[2-1. Error Analysis]

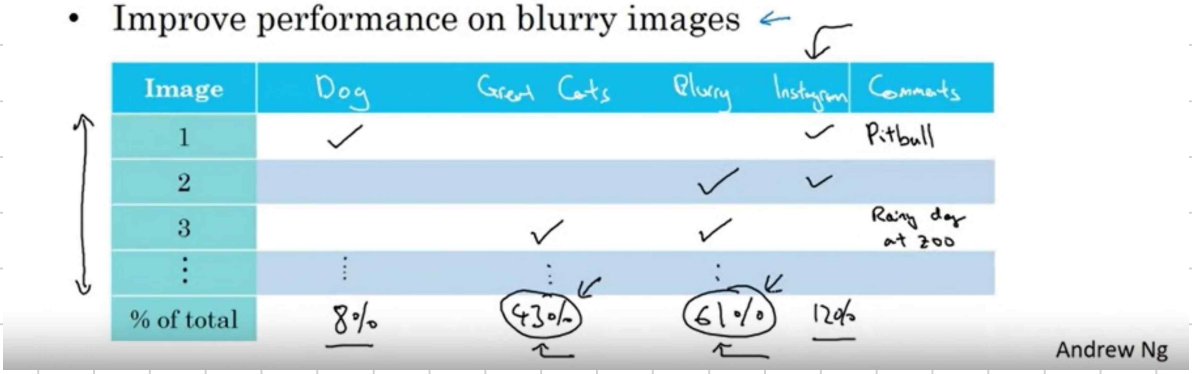
<Carrying Out Error Analysis>

- 오류분석 : dev sets 에서 잘못 표기된 예시를 찾고, False Positive와 False Negative를 찾는다. 그 후, 다양한 카테고리에 대한 오류의 총 개수를 세서 가장 비중을 많이 차지하는 것을 고치는 것이 가장 좋다.

Evaluate multiple ideas in parallel

Ideas for cat detection:

- Fix pictures of dogs being recognized as cats ←
- Fix great cats (lions, panthers, etc..) being misrecognized ←
- Improve performance on blurry images ←



<Cleaning Up Incorrectly Labeled Data>

Image	Dog	Great Cat	Blurry	Incorrectly labeled	Comments
...					
98				✓	Labeler missed cat in background
99		✓			
100				✓	Drawing of a cat; Not a real cat.
% of total	8%	43%	61%	6%	

- 훈련 세트:
  - 훈련세트는 무작위 오차에 대해 다소 둔감합니다. 잘못 라벨링 된 데이터의 학습 결과 오차가 무작위 오차와 크게 차이 나지 않을 경우 고치지 않아도 됩니다.
  - 하지만 무작위 오차가 아닌 시스템적인 오류(같은 라벨에 대해서 계속 오분류 하는 것)는 덜 둔감하기 때문에 문제가 있습니다.
- 개발 및 시험 세트:
  - 개발 및 시험 세트는 오차 분석시 잘못 라벨링으로 인한 오차의 비율을 구하시고, 전체 오차에서 얼만큼 차지하는지 살펴보고 결정하기를 권장합니다.
  - 개발 과 시험 세트의 분포는 같아야하기 때문에 동시에 살펴 볼 것을 권장합니다.

<Build your First System Quickly, then Iterate>

1. Set up dev/test set and metric
2. Build initial system quickly
3. Use Bias/Variance analysis & Error analysis to prioritize next steps.

[2-2. Mismatched Training and Dev/Test set]

<Training and Testing on Different Distributions>

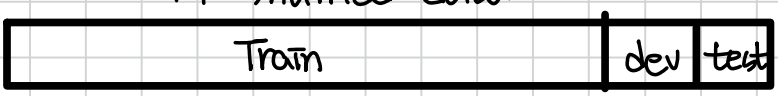
ex) Cat app example

- Data from webpages : 200,000 개의 고화질 데이터
- Data from mobile app : 10,000 개의 저화질 데이터

⇒ 이 두 데이터의 분포가 다르기 때문에 적용방법에 대한 고민이 필요하다.

(비추) Option 1) 무작위로 섞은 후, Train/dev/test로 나누기

210,000개의 shuffled data



205,000 개 (저화질) | 2,500 (저화질) | 2,500 (저화질)

- 장점 : train/dev/test가 같은 분포로 되어 있어서 다루기 쉽다.
- 단점 : 만약 내가 원한 데이터에서 좋은 결과를 얻기를 원한다면, 개수가 적어 힘들다.

Option 2) 일부는 Train에, dev와 test에 모두 사용

200,000개 (고) | 5,000개 (저) | 2500 (저) | 2500 (저)

- 장점 : 내가 원한 데이터에서 더 좋은 성능, 장기적으로 좋음
- 단점 : train/dev/test의 분포가 달라짐

<Bias and Variance with Mismatched Data Distribution>

* Training-dev set : Same distribution as training set, but not used for training.

training set 과 dev set의 차이는 ① 두 개의 분포의 차이 때문인지 ② model이 training set에 생긴 noise의 영향인지 구별하기 어려운데, 이때 training dev set을 사용함

Human error	0%	0%	0%	0%
Training error	1%	1%	10%	10%
Training-dev error	9%	1.5%	11%	11%
Dev error	10%	10%	12%	20%
	↓ Variance	↓ data mismatch	↓ avoidable bias	↓ & data mismatch avoidable bias

* Bias/variance on mismatched training and dev/test sets

Human level error
Training set error
Training-dev set error
Dev error
Test error

↕ Avoidable bias
↕ Variance
↕ data mismatched
↕ degree of overfitting to dev set

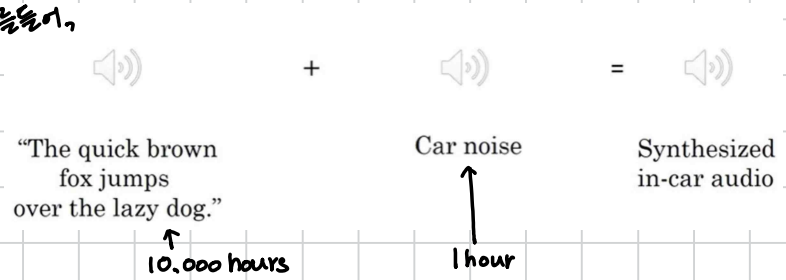
* More general formulation

	수집, 구매한 데이터	실제 사용될 어플리케이션에서 얻은 데이터
사람 수준	사람 수준 오차	
	↕ 회피가능 편향	
훈련된 데이터에서 생긴 오차	훈련 오차	
	↕ 분산 문제	
훈련되지 않은 데이터에서 생긴 오차	훈련-개발 오차 ⟷	개발/시험 오차
	데이터 불일치	

〈Addressing Data Mismatch〉

- Carry out manual error analysis to try to understand different between training and dev/test sets
- Make training data more similar; or collect more data similar to dev/test sets
  → Artificial data synthesis

예를들어,

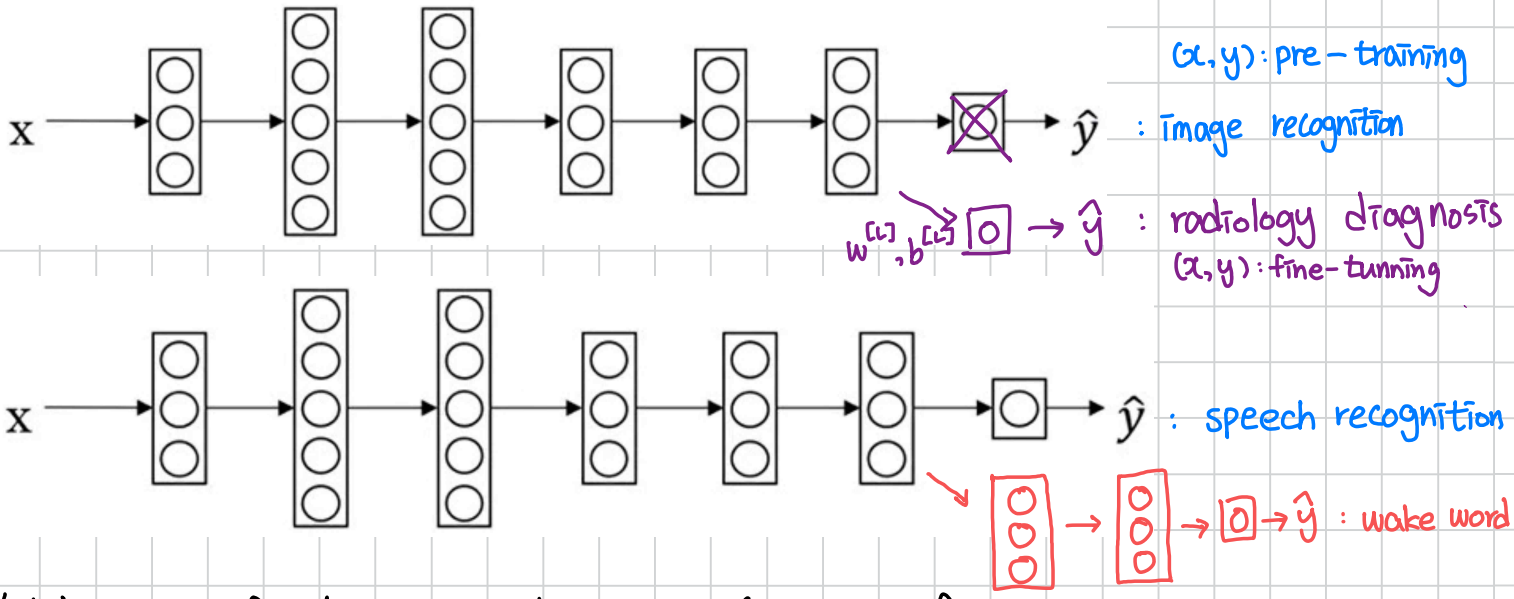


→ ① Overfit to 1 hour of car noise
② Synthesis data가 일부 데이터
①, ②의 문제가 발생할 수 있음

[2-3. Learning from Multiple Tasks]

〈Transfer Learning〉 (전이 학습)

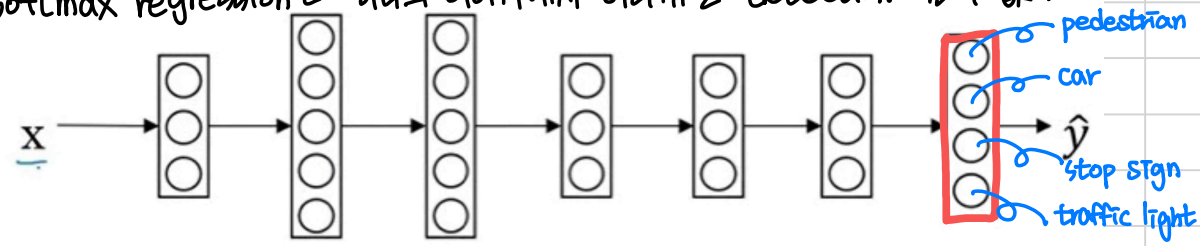
*Transfer Learning : 기존 학습된 모델에서 마지막 층을 제거한 후, 새로운 문제에 적합한 층을 연결하여 학습하는 것



* When transfer learning makes sense (Transfer from A to B)
  - Task A and B have the same input x.
  - You have a lot more data for Task A than Task B.
  - Low level features from A could be helpful for learning B.

〈Multi-task Learning〉

*Multi-task Learning(다중 작업 학습) : 하나의 신경망이 여러작업을 동시에 할 수 있도록 하는 것
※ softmax regression은 하나의 이미지에서 여러개를 detection 할 수 없다



*When multi-task learning make sense
  - Training on a set of tasks that could benefit from having shared lower-level features
  - Usually : Amount of data you have for each task is quite similar.
  - Can train a big enough neural network to do well on all the task.

[2-4. End-to-End Deep Learning]

<What is End-to-End Deep Learning?>

*End-to-End Deep Learning : 자료처리 시스템/학습시스템에서 여러 단계에 필요한 처리과정을 한번에 처리, 大量 데이터 필요

ex) Speech recognition example

일반적 : audio (x) —MFCC→ features —ML→ Phonemes → words → transcript (y)

end-to-end : audio → transcript

+) end-to-end는 大데이터를 필요로하기 때문에, 단계를 나눠서 학습하는 것이 효율적
즉, 복잡한 문제를 간단한 문제로 나눈 후, 데이터의 정보가 각각의 작업에 더 적합되도록 사용

<Whether to use End-to-End Deep Learning>

*Pros and cons of end-to-end deep learning

<Pros>

- Lets the data speak. 즉, 사람의 선입견 영향을 덜 받음
- Less hand-designing of components needed. 특성이나 중간 표현을 직접 설계하는데에 드는 시간을 줄일 수 있음

<Cons>

- May need large amount of data
- Excludes potentially useful hand-designed components. 이는 데이터가 적을 경우에 효율적인 특정 지식을 사용할 수 없다.

*Applying end-to-end deep learning

- key question: Do you have sufficient data to learn a function of the complexity needed to map x to y?