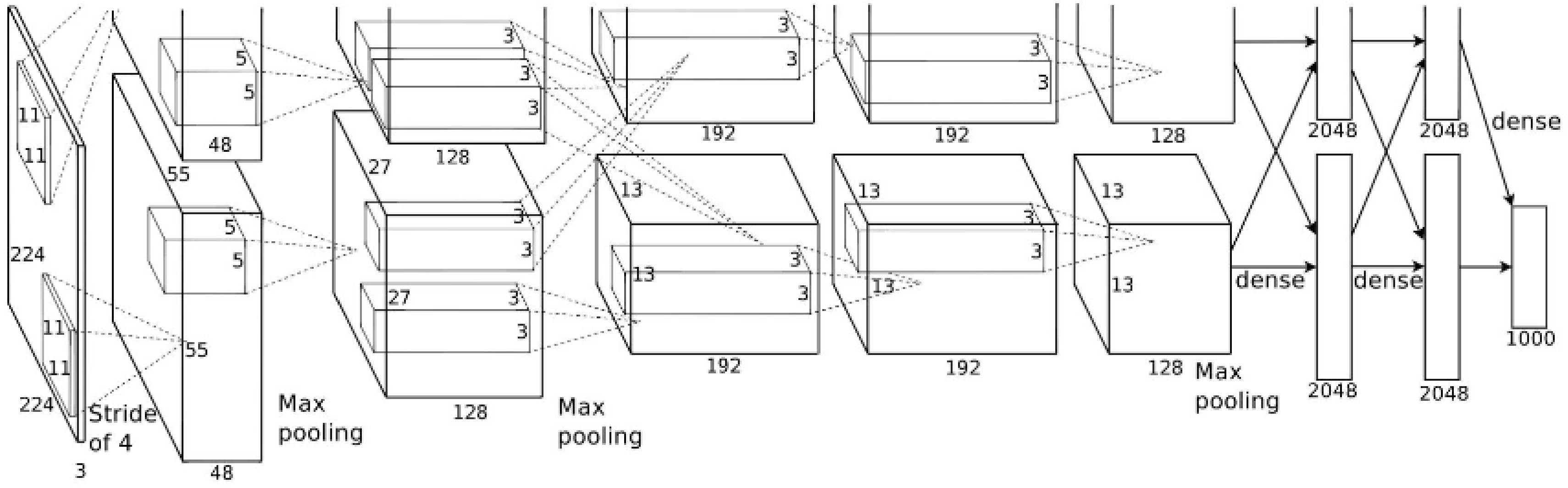
How ML works, and its application

Zhaohui Cheng
2020/04/21

Outcome

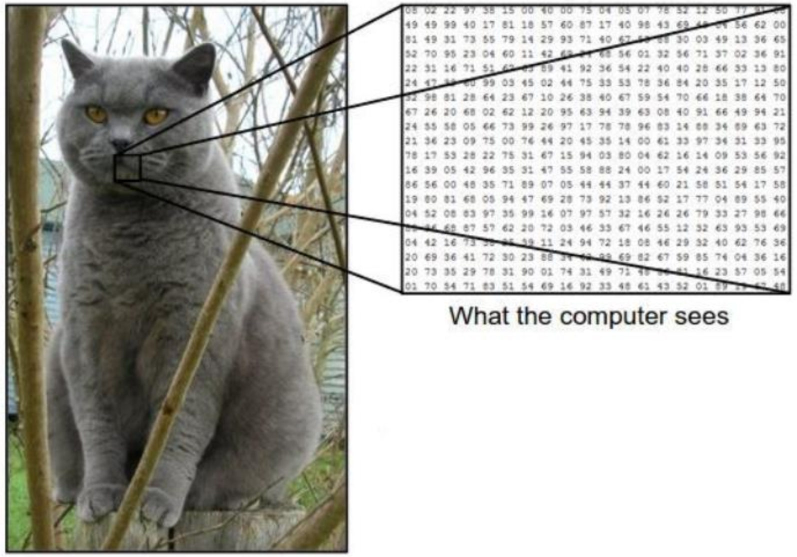
- Is machine learning magic?
- Does it require complex maths?
- How to read this?
- What is its application in our field?



How ML works, e.g. image classification

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat



An example dataset: CIFAR-10

Canadian Institute For Advanced Research

Collected by Alex Krizhevsky,

Vinod Nair, and Geoffrey Hinton.

60,000 colour images
Each 32x32x3 pixels

10 classes,
6000 images/class

50,000 training images
10,000 test images

airplane

automobile

bird

cat

deer

dog

frog

horse

ship

truck

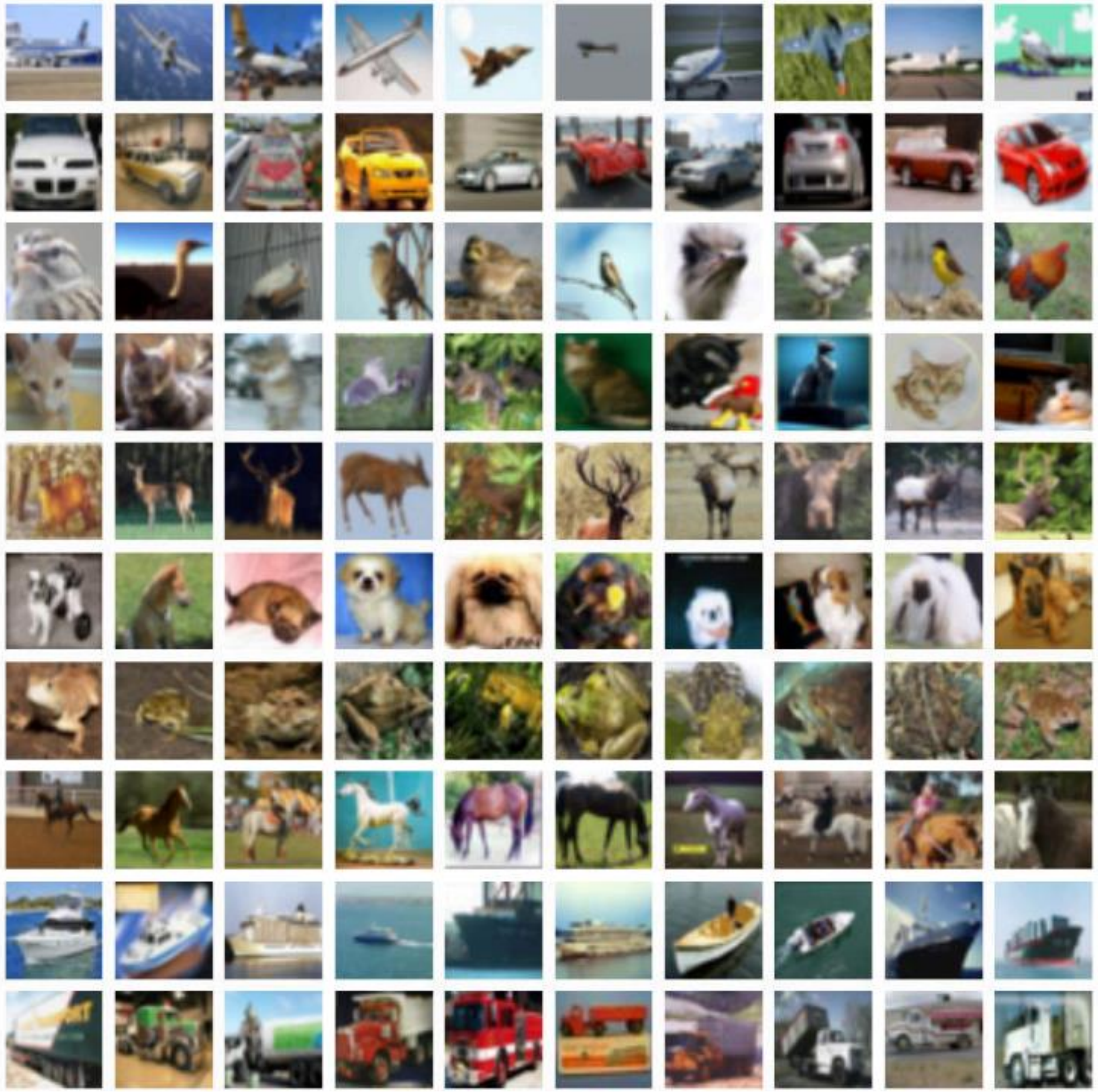


image parameters

$f(\mathbf{x}, \mathbf{W})$

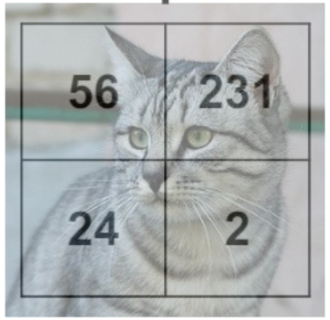
[32x32x3]
array of numbers 0...1
(3072 numbers total)

10 numbers,
indicating class
scores

The class with the highest score
is the classification result.

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Stretch pixels into column



Input image

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} * \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{y} \begin{array}{l} \text{Cat score} \\ \text{Dog score} \\ \text{Ship score} \end{array}$$

LOSS function

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

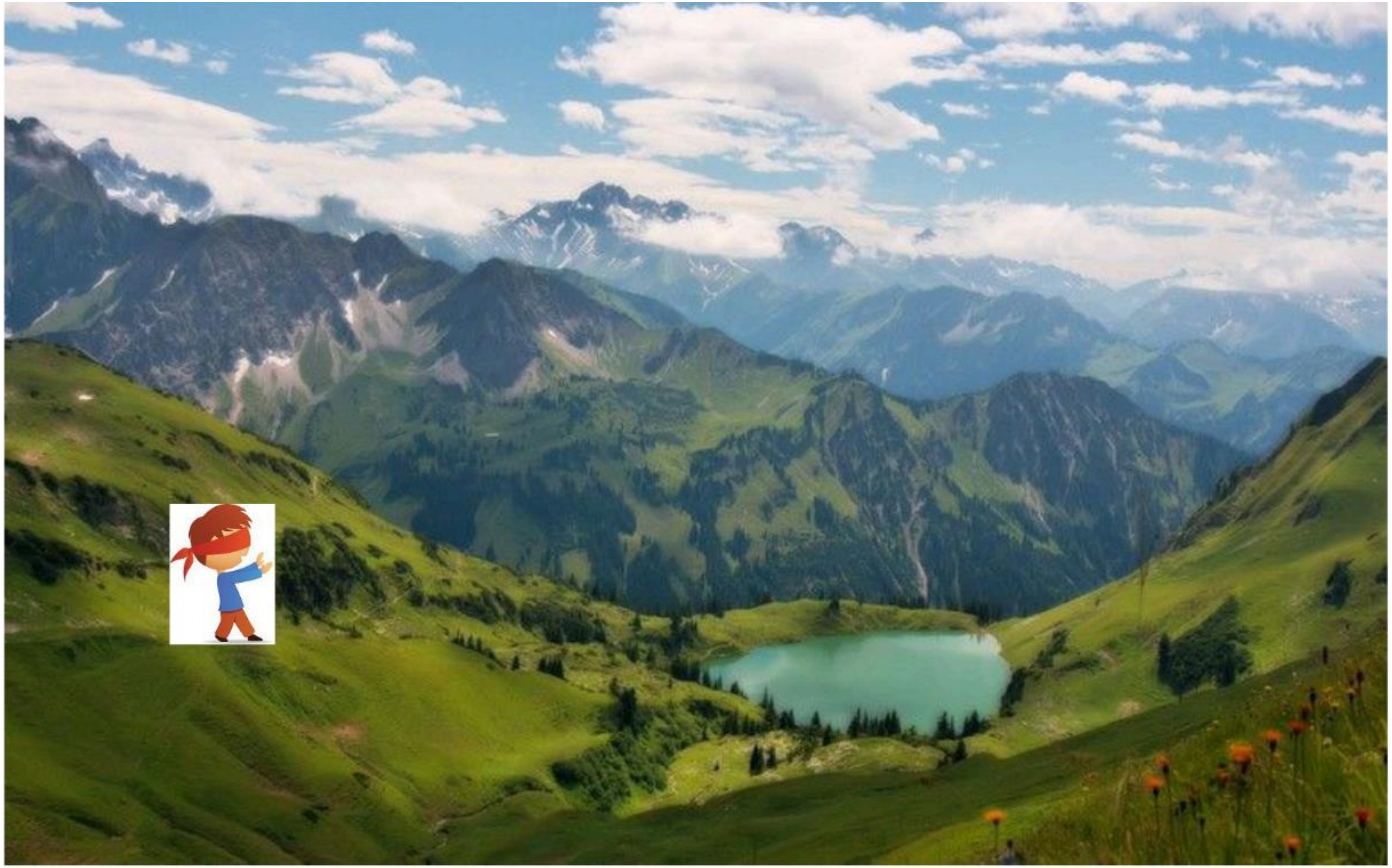
cat	**3.2**	exp →	**24.5**	normalize →	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

unnormalized log probabilities

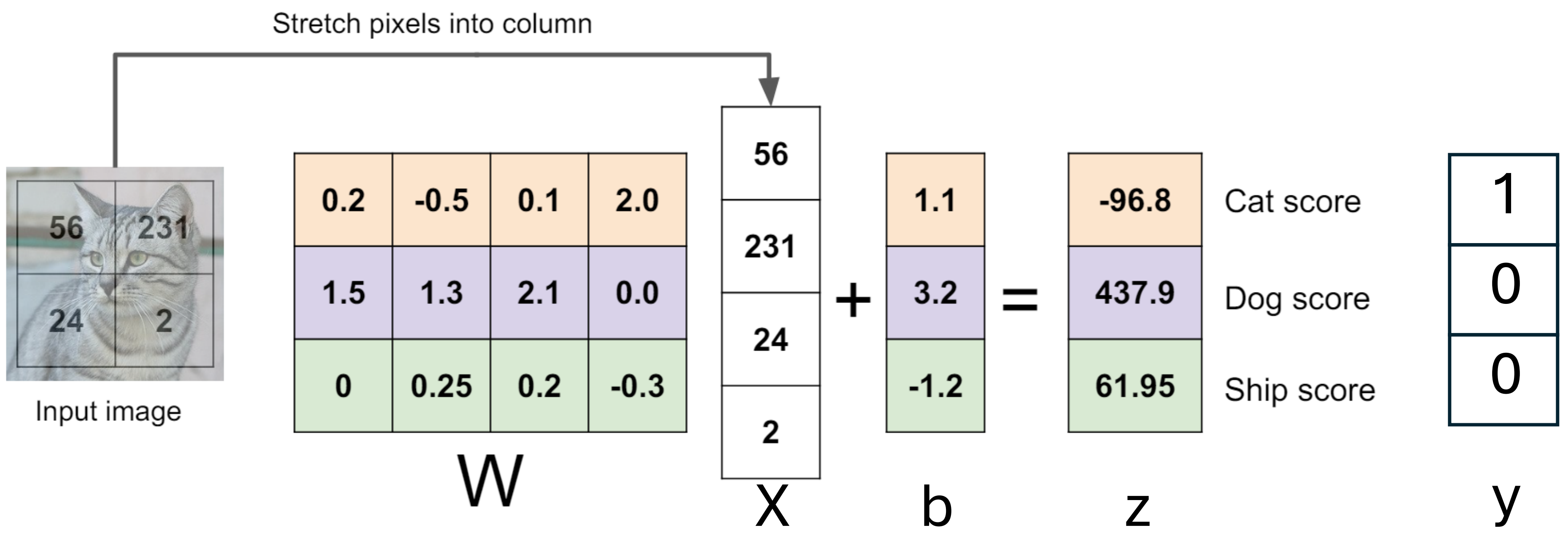
probabilities

→ L_i = -log(0.13)
= **0.89**

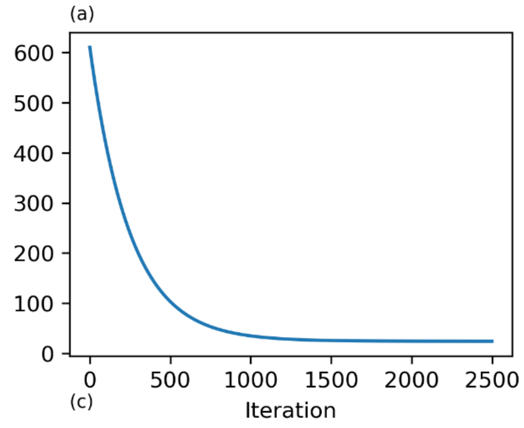
Minimize the loss



Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



1. Given a few images for training
2. Randomly initialize the weight (W) and offset (b) matrix
3. Calculate z by z=Wx+b
4. Compare z and y, calculate loss L
5. Calculate gradients dL/dw, dL/db
6. Update the weight (W2=W1-α dL/dw), and offset (b2=b1- α dL/db)
7. Iterate from 3 to 6 until the loss becomes convergent



App 1, regression

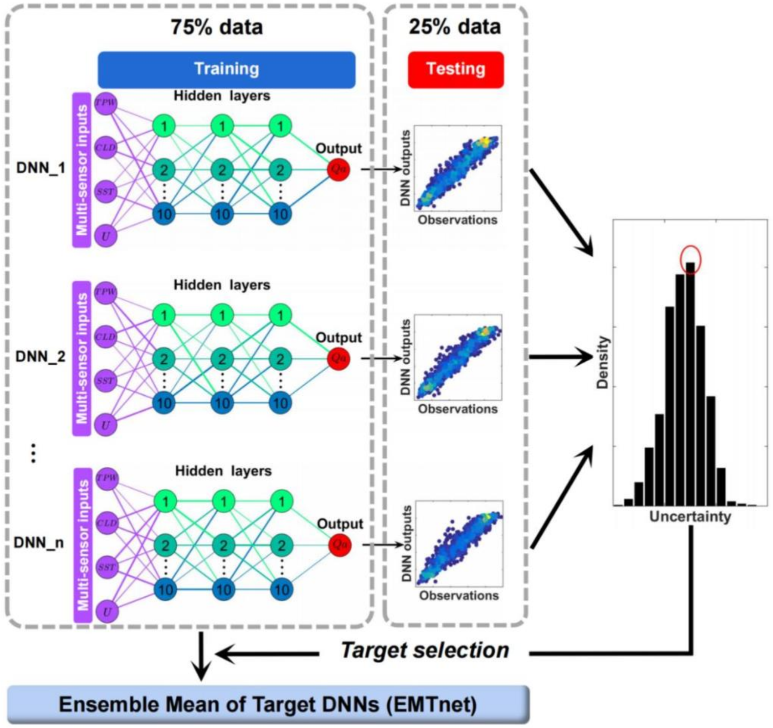


Figure 3. Architecture of the EMTnet. Each DNN_n (n = 1, 2, 3, . . .) uses 75% randomly sampled data from all in situ observations as training data, while the remaining 25% are used as testing data.

Deep Learning to Near-Surface Humidity Retrieval from Multi-Sensor Remote Sensing Data over the China Seas,
Rongwang Zhang

Cloud liquid water	U	SST	total column precipitable water
12	12	13	132
121	121	121	232
1212	121	121	34
...	...	...	...

$\xrightarrow{\text{Wx+b}}$ Near-surface humidity

$$\underset{W}{\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}} * \underset{X}{\begin{bmatrix} 12 \\ 12 \\ 13 \\ 132 \end{bmatrix}} + \underset{b}{\begin{bmatrix} 0 \end{bmatrix}} = \underset{z}{\begin{bmatrix} 0 \end{bmatrix}} \qquad \underset{y}{\begin{bmatrix} 32 \end{bmatrix}}$$

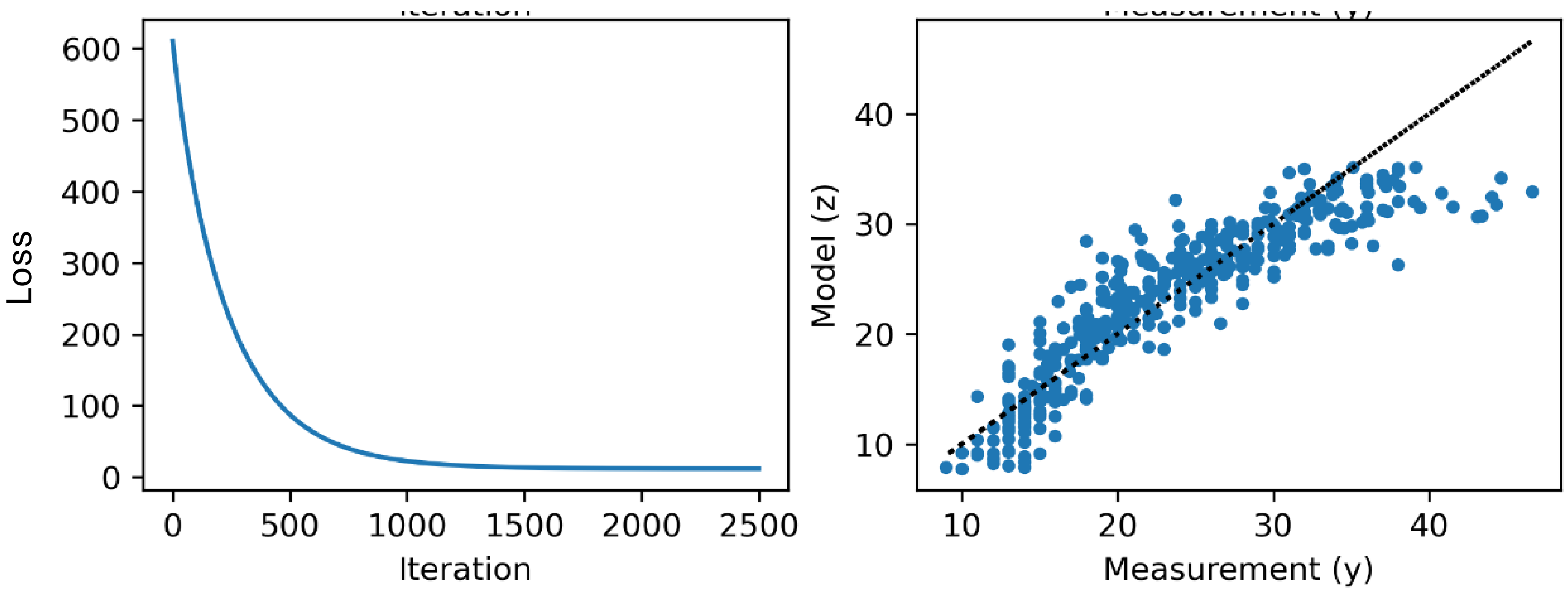
$$\underbrace{Z = \begin{bmatrix} 0 \\ 0 \\ \dots \end{bmatrix} \qquad y = \begin{bmatrix} 32 \\ 33 \\ \dots \end{bmatrix}}_{\text{Loss}}$$

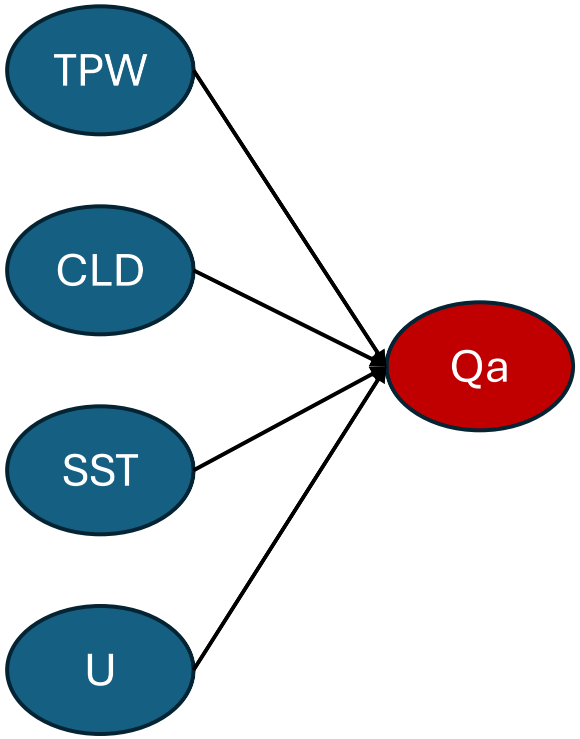
$$J = \frac{1}{n} \sum_{i=1}^{n} L_i$$

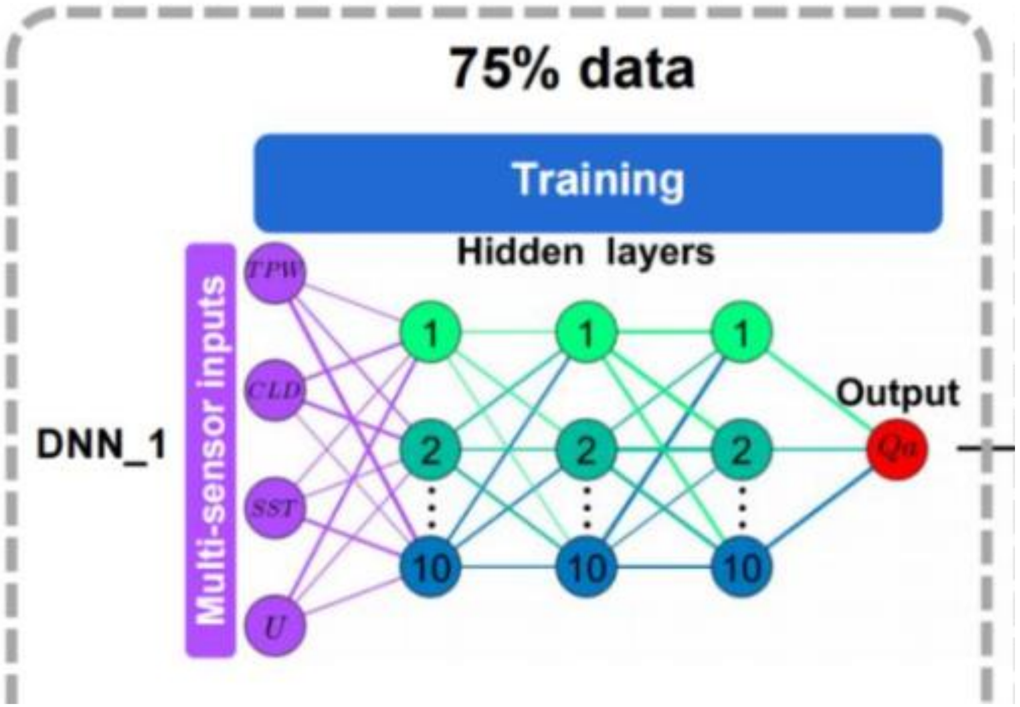
where $L_i = (y_i - z_i)^2$

```python
def initialize_parameters(width,Input_W=None,Input_b=None):

    if Input_W == None:
        Input_W=np.zeros((width,1))
    if Input_b == None:
        Input_b=0
    return Input_W,Input_b


def model_forward(Input_x,Input_W,Input_b):

    length,width=np.shape(Input_x)
    z=np.zeros((length,1))
    for i in range(length):
        z[i]=np.dot(Input_W.T,Input_x[i,:])+Input_b
    return z



def compute_cost(Input_y,Input_z):

    length=len(Input_y)
    L=np.zeros((length,1))
    for i in range(length):
        L[i]=(Input_y[i]-Input_z[i])**2
    J=np.mean(L)
    return J

def model_backward(Input_x, Input_y, Input_z):

    length,width=np.shape(Input_x)
    # calculate dJ/db
    temp1=np.zeros((length,1))
    for i in range(length):
        temp1[i]=(1/length)*(2*Input_z[i]-2*Input_y[i])
    dJdb=np.sum(temp1)

    #calculate dJ/dwj
    dJdwj=np.zeros((width,1))
    for j in range(width):
        temp2=np.zeros((length,1))
        for i in range(length):
            temp2[i]=Input_x[i,j]*(2*Input_z[i]-2*Input_y[i])/length
        dJdwj[j]=np.sum(temp2)

    return dJdb, dJdwj
```

```python
def update_parameters(Input_W,Input_b,Input_dJdb,Input_dJdwj,Input_learnrate):

    width=len(Input_W)
    new_W=np.zeros((width,1))
    for j in range(width):
        new_W[j]=Input_W[j]-Input_learnrate*Input_dJdwj[j]
    new_b=Input_b-Input_learnrate*Input_dJdb

    return new_W,new_b

def predict(Input_x,Input_W,Input_b):

    m,n=np.shape(Input_x)
    z=np.zeros((m,1))
    for i in range(m):
        z[i]=np.dot(Input_W.T,Input_x[i,:])+Input_b
    return z


def train_linear_model(x,y,iteration,learnrate):

    m,n=x.shape
    W,b=initialize_parameters(width=n)
    J=np.zeros((iteration,1))

    for itera in range(iteration):
        print('Completed '+str(100*itera/iteration)+'% of work...')
        z=model_forward( x, W, b)
        J[itera]=compute_cost(y, z)
        dJdb,dJdw = model_backward(x, y, z)
        W,b=update_parameters(W, b, dJdb, dJdw, learnrate)

    return W,b,J
```


Loss
600
500
400
300
200
100
0
0
500
1000
1500
2000
2500
Iteration
Model (z)
40
30
20
10
10
20
30
40
Measurement (y)


TPW
CLD
SST
U
Qa


75% data
Training
Hidden layers
DNN_1
Multi-sensor inputs
TPW
CLD
SST
U
1
2
10
Output
Qa

$$\underbrace{\begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix}}_{W} * \underbrace{\begin{bmatrix}12\\12\\13\\132\end{bmatrix}}_{X} + \underbrace{\boxed{0}}_{b} = \underbrace{\boxed{0}}_{z} \qquad \underbrace{\boxed{32}}_{Qa}$$

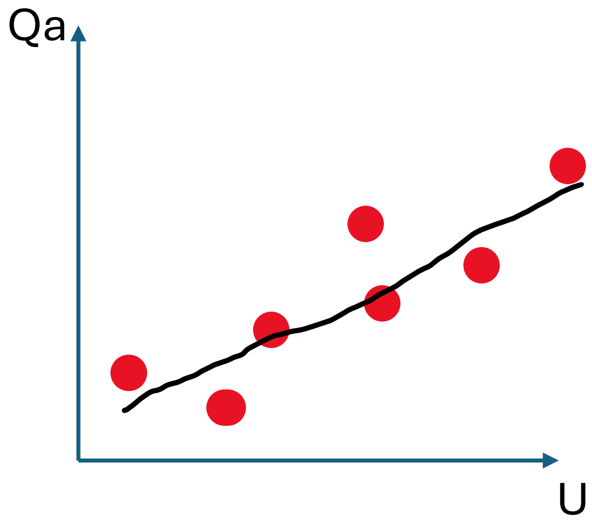
$h(g(f(x)))$

1st layer:

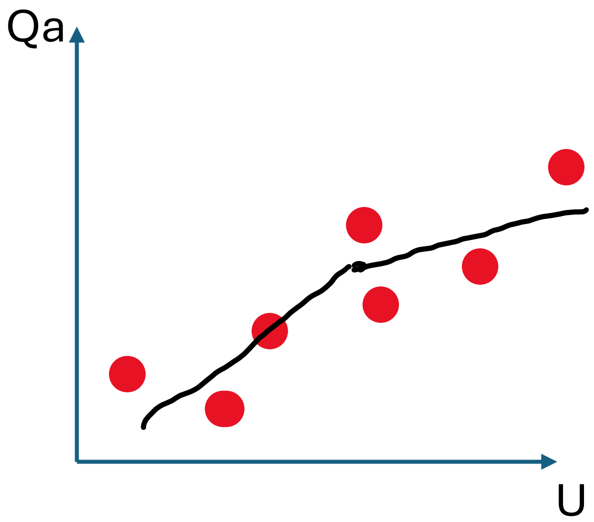
$$\underset{W1}{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}} * \underset{X}{\begin{bmatrix} 12 \\ 12 \\ 13 \\ 132 \end{bmatrix}} + \underset{b1}{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}} = \underset{z1}{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}$$

2st layer:

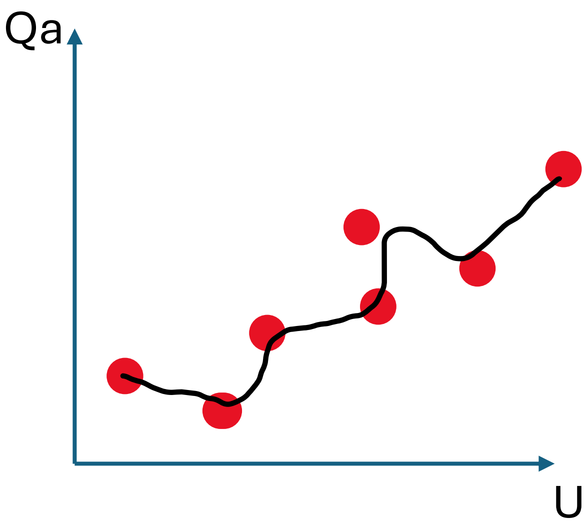
$$\underset{W2}{\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}} * \underset{z1}{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}} + \underset{b2}{\begin{bmatrix} 0 \end{bmatrix}} = \underset{z2}{\begin{bmatrix} 0 \end{bmatrix}} \qquad \underset{Qa}{\begin{bmatrix} 32 \end{bmatrix}}$$



1 linear layer

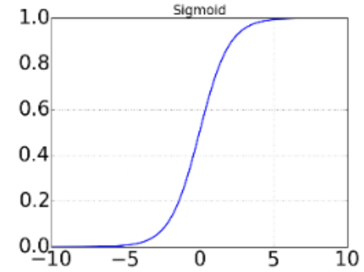


2 linear layer

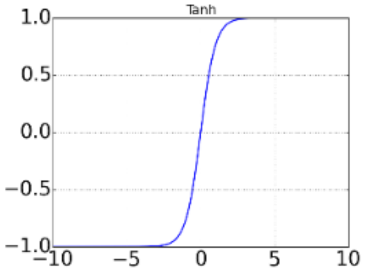


3 linear layer

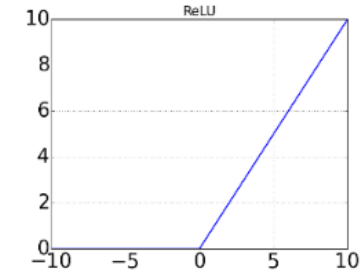
Activation functions



$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

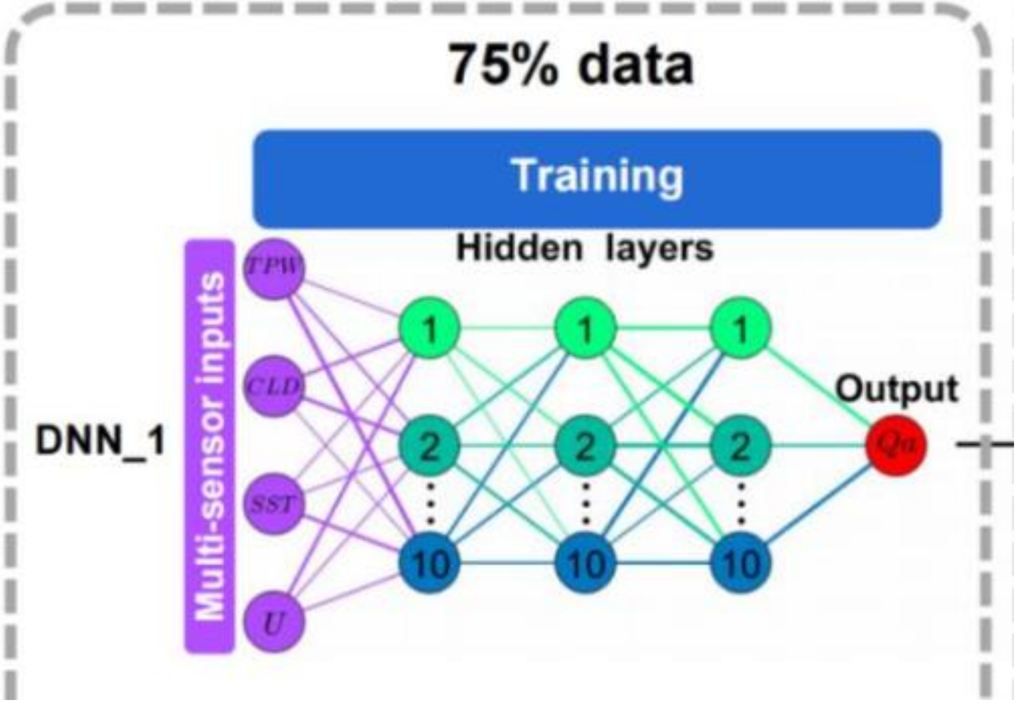


$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\, sigmoid(2x) - 1$$



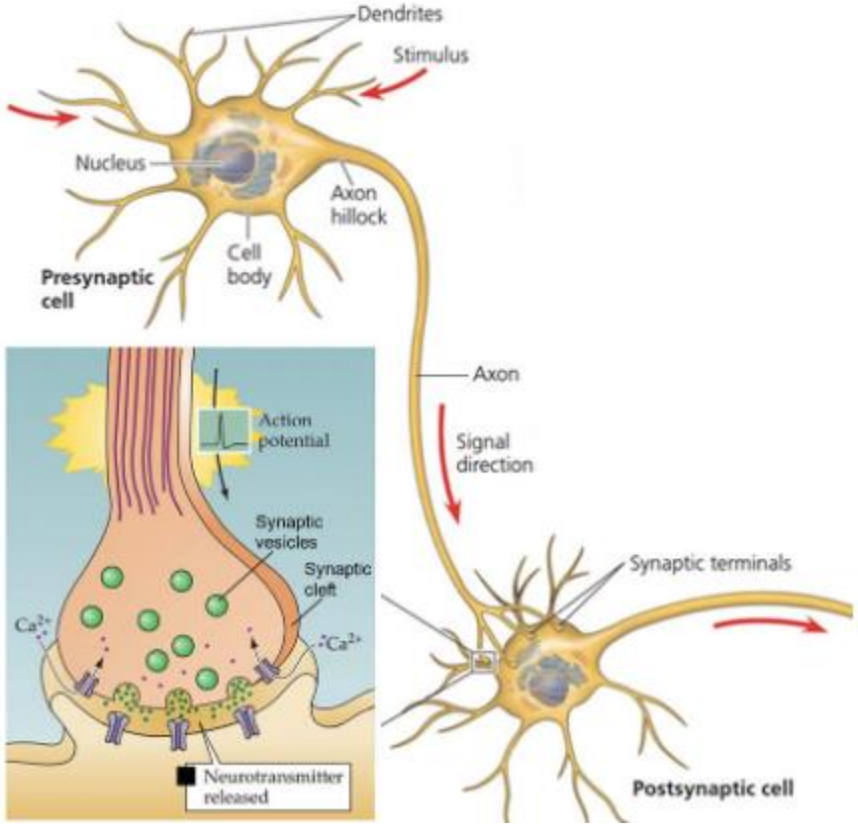
$$ReLU(x) = max(0, x)$$

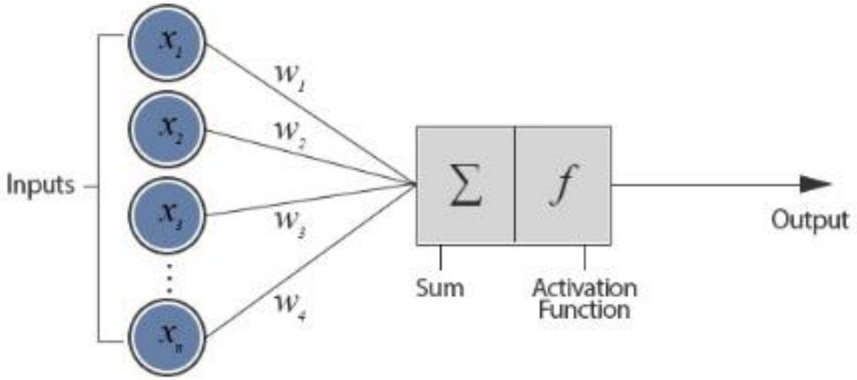
"Rectified Linear Unit" (diode function)



Hidden Layers, Black Box, Architecture, neural network
Artificially designed,
No scientific interpretation

Neuron and Artificial neuron





Src.http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7

$$output = h(\sum_{i=1}^{n} x_i w_i)$$

App2: ML for weather prediction

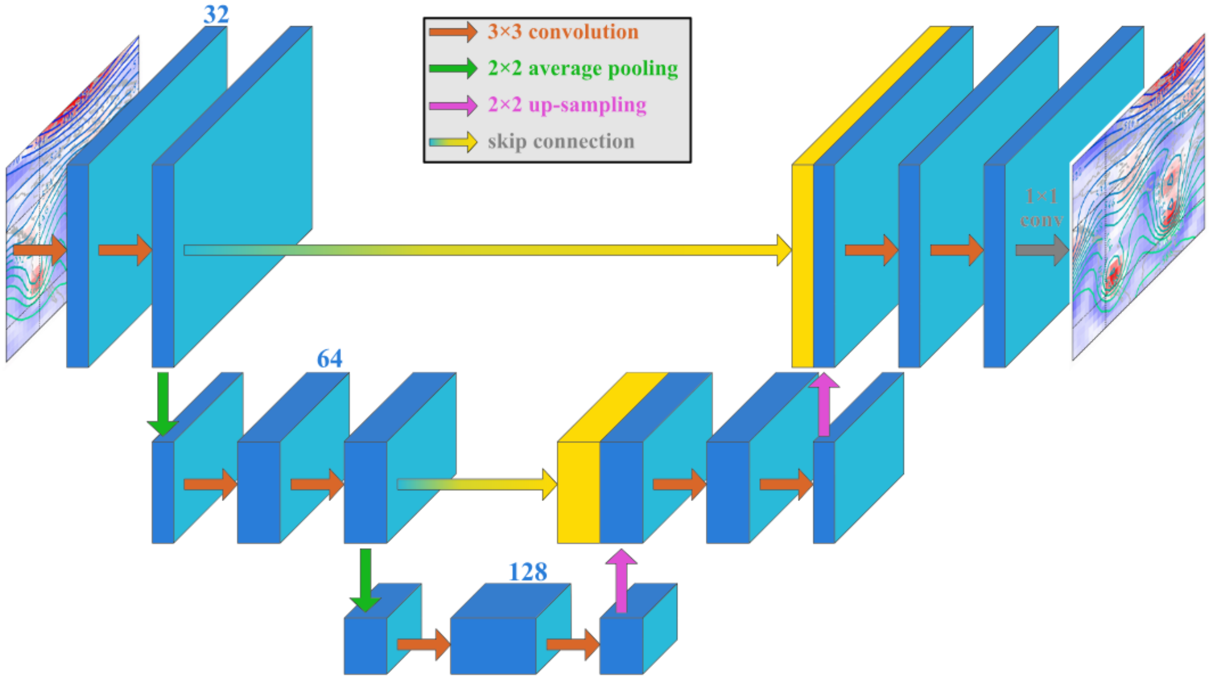


Figure 2. Schematic illustrating the architecture of our DLWP CNN based on the U-Net architecture. Each red arrow represents a 2-D convolution operating on each cubed-sphere face. Green and purple arrows indicate average-pooling and upsampling operations, respectively. The blue-to-yellow lines represent skip connections, whereby the blue state is copied exactly to the yellow state vector and concatenated to the new blue state vector along the channels dimension. The final gray arrow is a 1×1 convolution. The blue numbers indicate the number of convolutional filters (channels) at each stage of the network (channel width is to scale).

Improving Data-Driven Global Weather Prediction Using Deep Convolutional Neural Networks on a Cubed Sphere

2D data: CNN (convolutional neural network)

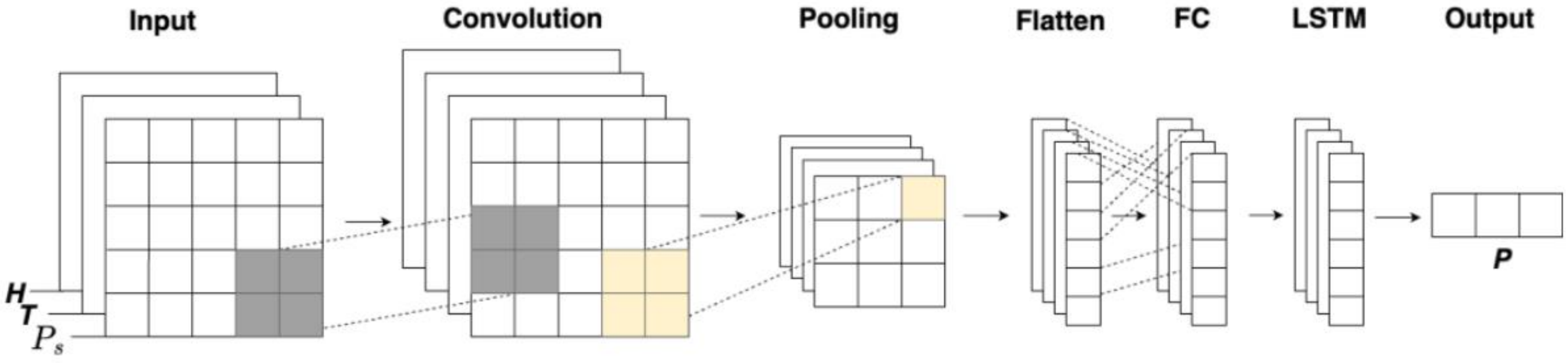
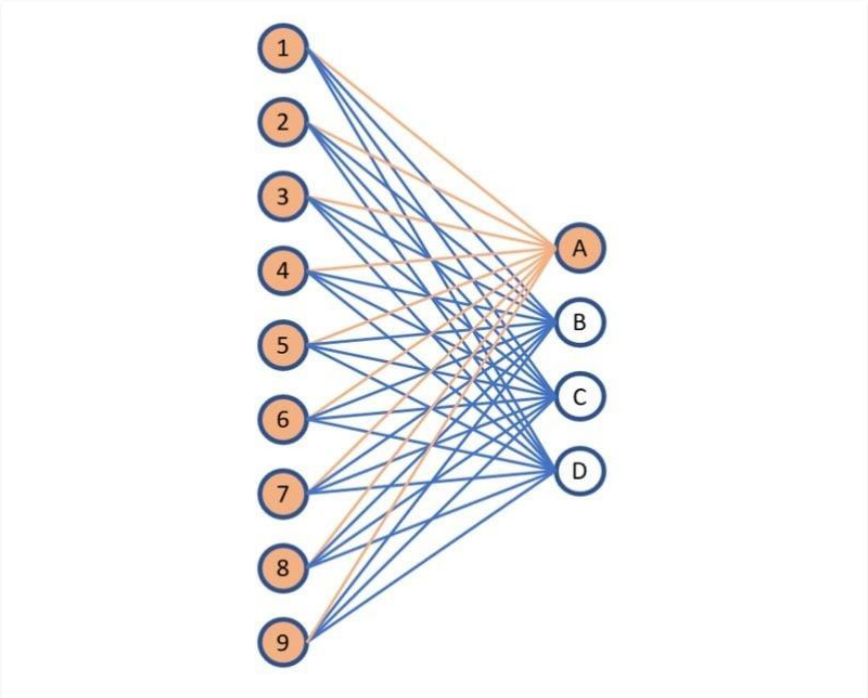
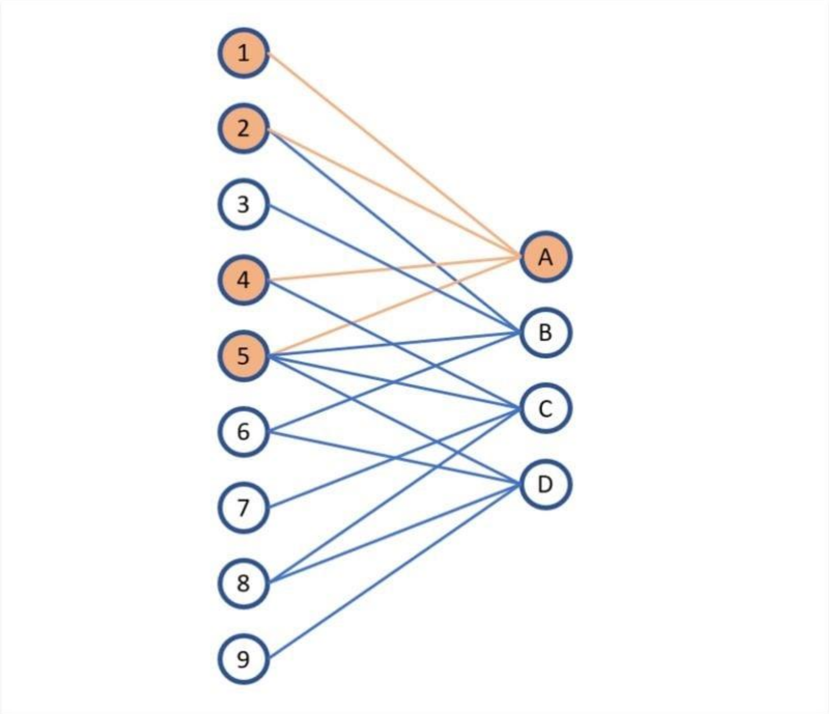
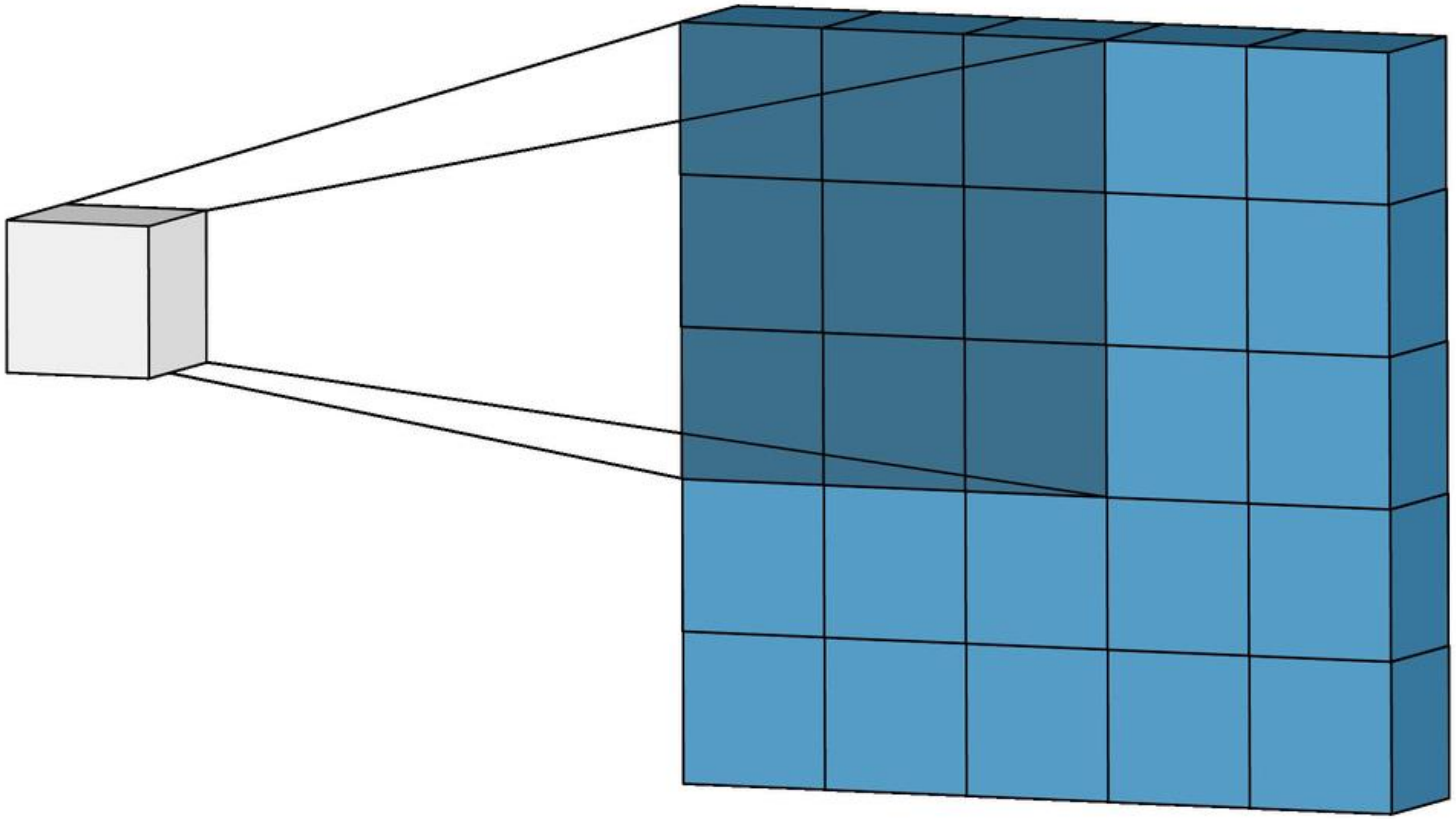


Figure 2. MetNet Structure.

Fully connected layers vs convolutional neural network

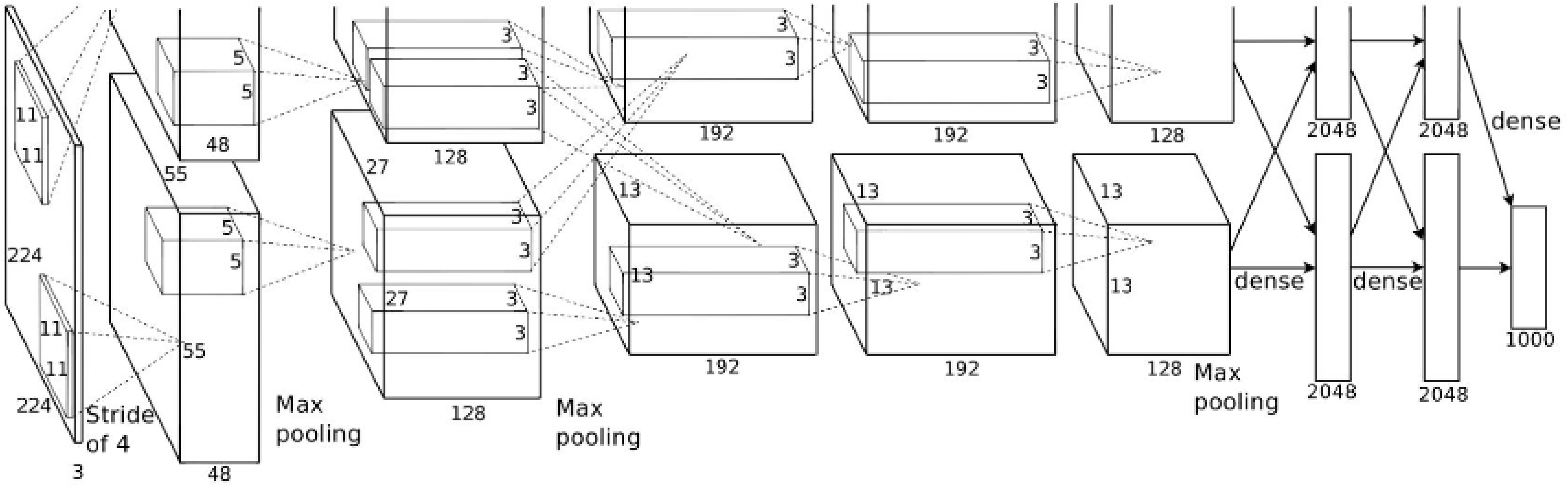






Alexnet

2012



Imagenet classification with **deep convolutional neural networks**

A **Krizhevsky**, I Sutskever… - Advances in **neural** …, **2012** - proceedings.neurips.cc

We trained a large, deep convolutional neural network to classify the 1.3 million high-resolution images in the LSVRC-2010 ImageNet training set into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 39.7\% and 18.9\% which is considerably better than the previous state-of-the-art results. The neural network, which has 60 million parameters and 500,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and two globally connected layers with a final …

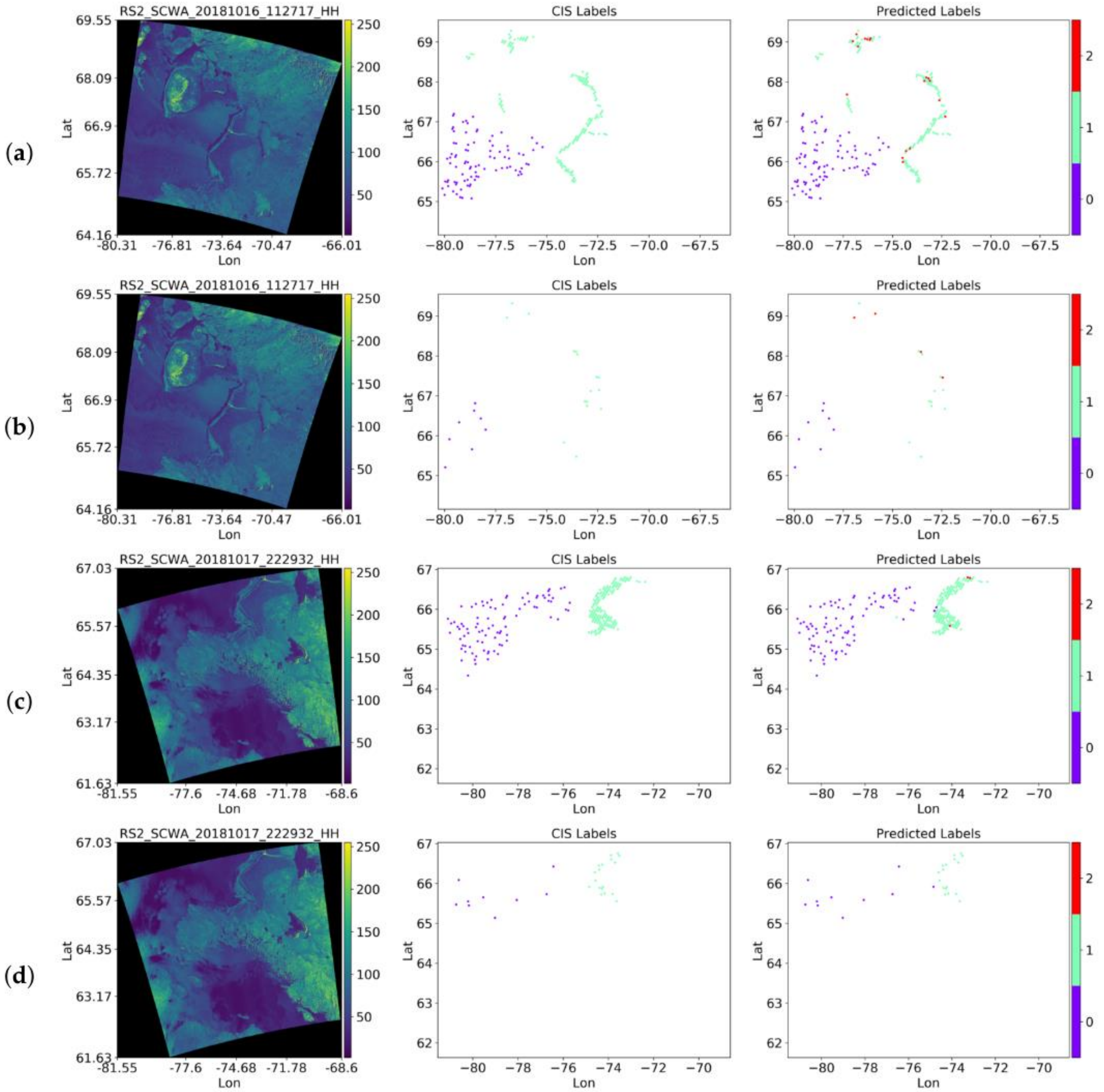
☆ Save 99 Cite Cited by 128198 Related articles All 102 versions »

Import pytorch

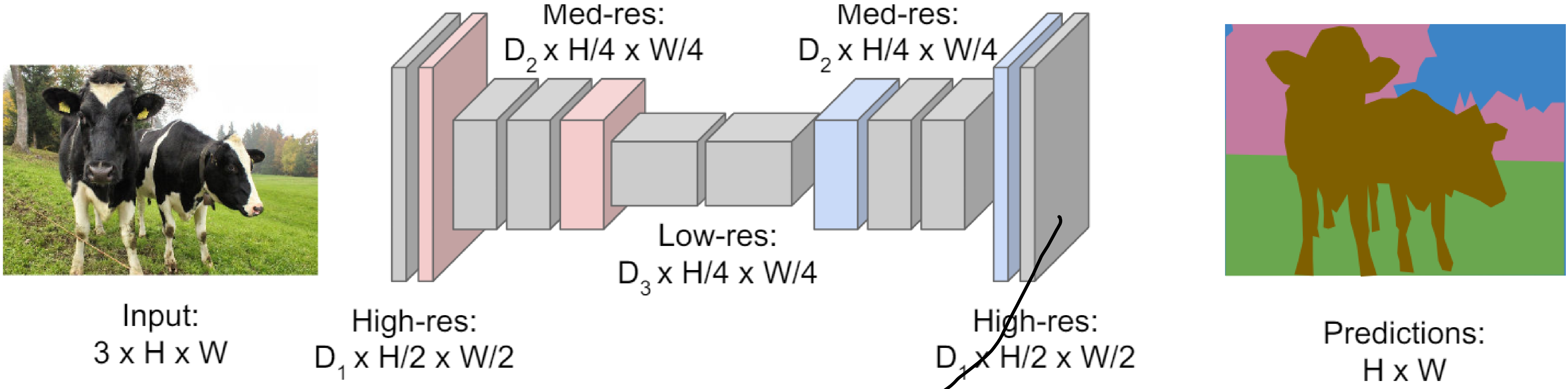
Use bricks in *pytorch* to build your own architecture.

```
# Define the CNN model
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # First convolutional layer
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        # Second convolutional layer
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        # Max pooling layer
        self.pool = nn.MaxPool2d(2, 2)
        # First fully connected layer
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        # Second fully connected layer
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        # First convolutional layer followed by ReLU activation and pooling
        x = self.pool(torch.relu(self.conv1(x)))
        # Second convolutional layer followed by ReLU activation and pooling
        x = self.pool(torch.relu(self.conv2(x)))
        # Flatten the tensor
        x = torch.flatten(x, 1)  # flatten all dimensions except batch
        # First fully connected layer followed by ReLU activation
        x = torch.relu(self.fc1(x))
        # Second fully connected layer
        x = self.fc2(x)
        return x
```

App 3, image segamentation



Proof of Concept for Sea Ice Stage of Development Classification Using Deep Learning

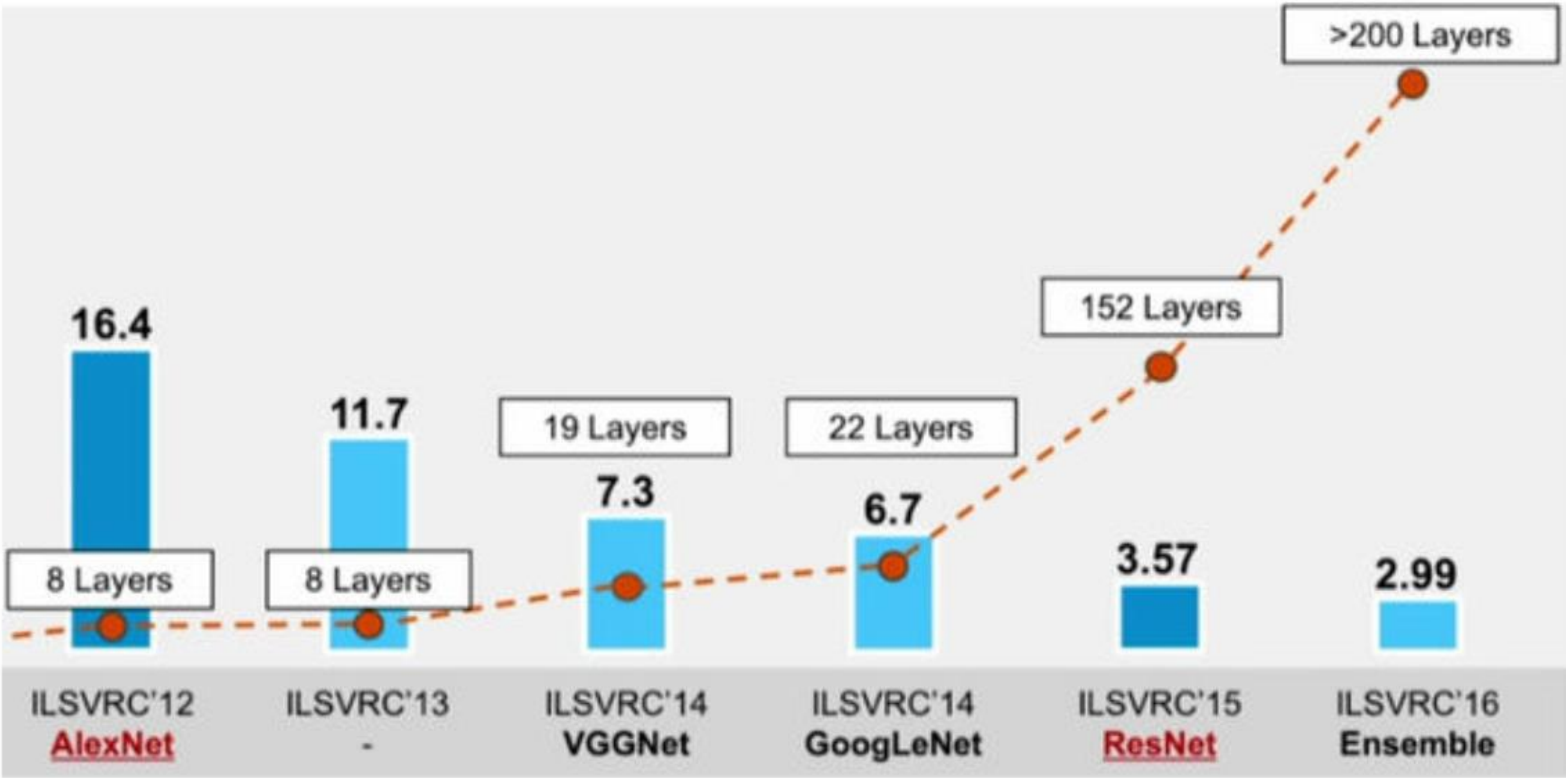


0	1	0	1	1
0	0	0	1	1
0	0	0	1	1
2	0	0	3	3
2	2	2	2	3

1: sky
0: cow
2: grass
3: tree

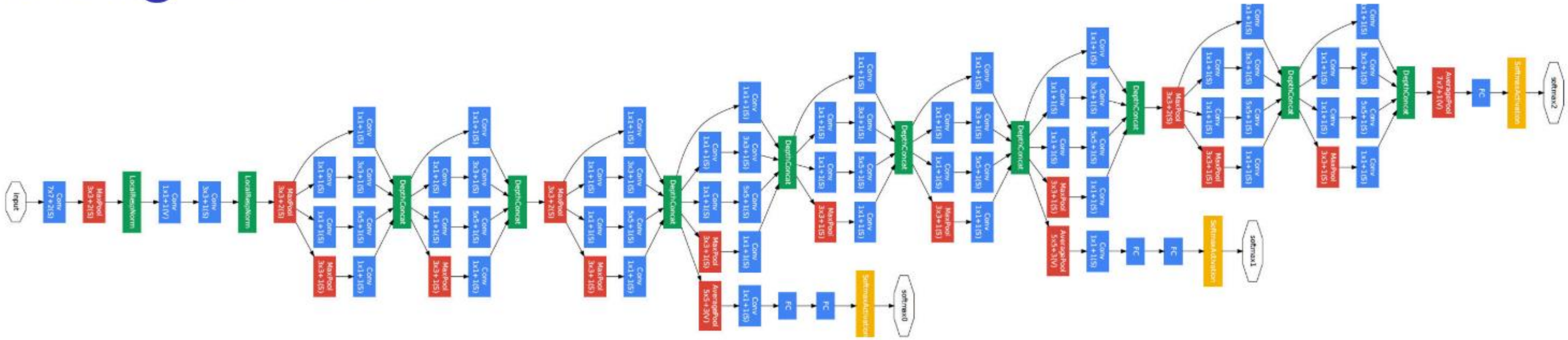
Outlook: deeper layers

Deeeeeep



Outlook: wider layers

GoogLeNet

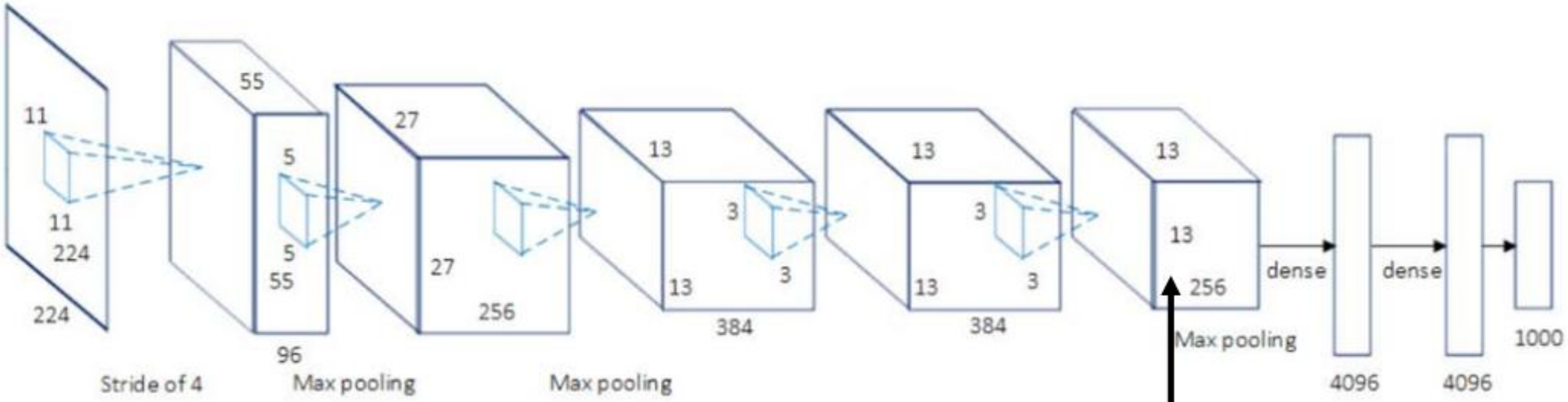


Improving machines → teaching humans

1. When AI is weaker than humans
   - Transparency → finding error modes
   - Goal = improving machines

2. When AI is at par with humans
   - Transparency → providing rationales
   - Goal = building trust with humans to drive adoption

3. When AI is stronger than humans
   - Transparency → explaining a complicated concept
   - Goal = teaching humans



"cat"

- Let's take a single value in an intermediate feature map and propagate its gradient back to the original image pixels
- What does this tell us?

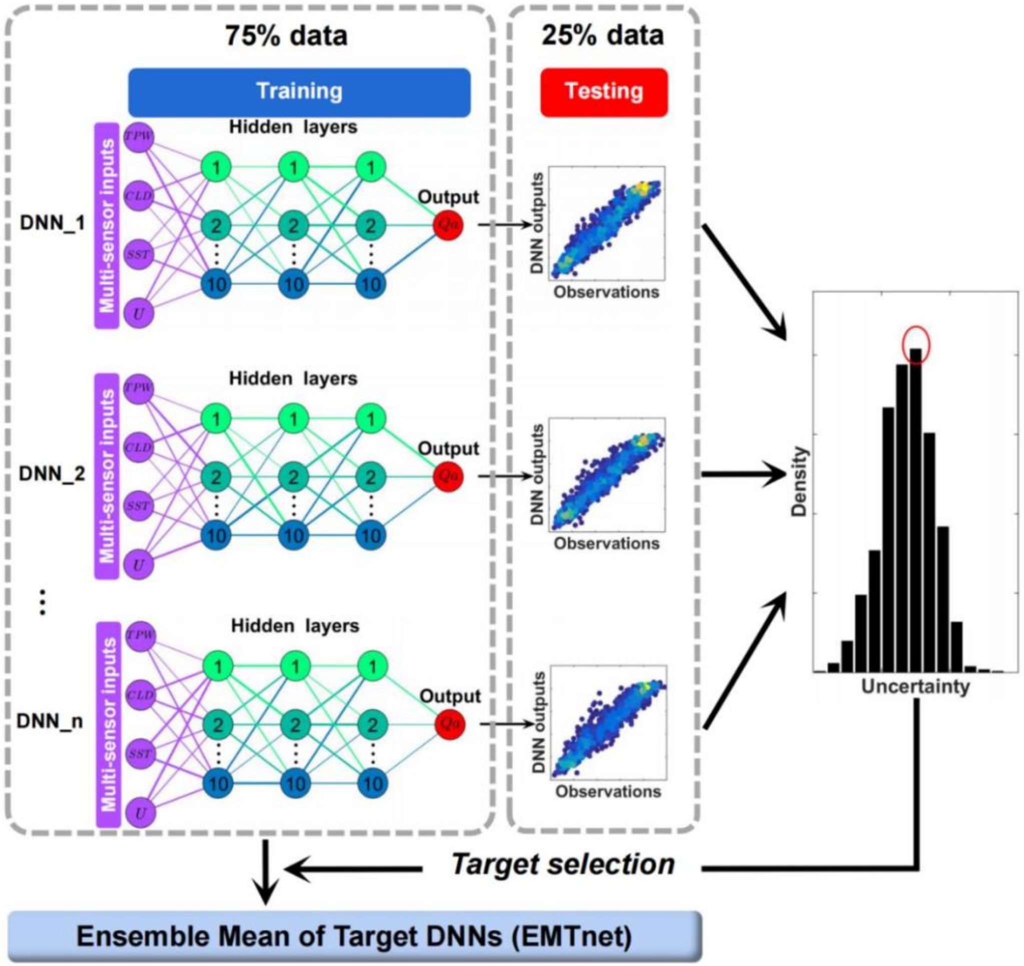
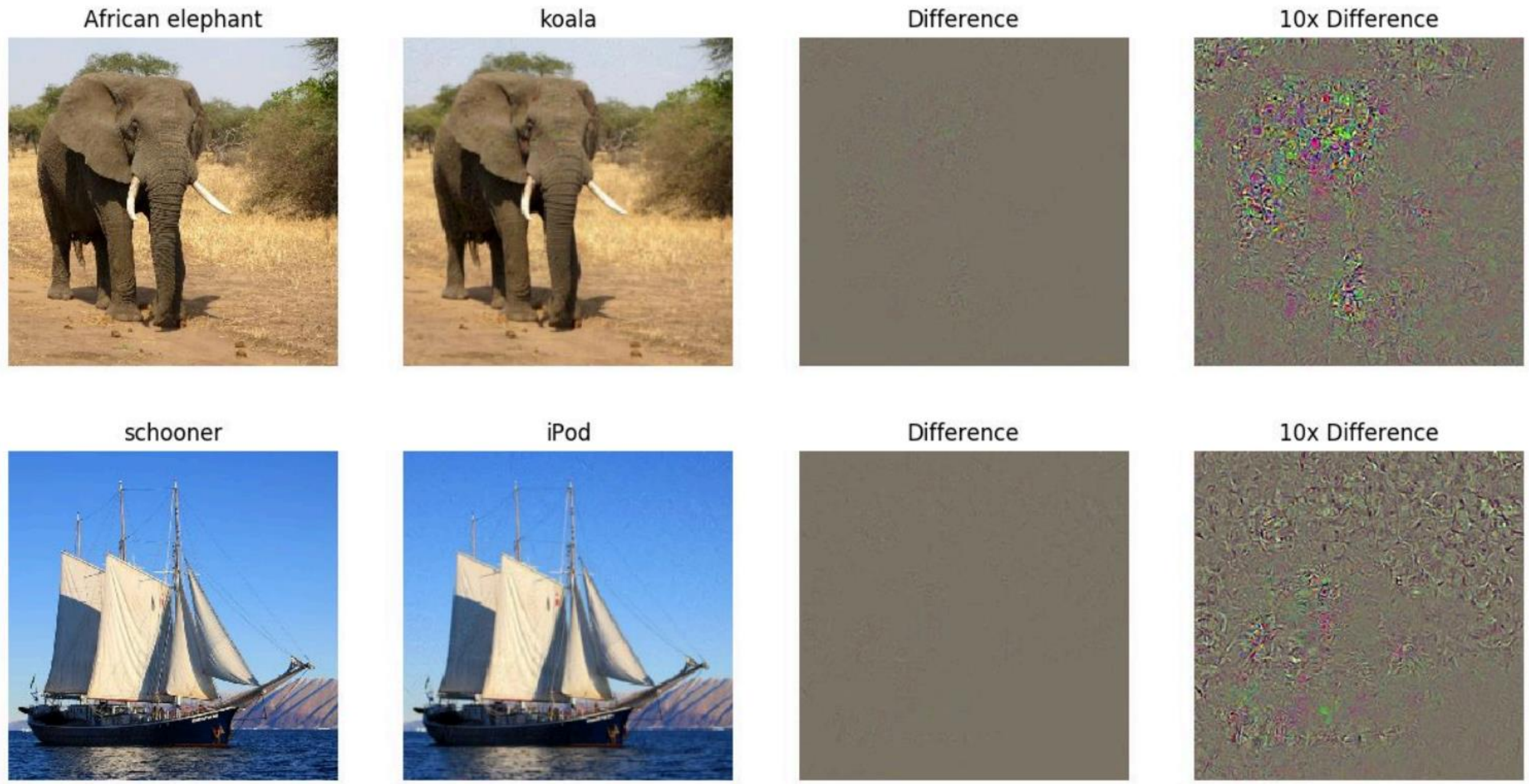
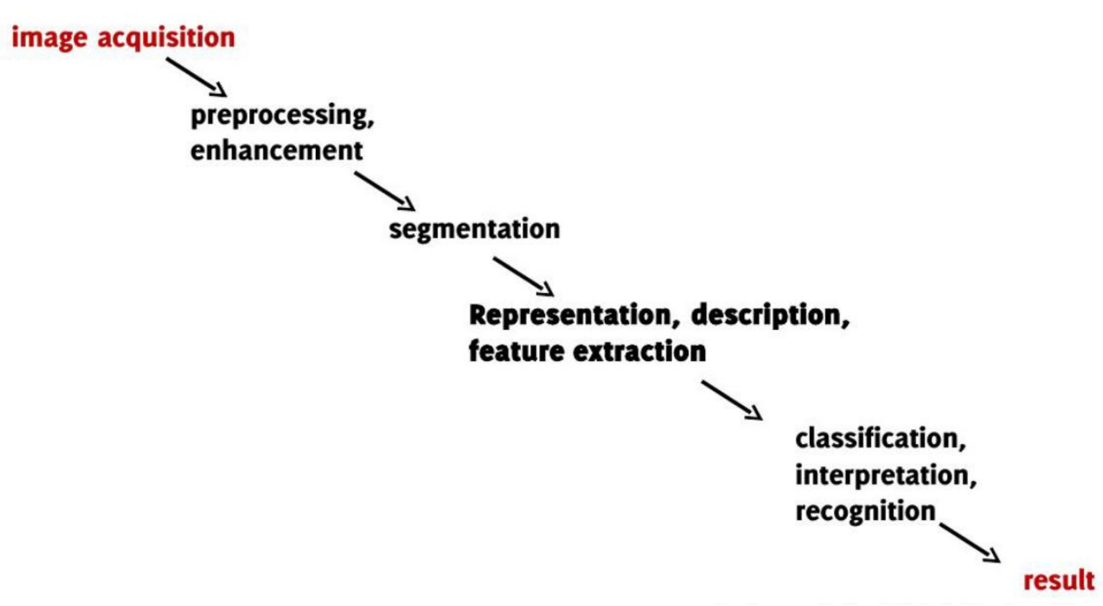


Figure 3. Architecture of the EMTnet. Each DNN_n (n = 1, 2, 3, ...) uses 75% randomly sampled data from all in situ observations as training data, while the remaining 25% are used as testing data.

Trained model is still Fragile



Conclusion



Conventional method VS machine learning

Conclusion

- Machine Learning is not always the best tool, and need to be used with care, augmentation, texture, bias…
- Can be rather demanding – lots of annotated data, compute heavy
- ML is great at interpolation, but less great at extrapolation
- Don’t use it to solve known maths:

  > ML is more useful to provide a starting guess

  > In many cases, combinations of classic and learning based methods provide the best solutions