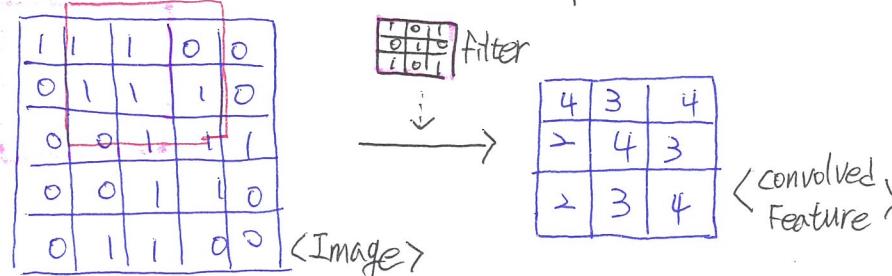


CNN의 주요용어 정리

* 합성곱 (Convolution)

: 예) 2차원 입력 데이터 (shape: (5,5))를 1개의 필터로 합성곱 연산.

합성곱 처리 결과로부터 Feature Map 만든다.



* 채널 (channel)

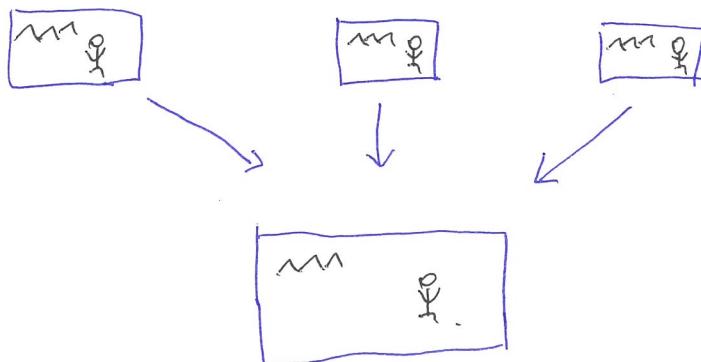
: 이미지 픽셀 하나 하나는 실수. 컬러 사진은 천연색 표현 위해, 각 픽셀을 RGB 3개의 실수로 표현한 3차원 데이터. → 컬러 이미지 채널 3개.

흑백 사진은 명암만 표현. 2차원 데이터 → 흑백 이미지 채널 1개.

ex) 높이 39 픽셀, 폭 31 픽셀, 컬러 사진의 데이터 shape : $(39, 31, 3)^3$

높이 39 픽셀, 폭 31 픽셀, 흑백 사진의 데이터 shape : $(39, 31, 1)$

Red channel Green channel Blue channel

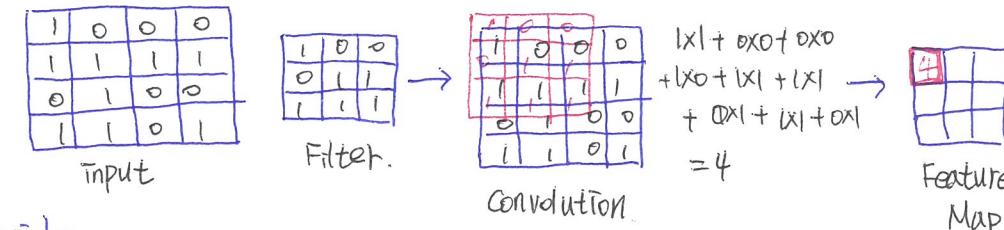


Color Image

* 필터 (Filter) & stride

: 필터는 이미지의 특징을 찾아내기 위한 공용 파라미터. (= kernel). 일반적으로 정사각 행렬. CNN의 학습 대상은 필터 파라미터. 겹쳐 : 3개. stride : 1
입력 데이터를 지정된 간격으로 순회하여 채널 별로 합성곱을 하고 모든 채널의 합성곱의 합을 Feature Map으로 만들기. 전체 입력데이터 \times Filter.

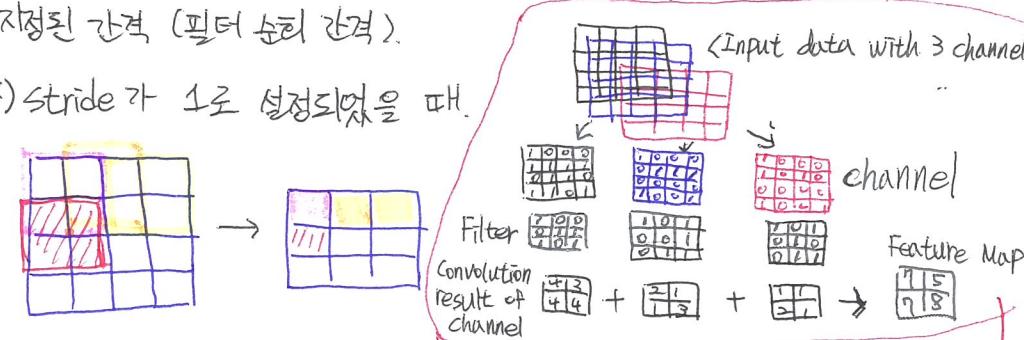
ex) 채널이 1개인 입력데이터를 (3,3) 크기의 필터로 합성곱 과정.



Stride

: 필터가 입력데이터를 지정한 간격으로 순회하면서 합성곱 계산할 때, 지정된 간격 (필터 순회 간격).

ex) stride가 1로 설정되었을 때.



입력데이터가 여러 채널을 가지고 있는 경우, 필터는 각 채널을 순회하여 합성곱 계산해 채널별 피처맵 만든다. 각 채널의 피처맵을 합산해 최종 피처맵 반한. 입력데이터는 채널 수와 상관 없이 필터별로 1개의 피처맵 생성

하나의 convolution Layer에 크기가 같은 여러개의 필터 적용 가능. 이때, Feature Map은 필터 개수 만큼 채널 만들어짐. 입력 데이터에 적용한 필터의 개수는 출력 데이터인 Feature Map의 채널 수가 됨.

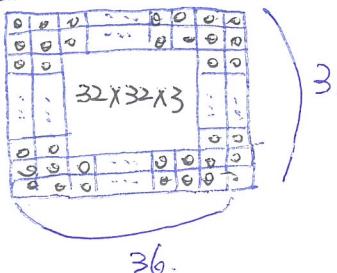
* Feature Map

: convolution layer의 입력데이터를 필터가 소화하며 합성곱을 통해 만든 출력을 Feature Map (= Activation Map). Feature Map은 합성곱 계산으로 만들어진 행렬. Activation Map은 Feature Map에 활성 함수 적용 결과. 즉, convolution layer의 최종 출력 결과는 Activation Map.

* 패딩 (Padding)

: convolution layer에서 Filter와 stride의 작용으로 Feature Map 크기는 입력 데이터보다 작음. convolution layer의 출력 데이터가 줄어드는 것을 방지하는 방법이 패딩임. 패딩은 입력데이터의 외곽에 지정된 퍼센트 만큼 특정값으로 채워넣는 것. 보통 패딩 값으로 '0'으로 채워넣음.

Ex). (32, 32, 3) 데이터를 외곽에 2pixel 추가해 (36, 36, 3) 행렬 만들기.



* 패딩을 통해

- ① convolution layer 출력 데이터 크기 조절
- ② 외곽을 '0'으로 둘러싸는 특징에서 인공신경망이 이미지의 외곽 인식 효과.

* Pooling layer

: 폴링레이어는 convolution layer의 출력 데이터를 입력으로 받아 출력 데이터의 크기를 줄이거나 특정 데이터 강조하는 용도. Pooling layer 처리 방법으로 Max pooling, Average Pooling, Min Pooling 있음. 정사각 행렬의 특정 역할인의 최댓값, 평균구하기. 일반적으로, Pooling 크기와 stride를 같은 크기로 설정하여 모든 원소가 한번씩 처리 되도록!

Ex)

12	20	30	0
8	12	2	0
34	10	31	7
112	100	22	12

Activation map.

20	30
112	31

MAX Pooling CNN에서 사용

13	8
79	18

Average Pooling.

cf. Pooling layer et Convolution layer 비교.

Pooling layer 는

- ① 학습 대상 파라미터 X. ② Pooling layer 통과시 행렬크기 ↓
- ③ Pooling layer 통과시 채널수 변경 X.

↳ 추가설명: convolution layer에서 입력 데이터의 채널이 3개여도

Feature map은 하나였음. 따라서 Pooling layer가 채널수 변경 X 가 중요X라고 생각할 수 있지만

Convolution layer를 3번 쌓고 pooling layer 쌓으면 이 특징 좋아!

CNN 주요 용어 정리 2

• 알고리즘이 iterative 하다.

- gradient descent와 같이 결과를 내기 위해서 여러번의 최적화 과정을 거쳐야하는 알고리즘.
→ 다뤄야 할 데이터가 너무 많고, 메모리 부족해 한번의 계산으로 최적화된 값 찾기는 힘들다. 따라서, ML에서 최적화 (optimization)를 할 때는 일반적으로 여러번의 학습과정 또 한번의 학습과정 역시 사용하는 데이터를 나누는 방식으로 세분화.
(epoch, batch size, iteration)

알고리즘.

* epoch.

One epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.

한번의 epoch는 완공신경망에서 전체 데이터셋에 대해 forward/backward 과정 거친 것.
즉, 전체 데이터셋에 대해 한 번 학습을 완료한 상태

→ 신경망에서 사용되는 역전파 알고리즘 (backpropagation algorithm)은 파라미터를 사용하여 입격부터 출격까지의 각 계층의 weight를 계산하는 과정을 거치는 순방향 패스 (forward pass)
forward pass를 반대로 거슬러 올라가며 다시 한번 계산 과정을 거쳐 기존의 weight를 수정하는 역방향 패스 (backward pass)로 나뉜다. 이 전체 데이터셋에 대해
forward pass + backward pass 가 완전히 완료되면 한 번의 epoch가 진행됨.

ex) epoch 40 : 전체 데이터를 40번 사용해 학습 거치!

⇒ 적절한 epoch 값을 설정해 모델 만들어야 underfitting과 overfitting 방지

[epoch 값 ↓ : underfitting 학률 有]

[epoch 값 ↑ : overfitting 학률 有]

* batch size

: Total number of training examples present in a single batch.

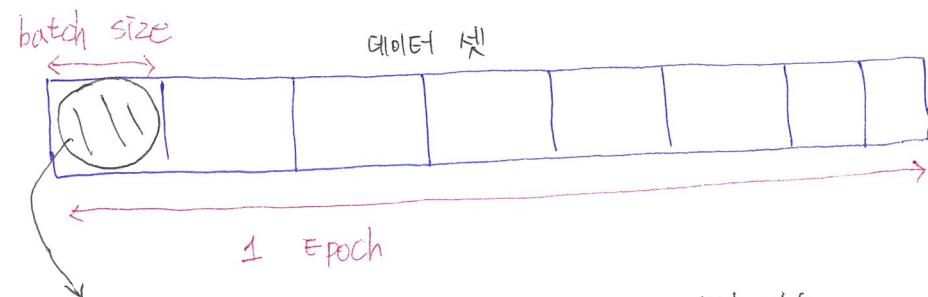
* Iteration

: The number of passes to complete one epoch.

⇒ batch size는 한 번의 batch마다 주는 데이터샘플의 size.

여기서 batch는 나눠진 데이터셋을 뜻하며 Iteration은 epoch를 나눠 실행하는 횟수
↳ 보통 mini-batch로 표현.

⇒ 메모리의 한계와 속도 저하 때문에 대부분의 경우에는 할 번의 epoch에서 모든 데이터를 한꺼번에 집어넣을 수 X. 그래서 데이터 나눠주게 되는데,
이 때 몇 번 나눠주는가를 Iteration. 각 Iteration마다 주는 데이터 사이즈 batch size



• mini-batch : 데이터 셋을 batch size 크기로 잘어서 학습.

예시

전체 2000개 데이터셋. epochs = 20, batch size = 500 일 때,

1 epoch = 4 iteration , 전체 데이터셋에 대해 20번 학습.
각각 데이터 size 500인 batch 들어감.
이터레이션 기준 80번 학습.

* Batch Normalization

: 학습의 효율 높이기 위해 도입. 배치 정규화는 Regularization 해주다고 볼 수 ○

- 장점.

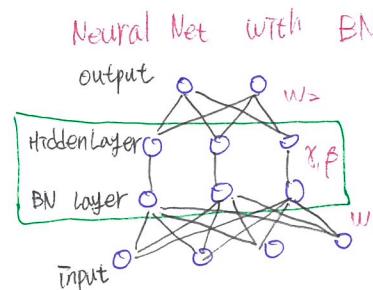
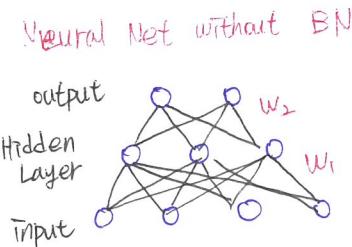
- ① 학습 속도 개선 : 학습률 높게 설정할 수 있기 때문
- ② 가중치 초기값 선택의 의존성 ↓ : 학습할 때마다 출력값 정규화
- ③ 과적합 (Overfitting) 위험 ↓ : 다른 이유 같은 기법 대체 가능
- ④ Gradient Vanishing 문제 해결

- 개념

: 배치 정규화는 학성화 할수의 값 또는 출력값을 정규화하는 작업을 말한다.
배치 정규화를 학성화 할수 이전에 하는지, 이후에 하는지 논의/실험 중.
신경망의 각 Layer에서 데이터(배치)의 분포를 정규화하는 작업.
인종의 노이즈를 추가하는 방법. 배치마다 정규화 향으로써 전체 데이터에 대한
평균과 분산의 값이 달라질 수 ○. 학습할 때마다 학성화값/출력값 정규화하기
때문에 초기화에서 자유로워질 수 있다.

- BN in Neural Net

: 각 hidden layer에서 정규화를 하면서 입력분포가 일정하게 되고, 이에 따라
Learning rate을 크게 설정해도 괜찮아진다. 결과적으로 학습 속도가 빠라짐.



- 입력분포의 균일화

: 학습할 때, hidden Layer의 중간에서 입력분포가 학습할 때마다 변화하면서
가중치가 엉뚱한 방향으로 갱신되는 문제 발생할 수 ○. 신경망 종이 깊어질수록
학습시에 가정했던 입력분포가 변화하여 엉뚱한 학습이 될 수 있다.

- 배치 정규화 알고리즘.

: 간단하게 배치 정규화는 학습 시 미니 배치를 한 단위로 정규화하는 것으로,
분포의 평균이 0, 분산이 1이 되도록 정규화하는 것.

Input : Values of x over a mini-batch : $B = \{x_1, \dots, x_m\}$
Parameters to be learned : γ, β

Output : $\{y_i = BN_{\gamma, \beta}(x_i)\}$.

$$\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad \# \text{mini-batch mean.}$$

$$\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2 \quad \# \text{mini-batch variance.}$$

$$z_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \# \text{Normalize}$$

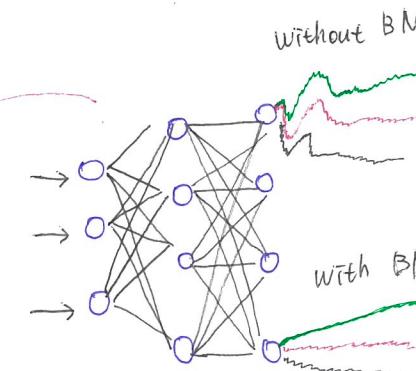
$$y_i \leftarrow \gamma z_i + \beta \quad \equiv BN_{\gamma, \beta}(x_i). \quad \# \text{scale and shift}$$

~로 정의하겠다!

INPUT으로 사용된 mini batch의 평균. 분산

hidden layer의 학성화값/출력값에 대해
평균이 0, 분산이 1인 정규화.

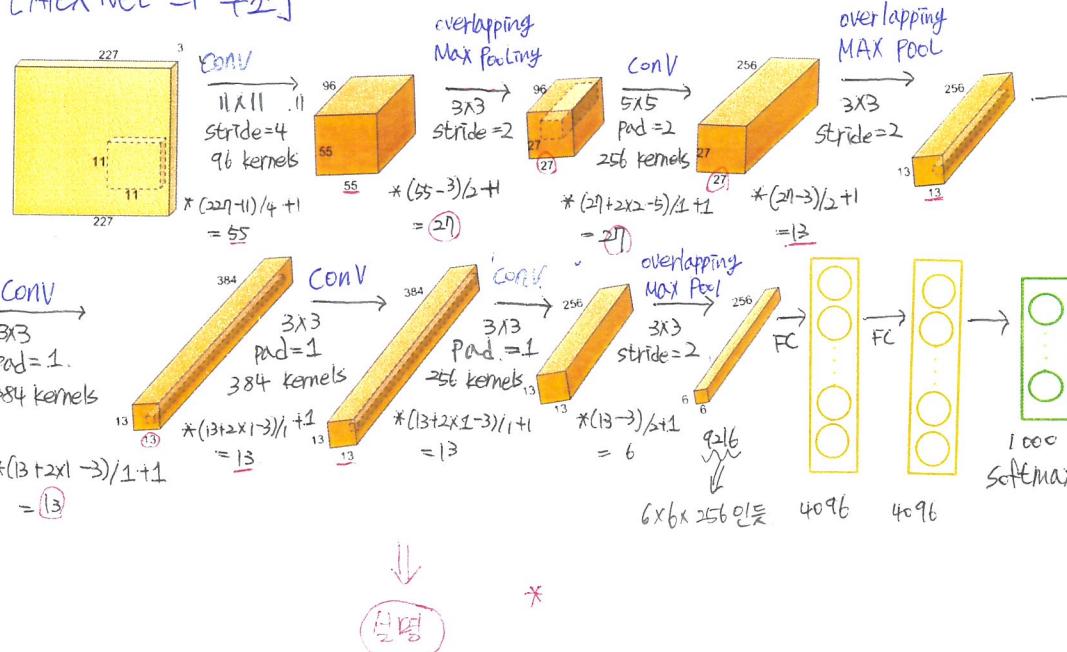
↓
데이터 분포가 덜 치우치게되고
배치 정규화 단계마다 학대 scale과
이동 shift는 변환 수행.



CNN에서 parameter 개수, tensor 계산

- AlexNet 기준 예시

[Alex Net의 구조]



- Input : 227x227x3 크기의 컬러 이미지
- Conv -1 : 11x11 크기의 커널 96개, stride = 4, padding = 0.
- Max Pool -1 : stride = 2, 3x3 max pooling layer.
- Conv -2 : 5x5 크기의 커널 256개, stride = 1, padding = 2
- Max Pool -2 : stride = 2, 3x3 max pooling layer
- Conv -3 : 3x3 크기의 커널 384개, stride = 1, padding = 1.
- Conv -4 : 3x3 크기의 커널 384개, stride = 1, padding = 1.
- Conv -5 : 3x3 크기의 커널 256개, stride = 1, padding = 1
- Max Pool -3 : stride = 3, 3x3 max pooling layer
- FC 1 : 4096 개의 fully-connected layer.
- FC 2 : 4096 개의 fully-connected layer
- FC 3 : 1000 개의 fully-connected layer

* Convolution layer의 output tensor size.

$$\text{Size (Width) of output Image} = \frac{I - k + 2p}{s} + 1$$

- I : Size (width) of input image

- K : Size (width) of kernels used in the Conv layer.

- P : padding size

- S : stride of the convolution operation.

(N)

+ N: Number of kernels. → 출력 이미지의 채널 수는 커널의 개수와 같음

ex) Alex Net에서 conv-1

입력 이미지 : 227x227x3, Conv layer : 11x11x3 크기, 커널 96개, stride = 4, padding = 0

$$\text{output image size} = \frac{227 - 11 + 2 \times 0}{4} + 1 = \frac{216}{4} + 1 = 55 \Rightarrow 55 \times 55 \times 96.$$

* Max Pooling layer의 output tensor size

$$\text{Size (Width) of output Image} = \frac{I - p_s}{s} + 1. \rightarrow p_s : \text{Pooling size.}$$

→ 출력의 채널 수는 입력의 개수와 동일 (conv. layer와 다름!)

conv layer의 수식에서 커널크기(k)를 p_s 로 대체. $p_s=0$ 으로 설정하면 동일한식.

ex) AlexNet에서 maxpool-1

stride = 2, 사이즈 : 3x3, 이전 단 (conv-1) 출력 크기 : 55x55x96.

$$\Rightarrow \frac{55 - 3}{2} + 1 = 27.$$

* Fully Connected layer의 output tensor size.

: FC Layer는 Layer의 뉴런 수와 동일한 길이의 벡터를 출력.

ex) AlexNet

Input : 227x227x3 → conv-1은 max pool 1 거치면서 55x55x96에서 27x27x96 변환

→ conv-2는 27x27x256에서 max pool 2 거치며 13x13x256으로 변환 → conv-3은 13x13x384

→ conv-4는 크기유지 → conv-5는 27x27x256 변환 → MaxPool-3은 6x6x256 줄임.

→ 이 이미지는 초기 4096x1 초기의 벡터로 변환되는 FC-1에 feed. → FC-2 크기유지.

→ FC-3은 size를 1000x1로 변환

* Convolution layer의 parameter 개수.

- CNN의 각 layer는 weight parameter와 bias parameter 존재.
- 전체 Network의 parameter의 수는 각 convolution layer 파라미터 수의 합.

Number of weights of the conv layer : $W_c = k^2 \times C \times N$

Number of biases of the conv layer : $B_c = N$

Number of parameters of the conv. layer : $P_c = W_c + B_c$

→ k : size(width) of kernels used in the conv. layer.

N : Number of kernels

★ kernel = filter ★

C : Number of channels of the input image

→ conv. layer에서 모든 커널의 깊이는 항상 입력 이미지의 채널 수와 같음.
파라미터, 모든 커널에는 $k^2 \times C$ 개의 parameter들이 있으며, 그러한 커널들이 N 개 존재.

ex) AlexNet의 Conv-1 : 입력 이미지의 채널 수 $C=3$, kernel size $k=11$.

전체 커널 개수 $N=96$. $\rightarrow W_c = 11^2 \times 3 \times 96 = 34,848$, $B_c = 96$ $\therefore P_c = 34,848 + 96 = 34,944$
conv-2, 3, 4, 5로 동일한 방법으로 614, 656, 885, 120, 1,321, 4,884, 992 개의

parameter 갖음. AlexNet의 conv. layer parameter 개수는 3,741,200개.

FC layer의 parameter 수가 더해지지 않았으므로 전체 네트워크의 parameter 개수 X.

→ conv. layer의 장점은 weight parameter가 모두 되므로 FC layer에 의해 매개변수 축소!

* Max Pool layer의 parameter 개수.

: Padding, stride, padding은 hyper parameter. ← 계산 X.

* Fully Connected layer의 parameter 개수.

: ANN에는 그 종류의 FC layer 존재

[마지막 conv. layer의 뒤에 붙는 FC layer]

[다른 FC layer에 연결되는 FC layer]

1 FC layer connected to a conv. layer.

a conv. lay.

$W_{cf} = O^2 \times N \times F$. (Number of weights of a FC layer which is connected to a conv. layer)

$B_{cf} = F$ (Number of biases of a FC layer which is connected to a conv. layer)

$P_{cf} = W_{cf} + B_{cf}$ (Number of parameters of a FC layer which is connected to a conv. layer)

↪ O : size(width) of the output image of the previous conv. layer.

N : Number of kernels in the previous conv. layer

F : Number of neurons in the FC layer.

ex) AlexNet

FC-1 layer : $O=6$, $N=256$, $F=4096 \rightarrow W_{cf} = 30,748,736 \quad B_{cf} = 4096$
 $P_{cf} = 30,752,832$

2. FC layer connected to a FC layer.

- Number of weights of a FC layer which is connected to a FC layer $\rightarrow W_{ff} = F_1 \times F$.

- Number of biases of a FC layer which is connected to a FC layer $\rightarrow B_{ff} = F$

- Number of parameters of a FC layer which is " " $\rightarrow P_{ff} = W_{ff} + B_{ff}$

→ F : Number of neurons in FC layer, F_1 : Number of neurons in the previous

→ $F_1 \times F$ 는 이전 FC layer에서의 neuron과 현재 FC layer의
neuron 사이의 총 연결 개수.

→ Bias parameter의 개수는 뉴런의 개수 F 와 같음.

ex) AlexNet - FC-3.

$F_1 = 4096$, $F = 1000 \rightarrow W_{ff} = 4096 \times 1000 = 4,096,000 \quad B_{ff} = 1,000$
 $P_{ff} = 4097,000$