# Weekly Report

Wangwon Lee, 2019/04/13

| This week | Next week |
| --- | --- |
| • **Visualization**<br>- Activation Maximization<br>- Evaluate (Implement)<br>- Evaluate (Softmax)<br>- Evaluate (Softmax to Linear) | • **Visualization**<br>- Apply 1-D model<br>- More clearly<br>- The other model(tanh, long filter)<br>- The other label<br>- CAM(Class Activation Map) |

## Interesting and new finding

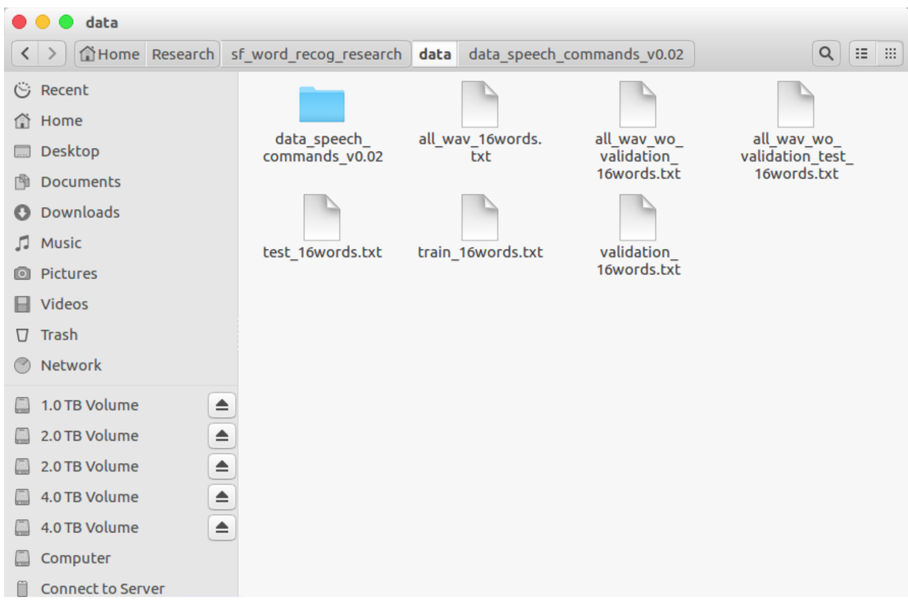
- Visualization
- Activation Maximization

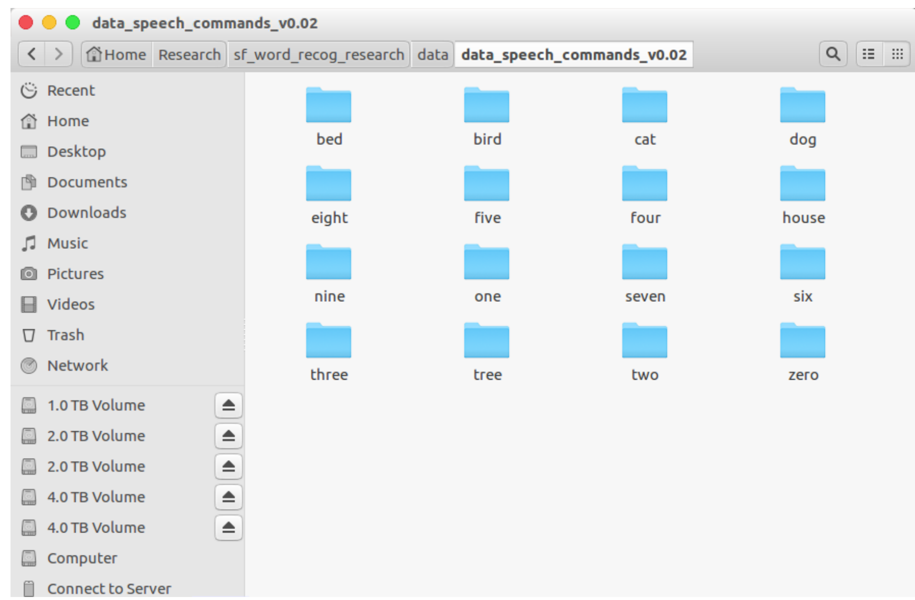
## The aim of this month / Discussion

- **The aim of this month:** To investigate about CNN and visualization.

# Audio Classification - Previous Work

- Data is low-waveform.
  - sec: 1, sampling rate: 16000, type: float32, channel: mono
- 16 class data.
  - 'zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'bed', 'bird', 'tree', 'cat', 'house', 'dog'
- Train: 40851(≒80%), Validation: 4796(≒10%), Test: 5297(≒10%)

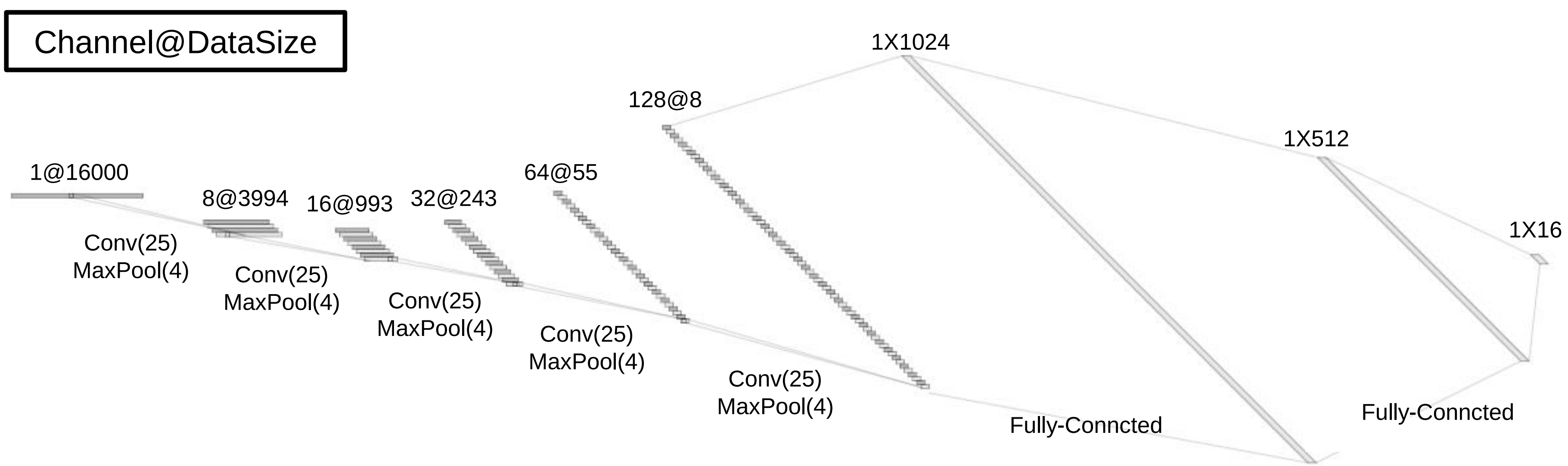


# Audio Classification - Previous Work

- For example, '5Conv, 2FC' baseline model's detail.
- It just flatten 2D model. (5X5 filter->1X25 filter, 2X2 stride->1X4 stride)

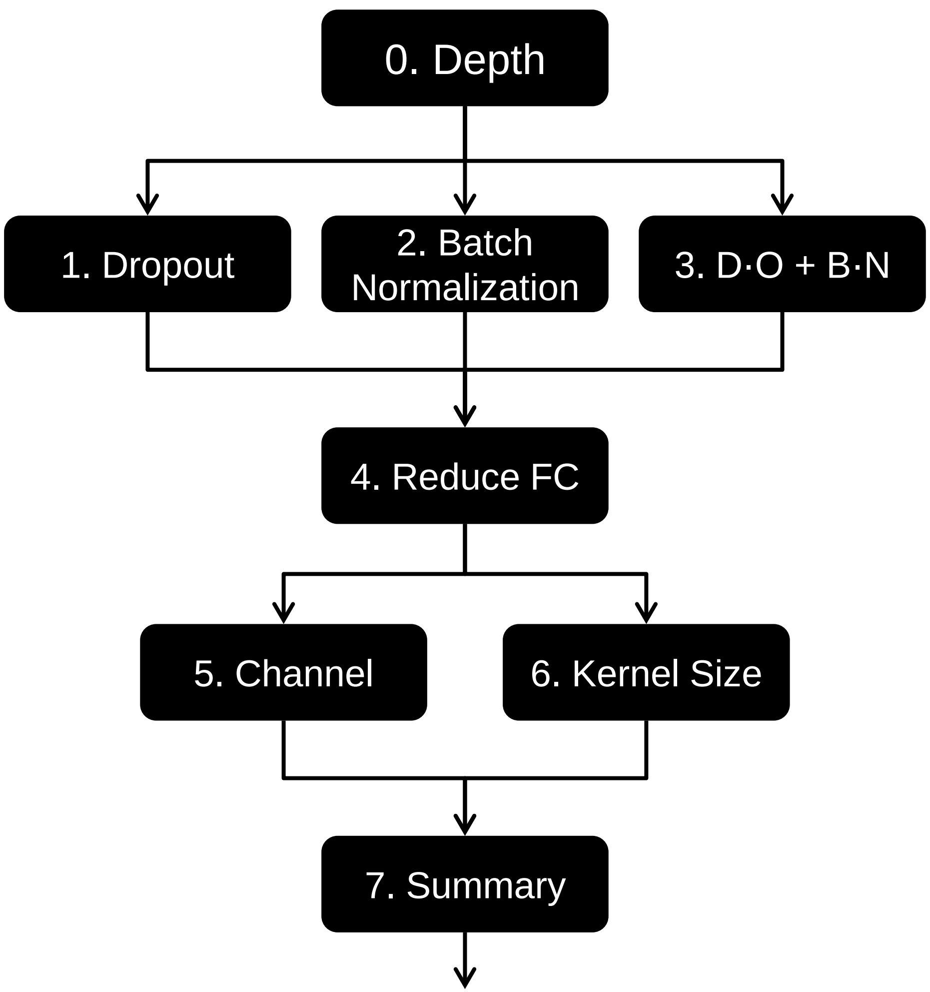
- Input: 16000X1 low waveform.
- Output:1x16 labeled one hot vector. ('zero', …, 'eight', …, 'house', 'dog')
- Loss: cross entropy loss
- Obtimizer: Adam

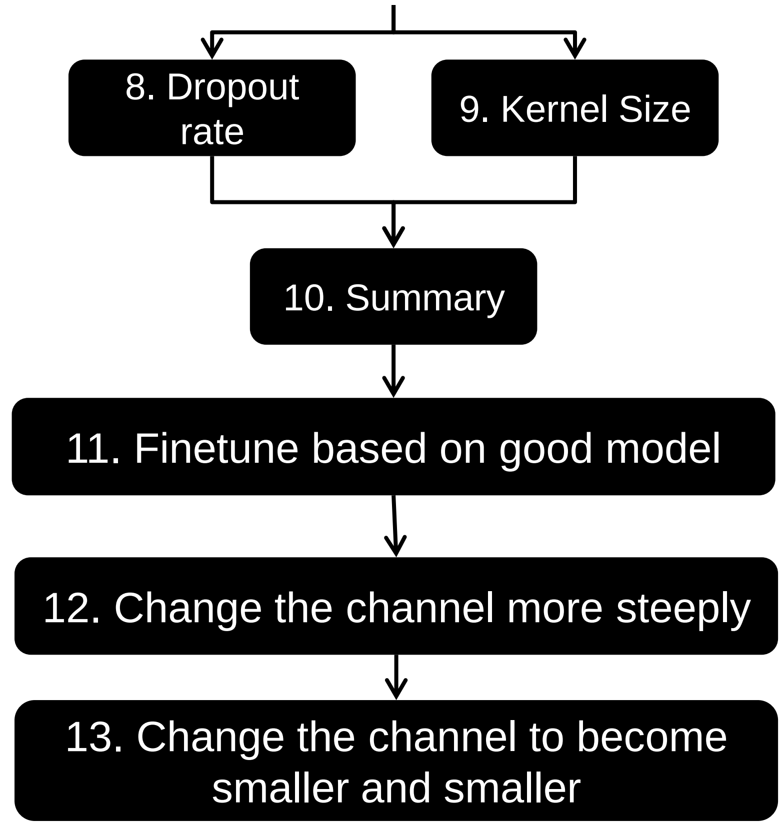



**[ Baseline Architecture]**

# Audio Classification - Previous Work

- Fine tuning task in 1D-CNN




Previous Work

# Audio Classification - Previous Work

- This is SOTA(State Of The Art) in current research.

| | Architecture (i = 0,1,2…) | 1D DO(0.5) | 1D BN | 1D DO+BN | Params |
|---|---|---|---|---|---|
| | baseline | | | | |
| base model | 5 Conv(25, 8*$2^i$), 5 Pool(4), 2 FC | 0.9090 | 0.9072 | 0.9240 | 1,855,056 |
| | Accuracy and Number of parameters | | | | |
| Custom channel 32 DO(0.75) | 8 CONV(5, 64) | 0.9533 | 0.9285 | 0.9391 | 94,768 |
| Custom channel 64 DO(0.75) | 8 CONV(5, 128) | 0.9589 | X | 0.9497 | 363,600 |
| Custom VGG style DO(0.75) | 16 CONV(3, 128) , 8 Pool | 0.9620 | 0.9423 | 0.9136 | 470,736 |
| | Only Accuracy | | | | |
| Custom channel 128 DO(0.25)+BN | 9 CONV(5, 512) | 0.9535 | X | 0.9701 | 2,071,184 |

# Activation Maximization

- Synthesize an input pattern image that can maximize a specific neurons activation in arbitrary layers.
- The preferred input can indicate what features of a neuron has learned.

$$x^* = \underset{x}{\operatorname{argmax}}\, a_{i,l}(\theta, x),$$

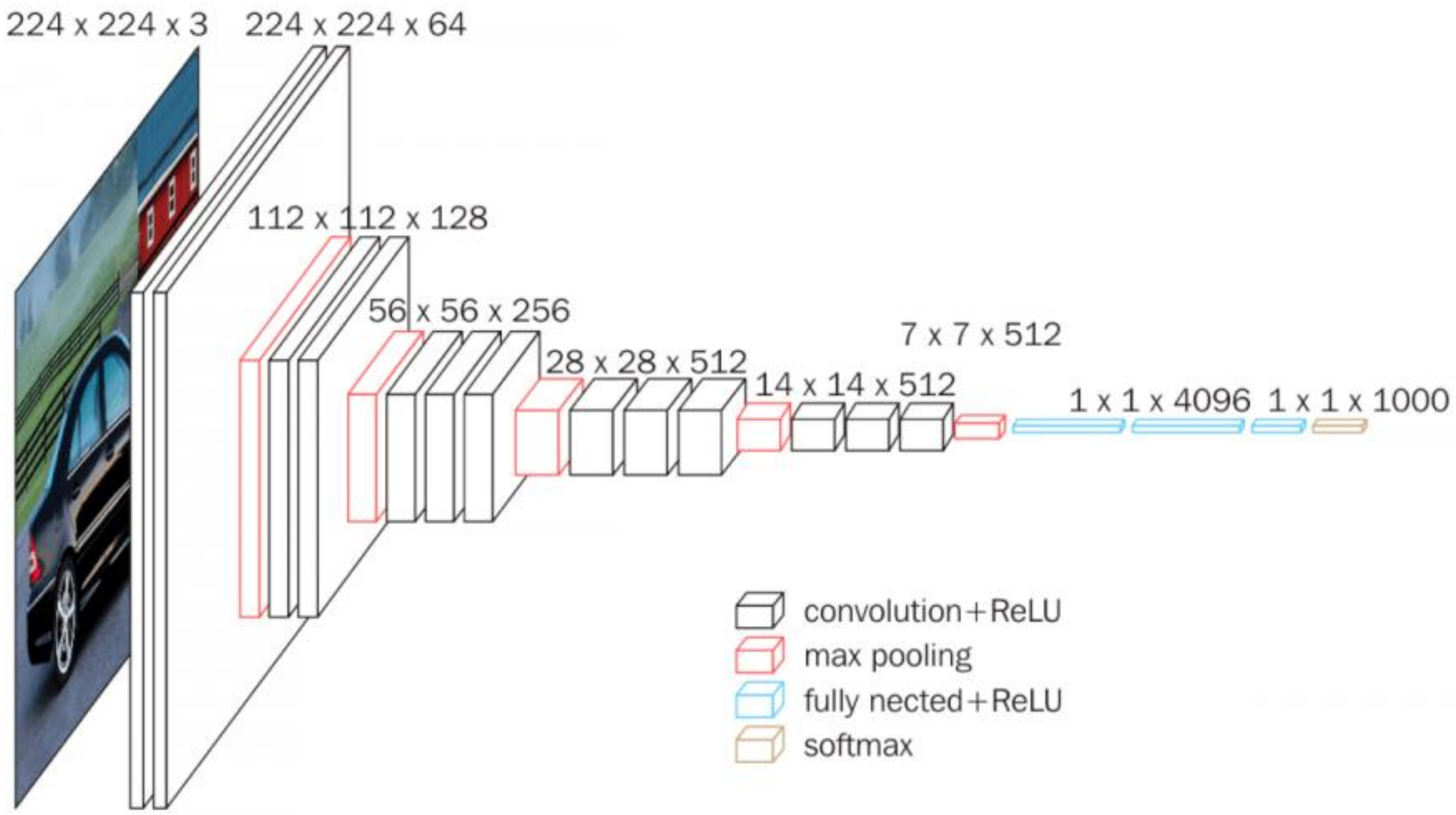
$$x \leftarrow x + \eta \cdot \frac{\partial a_{i,l}(\theta, x)}{\partial x}$$

a: filter
i: layer index
l: filter index
θ: weight, bias
x: input noise image

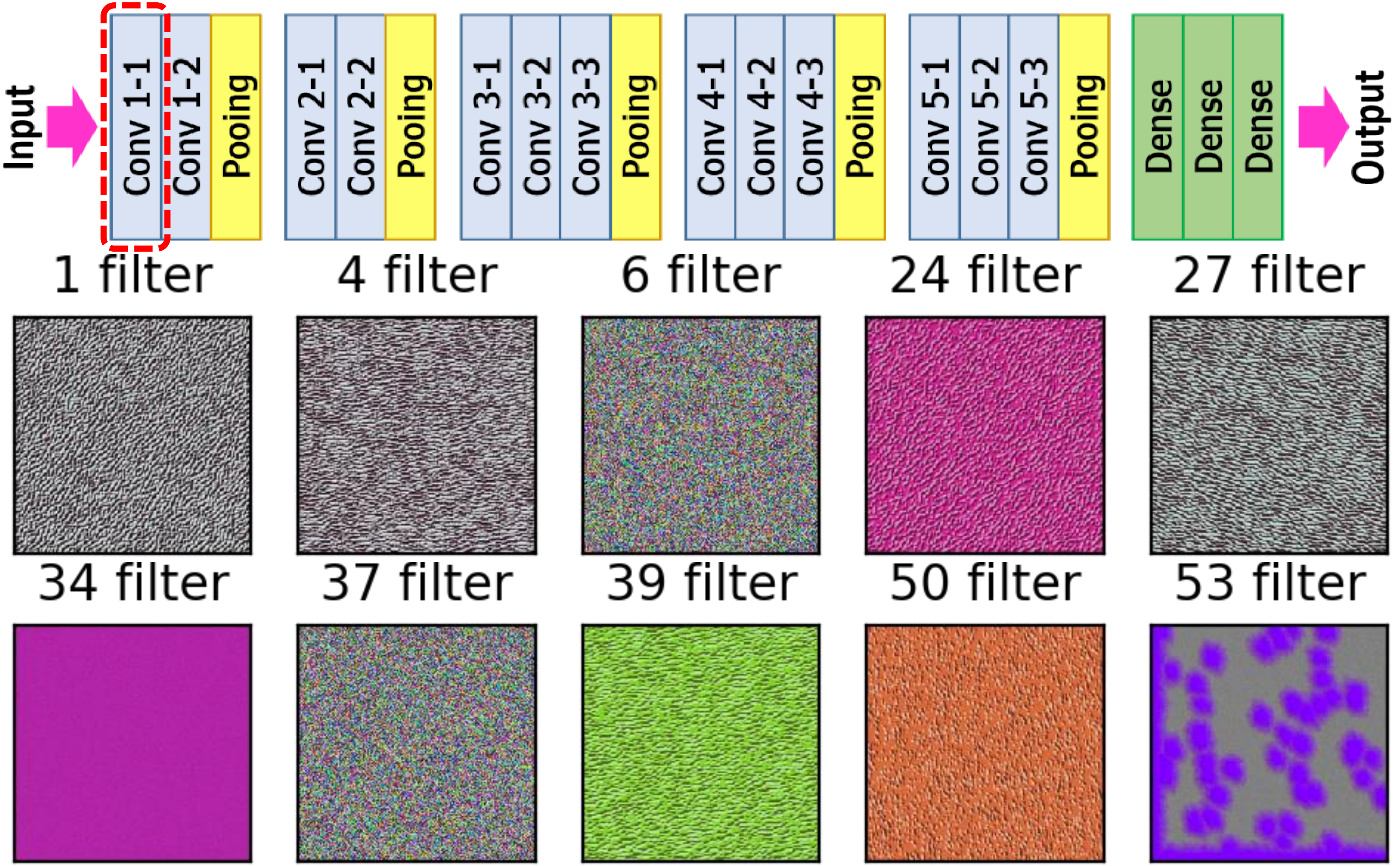
η: learning rate
(if η is low => focus on texture)
(if η is high => focus on shape)

# AM – Pretrained Model (VGG 16)

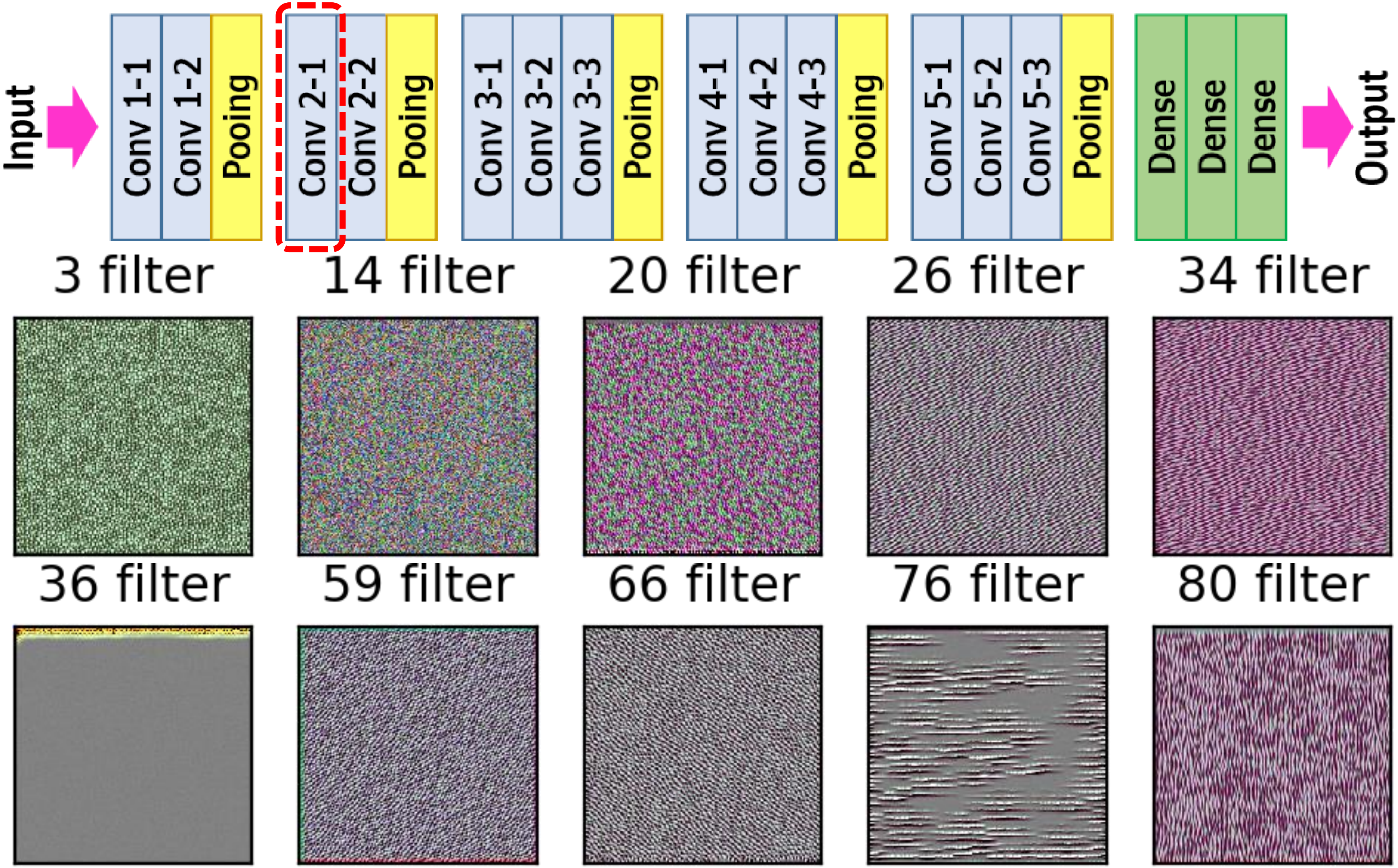
- Input: imagenet data (224*224*3)
- Output: 1000 class (ex. zebra, slug, hen, gold fish, etc… )



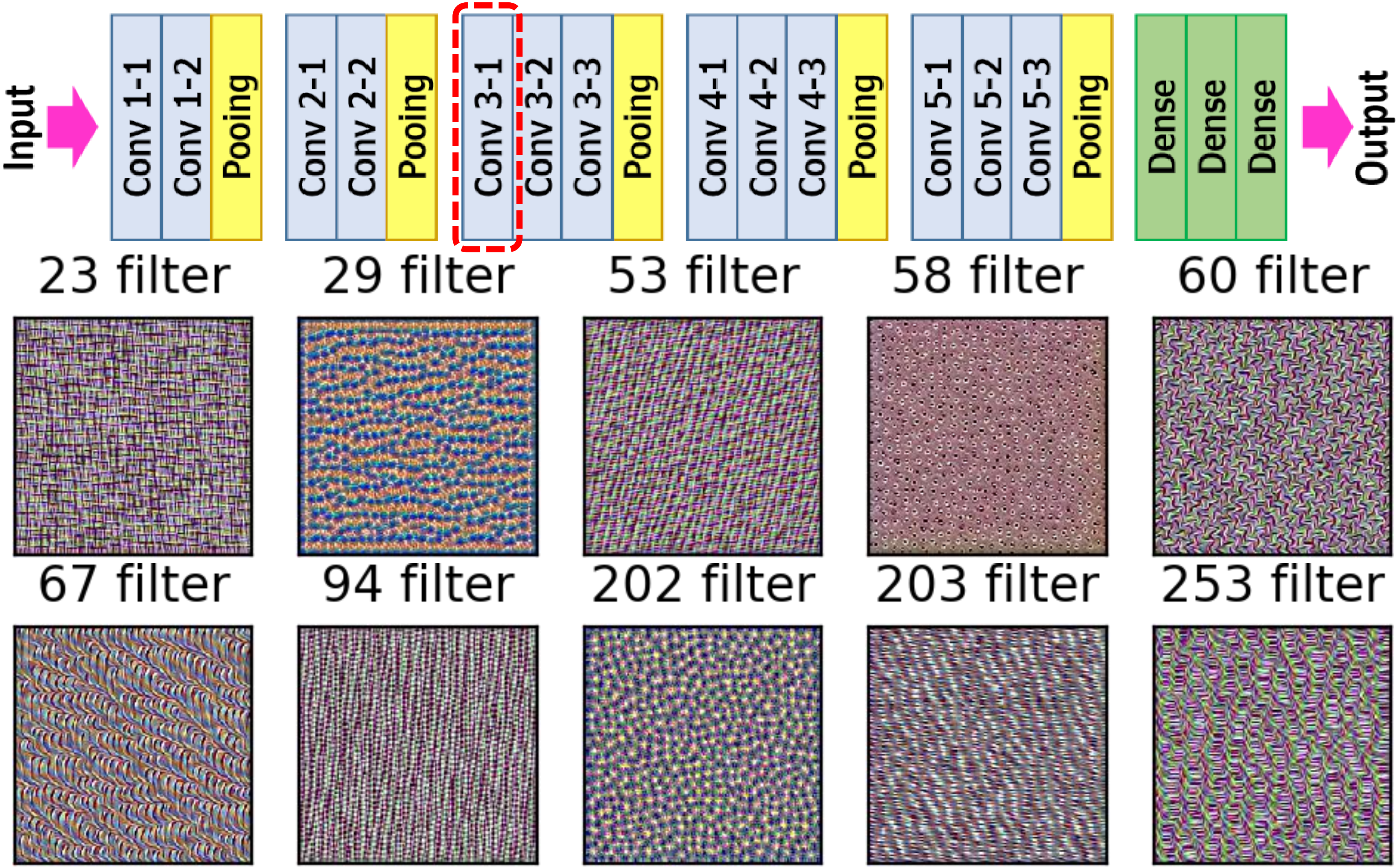
# AM – 1th Conv layer (VGG 16)



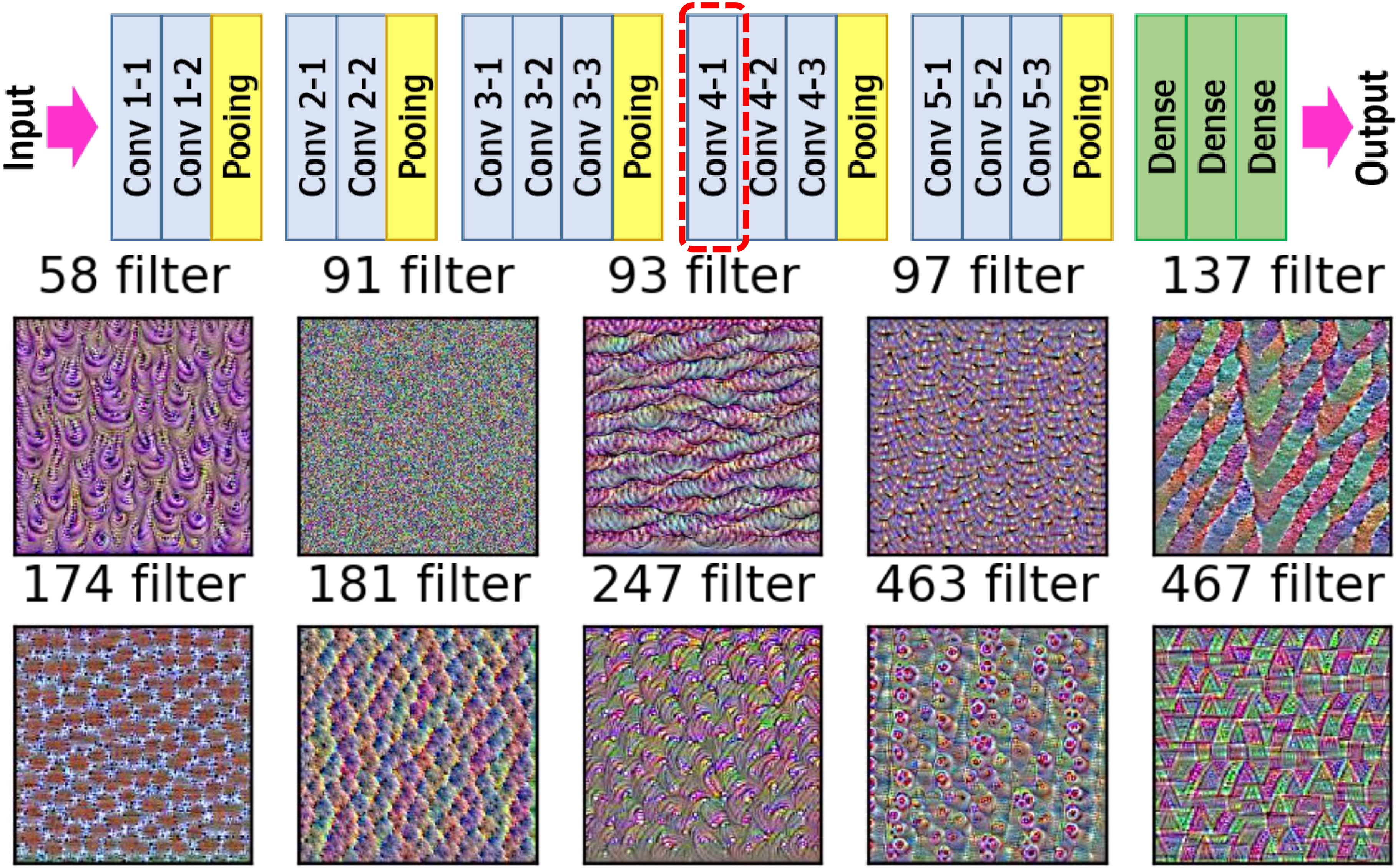
# AM – 2th Conv layer (VGG 16)



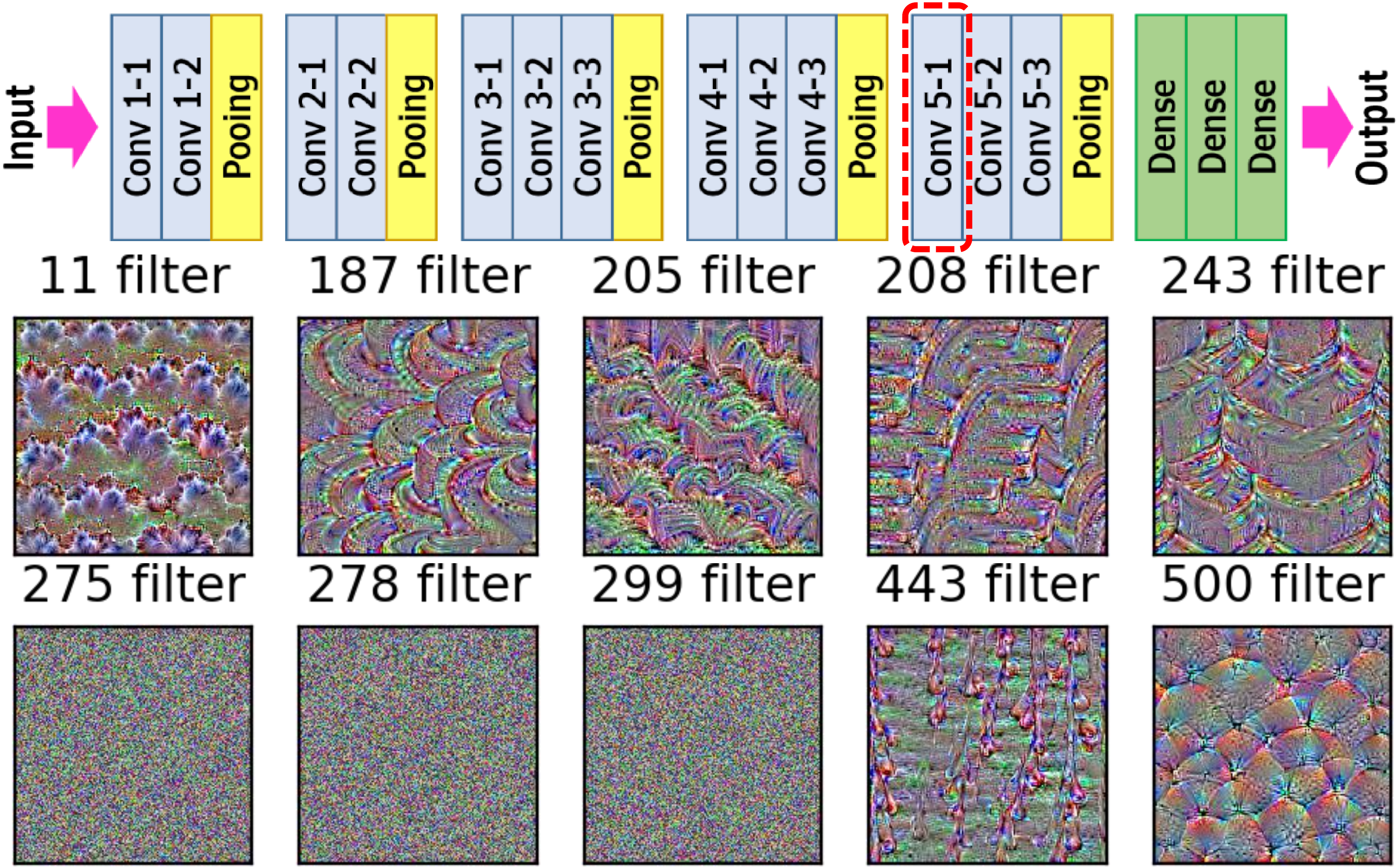
# AM – 3th Conv layer (VGG 16)



# AM – 4th Conv layer (VGG 16)



# AM – 5th Conv layer (VGG 16)



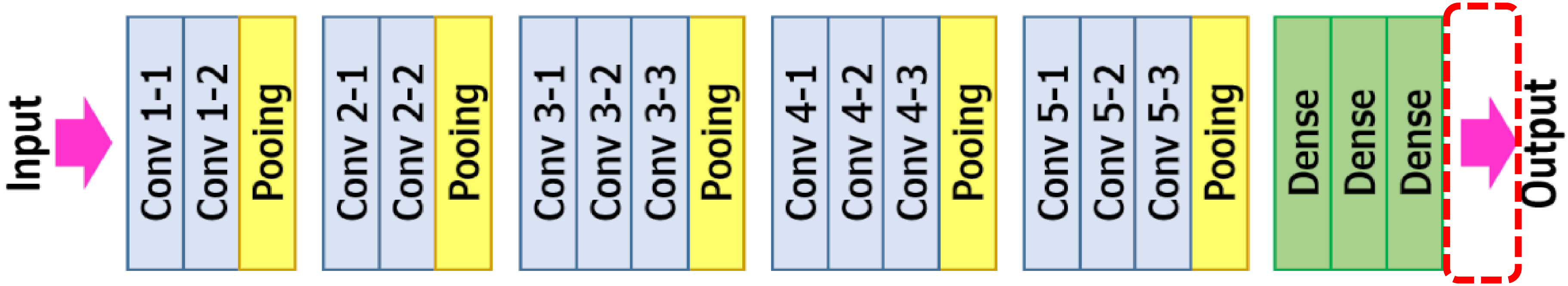
# AM – Evaluate (implement)

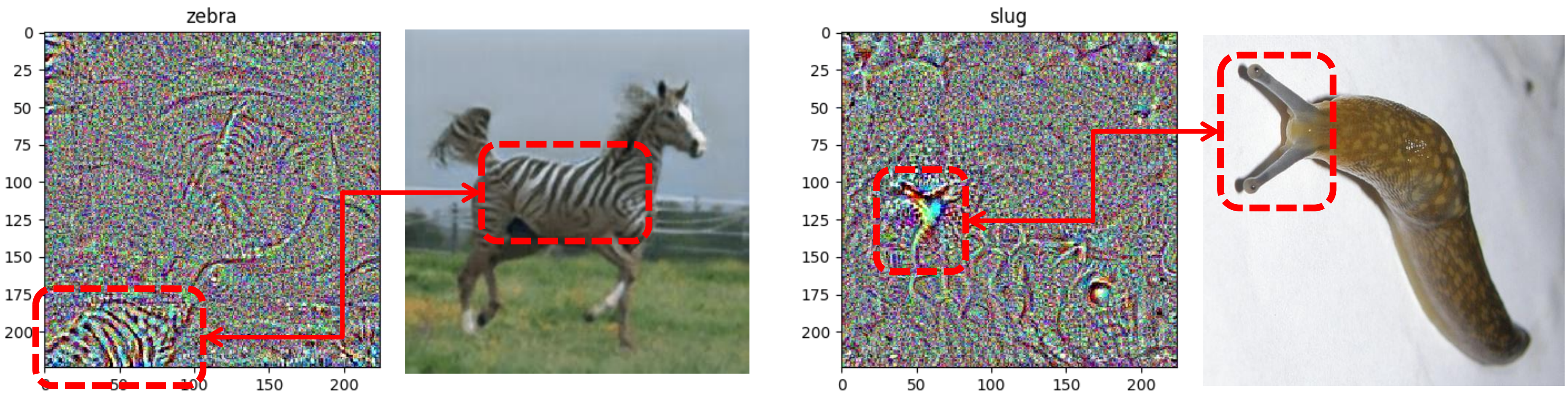
- Synthesize an input pattern image that can maximize a specific neurons activation in arbitrary layers.
- The preferred input can indicate what features of a neuron has learned.

```
def get_activation_maximization(model, layer_name, filter_index, step=2, epochs=200):
    layer_dict = dict([(layer.name, layer) for layer in model.layers])
    layer_output = layer_dict[layer_name].output
    loss = K.mean(layer_output[:, :, :, filter_index])

    # compute the gradient of the input picture wrt this loss
    grads = K.gradients(loss, model.input)[0]
    # normalization trick: we normalize the gradient
    grads /= (K.sqrt(K.mean(K.square(grads))) + K.epsilon())

    # this function returns the loss and grads given the input picture
    iterate = K.function([model.input], [loss, grads])

    output_dim=(224, 224)
    input_img_data = np.random.random(
        (1, *output_dim, 3))
    input_img_data = (input_img_data - 0.5) * 20 + 128

    # we start from a gray image with some noise
    for i in range(epochs):
        loss_value, grads_value = iterate([input_img_data])
        input_img_data += grads_value * step
    return input_img_data
```

$$x \leftarrow x + \boxed{\eta} \cdot \boxed{\frac{\partial a_{i,l}(\theta, \boxed{x})}{\partial x}}$$

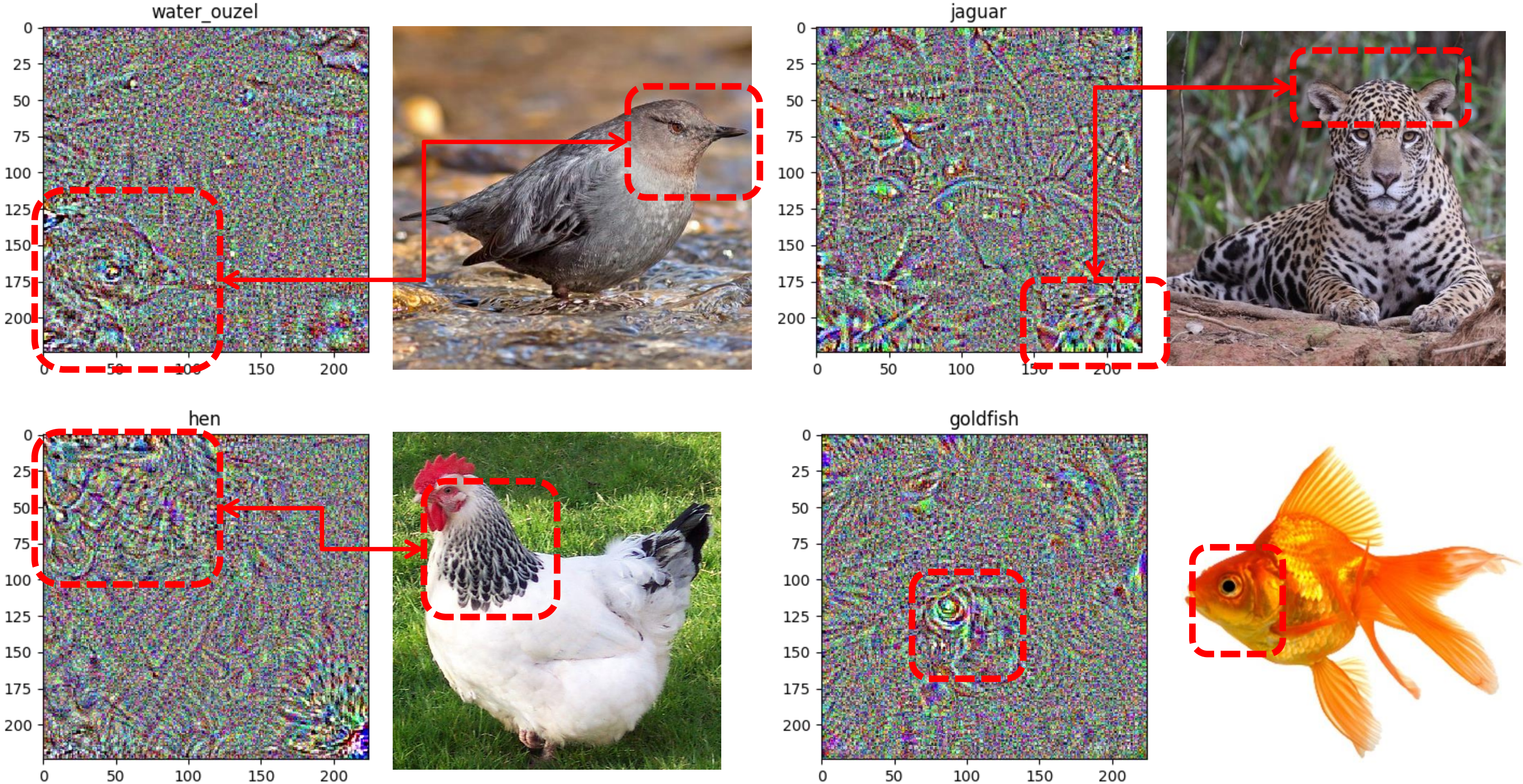
# AM – Evaluate (Softmax)



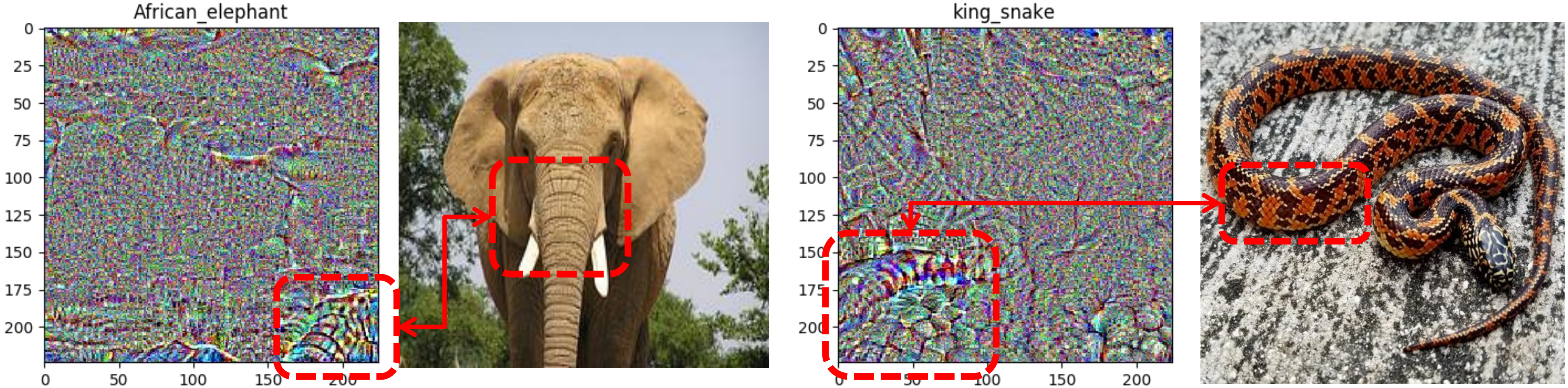

- In Softmax layer, if target node's activation is higher, that is representation of target label

# AM – Evaluate (Softmax)



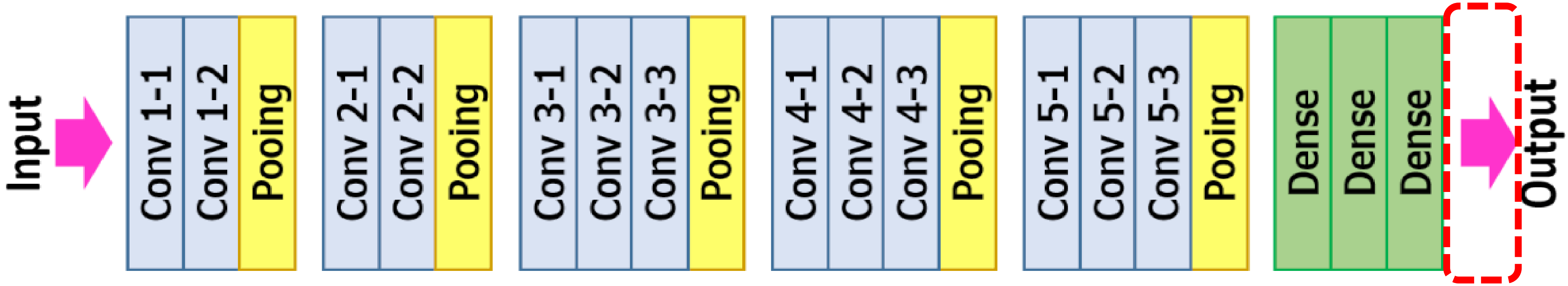

- In Softmax layer, if target node's activation is higher, that is representation of target label

# AM – Evaluate (Softmax)



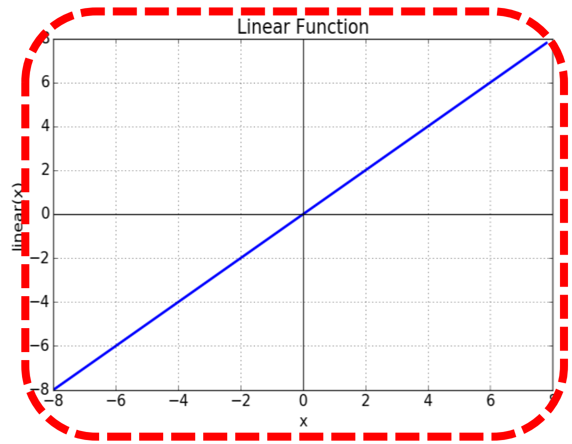

- We can find each pattern or shape or both.
- But, Is it well done…?
- The most of result is not clear.

# AM – Evaluate (Softmax To Linear)




0.0005801218
0.014823454
0.6863134
0.9589587
0.998198
0.999982
0.9999883
0.99999094
0.9999925
0.9999937
0.9999943
0.9999949
0.99999535
0.9999957
0.9999962
0.9999963
0.99999666

$$S(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{j} e^{y_j}}$$

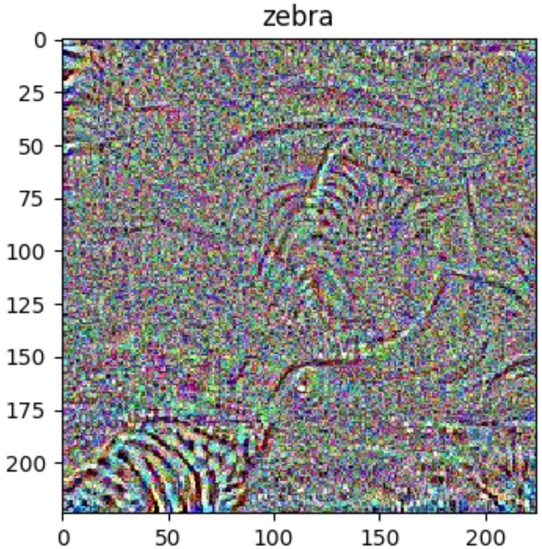


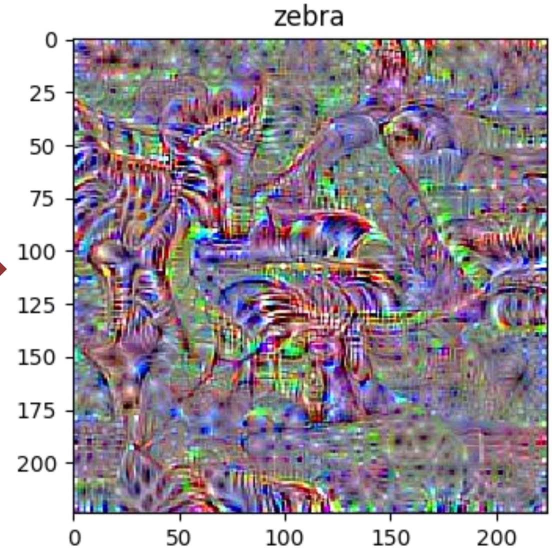

12.852844
37.658836
71.39896
100.16684
118.963936
154.92099
163.93884
197.65488
214.9229
235.99213
257.5274
281.22534
299.42224
321.27838
344.77408
361.18488
387.69177
403.08344

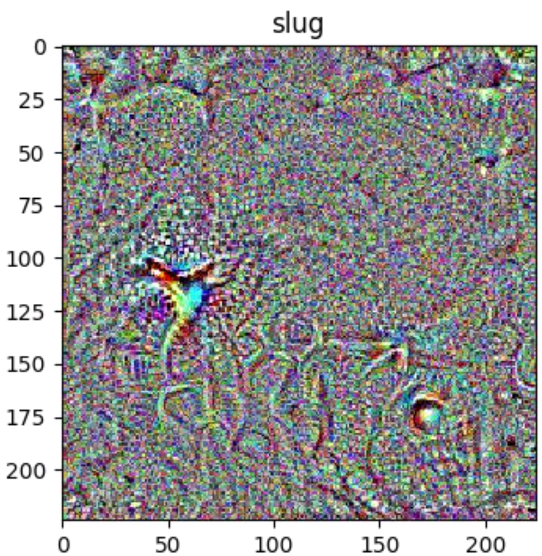
<Loss> < Softmax ($0 \leq y \leq 1$) > < Linear (-inf < y < inf) > <Loss>

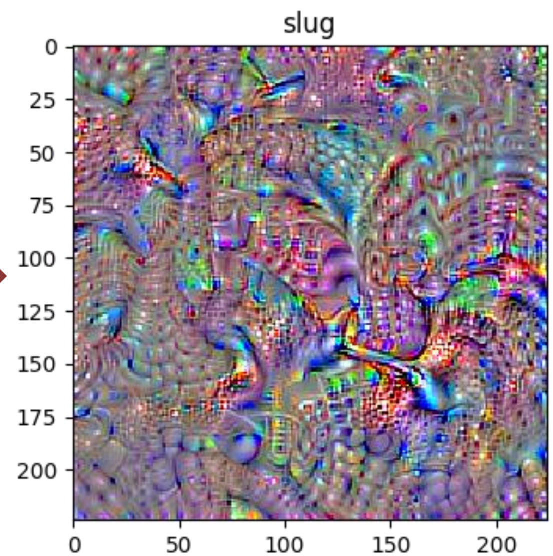
- Because of softmax range, Image is no longer updated
- To solve this, Change Softmax function to Linear function (It's Just trick)

# AM – Evaluate (Softmax To Linear)

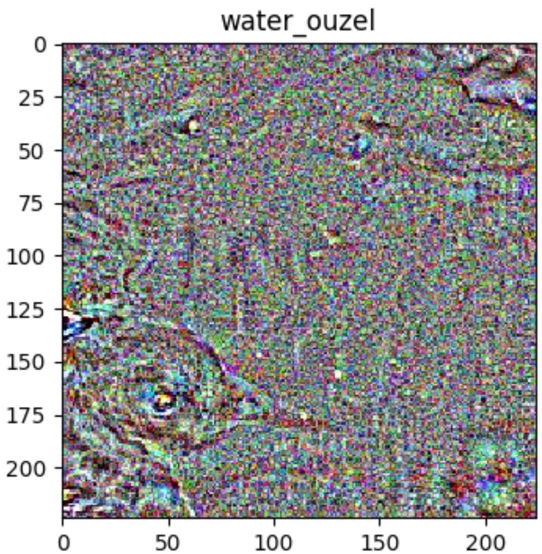


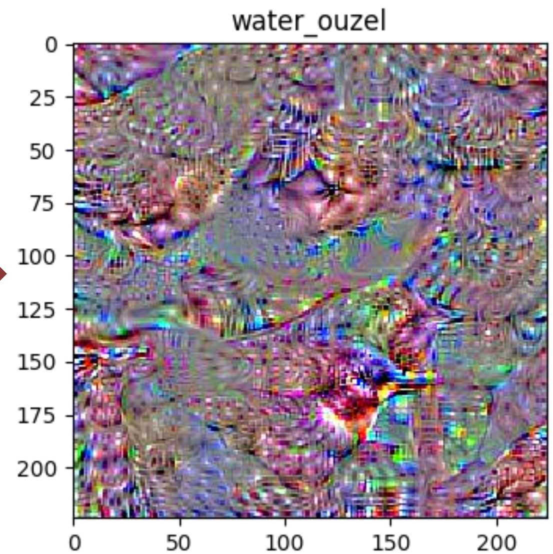



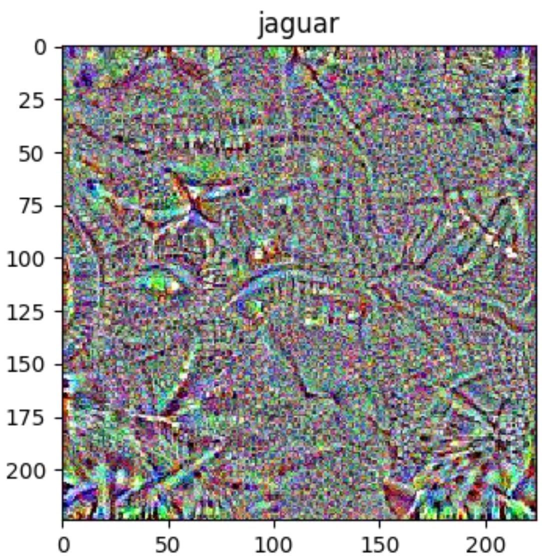



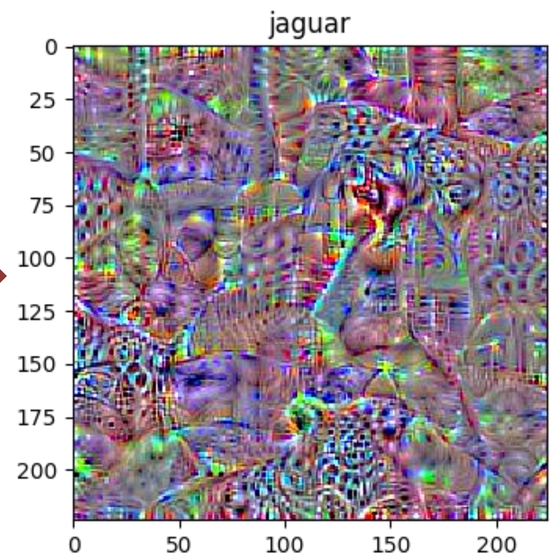




# AM – Evaluate (Softmax To Linear)

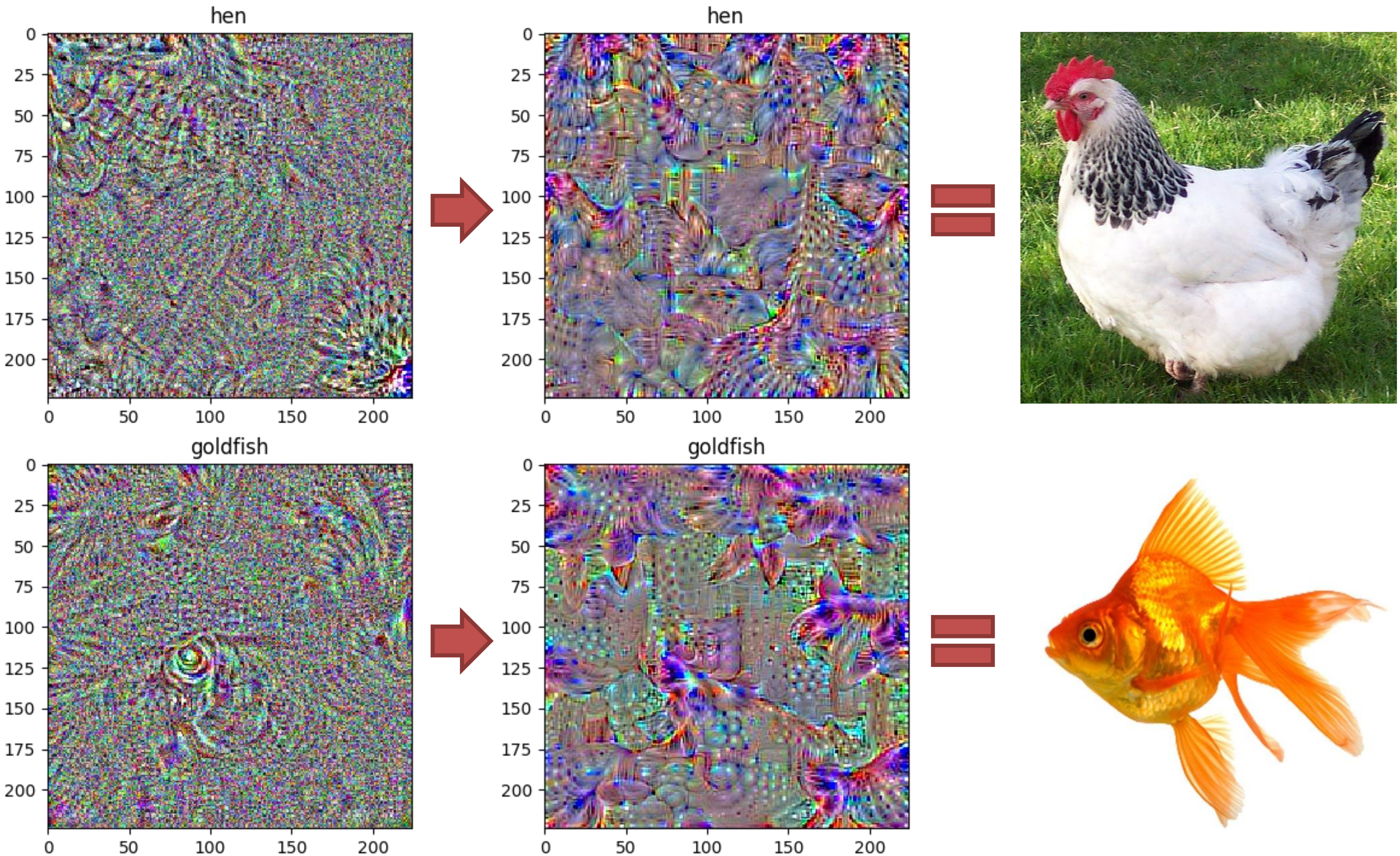


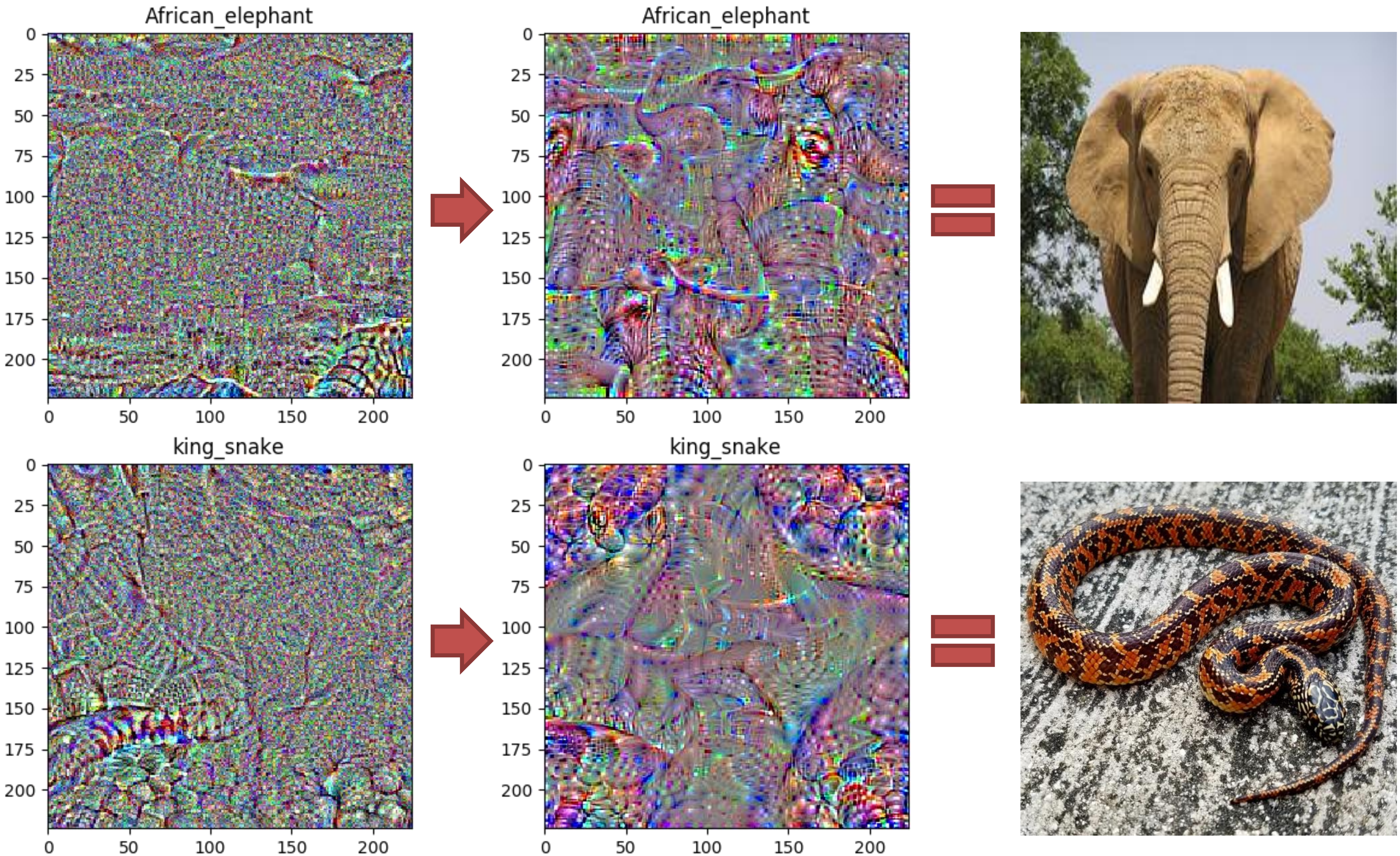




# AM – Evaluate (Softmax To Linear)

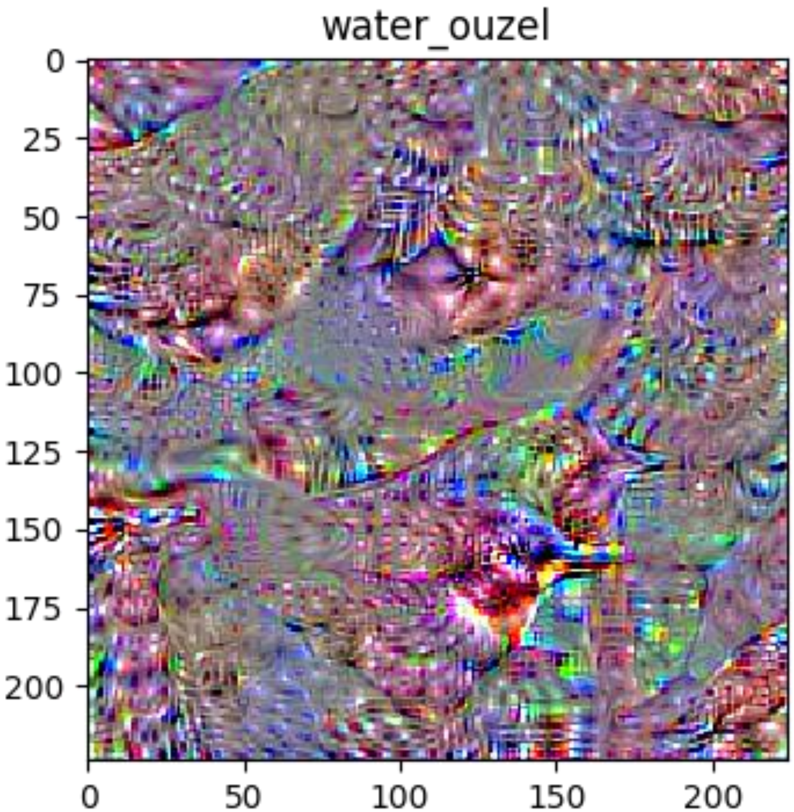


# AM – Evaluate (Softmax To Linear)

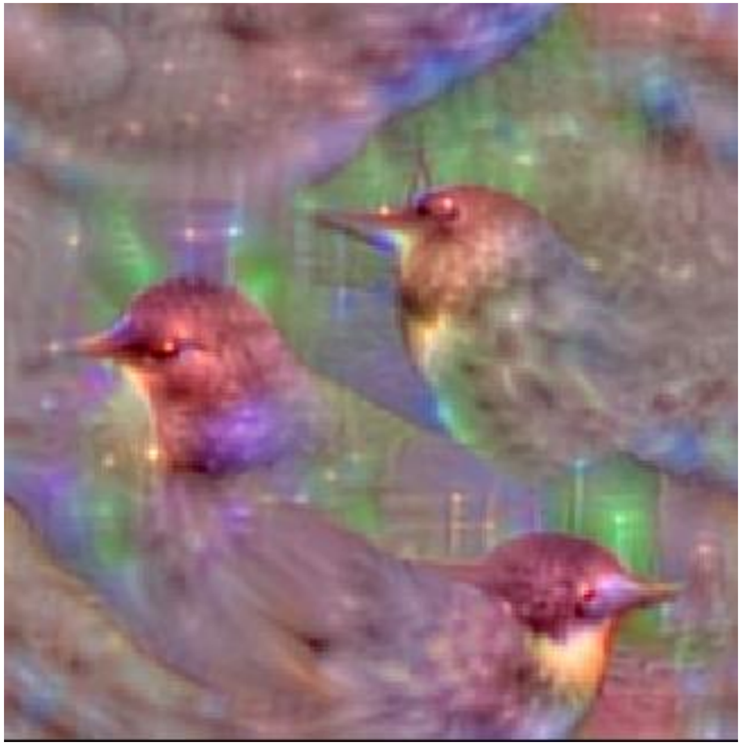


# To the next level

- There are many ways to improve the quality of visualization.
  - Zitter
  - Regularization (L1, L2… Lp space)
  - TotalVariation
  - GAN




< Current result >



< Our goal>

# Any Question?

Thank you