Image Classification and Convolutional Neural Network (CNN)

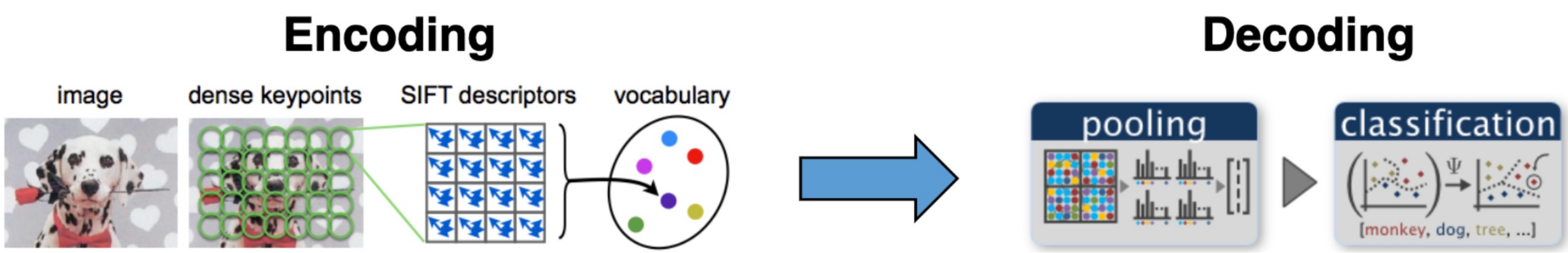
Sikan Li

Outline

- Introduction on basic ML/DL models
  - Multi-layer Perceptron (MLP)
  - Convolutional Neural Network (CNN)
- Introduction on PyTorch
  - DataLoader, Transform
- Case study: Natural Hazard Detection
  - Hands-on session

Traditional Methods

- Traditional image classification methods consist of two steps
    1. Feature extraction (encoding)
    2. Establishing a connection between features and image labels (decoding)



Typical Algorithms:

Scale invariant feature transform (SIFT)
Histogram of oriented gradients (HOG)
Binary robust invariant scalable keypoints (BRISK)

Typical Algorithms:

Support vector machines (SVM)
Decision trees
Neural networks

Source

What is Deep Learning?

- Traditional Approaches
  - Identify the features to solve a problem
  - Develop methods to extract these features
- Deep Learning Methods
  - Identifies the features it needs to solve a problem using the presented data

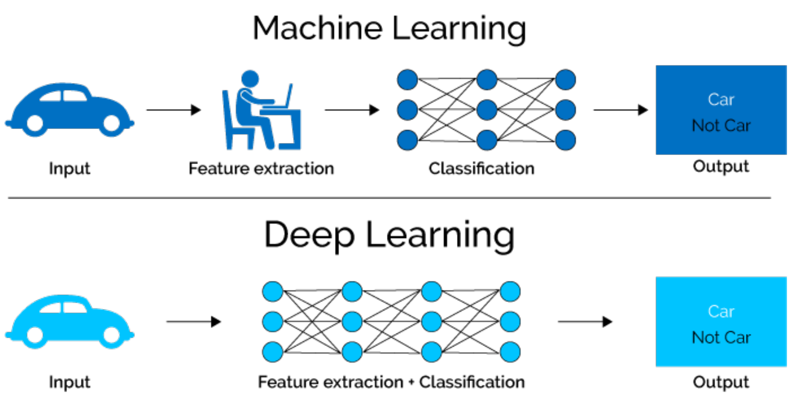


Image source

- Multi-Layer Perceptron (MLP)

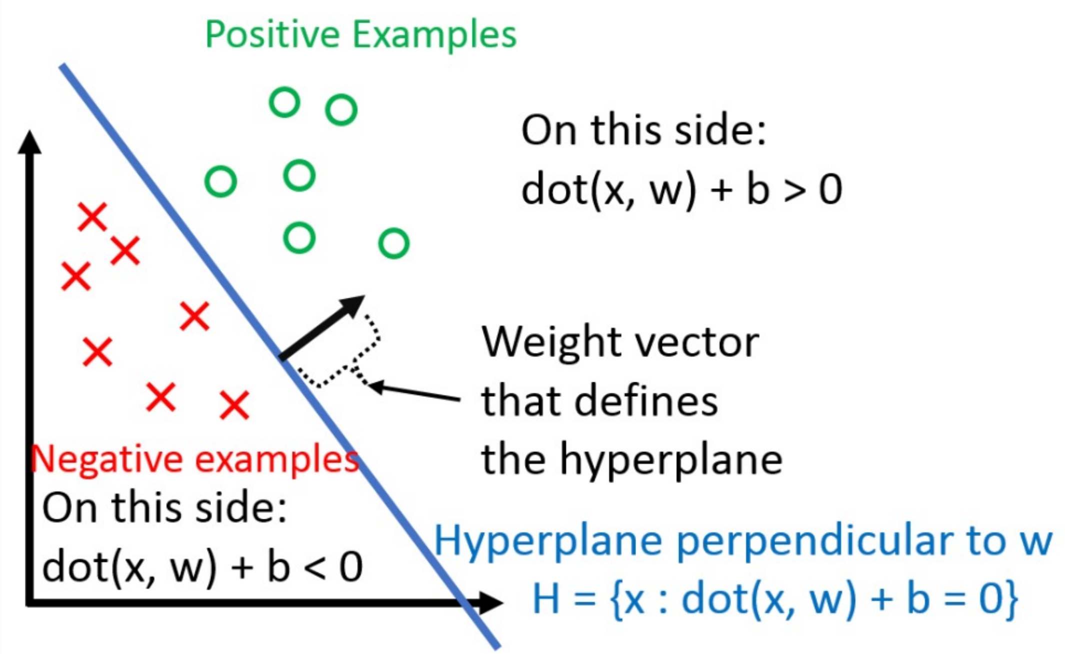
Perceptron

- MLP originates from the concept of perceptron.
- Suppose our dataset is linearly separable (i.e. there exits a hyperplane that can perfectly divide the dataset into two small groups) and has two classes with label +1 and -1, then perceptron can find such hyperplane in a finite number of steps.
- If the assumption is not held, perceptron will fail.

Perceptron

Classifier

$$h(\mathbf{x}_i) = \textbf{sign}(\mathbf{w}^\top \mathbf{x}_i + b)$$



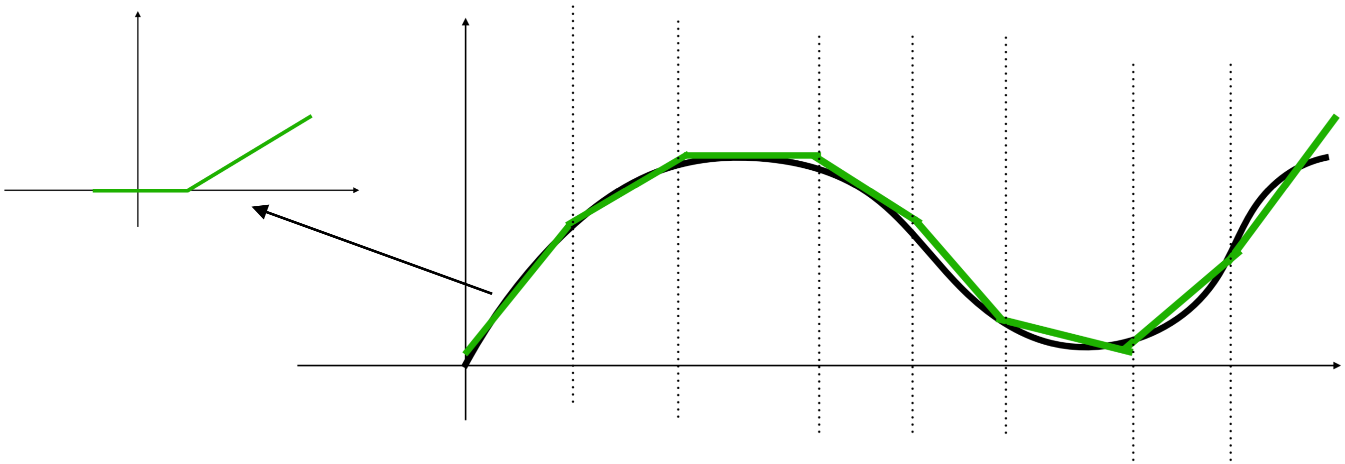
[Image source]()

Perceptron algorithm

```
Initialize w⃗ = 0⃗                          // Initialize w⃗. w⃗ = 0⃗ misclassifies everything.
while TRUE do                              // Keep looping
    m = 0                                  // Count the number of misclassifications, m
    for (x_i, y_i) ∈ D do                  // Loop over each (data, label) pair in the dataset, D
        if y_i(w⃗^T · x⃗_i) ≤ 0 then         // If the pair (x⃗_i, y_i) is misclassified
            w⃗ ← w⃗ + y x⃗                   // Update the weight vector w⃗
            m ← m + 1                      // Counter the number of misclassification
        end if
    end for
    if m = 0 then                          // If the most recent w⃗ gave 0 misclassifications
        break                              // Break out of the while-loop
    end if
end while                                  // Otherwise, keep looping!
```

From perceptron to MLP

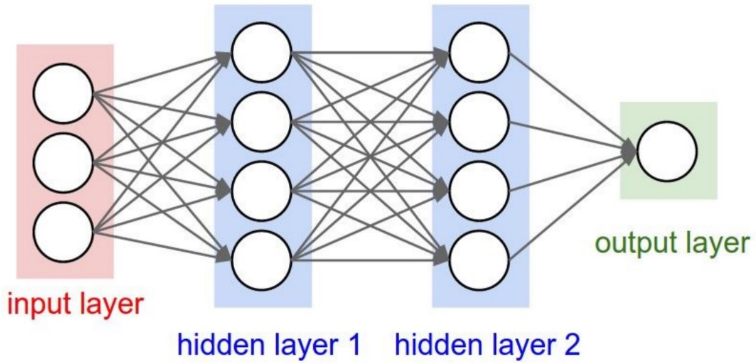
- When MLP is applied to a linear regression problem, it can be considered as building a piecewise linear function to simulate the target curve.



Multi-Layer Perceptron

- Layers in MLP are called linear layer or fully connected layer.
- Neurons in each layer are fully connected to the neurons in its following layer.
- MLP for binary classification is on the right.

A 3-layer neural network:



$w^{(i)}$: weight matrix of layer i
$o^{(i)}$: output of layer i
x: input vector(s), y: output scalar
σ: activation function (e.g. sigmoid, ReLU)

$$o^{(1)} = \sigma\left(\left(w^{(1)}\right)^T x\right)$$

$$o^{(2)} = \sigma\left(\left(w^{(2)}\right)^T o^{(1)}\right)$$

$$y = \left(w^{(3)}\right)^T o^{(2)}$$

Image source

MLP for multiclass classification

- For binary classification (i.e. dataset has only two possible labels), we can define one class having label 1 and the other class with label -1, and then use the sign of $y \in \mathbb{R}$ as the classification result (i.e. logistic reg.).
- For multiclass classification, we use one output per class that uses the softmax activation function.

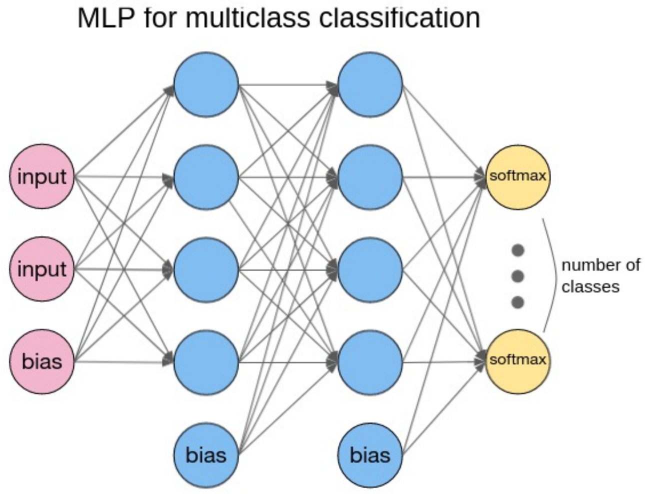


Image credit

Training and Testing

- Training
  - The process for making a model.
  - Training data is the data “observed” by the learning model

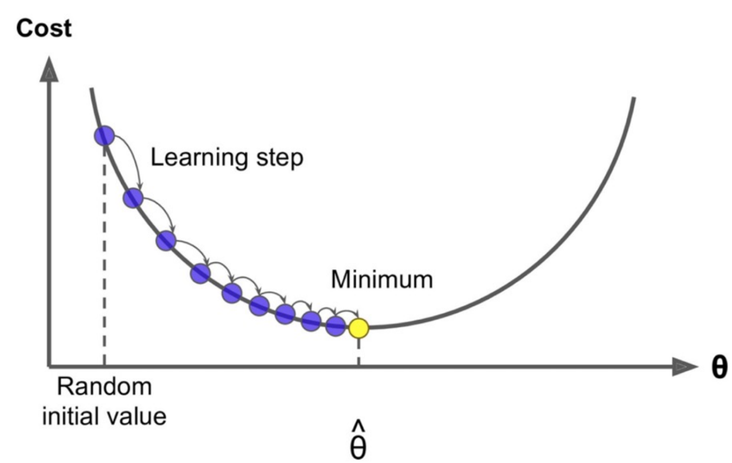
- Testing
  - Process for evaluating the performance of the model
  - Testing Data is data NOT observed by the learning model

- 80/20 split
  - 80% available data are randomly selected for training
  - 20% are used for testing

- Prediction
  - Apply model to data not in either training or testing data set
  - Assume the input data and its prediction are from the same process producing the training data.

Training Models by Minimizing "errors"

- A basic idea in many ML is to minimize loss function
- Numerical optimization methods like gradient descent are commonly used to find optimal values for w
- $f(w_t; x) = \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i)$
- $w_{t+1} = w_t - \alpha \nabla f(w_t; x)$



Backpropagation

- Fortunately, manual derivation for gradients of different neural networks is not needed.
- All ML/DL frameworks (e.g. PyTorch and Tensorflow) have auto-derivation engine, which is usually implemented based on computation graph and backprop.

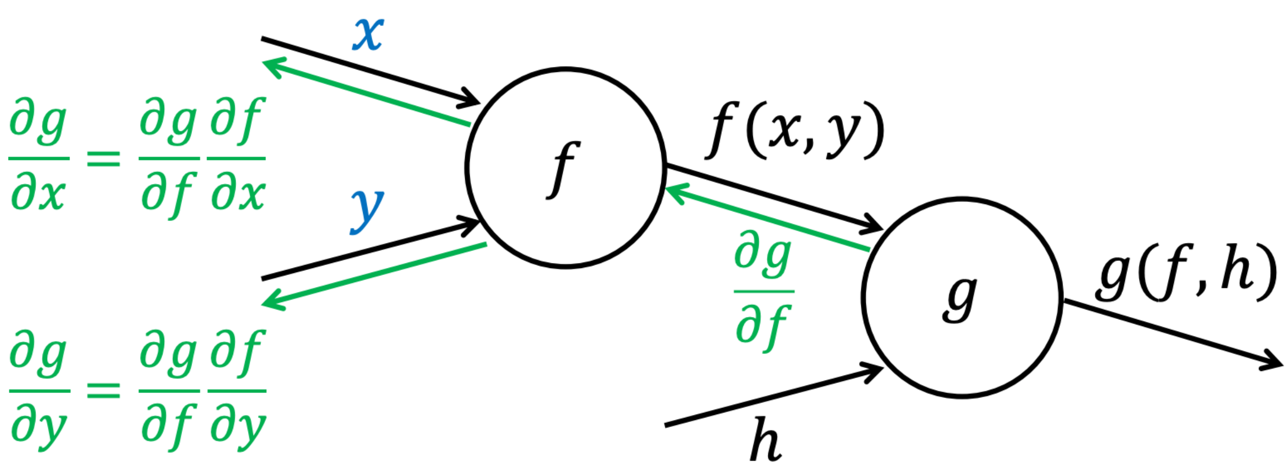


Image credit: Cornell ECE 5775 lecture14

MLP w/ Cross Entropy Loss: Softmax Regression

- Each neuron in the output layer generates the possibility for the corresponding class.
- The one with the highest possibility is the prediction label generated by the model.
- Softmax operation convers MLP's output to possibility.
- $p_i = (\text{softmax}(\mathrm{x}))_i = \frac{\exp(x_i)}{\sum_{j=1}^{c} \exp(x_j)}$
- Cross entropy loss: $L(x, y) = -\sum_{i=1}^{c} \underbrace{t_i}_{\text{True class distribution}} \log(\underbrace{p_i}_{\text{Predicted class distribution}})$

Underfitting vs Overfitting

- Underfitting,
  - Model cannot reflect all the relations from training data
  - Model performs poorly on training and testing data. “failed to learn”
  - Solutions: More features, more complex model, kernel method, boosting…
- Overfitting,
  - Model may lead to poor generalization on new testing data.
  - Performs very well on training data but poorly on testing data
  - Solutions: less complex model, more data, bagging…
- Hyperparameters impact under or over-fitting

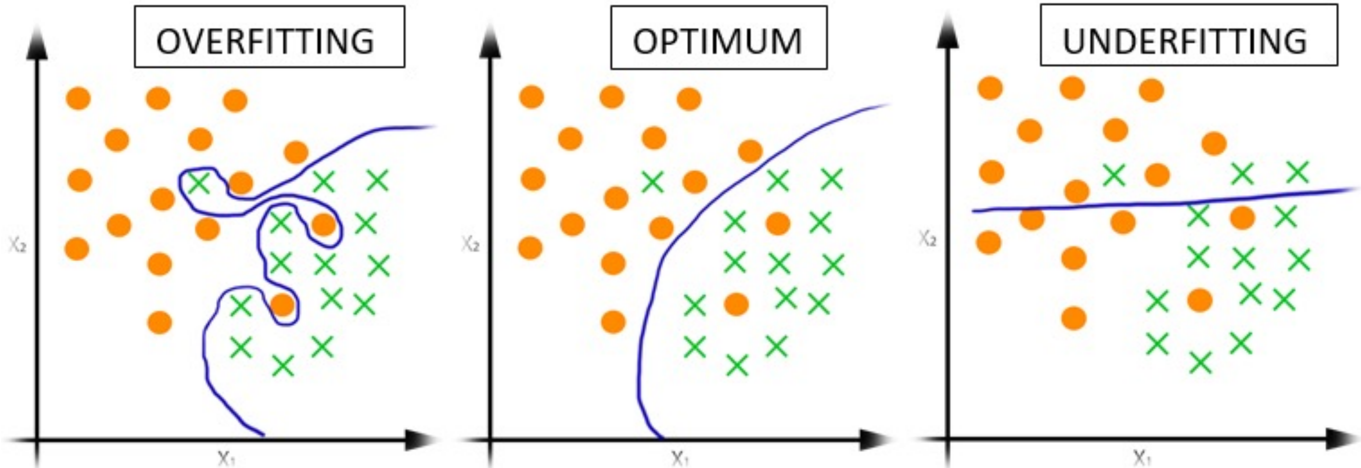


Image credit

- Convolutional Neural Network (CNN)

From MLP to CNN

- MLP can be trained to classify simple image dataset, such as MNIST dataset which contains handwritten digits and each image has 28 pixels in width, 28 pixels in height, and 1 greyscale channel.
- However, as for more complex image recognition tasks, its performance is not so good.

From MLP to CNN

- The main reason is that the fully connected layers fail to learn the spatial information.
  - Objects of a particular class usually has specific pixel patterns gathered.
  - When the object moves to another place in the image, model should still be able to recognize it, even it might not have seen such training image before.

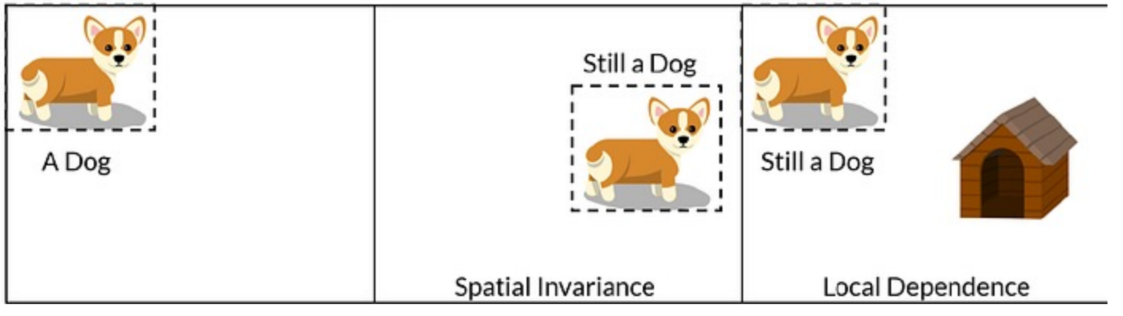
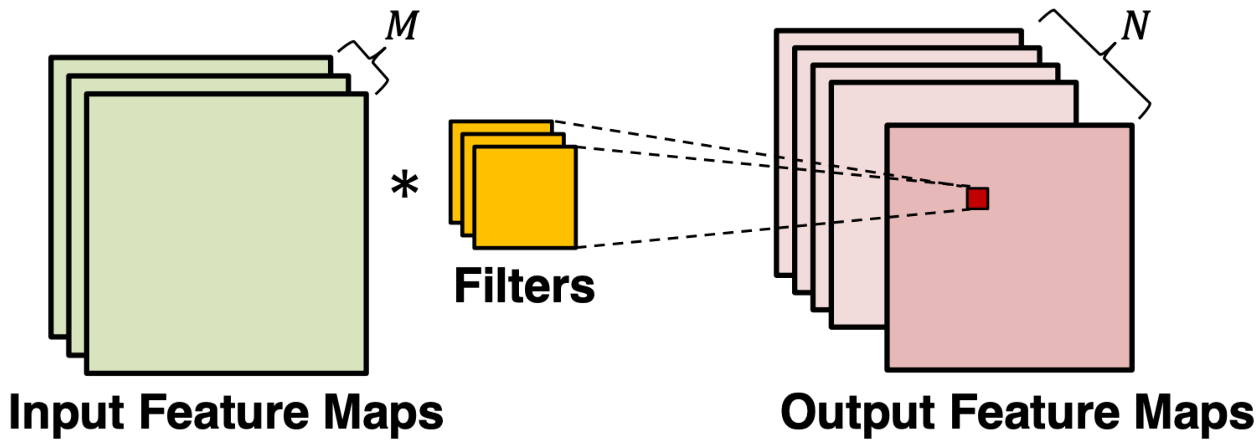


Image credit

Convolutional Neural Network

- Convolution on images



- Convolution layers ensures spatial invariance property
  - No matter where the object is, the model can detect it correctly
- RGB images are 3D tensors: $h \times w \times 3$
- Conv. on an image is done by swiping the kernels (usually a 3x3 matrix) through all pixels

Image credit: Cornell ECE 5775 lecture14

How Conv. Works

- Conv. on an image is done by swiping the kernels (usually a 3x3 matrix) through all pixels

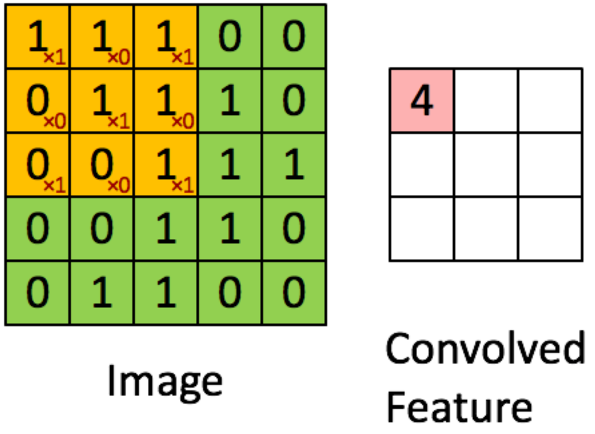


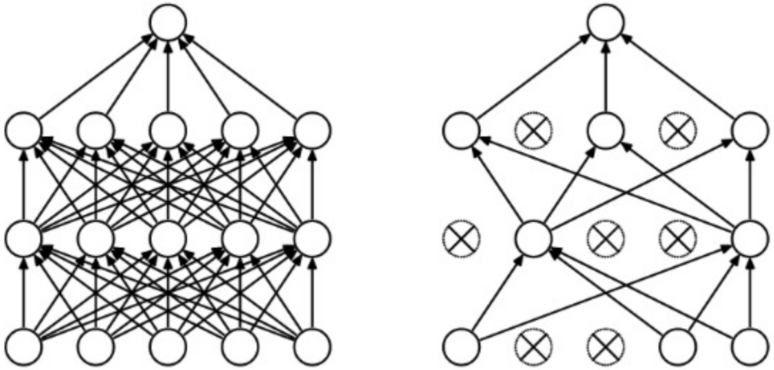
Image credit

Pooling Layer

- Convolution exploits the spatial information of pixels, but we still have the problems when object is moved, rotated, or shifted in the image.
- One common approach to cope this issue is to down sample the information from convolution results.
- Thus, only the large or important structural elements are preserved and then passed to the following layers.
- In CNN, we can down sample by using pooling layer.
- Common pooling operations: **max**, min, avg.

Dropout layer

- Dropout layer reduces the problem of overfitting.
- Given a probability of dropout, it randomly drops some neurons from the fully-connected layer.
- Adding such randomness into the network can increase its generalization to different situations, as it has been proven to be an effective regularization algo.



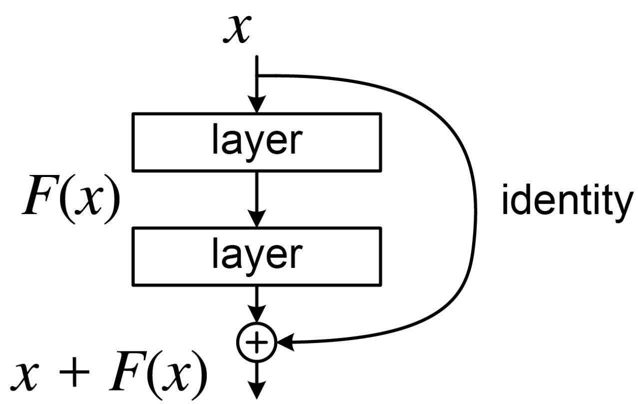
(a) Standard Neural Net (b) After applying dropout.

Dropout Layer

- Dropout only happens during the training stage.
- When we train the model, if a neuron is selected by the dropout layer, it will not contribute to the following layers, its gradient will not be computed, and its parameter will not be updated.

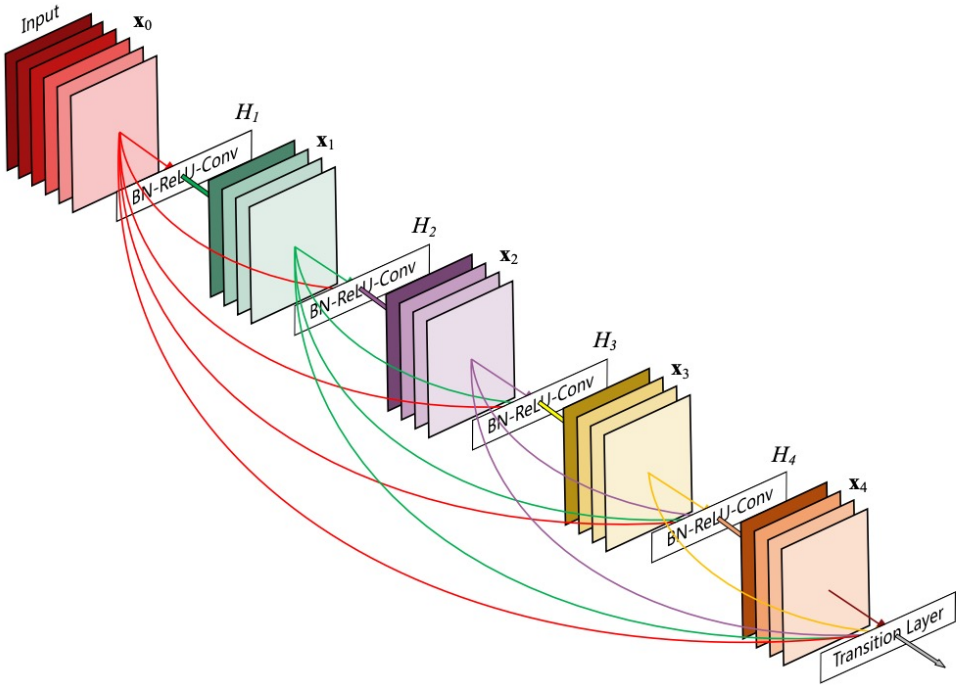
Residual Connection

- Deep CNN models usually have worse performance.
- This is not caused by overfitting, and adding more layers can lead to higher training error.
- One explanation is that during training process, gradient vanishing happens in some layers, causing all its preceding layers fail to learn.



Residual Connection

- Residual connection can help reduce model complexity.
- DenseNet, in which all the layers are connected through the residual signal, can achieve the same prediction accuracy with a smaller CNN model.



Data Augmentation

- What if our CNN model overfits the training dataset, and we are not able to collect more data?
- A simple solution is to just “make up” more data.
- This is the idea behind data augmentation.
- Every time we sample an image from the dataset, we randomly perform some operations on them.
    - For instance, shift, rotate, horizontal and vertical flip, clip..
- PyTorch has implemented such operations for us.

Transfer Learning

- Usually, the larger the model is, the better performance it can offer, if it does not overfit.
- To train large model while not causing overfitting, we need large dataset.
- However, for specific tasks, like natural hazard detection, there might not be enough images, so training large model with such small dataset will cause overfitting.
- A work around is to use transfer learning.

Transfer Learning

- We can use large model and train it again on our small dataset.
- To adapt large model to our small dataset with fewer number of classes, modifications on the “classification” block is usually needed.

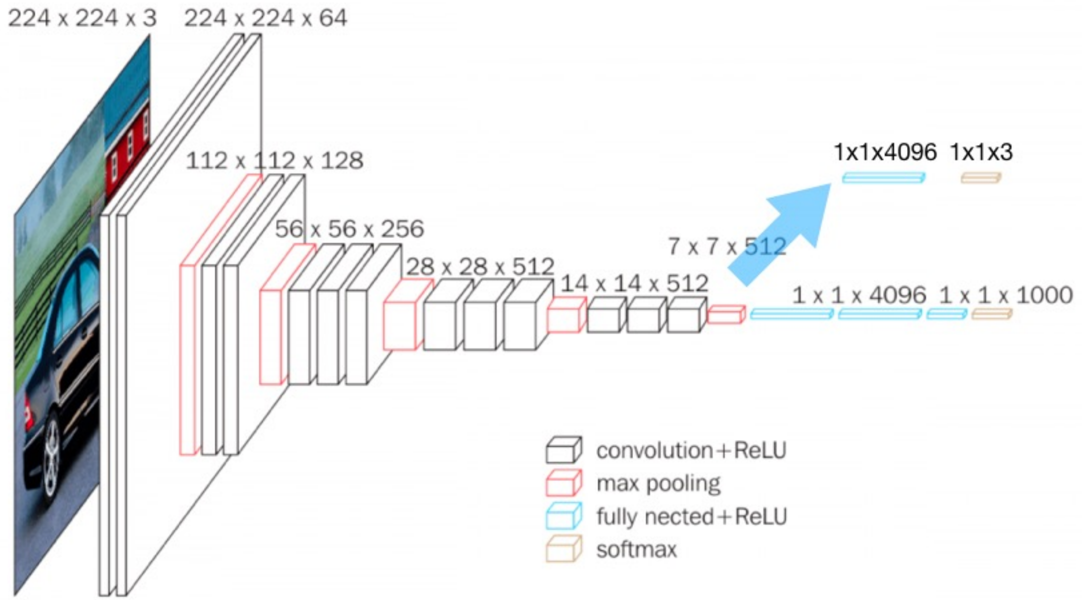
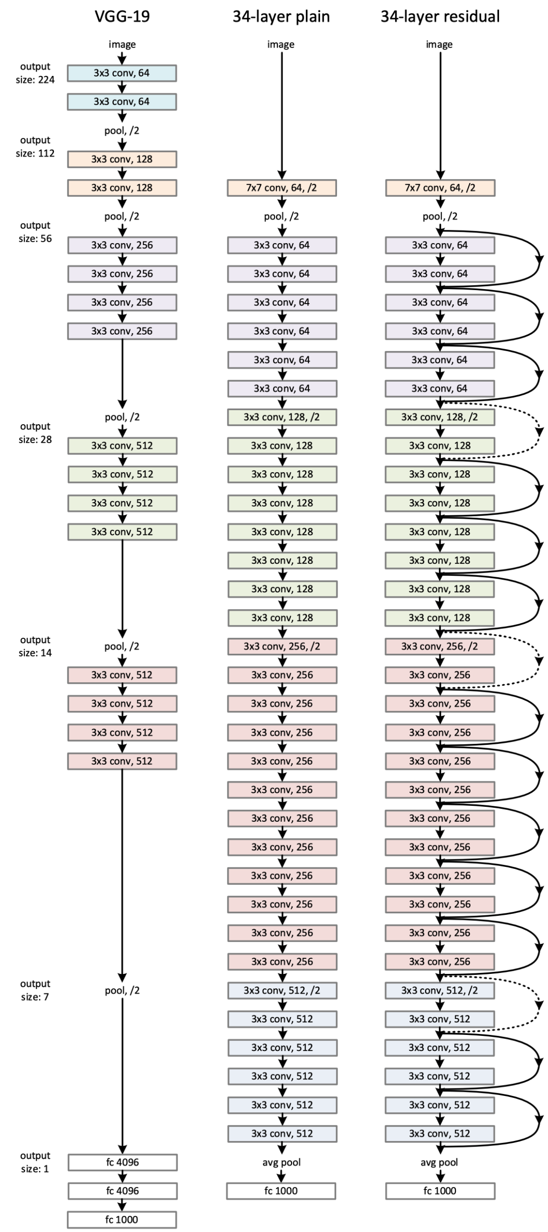


Image credit

Classic CNN Architectures

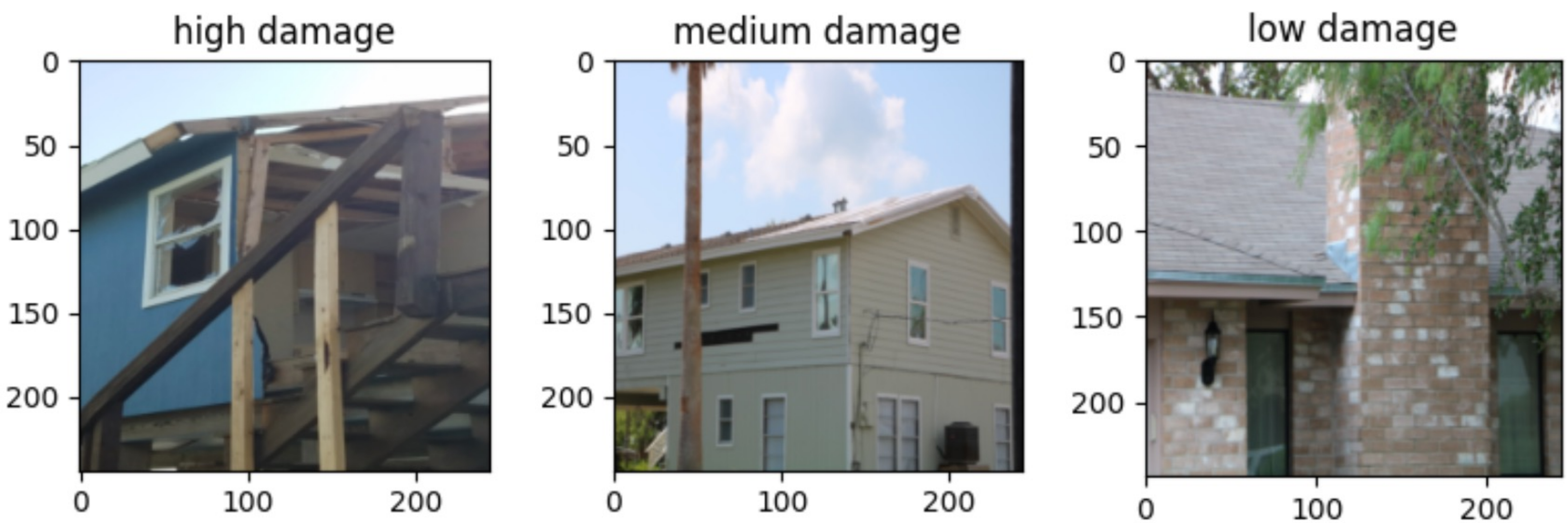
[ResNet]



Case Study: Natural Hazard Detection

Exercise

Image Classification with Hurricane Harvey Dataset

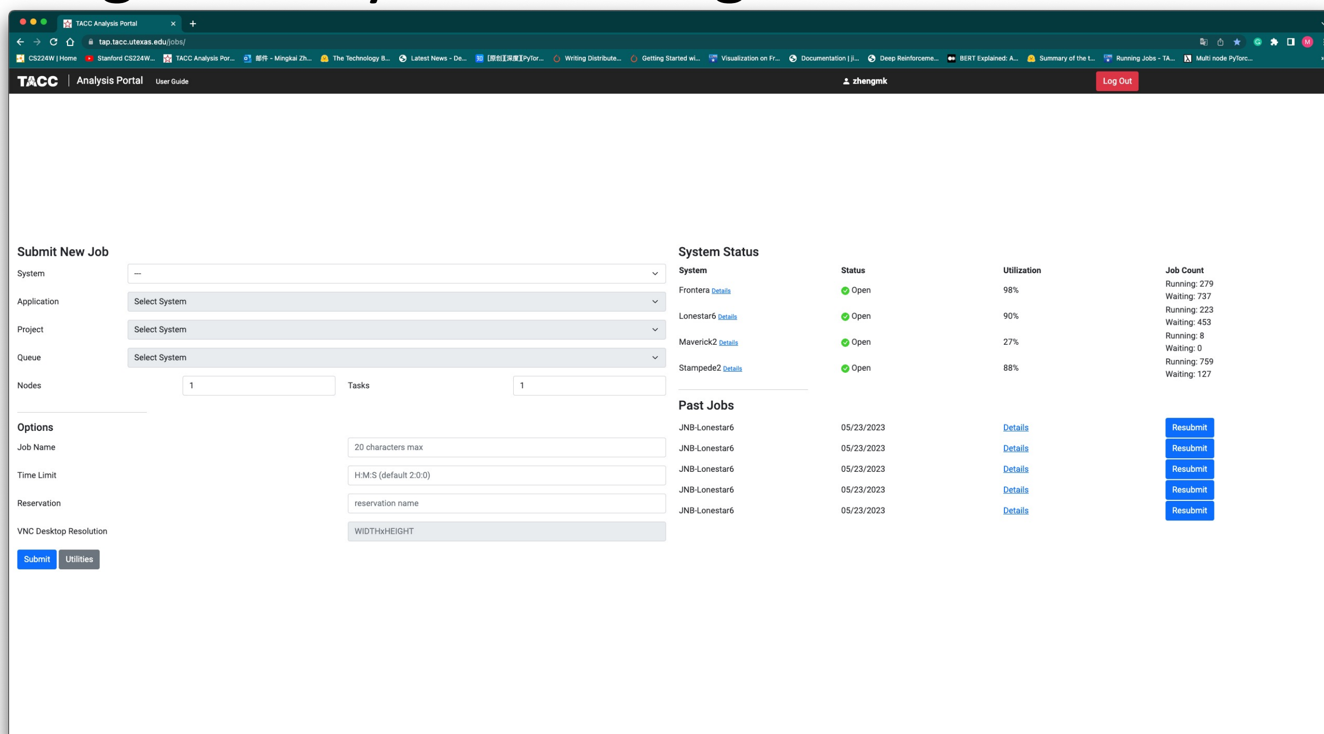


Building and Evaluating Classification Models

- In this exercise you will run two notebooks which build and evaluate CNN image classification models.

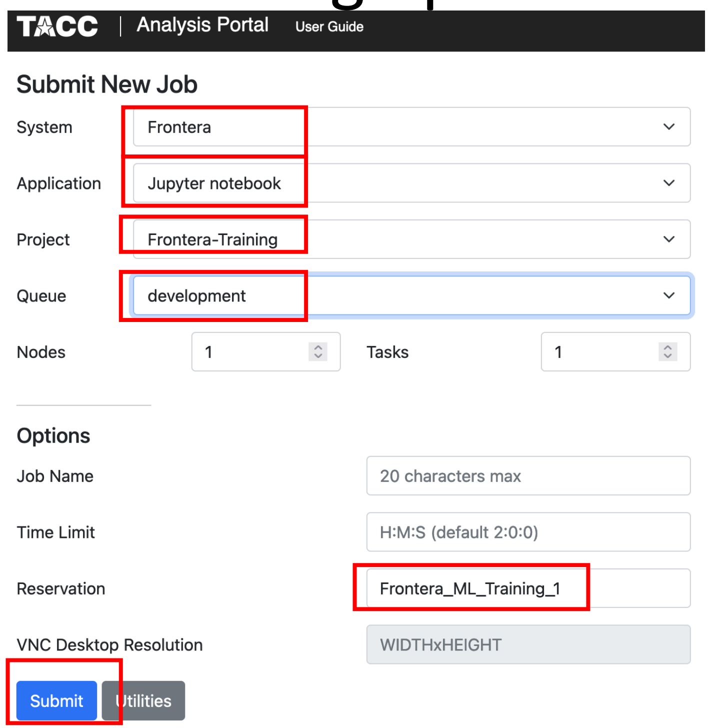
How to acquire TACC resources

- **Use [Analysis Portal]**
- Login with your training account



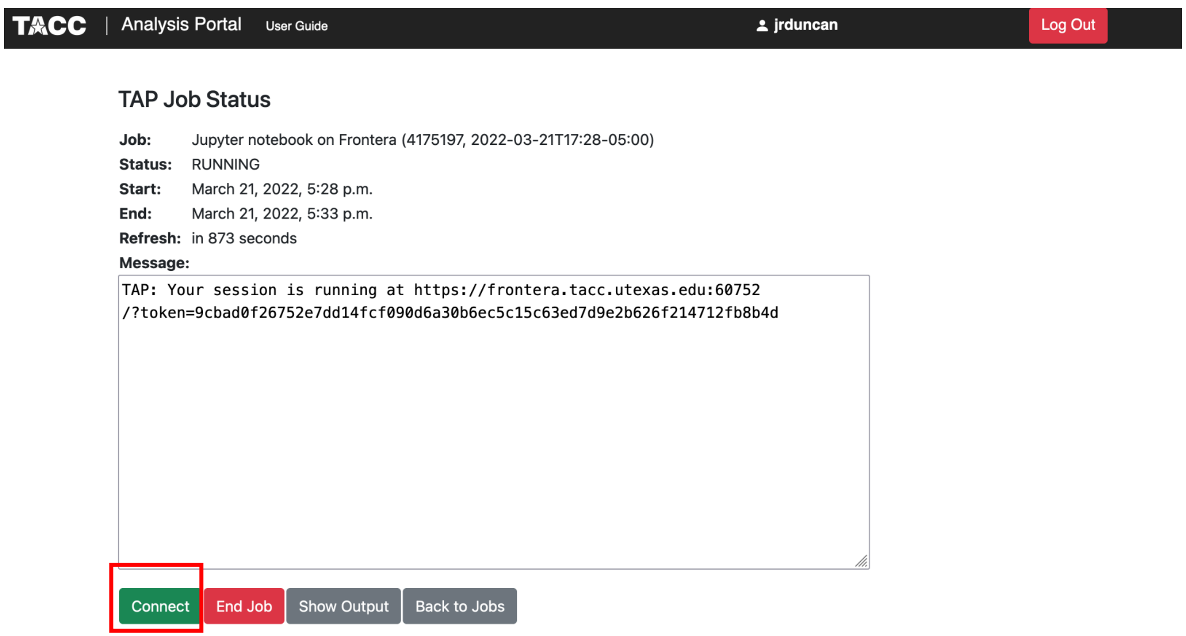
How to acquire TACC resources

- **Use [Analysis Portal]**
- Select the following options



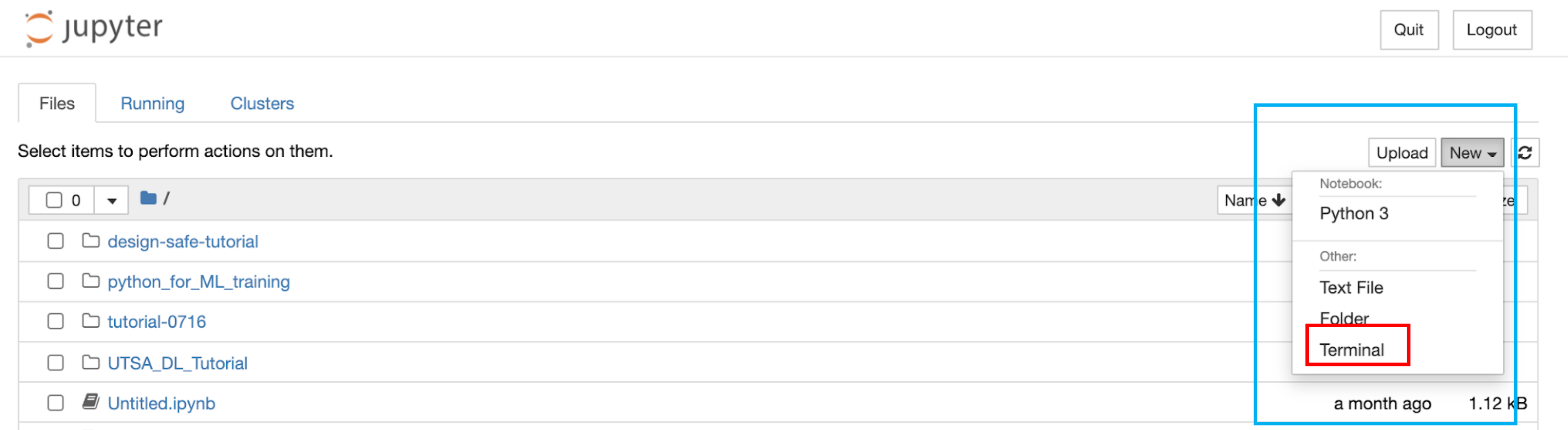
How to acquire TACC resources

- **Use [Analysis Portal]**
- Connect to the compute node(s)



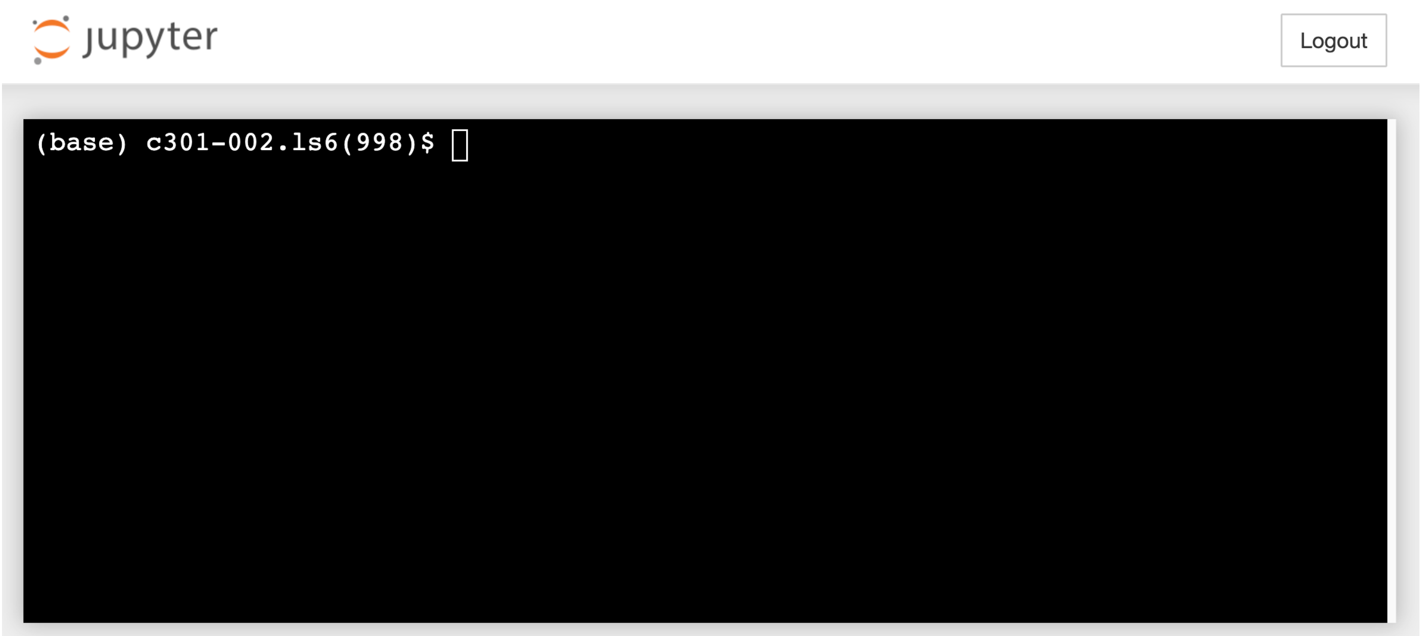
How to acquire TACC resources

- **Use Analysis Portal**
- Access the shell terminal through Jupyter Notebook



How to acquire TACC resources

- **Use Analysis Portal (for specific application only)**
- You should see this black window in your browser now



Copy Course Material

- Change directory to home directory and copy material to your home directory

  - `cd $HOME`
  - `rsync -ax --exclude=".*" /home1/07980/sli4/cnn-course .`

Install Containerized Jupyter Kernel

- Change directory into cnn-course and add execute permission for file install

  - `cd cnn-course`
  - `chmod +x install`

- Install the kernel: The script installs kernel.json in the ~/.local/share/jupyter/kernels/ space for the kernel we will use. Each Jupyter kernel has its own directory.

  - `./install`

Run Containerized Jupyter Kernel

- Close the tab on browser and launch your Jupyter session again
- Go to tap.tacc.utexas.edu and spawn a notebook you will find under the new pull down.