Delve Into Art and Creativity with Python

Aparna Krishnakumar

Next Tech Lab

SRM University

Next Tech Lab

SRM University

Deep Learning

what has been done and what can be done

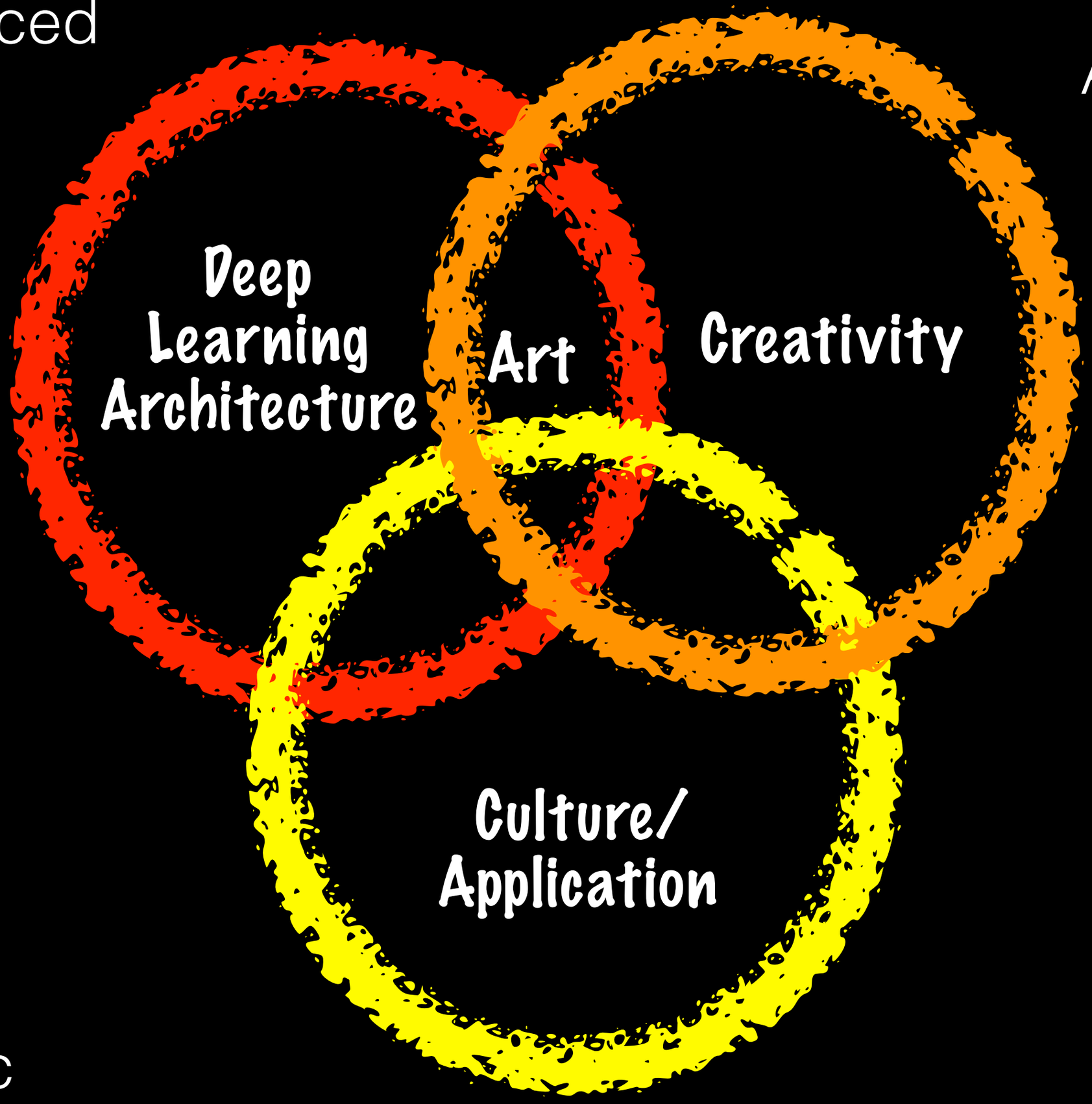
Advanced

Basic

Culture/
Application

'ART'

Libraries

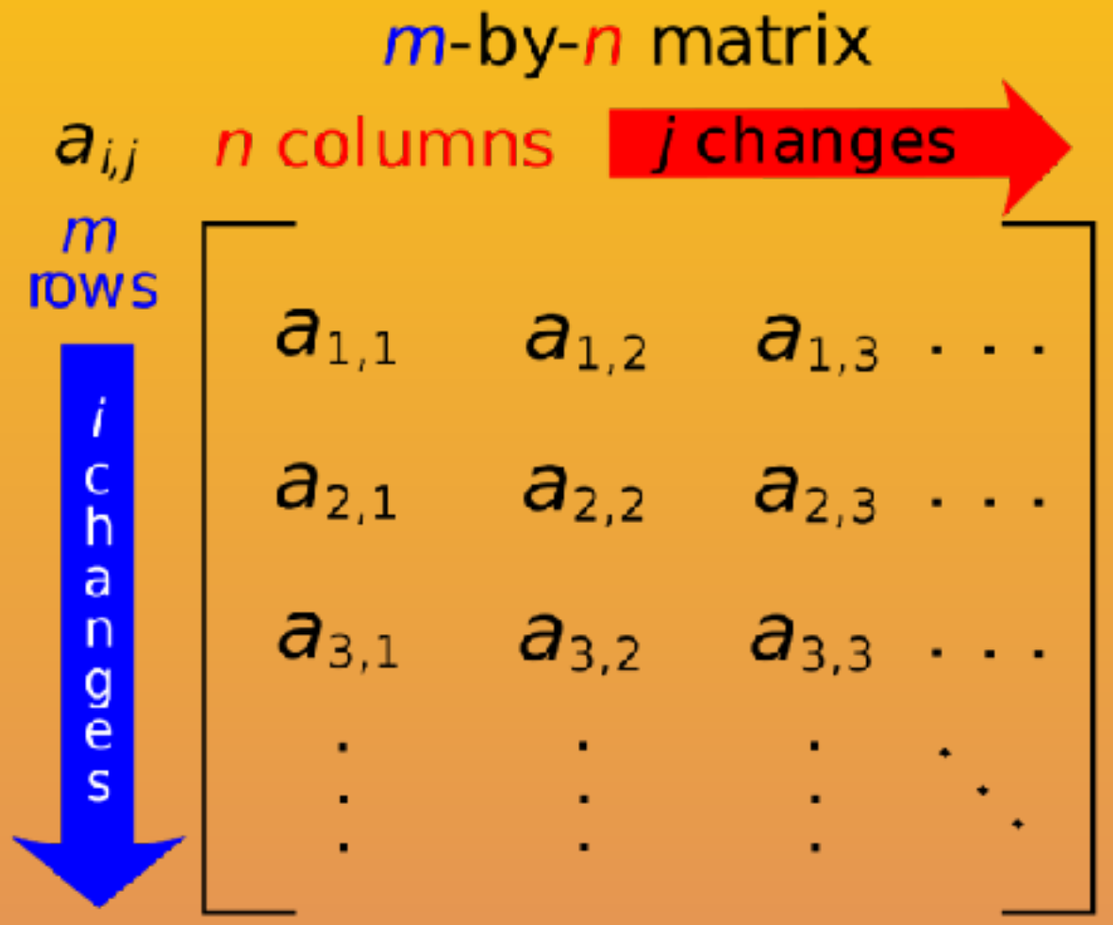


Why Python?

- Easy implementation
- Powerful, Image processing libraries
- comprehensive material
- Open Source
- AI Frameworks- TensorFlow, Keras, Pytorch
- Community
- Research Oriented

Artificial Intelligence-Deep Learning

Data

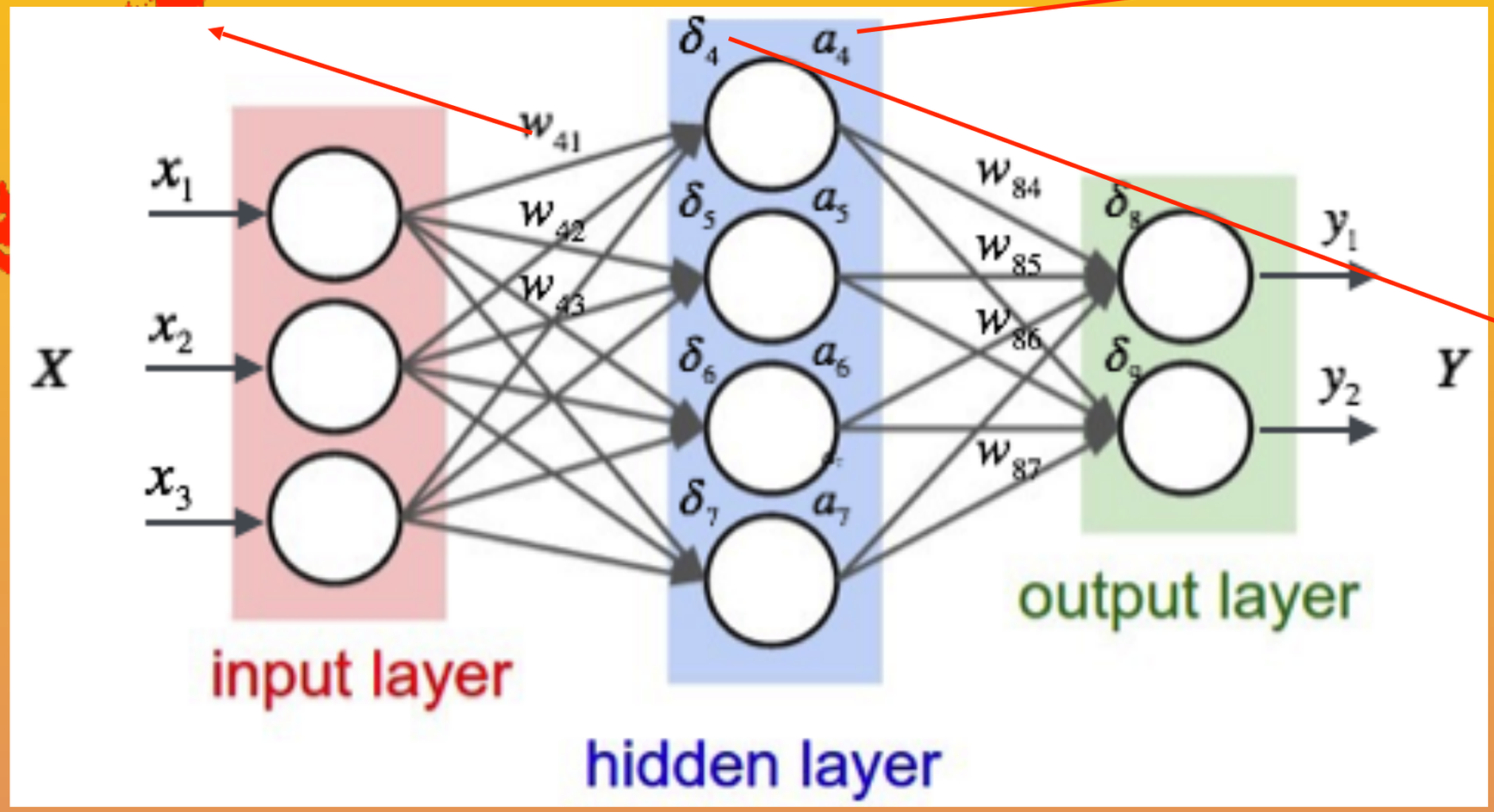


Training examples X features

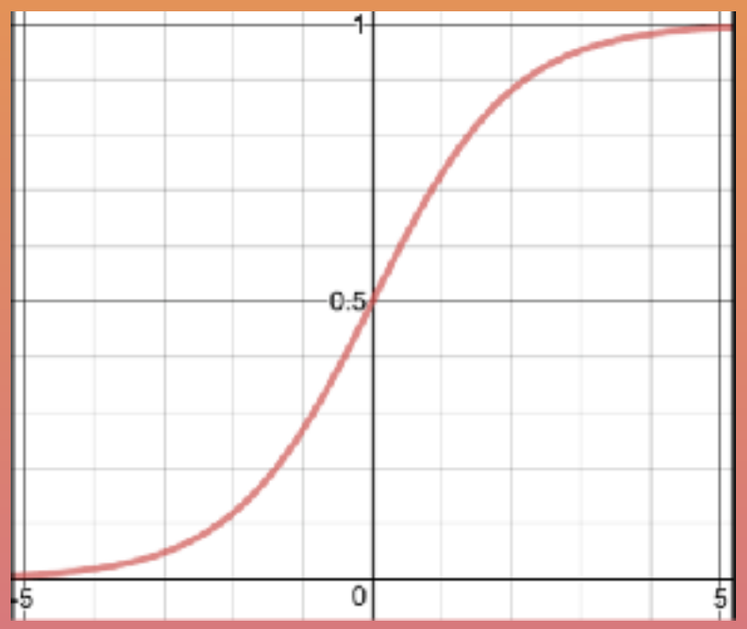
Neural Networks

Weights

after Activation function



back propagation Error



$a=g(w*x+b)$

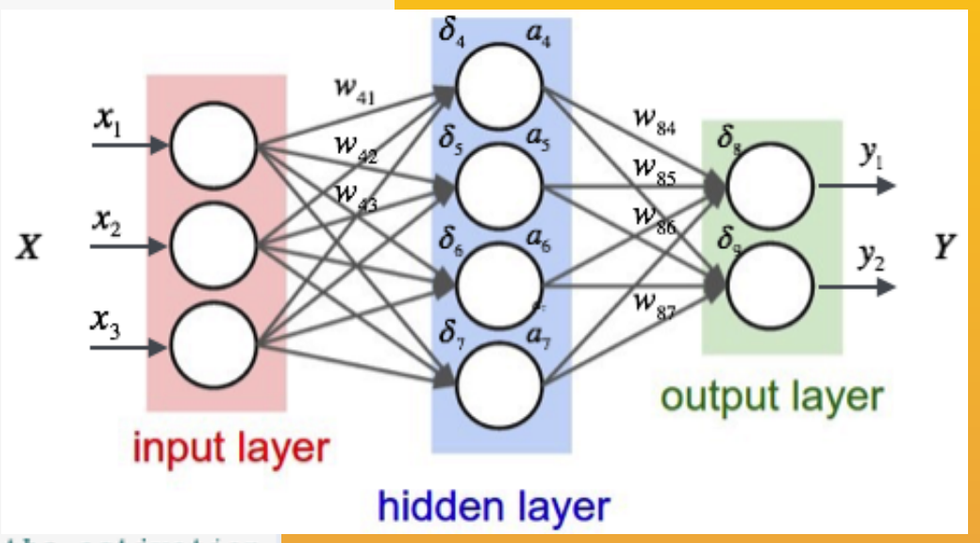
Sigmoid function

$$S(z) = \frac{1}{1+e^{-z}}$$

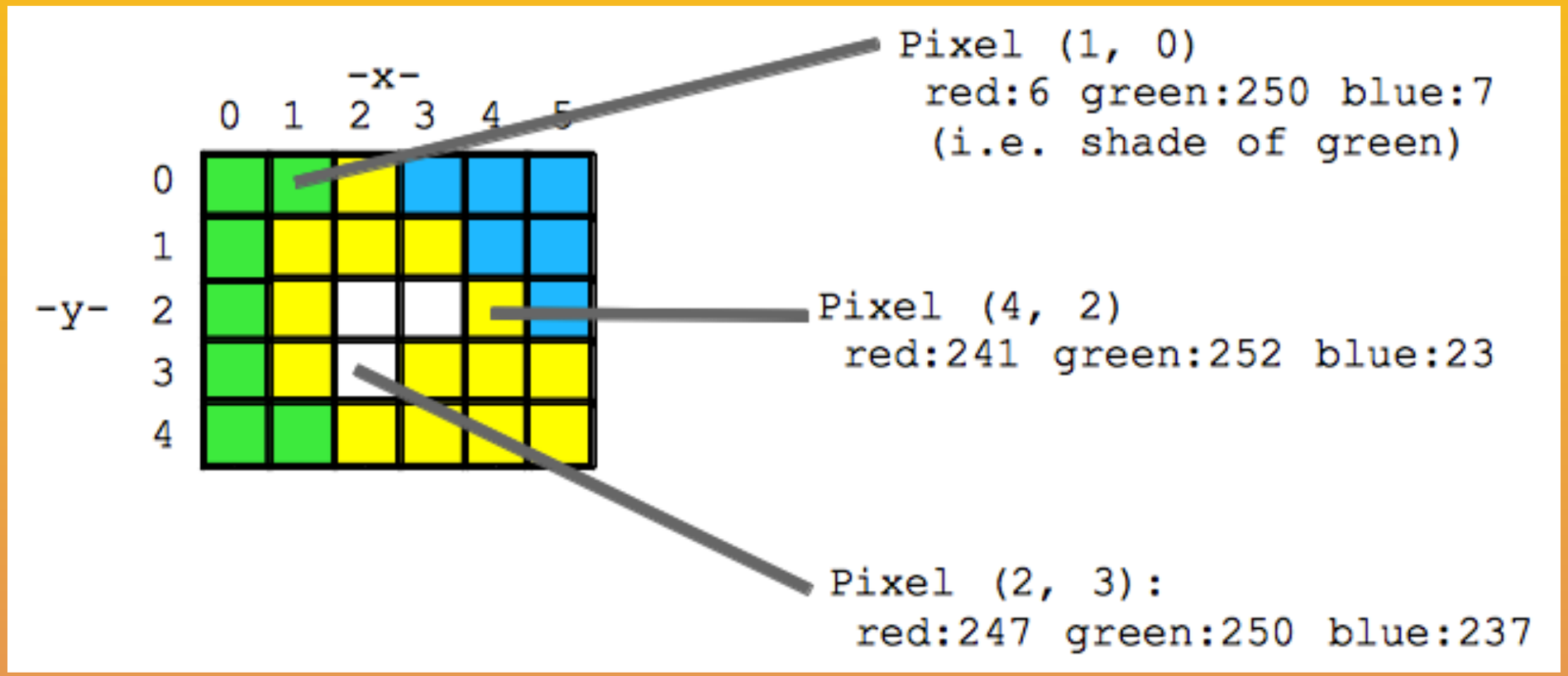
cost function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

```python
def sigmoid(z):
    """
    the sigmoid function is used as an activation fucntion to convert linear outputs
    to non linear outputs such that probability is outputed in between 0 and 1

    """
    return 1/(1+np.exp(-z))
def forward_propagate(X,theta1,theta2):
    """
    This function propagates through the network computing the output of
    every layer of the neural network with the given inputs and weights and
    computes the final output of the xor gate

    """
    #first, a column of biases is added to the input of the first layer
    a1=np.c_[np.ones(X.shape[0]),X]#mxn+1
    #the weights of the first layer are multiplied by the input of the first layer
    z1=a1.dot(theta1)#mxhidden
    #the input of the second layer is the output of the first layer, passed through the activation
 function and column of biases is added
    a2=np.c_[np.ones(X.shape[0]),sigmoid(z1)]#mxhidden+1
    #the input of the second layer is multiplied by the weights
    z3=a2.dot(theta2)
    #the output is passed through the activation function to obtain the final probability
    h3=sigmoid(z3)
    #print(h3.shape)
    return a1,z1,a2,z3,h3
```



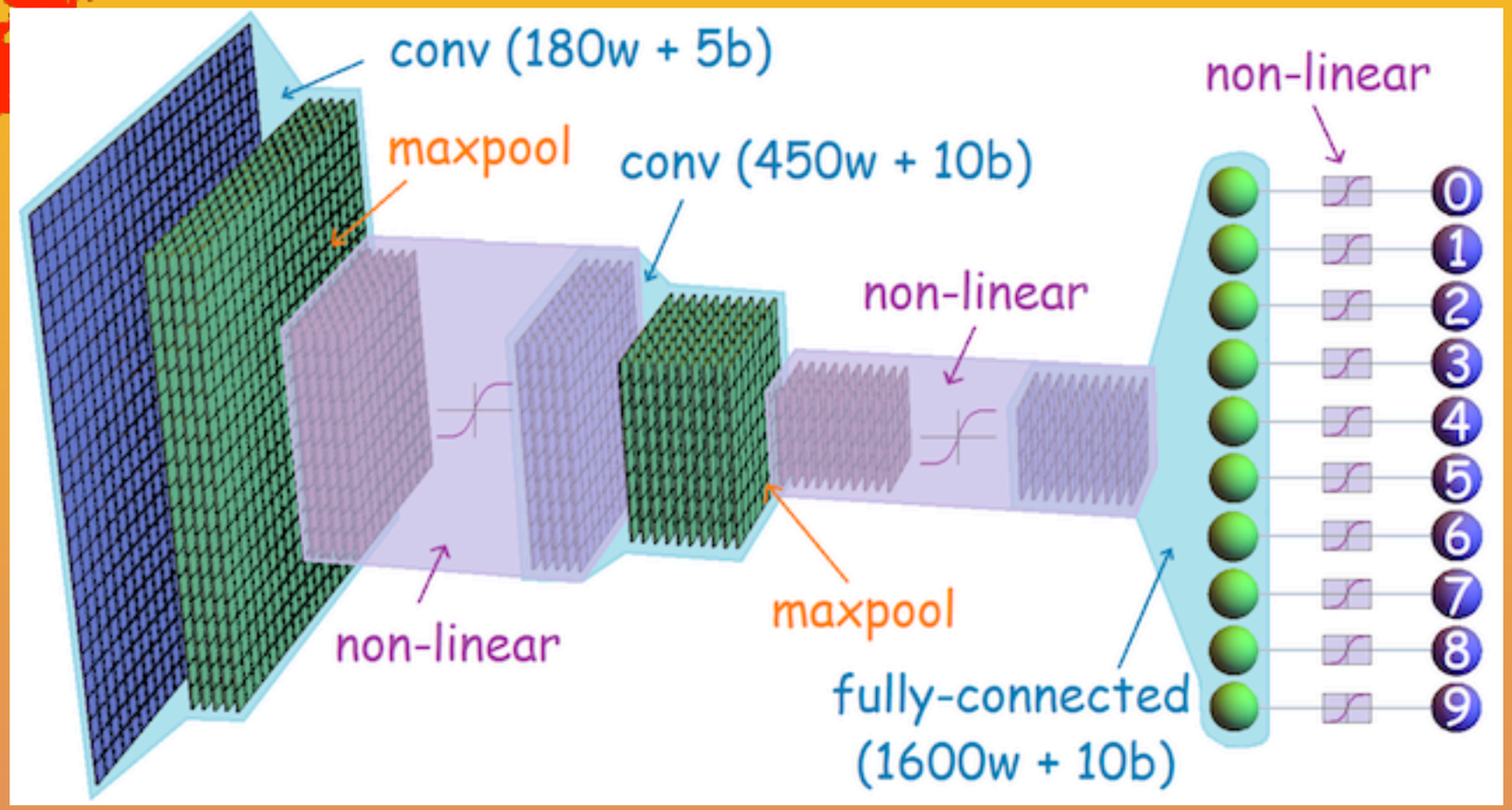

```python
for i in range(1000):
    a1, z1, a2, z3, hyp = forward_propagate(X, theta1, theta2)# for every iteration, forward propa
gation is carried out
    del_2= Y-hyp#the error of the final layer is calculated- the difference between the predicted
 and actual output
    #the error of the previous layer is found by computing the dot product of the error of the pre
vious layer and the weights of the second layer,without the column for biases.
    #this matrix is made to undergo element-wise multiplication with the output of the first layer
(taking into account the activation function)
    del_1=del_2.dot(theta2[1:,:].T)

    #the error of the second layer is multiplied element wise by the sigmoid gradient of the outpu
t of the second layer
    delta2=del_2

    #the error of the first layer is multiplied element wise by the sigmoid gradient of the output
 of the first layer
    delta1=del_1*sigmoid_grad(z1)

    #the parameters are updated using gradient descent
    theta2+=l_r*a2.T.dot(delta2)
    theta1+=l_r*a1.T.dot(delta1)
```



pixel values :0-255

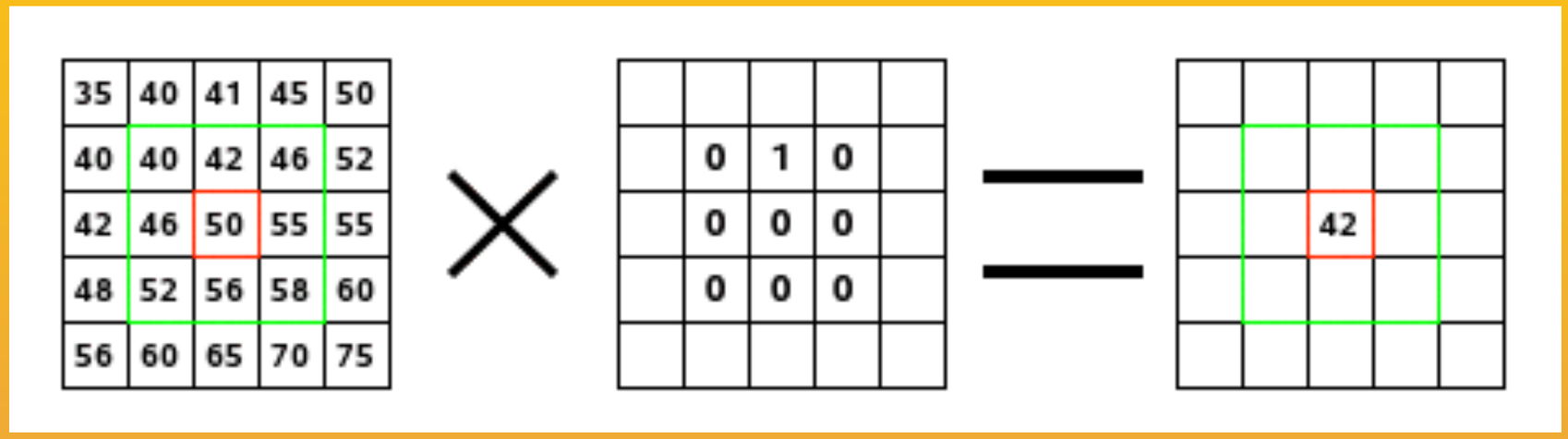
Height X width X channel

Convolutional Neural Networks

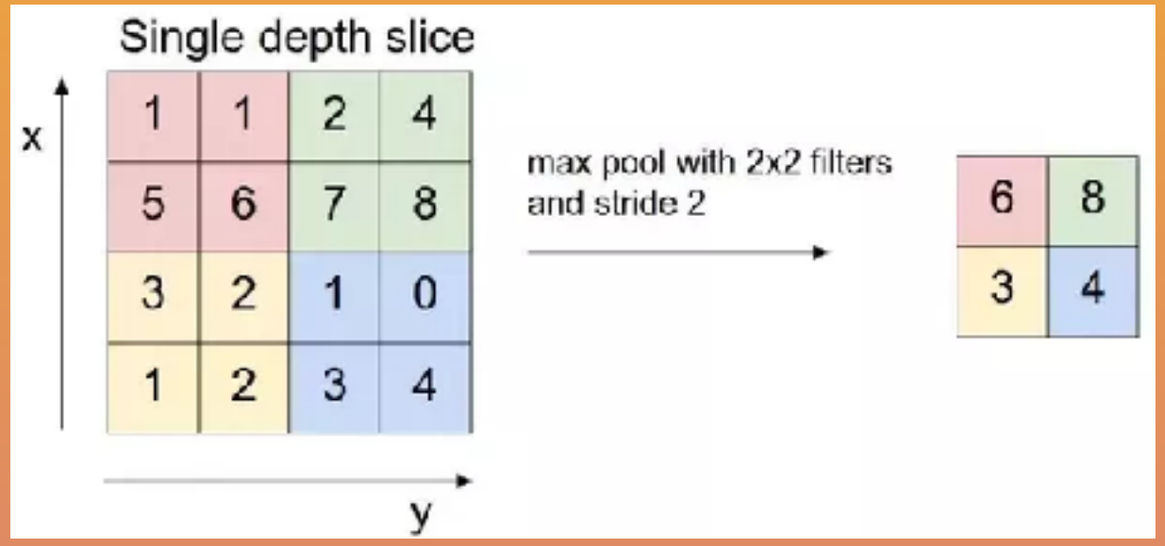


Training examples X features

1. Convolution



2. Max Pooling



3.RELU

4. Fully connected Layer

```python
def forward_propagation(X, parameters):
    W1 = parameters['W1']
    W2 = parameters['W2']

    Z1 = tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME')
    # RELU
    A1 = tf.nn.relu(Z1)
    # MAXPOOL
    P1 = tf.nn.max_pool(A1, ksize=[1, 8, 8, 1], strides=[1, 8, 8, 1], padding='SAME')
    # CONV2D
    Z2 = tf.nn.conv2d(P1, W2, strides=[1, 1, 1, 1], padding='SAME')
    # RELU
    A2 = tf.nn.relu(Z2)
    # MAXPOOL
    P2 = tf.nn.max_pool(A2, ksize=[1, 4, 4, 1], strides=[1, 4, 4, 1], padding='SAME')
    # FLATTEN
    P = tf.contrib.layers.flatten(P2)

    Z3 = tf.contrib.layers.fully_connected(P, 6, activation_fn=None)

    return Z3
```

```python
def compute_cost(Z3, Y):

    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=Z3, labels=Y))

    return cost
```

```python
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

Conv1 Conv2 Conv3 Conv4

→ more detail

Tate Britain AI exhibit

Recognition 2016

Reuters

Tate

Style Transfer

Leon A. Gatys, Alexander S. Ecker, Matthias Bethge

Image1

Conv 4

Alpa, beta

Max Pooling

Gram Matrix

Content

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^l - P_{ij}^l\right)^2 .$$

Style

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l .$$

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2$$

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

Experiment

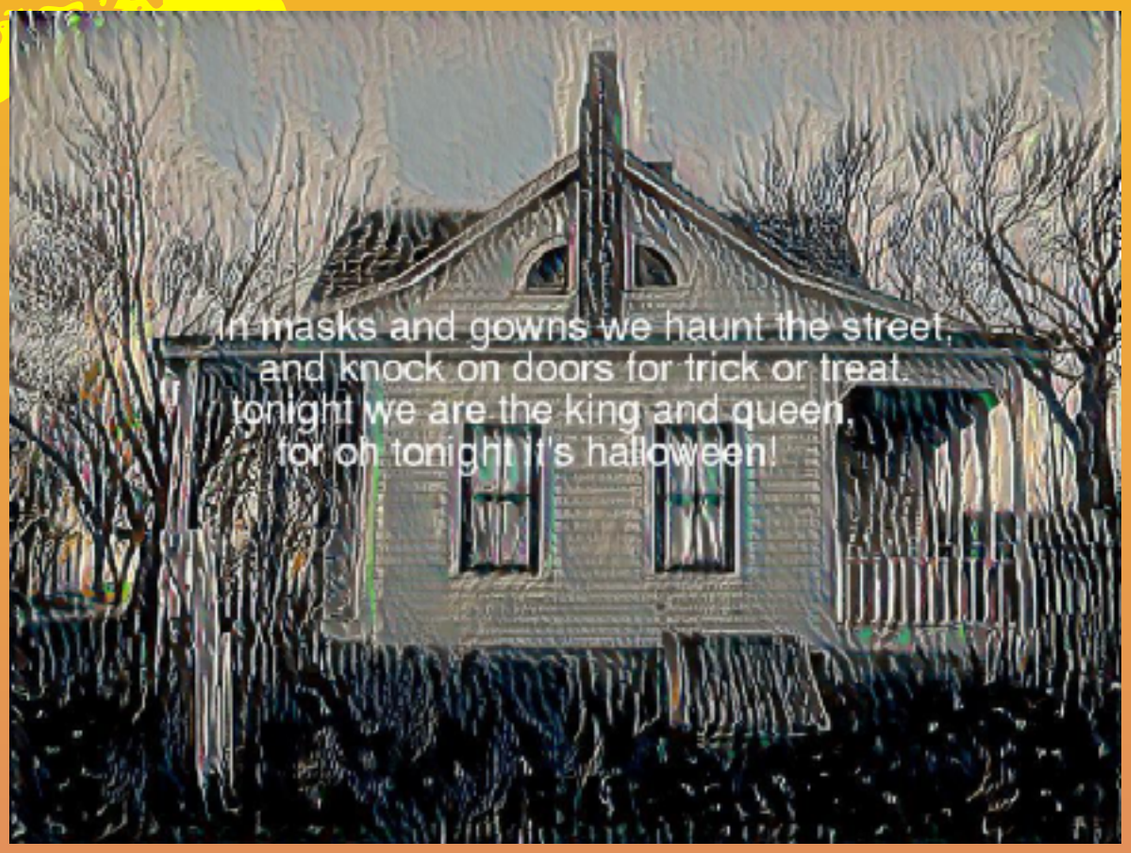
Style on style
genetic algorithms
Conv 2?
other types of pooling?

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

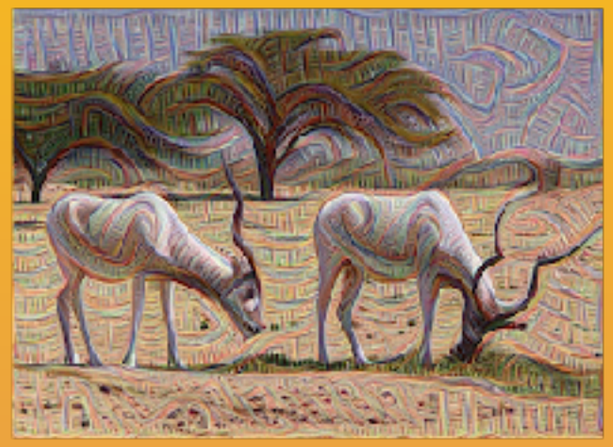
Noise → Using weights that minimise the cost function → Output

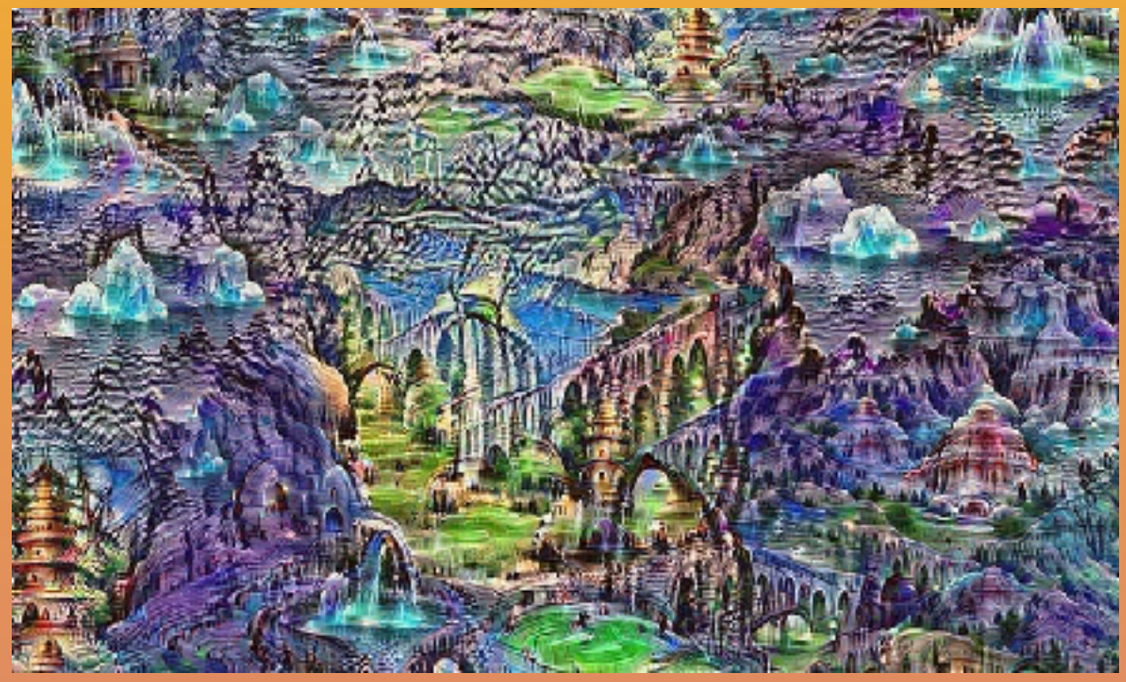
Style Transfer + Culture

Deep Greetings



Deep Dream





Experiment

Hybrid organisms
Fuse cultures
Scans/ medical Imaginary

Iterate over Outputs:

Features → Snapshot of inner workings → **feed as input**

Foster The People

Doing it for the Money

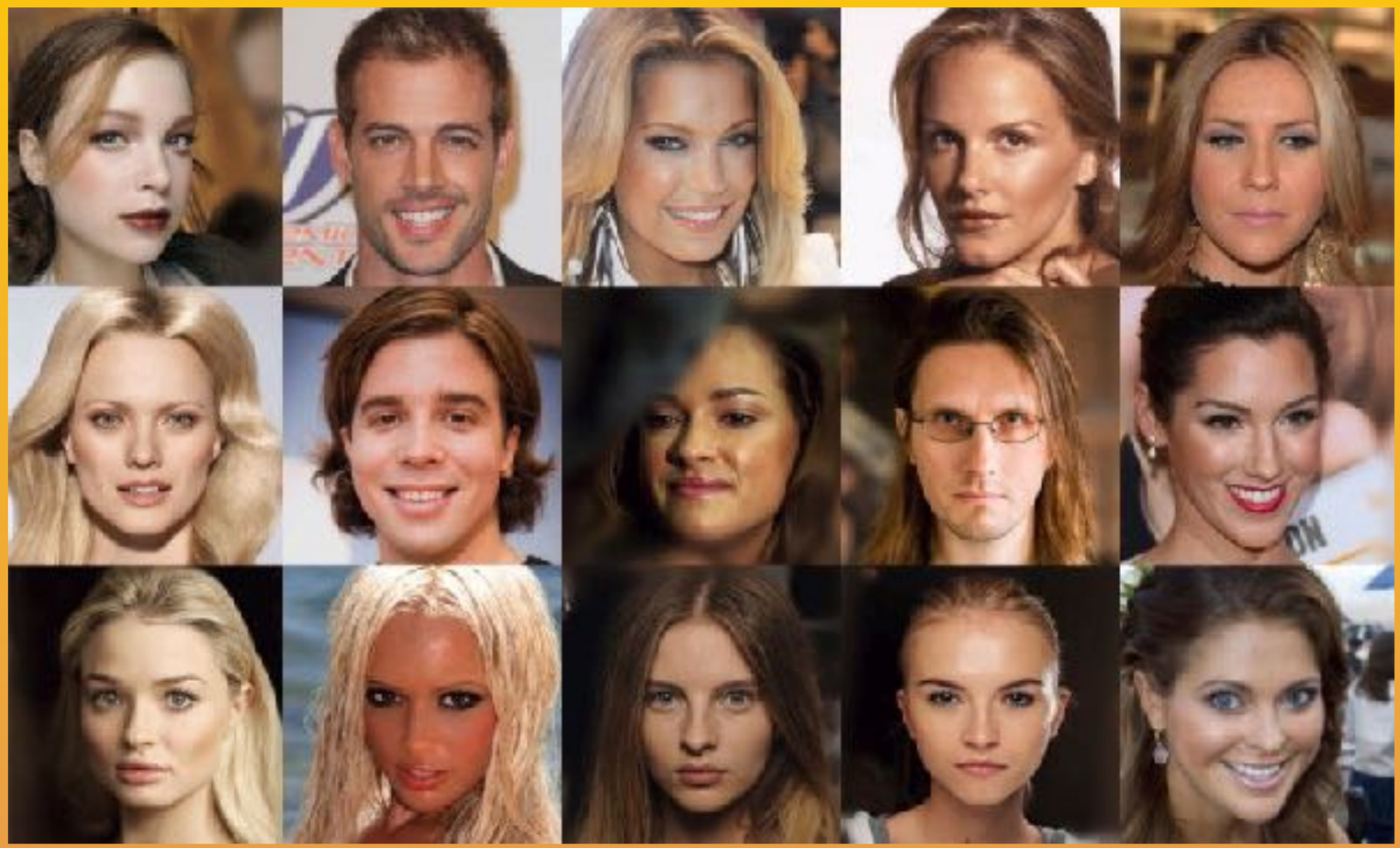
GANS

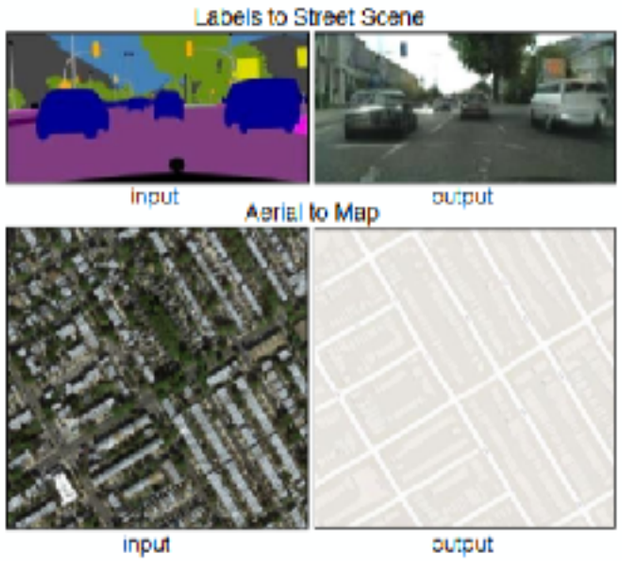
Discriminator

Generator

Training Data

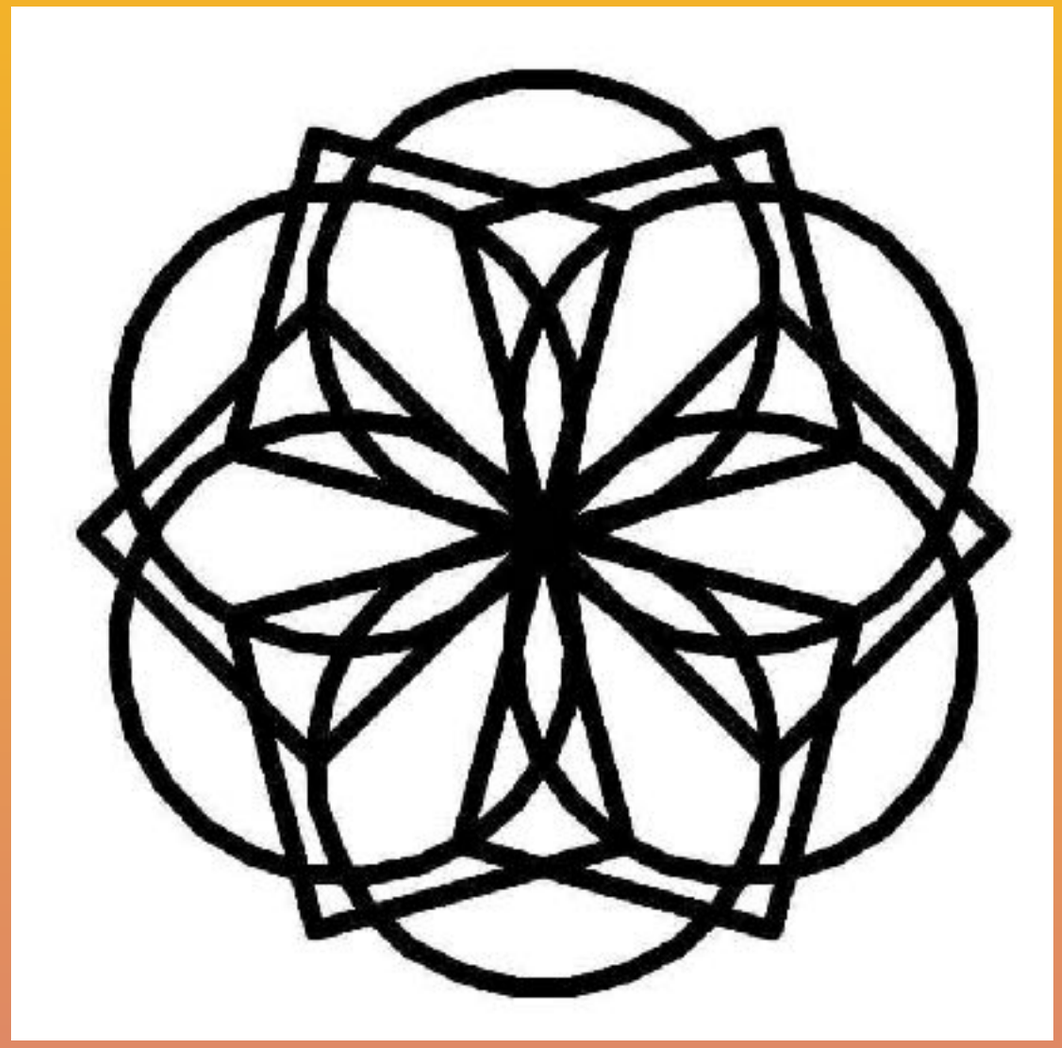
```python
def generator_fn(noise, weight_decay=2.5e-5, is_training=True):
    with framework.arg_scope(
            [layers.fully_connected, layers.conv2d_transpose],
            activation_fn=tf.nn.relu, normalizer_fn=layers.batch_norm,
            weights_regularizer=layers.l2_regularizer(weight_decay)), \
         framework.arg_scope([layers.batch_norm], is_training=is_training, zero_debias_moving_mean=True):
        net = layers.fully_connected(noise, 1024)
        net = layers.fully_connected(net, 800 * 800 * 1)
        net = tf.reshape(net, [-1, 800, 800, 1])
        net = layers.conv2d_transpose(net, 64, [4, 4], stride=2)
        net = layers.conv2d_transpose(net, 32, [4, 4], stride=2)
        # Make sure that generator output is in the same range as `inputs`
        # ie [-1, 1].
        net = layers.conv2d(net, 1, 4, normalizer_fn=None, activation_fn=tf.tanh)

        return net


def discriminator_fn(img, unused_conditioning, weight_decay=2.5e-5,
                     is_training=True):
    with framework.arg_scope(
            [layers.conv2d, layers.fully_connected],
            activation_fn=leaky_relu, normalizer_fn=None,
            weights_regularizer=layers.l2_regularizer(weight_decay),
            biases_regularizer=layers.l2_regularizer(weight_decay)):
        net = layers.conv2d(img, 64, [4, 4], stride=2)
        net = layers.conv2d(net, 128, [4, 4], stride=2)
        net = layers.flatten(net)
        with framework.arg_scope([layers.batch_norm], is_training=is_training):
            net = layers.fully_connected(net, 1024, normalizer_fn=layers.batch_norm)
        return layers.linear(net, 1)
```




Labels to Street Scene
input
output
Aerial to Map
input
output


Input
Ground truth
Output

Deep Rangoli



Coded Couture

- Abstract
- Style Transfer
- Creative Coding
- GANS
- Augmented Intelligence

Why should this be explored further?

- Evolution, expression
- Innovation
- applications
- Architectures
- logic
- socio-economic applications

" there's more than one way to do it"

@Aparnaakk

aparnakrishnakumar.com

@Aparnaakk