Implementing Deconvolution to Visualize and Understand Convolutional Neural Networks

Xiaozhe Yao

Objectives

- Understand how Convolutional and Deconvolutional Neural Networks work and implement them from scratch.
- Propose possible study problems for future studies.

Introduction

- Implement the primary components in CNNs: the ***convolution***, ***pooling*** and ***fully connected*** layer.
- Apply these components to train a classifier on cat images.
- Implement the inverse operation of these components: the ***deconvolution*** and ***max-unpooling*** layer.
- Apply the inverse components to visualize the learned features of the classifier.

Table of Contents

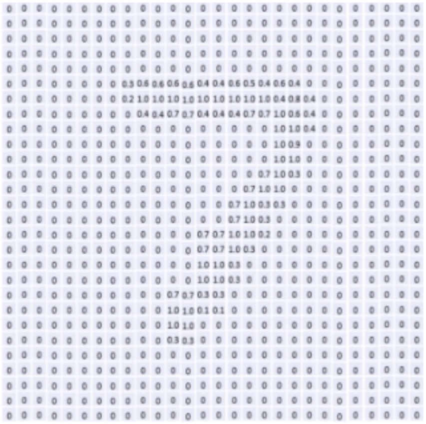
Approach

- Implement with pure Python, NumPy-like libraries.
- ***Correctness***: Each componenet is verified by comparing the output with PyTorch. Besides, an end-to-end test is used to verify the updating process.
- ***Performance:*** CPU/GPU support (with the help of CuPy).

Classification Models

Images → Tensors

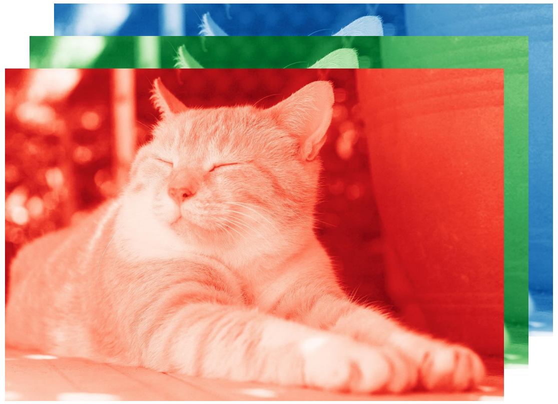
H



W

H

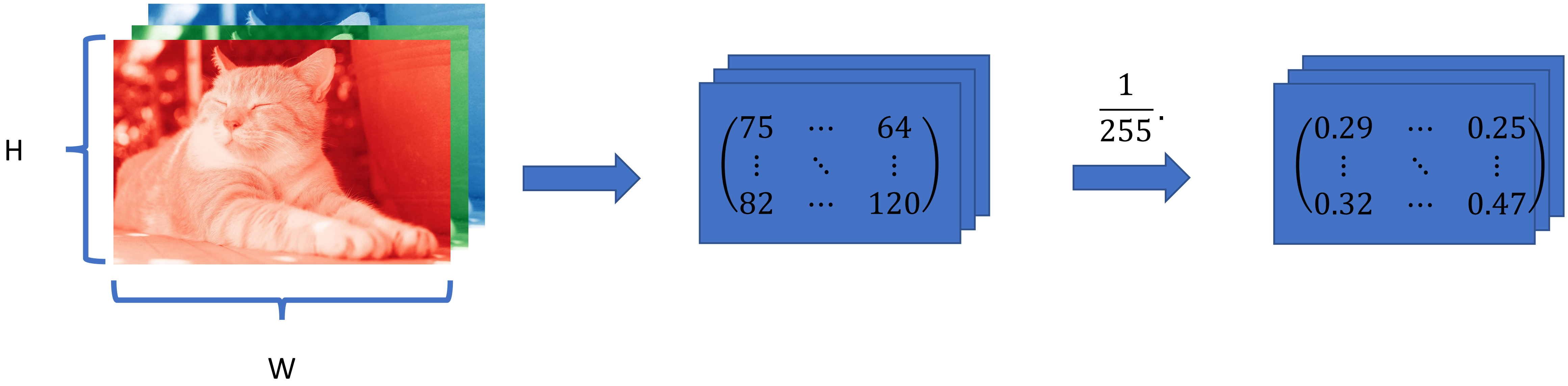
W



Images can be regarded as matrices.
Grayscale images are (H, W) matrices.
Each entry represents a pixel in image.

Color images usually consist 3 channels:
red, green and blue.
Each channel is a single (H, W) matrix.

Images → Tensors



A (H, W) color Image

A (3, H, W) tensor

A normalized (3, H, W) tensor

For simplicity, we will only use a single channel example in the following

Convolution Layer

Initialize the Kernel with a random 2x2 matrix: $\begin{bmatrix} 0.10 & 0.20 \\ 0.30 & 0.40 \end{bmatrix}$

Slide the kernel across the input by a unit stride.

We omit the bias for simplicity.

$$\begin{bmatrix} 0.10 & 0.15 & 0.20 & 0.25 & 0.30 \\ 0.35 & 0.40 & 0.45 & 0.55 & 0.60 \\ 0.65 & 0.70 & 0.75 & 0.80 & 0.85 \\ 0.90 & 0.95 & 0.10 & 0.15 & 0.20 \\ 0.25 & 0.30 & 0.35 & 0.40 & 0.45 \end{bmatrix} \otimes \begin{bmatrix} 0.10 & 0.20 \\ 0.30 & 0.40 \end{bmatrix} \rightarrow$$

0.305	0.355		

$$< \begin{bmatrix} 0.10 & 0.15 \\ 0.35 & 0.40 \end{bmatrix}, \qquad > = 0.10*0.10+0.15*0.20+0.35*0.30+0.40*0.40=0.305$$

$$< \begin{bmatrix} 0.15 & 0.20 \\ 0.40 & 0.45 \end{bmatrix}, \begin{bmatrix} 0.10 & 0.20 \\ 0.30 & 0.40 \end{bmatrix} > = 0.15*0.10+0.20*0.20+0.40*0.30+0.45*0.40=0.355$$

Convolution Layer

Initialize the Kernel with a random 2x2 matrix: $\begin{bmatrix} 0.10 & 0.20 \\ 0.30 & 0.40 \end{bmatrix}$

Slide the kernel across the input by a unit stride.

We omit the bias for simplicity.

$$\begin{bmatrix} 0.10 & 0.15 & 0.20 & 0.25 & 0.30 \\ 0.35 & 0.40 & 0.45 & 0.55 & 0.60 \\ 0.65 & 0.70 & 0.75 & 0.80 & 0.85 \\ 0.90 & 0.95 & 0.10 & 0.15 & 0.20 \\ 0.25 & 0.30 & 0.35 & 0.40 & 0.45 \end{bmatrix} \otimes \begin{bmatrix} 0.10 & 0.20 \\ 0.30 & 0.40 \end{bmatrix} \rightarrow$$

0.305	0.355	0.425	0.49
0.59	0.64	0.7	0.755
0.855	0.545	0.325	0.375
0.475	0.345	0.305	0.355

Max-Pooling Layer

Assume we have a 2x2 pooling kernel.

Slide the kernel across the input by a unit stride.

$$\begin{bmatrix} \boxed{\begin{matrix} 0.305 & 0.355 \\ 0.590 & 0.640 \end{matrix}} & \begin{matrix} 0.425 & 0.490 \\ 0.700 & 0.755 \end{matrix} \\ \begin{matrix} 0.855 & 0.545 \\ 0.475 & 0.345 \end{matrix} & \begin{matrix} 0.325 & 0.375 \\ 0.305 & 0.355 \end{matrix} \end{bmatrix} \rightarrow$$

0.64	0.70	0.755
0.855	0.70	0.755
0.855	0.545	0.375

$$\max\{0.305, 0.355, 0.590, 0.640\} = 0.640$$

Flatten

0.64	0.70	0.755
0.855	0.70	0.755
0.855	0.545	0.375

0.64	
0.70	Row 1
0.755	
0.855	
0.70	Row 2
0.755	
0.855	
0.545	Row 3
0.375	

Fully Connected Layer

The shape of weight is determined by the input shape and output shape.
In our example, we have a 2×9 matrix.
Initialize it randomly.

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 \\ 0.9 & 0.8 & 0.7 & 0.6 & 0.5 & 0.4 & 0.3 & 0.2 & 0.1 \end{bmatrix} \begin{bmatrix} 0.64 \\ 0.70 \\ 0.755 \\ 0.855 \\ 0.70 \\ 0.755 \\ 0.855 \\ 0.545 \\ 0.375 \end{bmatrix} = \begin{bmatrix} 2.9475 \\ 3.2325 \end{bmatrix}$$

Raw Confidence for Class 0

Raw Confidence for Class 1

Softmax: normalize raw confidence to $[0,1]$

Softmax is defined as $f(y_i) = \frac{e^{y_i}}{\sum e^{y_j}}$. In our case, $y_0 = 2.9475$, $y_1 = 3.2325$.

- $e^{y_0} = 19.058$, $e^{y_1} = 25.343$.
- $\sum e^{y_j} = 19.058 + 25.342 = 44.401$.
- $p(y_0) = f(y_0) = \frac{19.058}{44.401} = 0.4292$, $p(\mathrm{y}_1) = f(y_1) = \frac{25.342}{44.401} = 0.5707$.

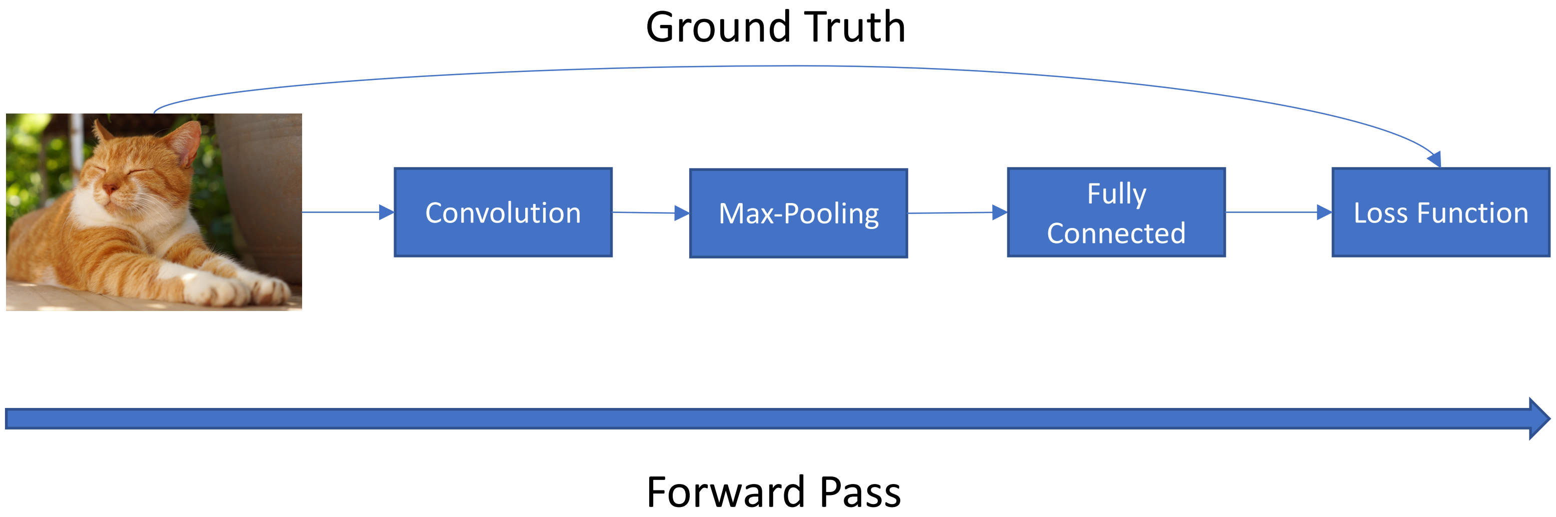
Normalized confidence for classes.

Cross Entropy: Loss function for classification

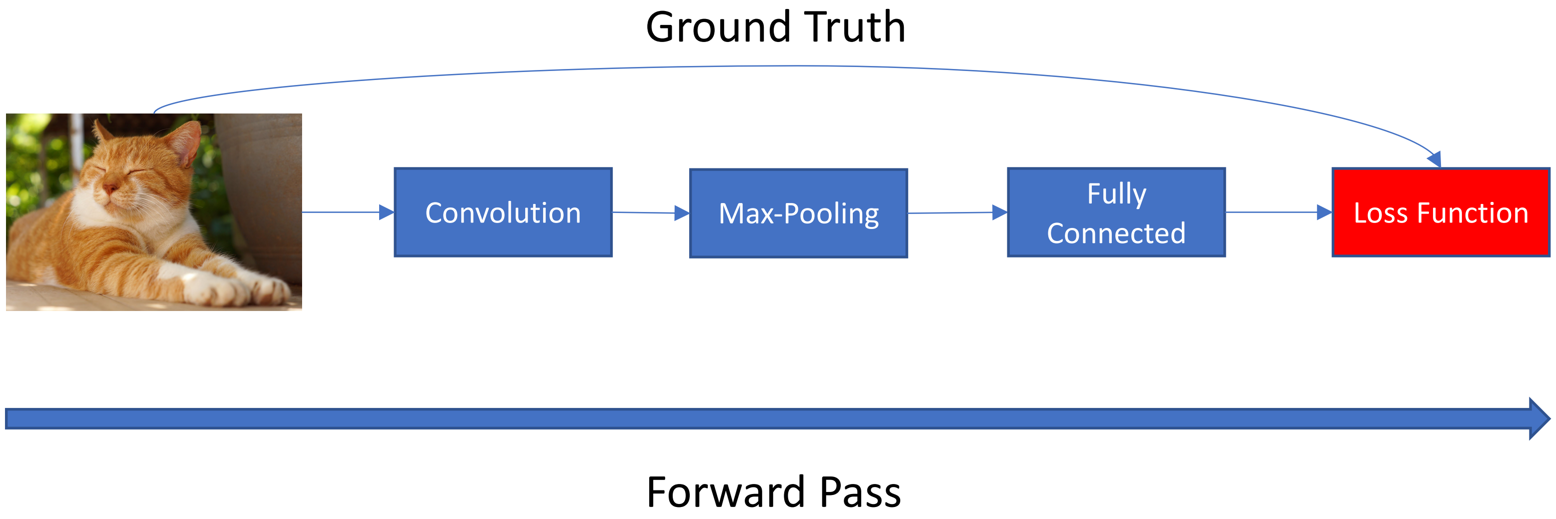
Cross Entropy loss is defined as $\ell = -\sum y_i log(p(y_i))$. In our case $p(y_0) = 0.4292, p(y_1) = 0.5707$.

- Ground truth is 1.
- $-y_0 lo\, g\big(p(y_0)\big) = -0 * \log(0.4292) = 0$
- $-y_1 lo\, g\big(p(y_1)\big) = -1 * \log(0.5707) = 0.5608.$
- $\ell = 0 + (0.5604) = 0.5608.$
- The loss value measures the distance between the predicted output and the ground truth.

What we have done so far?



What we have done so far?



Backward: Loss Function

How the output will impact the loss? → What's the $\frac{\partial \ell}{\partial y_0}$ and $\frac{\partial \ell}{\partial y_1}$?

- $y_0 = 2.9475, y_1 = 3.2325, \ell = -\sum y_i log(p(y_i))$, and $p(y_i) = \frac{e^{y_i}}{\sum e^{y_j}}$.
- $\frac{\partial \ell}{\partial y_i} = -\sum_j y_j \frac{\partial \log\left(p(y_j)\right)}{\partial y_i} = -\sum_j y_j \frac{1}{\mathrm{p}(y_j)} \frac{\partial p(y_j)}{\partial y_i}$.
- Among all j, there will be a $k = i$, such that $\frac{\partial p(y_k)}{\partial y_i} = \frac{\partial \frac{e^{y_i}}{\sum e^{y_j}}}{\partial y_i} = \frac{e^{y_i} \sum e^{y_j} - e^{y_i} e^{y_i}}{\left(\sum e^{y_j}\right)^2} = \frac{e^{y_i}}{\sum e^{y_j}} \frac{\sum e^{y_j} - e^{y_i}}{\sum e^{y_j}} = p(y_i)(1 - p(y_i))$.
- For others $k \neq i$, we have $\frac{\partial p(y_k)}{\partial y_i} = \frac{\partial \frac{e^{y_k}}{\sum e^{y_j}}}{\partial y_i} = \frac{0 - e^{y_k} e^{y_i}}{\left(\sum e^{y_j}\right)^2} = \frac{e^{y_k}}{\sum e^{y_j}} \frac{-e^{y_i}}{\sum e^{y_j}} = -p(y_i)p(y_k)$.
- Then, what about $\frac{\partial \ell}{\partial y_i}$?

Backward: Loss Function

- $\ell = -\sum y_i log\left(p(y_i)\right)$. We split it into two parts: i and $j \neq i.$
- $\frac{\partial l}{\partial y_i} = (-y_i)\frac{1}{p(y_i)}\ p(y_i)\left(1 - p(y_i)\right) - \sum_{i\neq j} y_j \frac{1}{p(y_j)}\left(-p(y_j)p(y_i)\right)$

$$= -y_i\left(1 - p(y_i)\right) - \sum_{i\neq j} -p(y_i)y_j$$

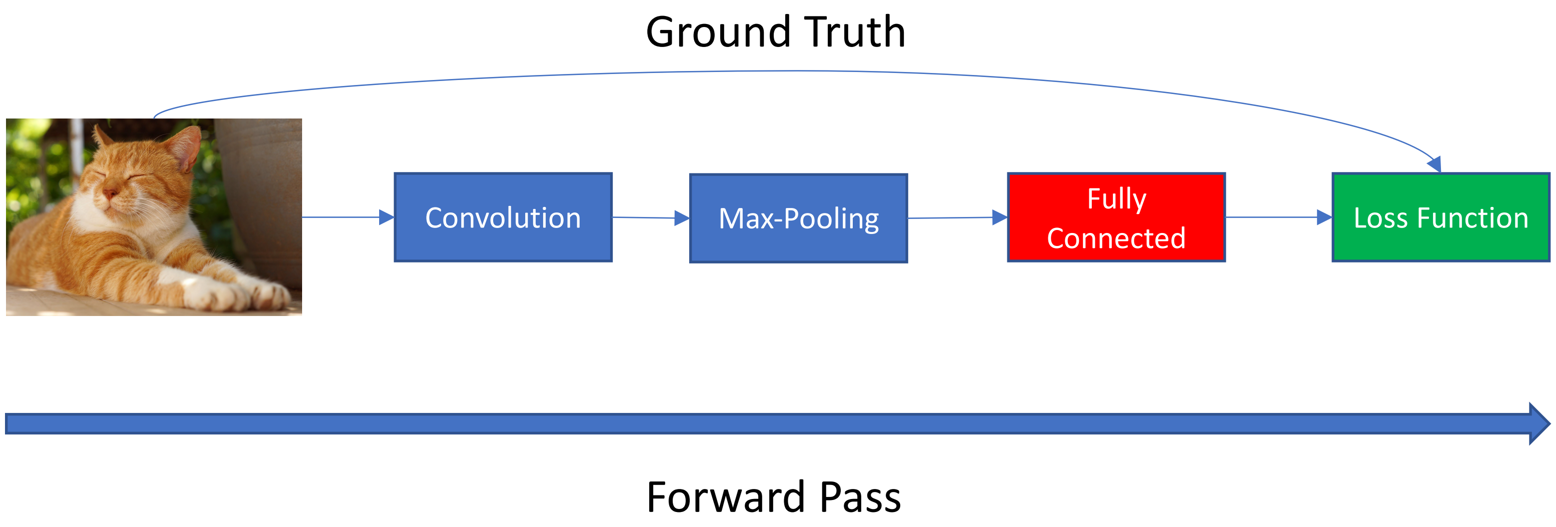
$$= -y_i + y_i p(y_i) + \sum_{i\neq j} y_j p(y_i)$$

$$= -y_i + p(y_i)\underbrace{\left(y_i + \sum_{i\neq j} y_j\right)}_{0+1=1} = -y_i + p(y_i)$$

Backward: Loss Function

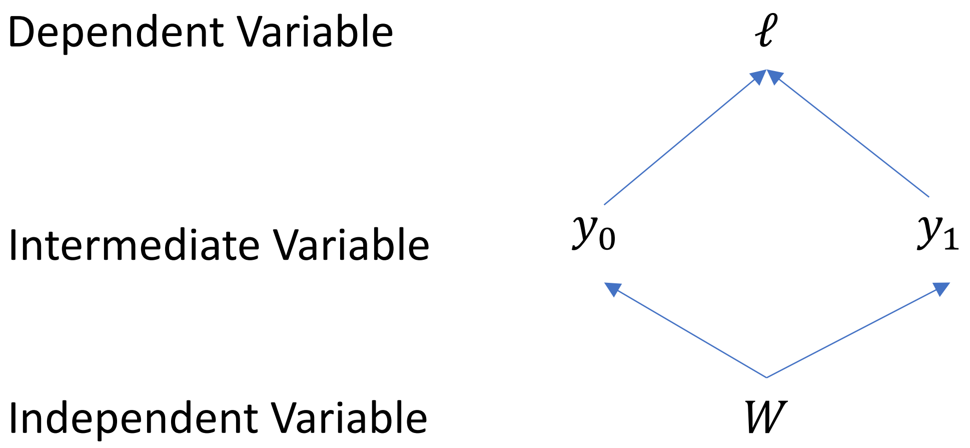
- In our example, we will have
  - $\frac{\partial \ell}{\partial y_0} = p(y_0) - y_0 = 0.4292 - 0 = 0.4292.$
  - $\frac{\partial \ell}{\partial y_1} = p(y_1) - y_1 = 0.5707 - 1 = -0.4292.$
- In matrix forms, we have $\frac{\partial \ell}{\partial Y} = \begin{bmatrix} \frac{\partial \ell}{\partial y_0} \\ \frac{\partial \ell}{\partial y_1} \end{bmatrix} = \begin{bmatrix} 0.4292 \\ -0.4292 \end{bmatrix}.$
- Then, how will the weight in fully connected layer affect the loss ℓ?

What we have done so far?



Backward: Fully Connected Layer.

- Using Chain rule: $\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial y_0}\frac{\partial y_0}{\partial W} + \frac{\partial \ell}{\partial y_1}\frac{\partial y_1}{\partial W}$



Backward: Fully Connected Layer

$$Y = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 \\ 0.9 & 0.8 & 0.7 & 0.6 & 0.5 & 0.4 & 0.3 & 0.2 & 0.1 \end{bmatrix} \begin{bmatrix} 0.64 \\ 0.70 \\ 0.755 \\ 0.855 \\ 0.70 \\ 0.755 \\ 0.855 \\ 0.545 \\ 0.375 \end{bmatrix} = \begin{bmatrix} 2.9475 \\ 3.2325 \end{bmatrix}$$

- $y_0 = \sum_i w_{0i}\, x_i \rightarrow \frac{\partial y_0}{\partial w_{0i}} = x_i, \frac{\partial y_0}{\partial w_{1i}} = 0.$
- $y_1 = \sum_i w_{1i}\, x_i \rightarrow \frac{\partial y_1}{\partial w_{1i}} = x_i, \frac{\partial y_1}{\partial w_{0i}} = 0.$

Backward: Fully Connected Layer

- $\frac{\partial \ell}{\partial w_{0i}} = \frac{\partial \ell}{\partial y_0}\frac{\partial y_0}{\partial w_{0i}} + \frac{\partial \ell}{\partial y_1}\frac{\partial y_1}{\partial w_{0i}} = \frac{\partial \ell}{\partial y_0} x_i.$
- $\frac{\partial \ell}{\partial w_{1i}} = \frac{\partial \ell}{\partial y_0}\frac{\partial y_0}{\partial w_{1i}} + \frac{\partial \ell}{\partial y_1}\frac{\partial y_1}{\partial w_{1i}} = \frac{\partial \ell}{\partial y_1} x_i.$
- $\frac{\partial \ell}{\partial y_i}$ is the ith row in $\frac{\partial \ell}{\partial Y}$, x_i is the ith column in X^T.

$$\frac{\partial \ell}{\partial W} = \underbrace{\begin{bmatrix} 0.429 \\ -0.4292 \end{bmatrix}}_{\frac{\partial \ell}{\partial Y}} \underbrace{\begin{bmatrix} 0.64 & 0.70 & 0.755 & 0.855 & 0.70 & 0.755 & 0.855 & 0.545 & 0.375 \end{bmatrix}}_{X^T}$$

Backward: Fully Connected Layer.

- $\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial Y} X^T$.
- Similarly $\frac{\partial \ell}{\partial X} = W^T \frac{\partial \ell}{\partial Y}$. (We will use this later)

$$\frac{\partial \ell}{\partial W} = \begin{bmatrix} 0.4292 \\ -0.4292 \end{bmatrix} \begin{bmatrix} 0.64 & 0.70 & 0.755 & 0.855 & 0.70 & 0.755 & 0.855 & 0.545 & 0.375 \end{bmatrix}$$
$$= \begin{bmatrix} 0.274 & 0.300 & 0.324 & 0.366 & 0.300 & 0.324 & 0.366 & 0.233 & 0.160 \\ -0.274 & -0.300 & -0.324 & -0.366 & -0.300 & -0.324 & -0.366 & -0.233 & -0.160 \end{bmatrix}$$

$$\frac{\partial \ell}{\partial X} = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 \\ 0.9 & 0.8 & 0.7 & 0.6 & 0.5 & 0.4 & 0.3 & 0.2 & 0.1 \end{bmatrix}^T \begin{bmatrix} 0.4292 \\ -0.4292 \end{bmatrix} = \begin{bmatrix} -0.343 \\ -0.2575 \\ -0.1716 \\ -0.085 \\ 0 \\ 0.085 \\ 0.1716 \\ 0.2575 \\ 0.3433 \end{bmatrix}$$

Backward: Fully Connected Layer

- Now we update the weight by gradient descent: $W^{new} = W - \lambda \frac{\partial \ell}{\partial W}$.
- λ is a preset hyper-parameter, here we set $\lambda = 0.01$.

$$W^{new} = \begin{bmatrix} 0.097 & 0.196 & 0.296 & 0.396 & 0.496 & 0.596 & 0.696 & 0.797 & 0.898 \\ 0.902 & 0.803 & 0.703 & 0.603 & 0.503 & 0.403 & 0.303 & 0.202 & 0.101 \end{bmatrix}$$

- Then we pass $\frac{\partial \ell}{\partial X}$ to previous layer.

$$\frac{\partial \ell}{\partial X} = \begin{bmatrix} -0.343 \\ -0.2575 \\ -0.1716 \\ -0.085 \\ 0 \\ 0.085 \\ 0.1716 \\ 0.2575 \\ 0.3433 \end{bmatrix}$$

Summary: Backward of Linear Transformation

- If we have two functions:
  - $g(X): R^{p\times n} \to R^{m\times n}, g(X) = Y = WX$, where $W \in R^{m\times p}$.
  - $f(X): R^{m\times n} \to R$.

Then we have:

- $\frac{\partial f}{\partial W} = \frac{\partial f}{\partial Y} X^T$
- $\frac{\partial f}{\partial X} = W^T \frac{\partial f}{\partial Y}$

We will reuse this result again later

Backward: Max-Pooling

- The fully connected layer gives us: $\frac{\partial \ell}{\partial X}$ (X is the input of Fully Connected Layer). It is equal to $\frac{\partial \ell}{\partial Y}$ (Y is the output of the Pooling).
- There's no weight in Pooling Layer. We only need to pass the gradient to previous layer, i.e. we only need to compute $\frac{\partial \ell}{\partial X}$.
- First reshape the received gradient to original size, i.e. 3×3.

$$\frac{\partial \ell}{\partial X} = \begin{bmatrix} -0.343 & -0.2575 & -0.1716 \\ -0.085 & 0 & 0.085 \\ 0.1716 & 0.2575 & 0.343 \end{bmatrix}$$

Backward: Max-Pooling

$$\begin{bmatrix} \boxed{0.305 & 0.355} & 0.425 & 0.490 \\ \boxed{0.590 & 0.640} & 0.700 & 0.755 \\ 0.855 & 0.545 & 0.325 & 0.375 \\ 0.475 & 0.345 & 0.305 & 0.355 \end{bmatrix}$$

$$\begin{bmatrix} -0.343 & -0.257 & -0.171 \\ -0.085 & 0 & 0.085 \\ 0.171 & 0.257 & 0.343 \end{bmatrix}$$

	-0.343	-0.257+0	-0.171+0.085
-0.085+0.171	0.257		0.343

Backward: Max-Pooling

- Other elements are all zeros.
- Result:

$$\frac{\partial \ell}{\partial X} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -0.343 & -0.257 & -0.085 \\ 0.085 & 0.257 & 0 & 0.343 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Revisit Convolution: Linear Transform

- 5×5 input and 2×2 kernel → 4×4 output.
- 25×1 input matrix, left multiplied by a 16×25 matrix W → 16×1 matrix.

$$\begin{bmatrix} 0.10 & 0.15 & 0.20 & 0.25 & 0.30 \\ 0.35 & 0.40 & 0.45 & 0.55 & 0.60 \\ 0.65 & 0.70 & 0.75 & 0.80 & 0.85 \\ 0.90 & 0.95 & 0.10 & 0.15 & 0.20 \\ 0.25 & 0.30 & 0.35 & 0.40 & 0.45 \end{bmatrix}$$

Input

$$\begin{bmatrix} 0.10 & 0.20 \\ 0.30 & 0.40 \end{bmatrix}$$

Kernel

$$\text{The first row: } \begin{array}{ccccccccc} [0.10 & 0.20 & 0 & 0 & 0 & 0.30 & 0.40 & \cdots] \\ \uparrow & \uparrow & & & & \uparrow & \uparrow & \\ 0.10 & 0.15 & & & & 0.35 & 0.40 & \end{array}$$

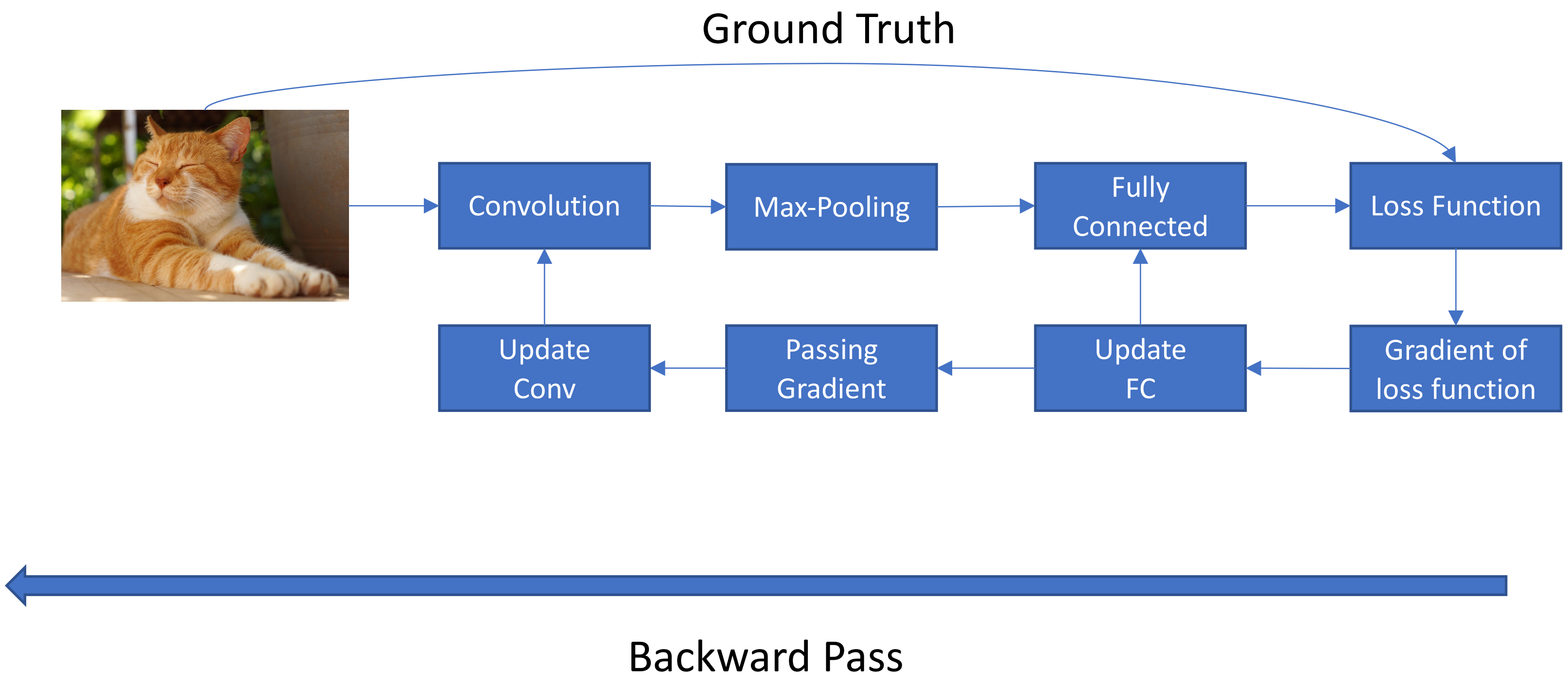
$$\text{The next row: } \begin{array}{ccccccccc} [0 & 0.10 & 0.20 & 0 & 0 & 0 & 0.30 & 0.40 & \ldots] \\ & \uparrow & \uparrow & & & & \uparrow & \uparrow & \\ & 0.15 & 0.20 & & & & 0.40 & 0.45 & \end{array}$$

Convolution operation becomes a matrix multiplication.

Backward: Convolution

- We will have $\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial Y} X^T$, by reusing the results from fully connected layer.
- Then update the weight: $W^{new} = W - \lambda \frac{\partial \ell}{\partial W}$.
- Result:
  - $W^{new} = \begin{bmatrix} 0.097 & 0.198 \\ 0.301 & 0.403 \end{bmatrix}$

What we have done so far?



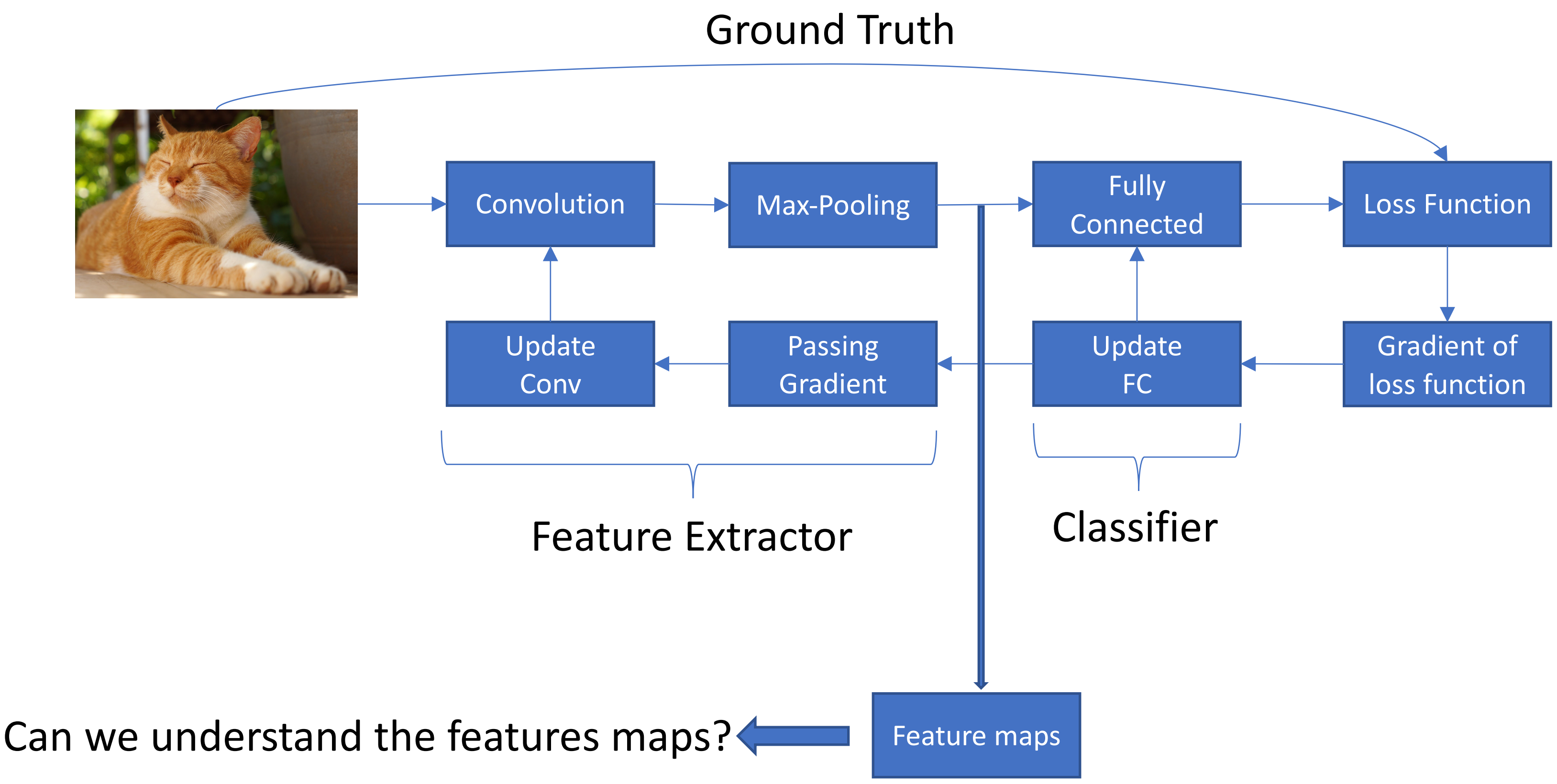
Is our model more confident?

- To verify, we perform forward pass again with updated weights.
- Output probabilities: $p(y_0) = 0.416$, $p(y_1) = 0.584$. Slightly more confident than $p(y_1)$ =0.571.
- Loss: $\ell = 0.5387$. Slightly closer to ground truth than $\ell = 0.5608$.

Interpretation Model

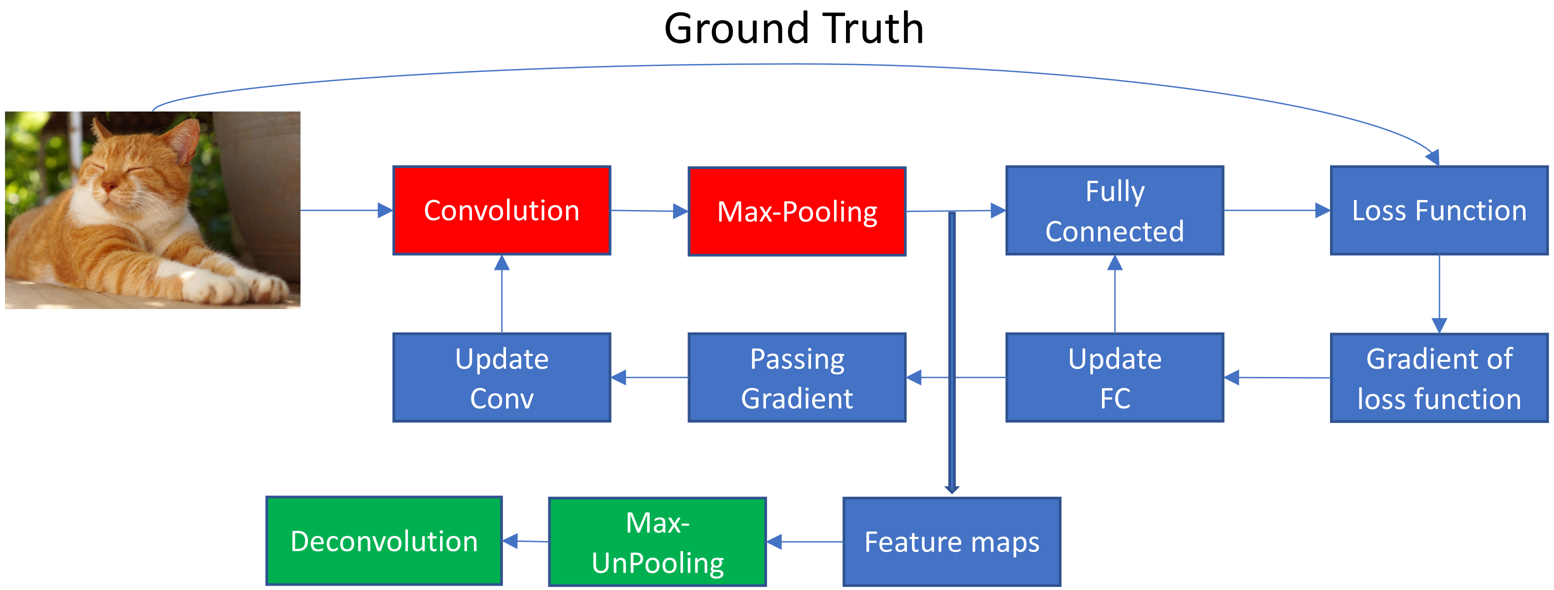
Problems in CNNs

- What kind of features does convolution + pooling extracted?



Problems in CNNs

- What kind of features that convolution + pooling extracted?



Max-UnPooling

0.64	0.70	0.755
0.855	0.70	0.755
0.855	0.545	0.375

	0.640	0.700	0.755
0.855	0.545		0.375

$$\begin{bmatrix} \boxed{\begin{matrix} 0.305 & 0.355 \\ 0.590 & 0.640 \end{matrix}} & \begin{matrix} 0.425 & 0.490 \\ 0.700 & 0.755 \end{matrix} \\ \begin{matrix} 0.855 & 0.545 \\ 0.475 & 0.345 \end{matrix} & \begin{matrix} 0.325 & 0.375 \\ 0.305 & 0.355 \end{matrix} \end{bmatrix}$$

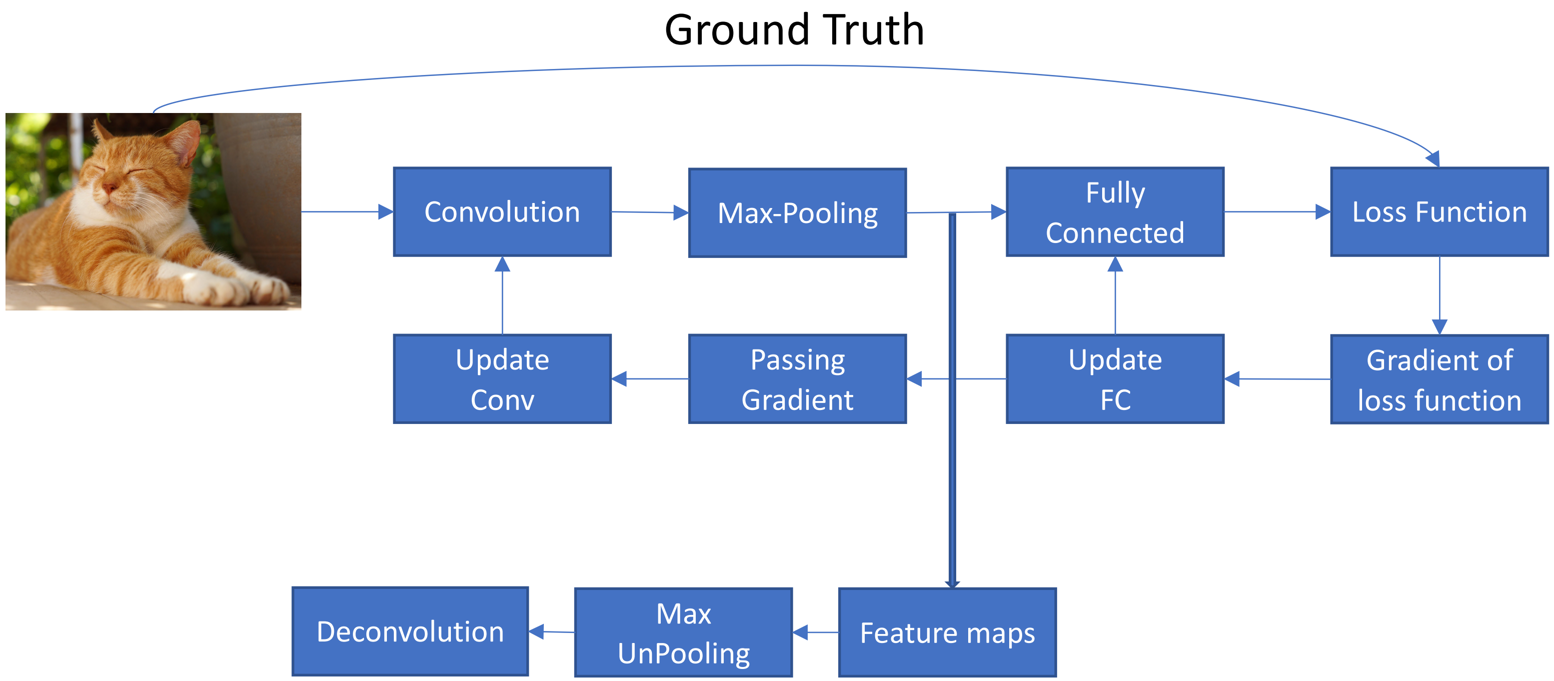
Other elements are all zeros.

Deconvolution: Transposed Kernel

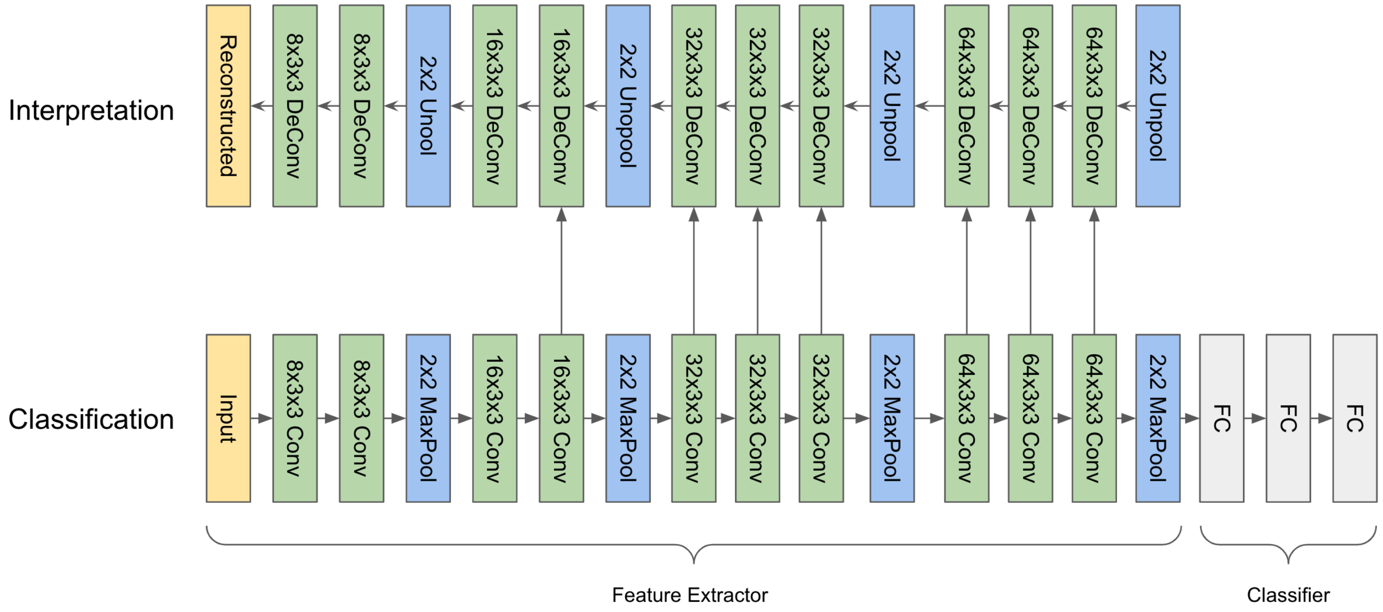
- Convolution: 5×5 input and 2×2 kernel → 4×4 output.
- Deconvolution: 4×4 input and 2×2 kernel → 5×5 output.
- In convolution, we have a 16×25 matrix. By transposing, we will have a 25×16 matrix.
- Then we multiply it with the 16×1 input (reshaped from 4×4), we will get a 25×1 vector.
- After that, we reshaped the output 25×1 vector back to 5×5 matrix.
- Result:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0.064 & 0.198 & 0.215 & 0.151 \\ 0.085 & 0.417 & 0.575 & 0.544 & 0.377 \\ 0.256 & 0.505 & 0.218 & 0.112 & 0.150 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

What we have done so far?



Experiments and Visualizations



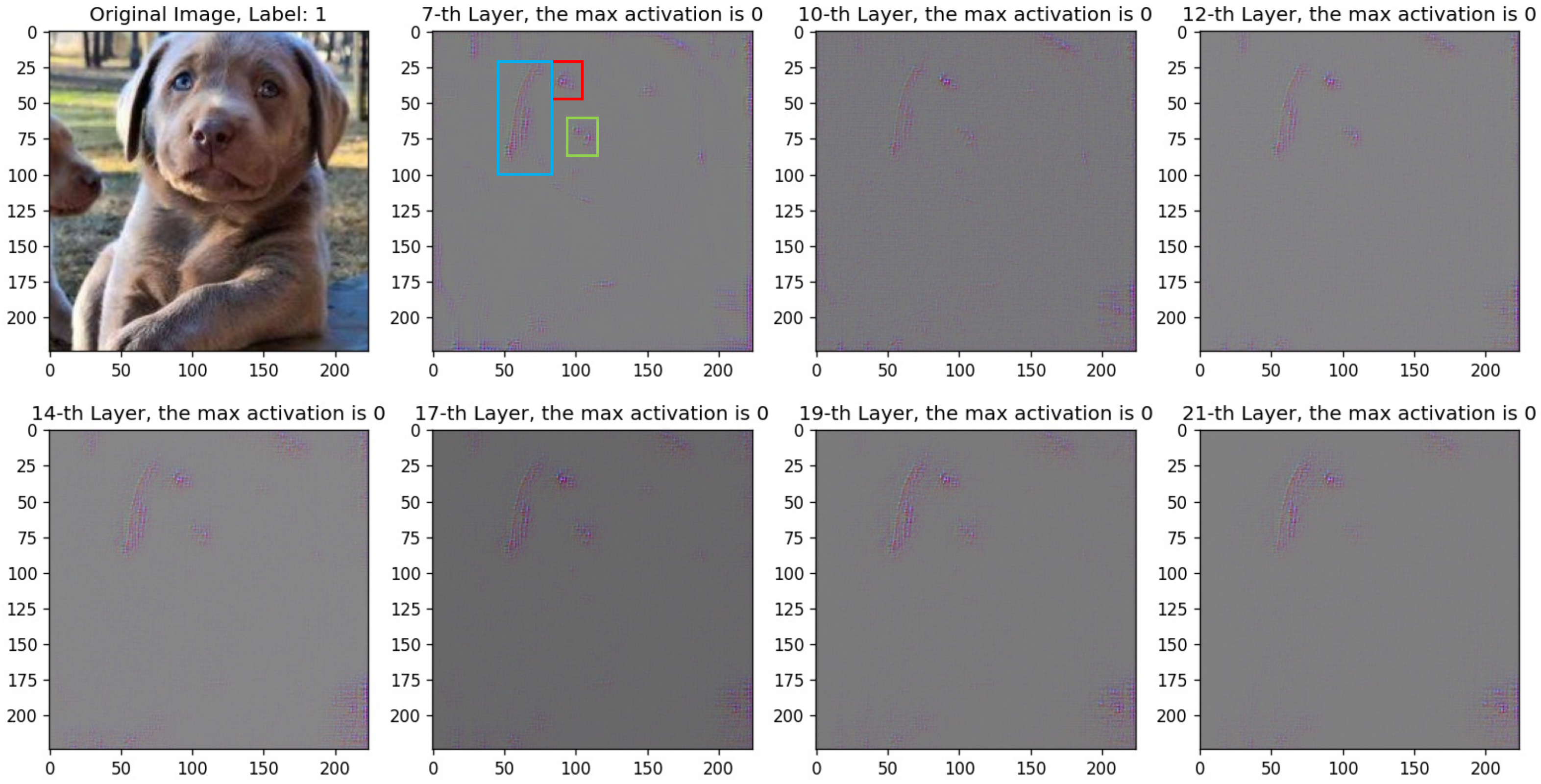
600 cat images + 600 not-cat images as train+val dataset. Then 90% for training and 10% for validation.

Each image is (3, 224, 224).

The learning rate, λ, is set to be 10^{-3}.

Acc=85% on validation set, 65% on test set.

Visualized Feature Maps



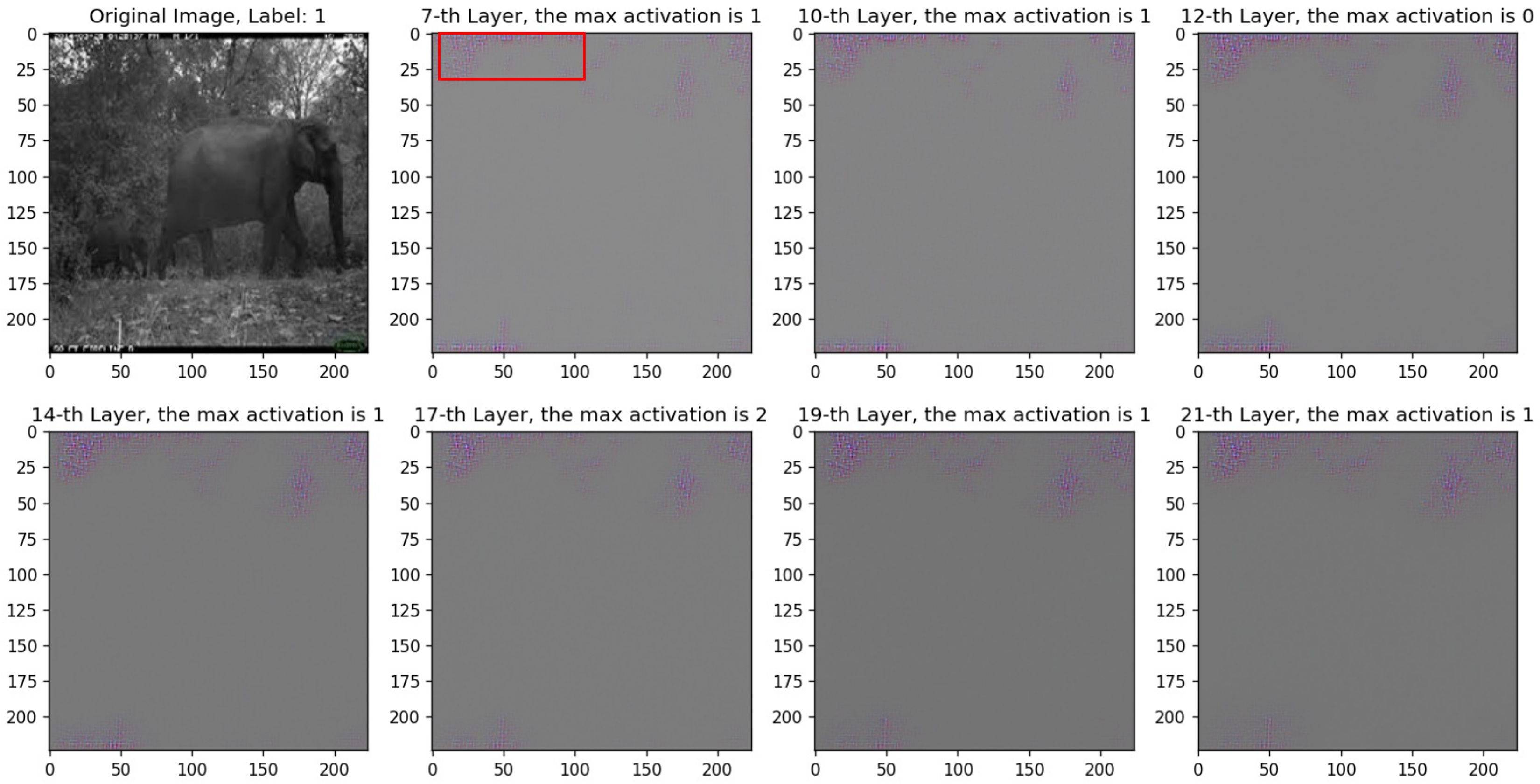
Right eyes

Nose

Hair

"These regions are discriminative for classification."

Sometimes it does not perform well.



Conclusion

- This approach reveals the regions that are discriminative to the classifier.
- From the visualizations, we can “guess” when our models will work, and when it will not work.
- It is only a qualitative analysis.
- Some regions cannot be understood by our humans.