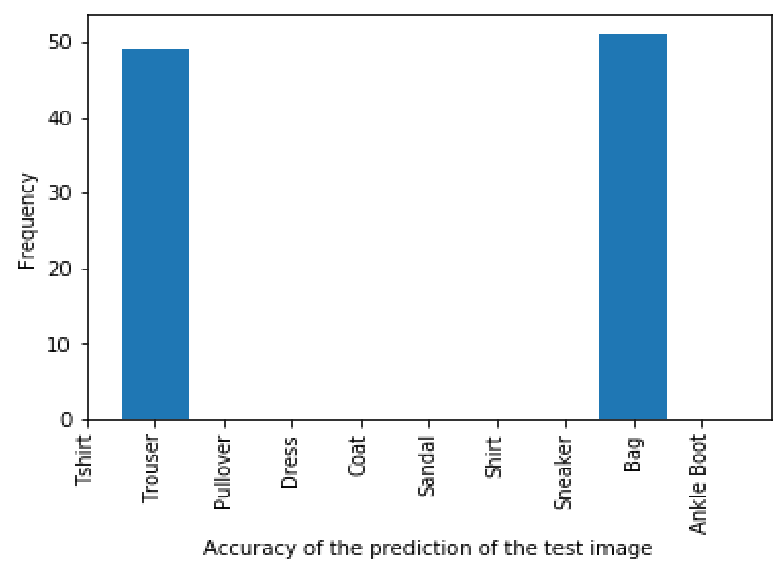
PyMC3, Edward, TensorFlow Probability

```
In [8]: x_train = x_train.reshape((x_train.shape[0],D))
        x_test = x_test.reshape((x_test.shape[0],D))
        print(x_train.shape)
        print(x_test.shape)

(60000, 784)
(10000, 784)

In [9]: from keras.utils import to_categorical
        y_train = to_categorical(y_train)
        y_test = to_categorical(y_test)
        print(y_train.shape)
        print(y_test.shape)

(60000, 10)
(10000, 10)

In [10]: ed.set_seed(314159)
         N = 100     # number of images in a minibatch.
         D = D       # number of features.
         K = 10      # number of classes.

         # Create a placeholder to hold the data (in minibatches) in a TensorFlow graph.
         x = tf.placeholder(tf.float32, [None, D])
         # Normal(0,1) priors for the variables. Note that the syntax assumes TensorFlow 1.1.
         w = Normal(loc=tf.zeros([D, K]), scale=tf.ones([D, K]))
         b = Normal(loc=tf.zeros(K), scale=tf.ones(K))
         # Categorical likelihood for classication.
         y = Categorical(tf.matmul(x, w) + b)

In [11]: # Contruct the q(w) and q(b). in this case we assume Normal distributions.
         qw = Normal(loc=tf.Variable(tf.random_normal([D, K])),
                       scale=tf.nn.softplus(tf.Variable(tf.random_normal([D, K]))))
         qb = Normal(loc=tf.Variable(tf.random_normal([K])),
                       scale=tf.nn.softplus(tf.Variable(tf.random_normal([K]))))

In [12]: def generator(arrays, batch_size = N):
             starts = [0] * len(arrays)  # pointers to where we are in iteration
             while True:
                 batches = []
                 for i, array in enumerate(arrays):
                     start = starts[i]
                     stop = start + batch_size
                     diff = stop - array.shape[0]
                     if diff <= 0:
                         batch = array[start:stop]
                         starts[i] += batch_size
                     else:
                         batch = np.concatenate((array[start:], array[:diff]))
                         starts[i] = diff
                     batches.append(batch)
                 yield batches
         cifar10 = generator([x_train, y_train], N)

In [13]: # We use a placeholder for the labels in anticipation of the traning data.
         y_ph = tf.placeholder(tf.int32, [N])
         # Define the VI inference technique, ie. minimise the KL divergence between q and p.
         inference = ed.KLqp({w: qw, b: qb}, data={y: y_ph})
         # Initialse the infernce variables
         inference.initialize(n_iter=50000, n_print=100, scale={y: float(x_train.shape[0]) / N})
         # We will use an interactive session.
         sess = tf.InteractiveSession()
         # Initialise all the vairables in the session.
         tf.global_variables_initializer().run()
         # Let the training begin. We load the data in minibatches and update the VI infernce using each new batch.
         for _ in range(inference.n_iter):
             X_batch, Y_batch = next(cifar10)
             #X_batch = X_batch.reshape(N,-1)
             # TensorFlow method gives the label data in a one hot vetor format. We convert that into a single label.
             Y_batch = np.argmax(Y_batch, axis=1)
             info_dict = inference.update(feed_dict={x: X_batch, y_ph: Y_batch})
             inference.print_progress(info_dict)

50000/50000 [100%] ██████████████████████████████ Elapsed: 221s | Loss: 85453.266

In [14]: # Generate samples the posterior and store them.
         n_samples = 100
         prob_lst = []
         samples = []
         w_samples = []
         b_samples = []
         for _ in range(n_samples):
             w_samp = qw.sample()
             b_samp = qb.sample()
             w_samples.append(w_samp)
             b_samples.append(b_samp)
             # Also compute the probabiliy of each class for each (w,b) sample.
             prob = tf.nn.softmax(tf.matmul(x_test, w_samp) + b_samp)
             prob_lst.append(prob.eval())
             sample = tf.concat([tf.reshape(w_samp,[-1]),b_samp],0)
             samples.append(sample.eval())

In [15]: # Compute the accuracy of the model.
         # For each sample we compute the predicted class and compare with the test labels.
         # Predicted class is defined as the one which as maximum probability.
         # We perform this test for each (w,b) in the posterior giving us a set of accuracies
         # Finally we make a histogram of accuracies for the test data.
         accy_test = []
         for prob in prob_lst:
             y_trn_prd = np.argmax(prob, axis=1).astype(np.float32)
             acc = (y_trn_prd == np.argmax(y_test, axis=1)).mean()*100
             accy_test.append(acc)

         plt.hist(accy_test)
         plt.title("Histogram of prediction accuracies in the CIFAR10 test data")
         plt.xlabel("Accuracy")
         plt.ylabel("Frequency")
         plt.show()
```

Prediction







National Bioinformatics Infrastructure Sweden (NBIS)