

Lecture 3 | Loss Functions and Optimization

이번 강에서는, 저번 강에 이어서 선형 분류(Linear Classification)에 대해 알아보고자 한다.

손실 함수(Loss function)

손실 함수는 현재 분류기가 얼마나 정확한지를 알려주는 척도이다. 즉, W 와 b 가 얼마나 잘 세팅되었는가를 알려주는 함수라고 생각할 수 있다.

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Multiclass Support Vector Machine (SVM)

우선, 우리는 Multiclass SVM loss를 이용하여 현재 분류기의 loss를 계산하고자 한다.

Multiclass SVM loss를 계산하는 방법은 아래와 같다.

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases} \\ &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \end{aligned}$$

각 이미지와, 이것의 올바른 라벨이 있을 것이다. 그러면 올바른 라벨의 점수와 나머지 라벨들의 점수를 비교하는 식으로 계산된다. 만약 올바른 라벨이 다른 라벨보다 1이상 크다면 0으로, 아니라면 다른 라벨의 점수에 1을 더한 값과 올바른 라벨의 점수의 차이로 하여 모든 라벨과 비교하여 계산한 값들의 합을 L_i 라고 한다. 그러면 각 이미지당 L_i 가 하나씩 계산될 것이고, 그들의 평균이 현재 분류기의 손실(Loss)이 된다.

즉, 각각의 올바른 라벨과 그 이외의 라벨에 대해 $\max(0, s_j - s_{y_i} + 1)$ 를 계산하여 이들의 합을 계산하면 된다.



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

위 그림에 주어진 값을 이용하여 직접 계산해보자.

고양이 이미지에 대해서는, 고양이라벨과 자동차라벨, 고양이라벨과 개구리 라벨의 점수를 비교해야 한다.

즉, $L_{cat} = \max(0, 5.1 - 3.2 + 1) + \max(-1.7 - 3.2 + 1) = 2.9$ 이다.

동일하게 L_{car} , L_{frog} 를 계산하면 각각 0, 12.9가 나온다.



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	12.9

car의 loss가 0이라는 말은 자동차 이미지에 대한 분류는 상당히 정확도가 높다는 것이다. 이에 비해 frog의 loss는 12.9로 상당히 높은 것을 보아 개구리 이미지는 정상적으로 분류하기 힘들다는 것이다.

이 loss들을 통해 전체 분류기의 loss를 계산해보면, $L = (2.9 + 0 + 12.9)/3 = 5.27$ 이다.

몇몇 Question들

- ▼ Q1 : 자동차의 점수가 조금 변하면 어떤 일이 발생할까

A1 : 위에서 자동차 이미지에 대해서는 다른 라벨들의 점수에 비해 자동차 라벨의 점수가 많이 크기 때문에 조금 변하여도 Loss는 0으로 유지될 것이다. 이에 비해 고양이와 개구리 이미지에 대해서는 기준 라벨들의 점수보다 자동차 라벨의 점수가 더 크다. 그러므로 자동차의 점수가 증가하면 이 두 Loss들이 커질 것이며 자동차의 점수가 작아지면 이 두 Loss들이 작아질 것이다. 물론, 많이 변하여 대소에 변화가 생긴다면 답은 달라질 것이다.

- ▼ Q2 : 가능한 최대/최소 손실은 얼마인가?

A2 : 최소 손실은 0이고, 최대 손실은 무수히 큰 값이다. 모든 이미지에 대해 기준 라벨의 점수가 다른 라벨들의 점수들보다 1 이상 높으면 손실이 0이 될 것이다. 반면 기준 라벨보다 다른 라벨의 점수가 높다면, 높으면 높을수록 손실이 커질 것이다.

- ▼ Q3 : 초기의 W는 대부분 0에 가까운 작은 값들로 구성되어있다. 이때의 손실은 얼마인가?

- ▼ Q4 : 모든 라벨들에 대해 손실을 계산한다면($j = y_i$ 인 경우 포함) 결과는 어떻게 변하는가?

A4 : $\max(0, s_j - s_{y_i} + 1)$ 을 계산할 때, $j = y_i$ 가 된다면 반드시 결과는 1이 된다. 그렇기에 모든 loss들이 1씩 증가하게 되고 평균도 1이 증가하게 된다.

- ▼ Q5 : 합이 아니라 평균으로 계산을 하면 어떻게 될까?

A5 : 그냥 결과가 $\frac{1}{N-1}$ 이 될 뿐 비교에는 큰 타격이 없다.

- ▼ Q6 : 만약 $\sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$ 을 통해 계산하면 어떻게 될까?

A6 : 결과 값이 커질 뿐, 이 또한 비교에는 큰 영향을 끼치지 않는다.

- ▼ Q7 : L=0인 W를 찾았다고 가정하자. 이는 유일한가?

A7 : 유일하지 않다. L=0일 때의 Weight를 W_k 라고 하면 $2 * W_k$ 를 이용하여 연산을 하여도 L=0이 된다.

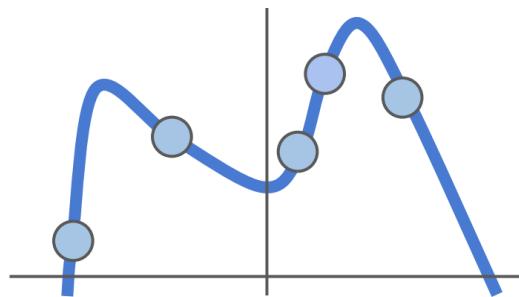
하지만 위에서 우리가 한 행동은 과연 좋은 결과를 낳을것인지 생각해볼 필요가 있다. 위에서 우리는 training data들을 가지고 좋은 W를 찾았다. 그러나 우리가 원하는 것은, 새

로운 test data들이 주어졌을 때 적은 loss를 가지는 것이다. 하지만 과연 위 W 가 test data에 대해서도 만족할까? loss가 0인 W 는 유일하지 않은 것처럼, training data를 이용해 구한 적합한 W 가 test data에서는 적합하지 않을 수도 있다.

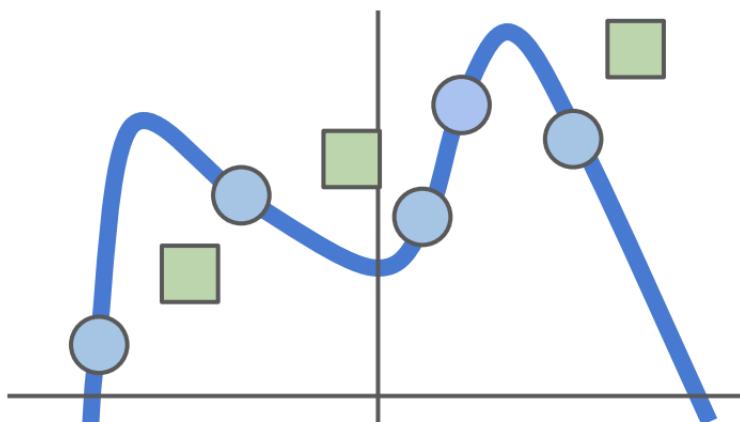
training set들을 이용하여 아래의 그래프와 같은 결과를 얻어내었다고 가정하자.

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)$$

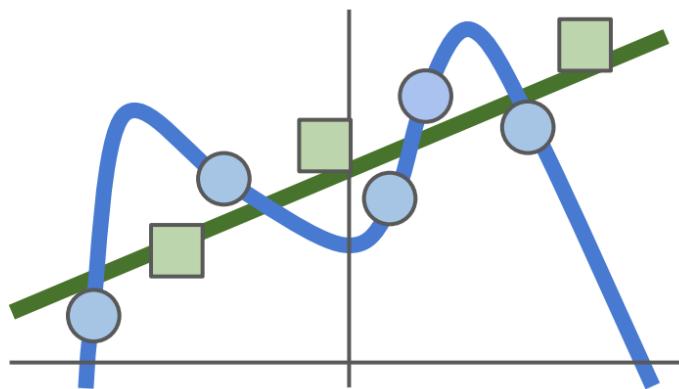
Data loss: Model predictions should match training data



이 때, 새로운 데이터(초록색 사각형으로 표시)가 아래와 같이 들어온다면 우리가 구한 W 는 작은 loss를 만들어내지 못할 것이다.



차라리 아래의 초록색 그래프처럼 만드는 것이 더 높은 정확도를 만들어 낼 것이다.



그래서 위에서 사용한 식만을 사용해서 W 를 예측하는 것이 아니라 우리는 추가적인 작업이 필요하다. 그것이 Regularization이다. 아래의 식에서 $\lambda R(W)$ 가 Regularization 부분이다.

$$L = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i) + \lambda R(W)$$

앞쪽 식은 training data에서 올바른 동작을 하게끔 하는 부분이라고 보면 뒤쪽 식은 test data에서도 올바른 동작을 하게끔 하는 부분이라고 보면 된다.

Occam's Razor : 가설을 세울 때는 가장 단순한 것이 좋다.

Regularization

regularization의 종류에는 다음과 같은 것들이 있다.

- L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$
- L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$
- Elastic net(L1+L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$
- Max norm regularization (might see later)

- Dropout (might see later)
- Fancier : Batch normalization, stochastic depth

일반적으로 우리는 L2 regularization을 사용한다.

소프트 맥스(Softmax Classifier)

이번에 알아볼 것은 또다른 분류기인 Softmax Classifier이다.

소프트 맥스에서는 Softmax function을 이용하여 Loss를 계산한다. Softmax function은 아래와 같다. 아래의 식을 해석해보자면, 각각의 데이터들을 e의 지수승을 한 후, 각각의 비율을 구하는 것이다.

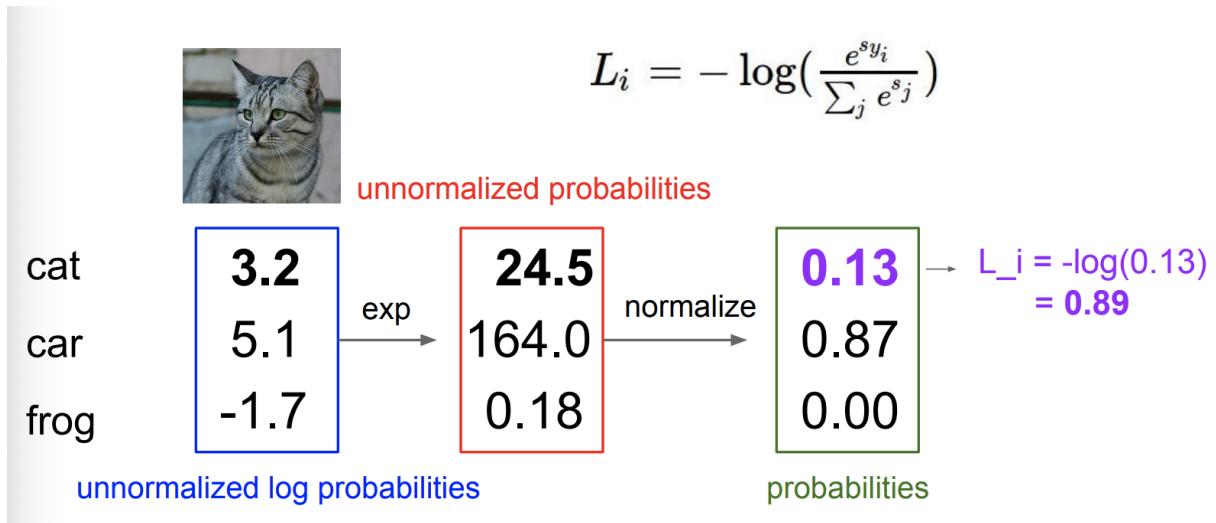
$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where } s = f(x_i, W)$$

또, 우리는 이 Softmax function을 이용하여 L_i 를 아래와 같이 정의한다.

$$\begin{aligned} L_i &= -\log P(Y = k | X = x_i) \\ &= -\log\left(\frac{e^{s_k}}{\sum_j e^{s_j}}\right) \end{aligned}$$

그럼 이를 이용해서 직접 계산을 해보자.

아래의 그림을 보면 데이터들이 주어져 있다. 이들이 s_j 에 해당하는 값들이다. 우선 softmax에서는 e^{s_j} 의 형태로 활용하기 때문에 연산을 해 주고 각각의 비율을 계산해 주면 된다. 그 후, cat 라벨에 해당하는 값에 $-\log$ 만 취해주면 $L_{cat} = 0.89$ 라는 결과를 얻을 수 있다.



몇몇 Question들

▼ Q1 : Softmax Classifier에서 최대/최소 L_i 는 얼마인가?

A1 : $-\log$ 를 취하기 전 최소는 0 최대는 1이다. 그렇기에 L_i 의 최소는 $-\log(1) = 0$ 이고, 최대는 $-\log(0) = \infty$ 이 된다.

▼ Q2 : 초기의 W는 대부분 0에 가까운 작은 값들로 구성되어있다. 이때의 손실은 얼마인가?

▼ Q3 : 데이터에 조금 변화가 생기면 어떻게 될지 SVM과 Softmax를 비교하여 설명하시오.

앞에서 설명한 것처럼 SVM에서는 데이터에 변화가 생겨도 max함수를 사용하기 때문에 영향을 끼치지 않을 수가 있다. 하지만 Softmax의 경우 전체중에 차지하는 비중으로 계산하고, 또 지수함수를 이용하기 때문에 변화가 큰 영향을 끼친다.

Optimization

그렇다면 우리는 어떻게 적합한 W를 선정해야 할까? W를 선정하는 것을 "산에서 가장 낮은 봉우리가 어디인지 찾는 것"으로 비유하여 설명하도록 하겠다.

Random search

임의로 선택하는 방법은 상당히 좋지 않은 방법이다. Random으로 W를 선정하여 계산해 본 결과 정확도가 15.5%로 상당히 낮은 수치가 나왔다.

```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

Follow the slope

현재 위치를 기준으로 내리막길을 찾아 이동해가면서 가장 낮은 곳을 찾는 방법이다. 이를 "경사 하강법(Gradient Descent)"라고 한다.

현재 우리가 loss계산하기 위해 사용한 W를 아래와 같다고 해보자.

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]

loss 1.25347

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,
?,...]

여기서 우리는 각각의 값들을 조금씩 변화시켜보고 그 loss의 변화율을 계산한다.

current W:	W + h (first dim):	gradient dW:
[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...] loss 1.25347	[0.34 + 0.0001 , -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...] loss 1.25322	<p style="color: blue;">[-2.5, ?, ?, ?, ?, ?, ?, ?, ?,...]</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> $\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$ </div>

이를 반복하면 우리는 감소하는 방향을 알 수 있다. 하지만 이렇게 하면 꽤 많은 시간이 소요된다.

우리가 궁금한 것은 W가 변화했을 때의 loss값이므로 미분을 통해 연산이 가능하다.

우리가 직접 하나하나 연산하는 방법을 Numerical gradient라고 하는데, 이는 느리지만 정확하다는 특징이 있다.

반면, 미분을 통해 연산하는 것은 Analytic gradient라고 하는데, 이는 연산 속도는 빠르지만 에러가 나올 확률이 높다.

우리는 주로 Analytic gradient를 사용하고 확인이 필요할 때 numerical gradient를 사용한다. 이를 gradient check라고 한다.

또, 만약 N이 상당히 커지면 Analytic gradient더라도 시간이 많이 소요된다. 그 때 우리가 사용할 것이 바로 Stochastic Gradient Descent(SGD)이다. W를 32, 64, 128 등의 길이의 조각으로 쪼개고 나누어 계산하는 것이라고 생각하면 된다.

Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$
$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive
when N is large!

Approximate sum
using a **minibatch** of
examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent

while True:
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```

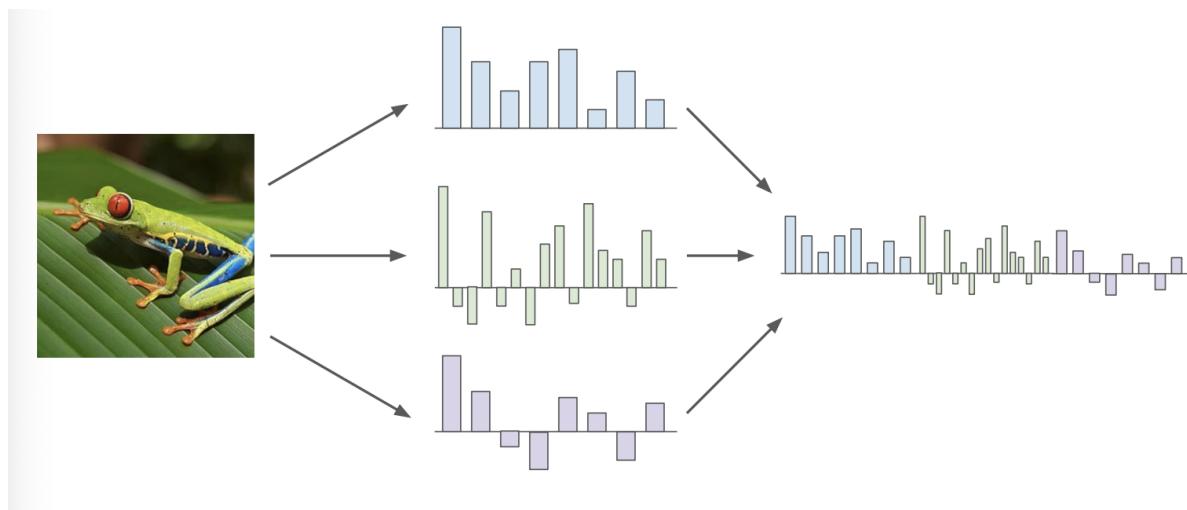
그 외의 방법들)

지금까지 우리는 사진의 데이터를 있는 그대로 대입하여 결과를 얻어내려고 하였다. 하지만 이러는 방법 말고 다른 방법도 존재한다.

Image Features

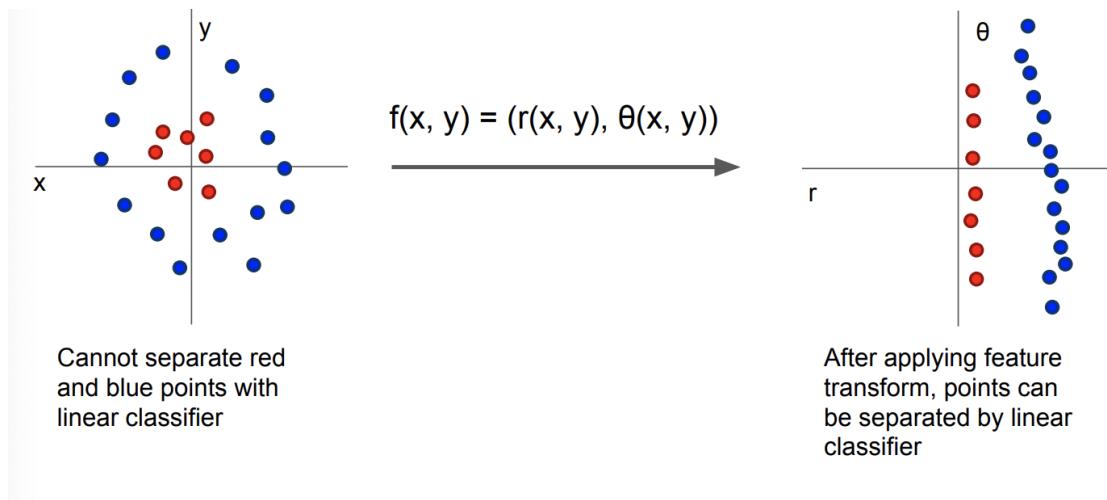
이미지에서 특징들을 뽑아내고, 그 특징들을 가지고 분류를 하는 것이다.

아래의 그림을 보면 개구리 사진에서 특징들을 추출해내고, 그 추출된 데이터들을 모아서 분류할 준비를 한다.



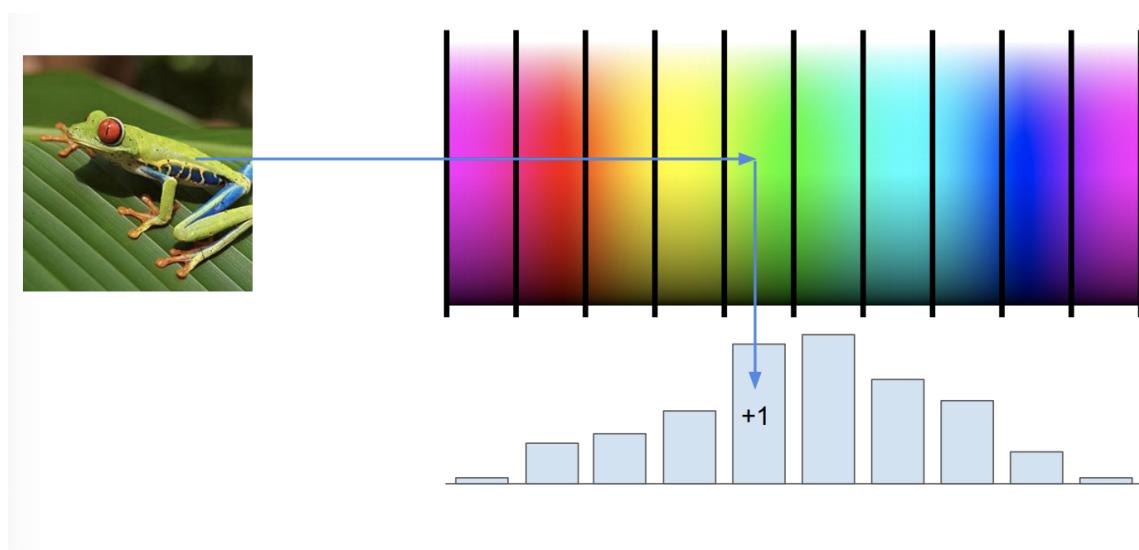
이 방법에 대한 동기는 아래와 같다.

왼쪽 사진은 평면 좌표에 점들을 표현한 것인데, 우리는 빨간 점과 파란 점들을 선형 분류기로 나누는 것이 쉽지 않다. 앞에서 말했던 것처럼, 일차함수, 즉 직선을 사용하기 때문이다. 그런데 이를 극좌표로 바꾸어보면 우리는 선형 분류기를 통해 쉽게 분류가 가능하다.



Color Histogram

또 다른 방법으로는 사진에서 사용되는 색들에 대한 데이터를 뽑아내어 활용하는 것이다. 개구리라는 사진을 보면 초록색의 비중이 높기 때문에 이를 이용하여 분류하는 것도 가능하다.

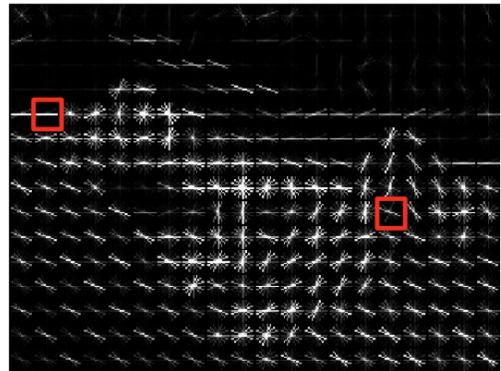


Histogram of Oriented Gradients(HoG)

이는 그림을 픽셀별로 쪼개고, 각 픽셀마다 어떠한 방향의 벡터들이 존재하는지에 대한 데이터를 이용하여 분류하는 방법이다.



Divide image into 8x8 pixel regions
Within each region quantize edge direction into 9 bins



Example: 320x240 image gets divided into 40x30 bins; in each bin there are 9 numbers so feature vector has $30 \times 40 \times 9 = 10,800$ numbers

Bag of Words

우선 training data를 픽셀별로 쪼개고 데이터를 추출하여 코드북을 만들어 둔다. 이후, 새로운 이미지가 들어왔을 때 그 이미지도 픽셀별로 쪼개고 가지고 있는 코드북과 비교하여 분류하는 방법이다.

