C3W2

Carrying out & error analysis:-

Let we have a cat classifier whic have 90% accuracy. We want to improve it. After analysis we saw it classified some dog as cat.

should we try to make cat classifier better or work on dog classifier?

Recommendation:-

Error analysis:-

- Get 100 mislabaled dev set examples.
- Count up how many are dogs manually.

Let say we found 5 dogs there. So if we works even few months on dog classification and fix this issue then our accuracy will be 90.5%. This is the best case scenario it is also called ceiling.

But instead of getting 5 dog if we get 60 dog then it might worth spending few months to classify dog problem.

Idea for cat detection:-

- fix picture of dogs being recognized as cats.
- Fix big cat (lions, panthers etc) being misrecognized.
- Improve performance on blury image.

Image	Dog	Great cats	blury image	comment
1	✓			
2				

2				
3			✓	
⋮		✓	✓	rainy day at zoo.
% total	8%	43%	61%	

we should work on blurry images & great cats to improve performance

Clean up incorrect label data:-

if we can get a desired bumpup by correcting the incorrect lablel then we should try to correct that Otherwise no.

Let we have an model that have erro percentage :-

Image	Dog	Great cat	blurry	incorrect label	comments
1					
2					
⋮					
total	8%	43%	61%	6%	

its better to focus on blurry image instead of incorrect label here.

If we plan to correct the label:-

⇒ Apply same process to your dev & test sets.

⇒ Check both the images that algorithms got right & wrong.

⇒ [illegible]

→ Train & dev/test might come from slightly different distribution.

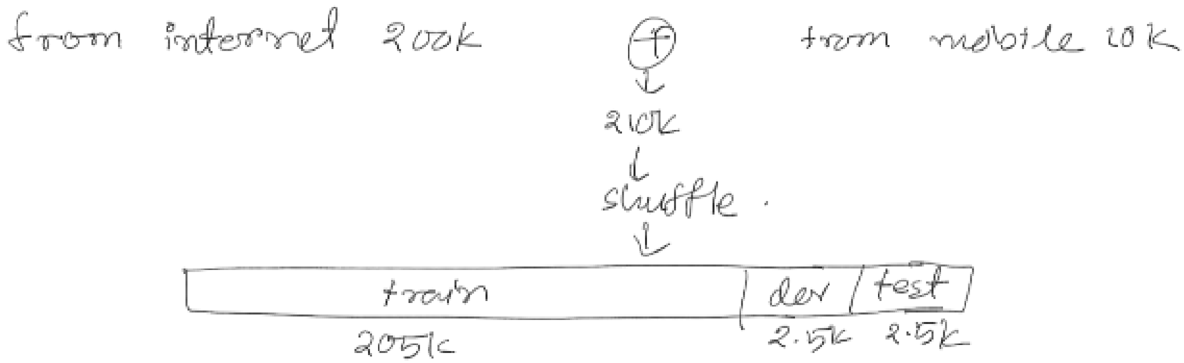
Build your initial system quickly then iterate

Training & testing on different distribution:-

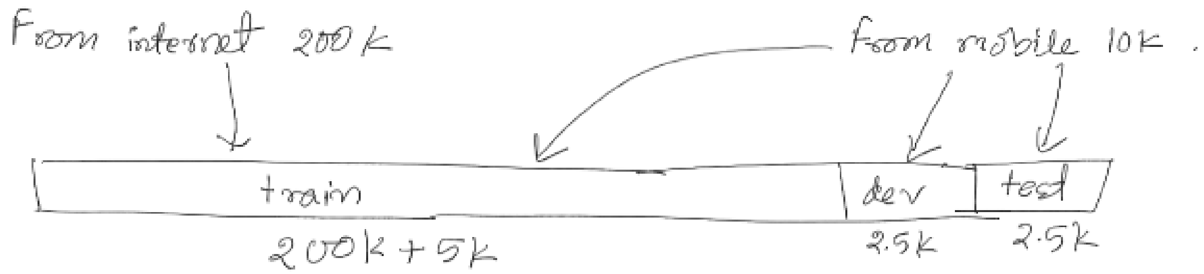
Let we are making a cat clasifier for mobile app. we collect 200k images from internet & 10K image from mobile device (low resolution, blurry, bad shapped) How should we use the data?.

Deep learning has a lot of hunger for data. so we should use as much data as possible. so we should use all 210k images. How to split the data:-

Option 1:- Not recommended cause target will be moved



Option 2:- Recommended.



Let we use the option 2 and got following scenario.

→ Bayesian error 0%

→ training error 1%.

→ Dev error 10%.

we really don't know if it's a problem of variance or data mismatch (dev & train datas are from different distribution)

To find the problem split the train data in two category train & dev train (both from same distribution)

train	train-dev	dev	test

if we get the following scenario:-

	Case 1	Case 2	Case 3	Case 4
Bayesian error (%)	0	0	0	0
train error (%)	1	1	10	10
train-dev error (%)	9	1.5	11	11
Dev error (%)	10	10	12	20
issue	variance	data mismatch	Bias	Bias + Data mismatch

Addressing data mismatch:-

→ Carry out manual error analysis to try to understand the difference between training & dev test set

For example if we use the regular user speech in a self driving car NLP then training & test data will have mismatch as in car the speech are pretty noisy.

→ Make training data more similar: Collect more data similar to dev/test set.

For example simulate noisy environment in the training data.

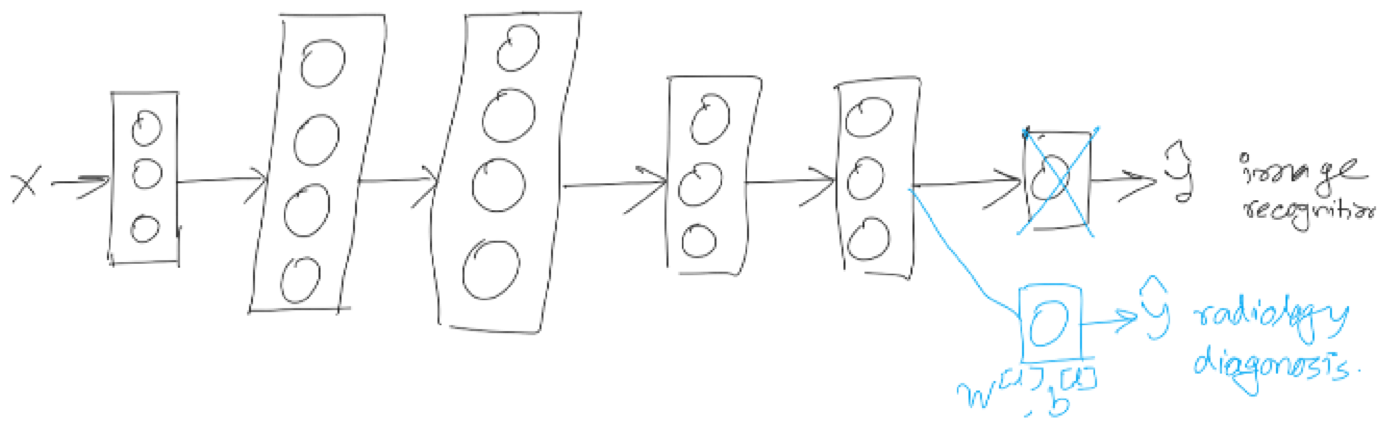
Artificial data synthesis:-

Clear data + car noise = synthesize in-car audio.

if we have 10,000 hours of data but only 1 hours of car noise then we can repeat the car noise 10,000 times and synthesize the data. But in this way we will overfit our NN to that 1 hour of car noise data.

Transfer Learning:-

take knowledge from a neural network that has learned from one task & apply that knowledge to some other task. Such as a cat classifier or part of the cat classifier can be used to recognize an x-ray.



Some of the layer already learned the curve/shape/edge from image recognition. so we don't need to train the whole network. We just need to retrain the last couple of layer. This retraining is called **fine-tuning**. the layer which isn't fine-tuned are called **pre-trained**. How many layer we will fine tune depends on the data we have.

Transfer from $A \rightarrow B$

Transfer learning makes sense if we have a lot of data for the problem we are transferring from & relatively less data to the problem we are transfering to.

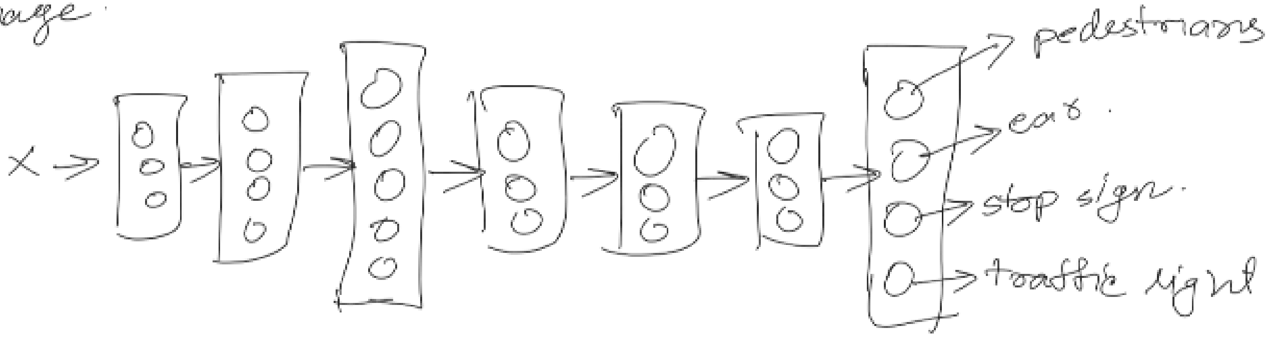
→ the input type should be same for both task(A, B). like both of them are image or both are speech.

→ Low level feature of A could be helpful for learning B.

Multi-task learning :-

A neural network can learn multiple things at the same time.

For example for a self driving car. the car needs to detects multiple things at the same time. for example. pedestrians, stop sign, other cars, traffic light. Unlike softmax regression all of them needs to be detected in a single image.



Loss: $\hat{y}^{(i)}$ $(4,1)$

$$\rightarrow \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{4} \mathcal{L}(\hat{y}_j^{(i)}, y_j^{(i)})$$

↳ usual logistic loss.

/ usual logistic loss:

$$-y_j^{(i)} \log \hat{y}_j^{(i)} - (1-y^{(i)}) \log (1-\hat{y}_j^{(i)})$$

$$Y = \begin{bmatrix} | & | & | & & | \\ y^{(1)} & y^{(2)} & y^{(3)} & \cdots & y^{(m)} \\ | & | & | & & | \end{bmatrix}$$

$(4, m)$

sometimes some label might be missing.

$$Y = \begin{bmatrix} 1 & 0 & ? & ? & \\ 0 & 1 & ? & ? & \cdots \\ 1 & ? & 0 & ? & \\ ? & ? & 1 & ? & \end{bmatrix}$$

we can use this kind of label as well. whenever we see a (?) then we have to omit that from loss function

<u>When does multitask learning make sense?</u>.

→ Training on a set of tasks that could benefit from having shared lower level feature

→ 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.

End-to-End deep learning:-

there have been some data processing systems or learning systems that require multiple stages of processing. End to end deep learning takes all those steps multiple stage & replace it with usually a single neural network.

$X \xrightarrow{MFCC}$ feature $\xrightarrow{ML}$ phonemes $\rightarrow$ words $\rightarrow$ transcript $\rightarrow y$
audio

End to end deep learning:-

audio $\longrightarrow$ transcript

Pros:-
→ Let the data speak.

→ Less hand designing component needed.

Cons:-

→ Need lot of data.

→ Excludes potentially useful hand-designed component.