**Supervised (Inductive) Learning:** Train + labels (desired output) e.g. classification, regression, obj detection, semantic segmentation
**Unsupervised:** Train (no desired output) e.g. clust, dim reduction
**Semi-supervised:** Train +few desired output
**Self-supervised:** Pretext task (e.g. predict if rotated image), learn the latent features (output labels) intrinsically from data
**RL:** Rewards from seq actions

**Evaluation:** Acc (must be class specific in case of imbalanced data)
E.g. If only 30% 1 & predict all 0, we have 70% acc! totally biased

| | | Predicted: +ve | Predicted: -ve | |
|---|---|---|---|---|
| Actual | +ve | TP | FN (Type II Error) | **Recall/ Sensitivity**: TP/(TP+FN) |
| | -ve | FP (Type I Error) | TN | **Specificity**: TN/(TN+FP) |
| | | **Precision**: TP/(TP+FP) | **-ve Pred Val**: TN/(TN+FN) | **Acc**: (TP+TN)/(All) |

**F1**: (Precision * Recall)/(Precision + Recall)

**MSE (L2 Loss) for regression:**
$$MSE = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

**Receiver Operating Curve (ROC):**
TPR vs FPR (plot against all threshold, AUC = 1 best)

**Information:** (Generally) Measured in bits ($\log_2$): Amt of "surprise" arising from a given (stochastic) event E: $I(E) = -\log(P(E))$

E.g. Three boys: Tom (20%), Bob (1%), Sam (79%)
Bob = $-\log_2(0.01) = 6.6$, high surprise vs
Sam = $-\log_2(0.79) = 0.34$, low surprise

**Entropy:** Avg lvl of information inherent in possible outcomes:
Higher Entropy signifies more uncertainty in what action an agent would choose: $H(X) = -\sum_i P_X(x_i) \log_b P_X(x_i)$

$$Entropy = -(0.20 \log_2(0.20) + ... + 0.79 \log_2(0.79)) = 0.26$$

**KL-Divergence (relative entropy):**
Expression of Surprise, diff btw 2 prob distribution P & Q:
Under the assumption P & Q are not close, KL divergence high (surprise).
If close then KL divergence low

**Likelihood ratio (n indep samples):** $LR = \prod_{i=0}^{n} \frac{p(x)}{q(x)}$
(how much more likely samples came from P dist than Q dist)

**Log likelihood ratio:** $LLR = \sum_{i=0}^{n} \log\left(\frac{p(x)}{q(x)}\right)$

**Expected LLR:**
(Avg, how much each sample gives evidence of P(x) over Q(x))
$$D_{KL}(P \parallel Q) = \sum_{i=0}^{n} p(x_i) \log\left(\frac{p(x_i)}{q(x_i)}\right)$$
$$D_{KL}(P \parallel Q) = \sum_{i=0}^{n} p(x_i) \log(p(x_i)) - \sum_{i=0}^{n} p(x_i) \log(q(x_i))$$

Entropy (Info content of P) - IC of Q weighted by P
If P is the "true" distribution, then the KL divergence is the amount of information "lost" when expressing it via Q.

**Cross-Entropy (min unlike max likelihood):**
Cross entropy is, at its core, a way of measuring the "distance" between two probability distributions P and Q. (Entropy on its own is just a measure of a single probability distribution)

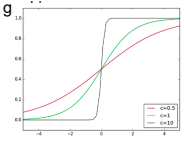
CE = Combination of the entropy of the "true" distribution P and the KL divergence between P and Q:
$$H(p, q) = H(p) + D_{KL}(p \parallel q) = -\sum_{i=0}^{n} p(x_i) \log(q(x_i))$$

E.g. Predict one-hot vector: True {0, 0, 1, 0, 0}
Output (of softmax layer): Predicted {0.01, 0.02, 0.75, 0.05}
Predicted class (argmax): 2 (zero index)

$$-(0 \cdot \log(0.01) + 0 \cdot \log(0.02) + 1 \cdot \log(0.75) + 0 \cdot \log(0.05))$$

| **Cross-entropy:** | **KL-Divergence:** |
|---|---|
| Avg # of total bits needed to encode data from a source with dist p when we use model q | Avg # of extra bits needed to encode the data, when using dist q instead of the true dist p. |

**Logistic Regression:**
Given an input space X, the goal is to predict for every sample x ∈ X to which class it belongs. For 2 class classification, the g
predict output space $Y = \{0, 1\}$ or $\{-1, +1\}$



Linear: $f(x) = w.x + b$, unbounded space
Need to map $(-\infty, \infty)$ to $(0, 1)$ $st$ 0 map to 0.5

**Use logistic sigmoid fn** $s(a)$:

$$s(a) = \frac{\exp(a)}{1 + \exp(a)} = \frac{1}{\exp(-a) + 1} \frac{\exp(a)}{\exp(a)} = \frac{1}{\exp(-a) + 1}$$

Note if multi class (k) class, use Softmax:
$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for} \quad i = 1, ..., K \quad \text{and} \quad \mathbf{z} = (z_1, ..., z_K) \in \mathbb{R}^K$$

$h(x) = s(f_{w,b}(x)) \in (0, 1)$ aka prob(sample x class label $y = +1$)

**Cross-Entropy loss (bin class):**

**CE Loss:** (1 sample: $(x_i, y_i)$)
$-\overline{y}_i \log(p_1(x_i)) - (1 - \overline{y}_i) \log(1 - p_1(x_i))$ where $p_1(x_i) = 1 - p_2(x_i)$

**Avg CE Loss:** (dataset: n indep samples)
$$\frac{1}{n} \sum_i -\overline{y}_i \log(p_1(x_i)) - (1 - \overline{y}_i) \log(1 - p_1(x_i))$$

Opti problem (convex) to be solved is (finding best parameter w min CE loss):
$(w^\star, b^\star) = \text{argmin}_{w,b}$ (avg CE Loss)
where $p_1(x_i) = h(x_i) = \frac{1}{\exp(-w \cdot x - b + 1)} = s(w \cdot x_i)$

$$\nabla_w L = \nabla_w \left( (-1) \cdot \sum_{i=1}^{n} y_i \log(s(w \cdot x_i)) + (1 - y_i) \log(1 - s(w \cdot x_i)) \right) = \sum_{i=1}^{n} x_i (s(w \cdot x_i) - y_i) = \sum_{i=1}^{n} x_i (h(x_i) - y_i)$$

**Optimizing Param:**
1. Init weights randomly $\sim N(0, \sigma^2)$
2. Loop until convergence:
   a. Compute gradient: $\frac{\partial J(W_n)}{\partial W_n}$
   b. Update coeff:
   $W_{n+1} = W_n - \alpha \cdot \frac{\partial J(W_n)}{\partial W_n}$
3. Return weights, grad to 0

Hence gradient wrt $w$: (sum of points $x_i$ weighted by the diff btw the predicted value $h(x_i) = s(w \cdot x_i)$ and true val $y_i$

**Data Set:** (No best split) **Training:** run your learning algo
**Development:** tune parameters **Test:** evaluate perf

**Common Error:**
1) Data mismatch (e.g. diff. Resolution img)
2) Bias and variance
   **Bias**: error rate on the training set
   **Variance**: how much worse test set error than train (e.g. 15% error (85% acc) on train, bias = 15%. But 16% error on test, variance = 1%)

**Set Distribution:**
If all same dist, then optimising for web imgs mostly here, which is diff from target dist.

| 100k imgs (from Web) | 8k imgs (from Target dist) |
|---|---|

| Train (104k imgs) | Dev (2k imgs) | Test (2k imgs) |
|---|---|---|

Instead make test & dev from the same dist such that optimizing to perform well on the target dist.

| 100k imgs (from Web) | 4k imgs | 4k imgs |
|---|---|---|

| Train (104k imgs) | Dev (2k imgs) | Test (2k imgs) |
|---|---|---|

However, the training dist is now diff from the dev/test distribution: it will take longer & more effort to optimize the model. More importantly, cannot easily tell if the classifier error on the dev set relative to the error on the train set is a variance error, a data mismatch error, or a combination of both.

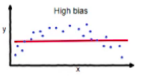
You can use a **bridge set (training data not used for training)** to trouble shoot where the error comes from.

| 100k imgs (from Web) | 4k imgs | 4k imgs |
|---|---|---|

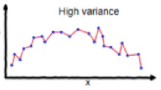
| Train (102k) | Bridge (2k) | Dev (2k) | Test (2k) |
|---|---|---|---|
| Error 2% | 3% | 10% | |

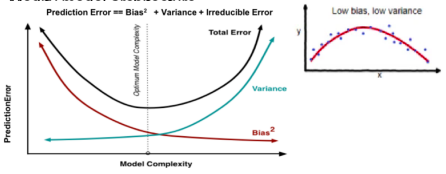
Error 8% (btw dev & train): 1% var + 7% data mismatch
Hence our classifier performed well on data from the same distribution but poorly when from a diff distribution, (data mismatch problem). Else if higher var & lower data mismatch error, implies overfitted to train set.

**Rules of thumb:**
**High avoidable bias: (underfitting)**
- Increase the size of your model
- Modify the input features based on error analysis
- Reduce or eliminate regularisation (reduces bias but increases variance)
- Modify model architecture




**High variance: (overfitting)**
- Add data to your training set (e.g. by data augmentation)
- Add regularisation
- Add early stopping
- Feature selection to decrease # of features (may increase bias)
- Decrease the model size (May increase bias)
- Modify input features based on error analysis




**Batch-norm layers:**
- Faster training (allow higher lr), improves reg. & acc, reduces dependencies on init
- Ensures received input batch has mean 0 & std dev 1

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma, \beta$
**Output:** $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$ // mini-batch mean
$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$ // mini-batch variance
$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$ // normalize
$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$ // scale and shift

**MSE (of unseen test pt)**: bias & var tradeoff + inherent noise, $\varepsilon$:
$y = f(x) + \varepsilon$ where $\varepsilon \sim rv\left(0, \sigma_\varepsilon^2\right)$
$$\text{Total Error} = E_x\left[\text{bias}\left[\widehat{f}(x)\right]^2\right] + E_x\left[\text{var}\left(\widehat{f}(x)\right)\right] + \sigma_\varepsilon^2$$

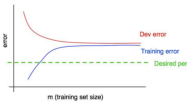
**MSE:** Avg squared diff of a pred $\widehat{f}(x)$ from its true val y:
$$MSE = E\left[\left(y - \widehat{f}(x)\right)^2\right]$$

**Bias:** Diff of avg val of pred (over diff realization of train data) to the true underlying function $f(x)$ for a given unseen test point $x$:
$$bias\left[\left(\widehat{f}(x)\right)\right] = E\left[\left(\widehat{f}(x)\right)\right] - f(x)$$

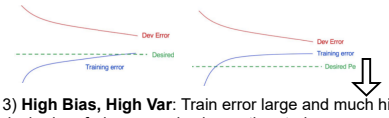
Note realization implies access to only a portion of the underlying data as training data (unrepresentative of the underlying population)

**Variance:** mean squared deviation of $\widehat{f}(x)$ from its expected value $E\left[\widehat{f}(x)\right]$ over diff realizations of training data:
$$var\left[\left(\widehat{f}(x)\right)\right] = E\left[\left(\widehat{f}(x) - E\left[\widehat{f}(x)\right]\right)^2\right]$$

**Learning Curves:**
1) **High Bias**: Error plateau, dev error higher than train error, very high train error




2) **High Var, Low Bias**: Train error relatively low, dev error much higher than train error, bias is small but variance is large. Perhaps add more training data will close the gap




3) **High Bias, High Var**: Train error large and much higher than desired perf, dev error also larger than train.

**Regularisation:**
Used to discourage the complexity of the model by penalizing the loss function (help solve overfitting)

**1) L1 (Lasso/L1 Norm)**
Feature selection by assigning insig input features with 0 weight & useful features with a non-0 weight

$\lambda$ is the penalty (reg) param: If 0, back to original loss function no penalty, if too large, weights become close to 0, hence high bias (underfitting)

**2) L2 (Ridge)**
Forces the weights to be small but $\neq 0$ (non-sparse)
Not robust to outliers as square terms blows up the error differences of the outliers & $\lambda$ tries to fix it by penalizing the weights.

Performs better when all the input features influence the output and all with
weights are of roughly equal size

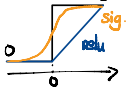
$$L(x, y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{p} |\theta_i| \text{vs} \sum_{i=1}^{p} \theta_i^2$$

$\theta_i$ is the coeff of the i-th term

Put the L1/L2 into the optimizer or use torch.norm(), total_loss = loss + L1/L2, then backward prop

$\hat{y} = g(w_0 + X^T w)$ linear decis$^n$ no matter network size
Nonlinear $g(a) \Rightarrow$ approx of arbit complex f$^n$

i) Threshold : $[a > 0]$
ii) Sigmoid $\sigma$ : $\frac{1}{1+e^{-a}}$
iii) ReLu : $\max(0, a)$
- non 0 centered
- nondiff @ 0, dying problem




XOR : simplest linearly inseperable problem

a) Reg. : Linear $(-\infty, +\infty)$ / ReLu $(0, +\infty)$ & MSE loss
b) Bin class. : $\sigma(0,1)$ & BCE loss
c) Categorical : Softmax (0,1) $\forall$ class, $\Sigma$ to 1 & CE loss
d) Multiclass cat : —"— & BCE loss $\forall$ class

Empirical loss: Total loss over entire data

$J(w) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f(x^{(i)}, w), y^{(i)})$ (pred., act.) MSE: $\frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - f(x^{(i)}, w))^2$

BCE: $\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)} \log(\sigma(x^{(i)}, w)) + (1-y^{(i)}) \log(1 - f(x^{(i)}, w))\right]$

Training : Find $w^* = \arg\min_w J(w)$, lowest loss
i) Init weights rand $\sim N(0, \sigma^2)$
ii) Loop until convery. $\nabla w = \frac{\partial J(w)}{\partial w}$, $w \leftarrow w - \eta \nabla w$
iii) Return w
- Small $\eta$ : converge quickly (local optima) } Use adap!
- large $\eta$ : unstable & diverges
- Easy compute (implement$^n$ & understand$^n$)
- May trap at local minima (need convex f$^n$ too)
- Update weights only after calculating $\nabla$ on entire dataset (too large $\Rightarrow$ years to compute + large mem)

SGD: Update model param more freq.
- Converges in less time & may get new minima's
- But high var in model param (may shoot ↑ even after reaching global min)
- To get some convergence as GD, need slowly ↓ $\eta$

Pick Batch B of data
$\nabla w = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_w(w)}{\partial w}$
} Fast compute & allow // compute better est of true $\nabla$ than 1 datapoint, smoother converge, allow larger $\eta$

Adagrad : Changes $\eta$ $\forall$ param & t (Adadelta, RMS Prob)
- No need tune $\eta$ manually & trainable on sparse data
- $\eta$ always ↓ -> slow training
- Computationally exp (need 2nd order der.)

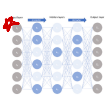
ADAM : momentum of 1st+2nd order
- Very fast converge + rectifies decaying $\nabla$ & high var
- Comp. exp.

Backprop
$\frac{\partial J(w)}{\partial w_1} = \frac{\partial J(w)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$




---

**Generative Modeling:** Take input train samples from some dist and learn a model that represents that dist (Debiasing)



- Capable of uncovering underlying features in a dataset
- Detect outliers in the dist for training

**AE:** Encoder: Unsupervised learning of a lower dim feature space (under complete), $z$ (salient features + easier to work with) representation from unlabelled train data
Decoder: Use $z$ to reconstruct the obs $\hat{x}$. Note no labels used.
$L(x, \hat{x}) = ||x - \hat{x}||^2$




- Form of compression, smaller $z$ -> larger training bottleneck
- Bottleneck hidden layers forces network to learn compressed latent rep. while reconstruct loss forces $z$ to capture as much information of the data

**Regularizer AE:** Overcomplete case with a loss function that encourages the model to have other properties beside copy-paste

- sparsity rep, smallness of rep derivatives (similar img, sim lv), robust to noise/missing inputs

**Sparse AE:** $L(x, g(f(x))) + \Omega(z)$ decoder, encoder output + sparse penalty
**Denoising AE:** Add noise to input image, AE learns to remove noise

---

Bag word (BOW): Col : sentences, Row : all words } Bin 0/1 -> SPARSE 1-hot encode

Term-freq-Inv doc freq (TF-IDF) : rare words carry more meaning
$TF = \frac{\#\text{ appearances for a term}}{\text{Total \# words in doc}}$
$IDF = \log(\#\text{ docs} / \#\text{ docs term appear in})$
} TF-IDF = TF * IDF

Less-space rep. : cooccurence models (SVD)
vector embedding models (word2vec)

Word embeddings : ea. word = pt in n dim space, (1-layer) rep by vector of len n (~100-300)
i) Cont-BOW : Pred center word from sum context >—
ii) Skip-grams : Pred sum context words from given center word —<
- Max. likelihood of context, given Row word
eg. I love pizza, P(I | pizza) & P(love | pizza)

Train word2vec : (weights of layer) But X prepare order!
- Decide window size & embedding size & vocab size
- Training obj : Negative sampling (bin LR class.)
- Fast accurate model that capture semantic content
"Gensim" lib, pretrained wc.
- Interested in the 1 hidden (latent rep.) layer, not the output

---

CNN : Preprocessing vals eg. normalize to ↑ perf
- ↓ # params to learn, preserves locality (no need flatten img)
- $\forall$ conv layer, # activat$^n$ map = # filters / feature maps / kernels
- For 2D image H & 2D kernel F, conv. operation: more complex

$G = H * F$ $\quad G[i,j] = \sum_{u=-k}^{k}\sum_{v=-k}^{k} H[u,v] F[i-u, j-v]$

However in CNN, do not flip kernel, instead use cross-correlat$^n$ operation : $G = H \otimes F$ [=] kernels transposed, symm, same output!

$G[i,j] = \sum_{u=-k}^{k}\sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$

Pos in filter (given dim 2k+1) pos in resulting activat$^n$ map, centered @ 0

- Still no diff in results, just convent$^n$
- Stride : Output size = $\frac{N, \text{ input size} - F, \text{ feature size}}{\text{Stride}} + 1$
- Padding : Add padding to output ↓ prob size
- Pooling : non linear down sampling (max/avg), No learning
Stride size & Pooling window size

Input -> [Conv+Relu, Pooling] ... -> [Flatten+FC+Softmax]
Feature learn$^n$ (L->R, T->D) Classificat$^n$

- Can be used for any vector/matrix rep eg. Audio

Audio as Img (waveform, spectogram : y axis freq., x time)
- Effective : Mel Freq. cepstrum (MFC, nonlinear) fit human hear

---

**Variational AE (VAE):** Sample from mean & std dev to compute $z$ (probabilistic, no longer deterministic).

eg. $z$ = [smile, latent dist.] sample 1, sample 2 $\quad p_\phi(z|x)$ $q_\theta(z|x)$

$L(\Theta, \theta, x) = \text{reconst loss} + \text{reg term}$
$= ||x - \hat{x}||^2 + \sum_j KL(P_j(z|x) \,||\, p(z))$ (fixed prior ld, inferred ld) } KL diverg.

- Choice of prior on ld: common $p(z) \sim N(0,1)$, encourages encodings to distribute evenly around center of $z$, penalizes if tries to cluster points tgt (memorise data)

- Generation (only sampling for $z$ + decoder) vs
- Training (Encoder: produces $z$ ($\mu$ + $\sigma$), calc reg loss, then ....., calc reconstruct loss, minimize both losses tgt

- Cannot backdrop grad through sampling layers, ∴ reparametrize sampling layer (sum fixed $\mu$ & $\sigma$ scaled by random const drawn from prior dist. $z = \mu + \sigma \cdot \varepsilon \sim N(0,1)$

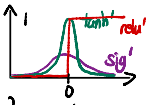
- Goal is to find disentangled uncorrelated meaningful lv
Latent Perturbation: Slowly increase/dec 1 lv, keep rest same for intepretation

**Conditional VAE:** Feed label $Y$ into model, since VAE no control over the generated data

---

RNN : Output : $\hat{y}_t = W_{hy}^T h_t$ Input : $x_t$ vector
Update hidden state : $h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t)$
i) Many values > 1 (Exploding $\nabla$) : $\nabla$ clipping
ii) Many values < 1 : Small # mult, $\nabla$ ↑ smaller, bias param unable to capture short term dependencies

a) $g(a)$ : ReLu, prevent f' from shrinking $\nabla$ when $x > 0$




b) Weight Init: Weights -> $\mathbb{I}$ matrix, Bias -> 0 } prevent weights shrink to 0

c) Using gates to track & control info thru time

LSTM: - Forget irrelevant info of prev state
$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
- Store relevant new info into cell state
$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ $\quad \tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$
- Update cell state selectively $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
- Output gate : control what to send to next t
$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$ $\quad h_t = o_t * \tanh(C_t)$

GRU or attention networks (transformers)

---

**Transfer Learning (Opti):** Higher start, slope & asymptote
- Identify related task with abundant data, reuse pretrained model

E.g. LeNet 5, AlexNet, VGG-16, Inception (GoogLeNet)
- More layers -ve effect, degradation problem

**ResNet:** residual blocks at intermediate layers
**ResNext:** "Split-transform-merge" + residual block connection


**DenseNet:** Each layer's feature map concatenated to the input of every successive layer within dense block

---

**Generative Adversarial Networks (GAN):**

Generator : noise -> imitat$^n$ of data to try trick discriminator
Discriminator : Tries identify data from fakes generated by G


Data: Real -> D use as +ve e.g., Fake -> D use as -ve e.g.
Alternative training:
D Training: D ignores G loss & uses its own loss
- Sample random noise to produce G output
- classify & calculate D loss
- BP. through D & G weight, only update G weights
G Training: Opposite
- G loss penalizes G for failing to fool D

If G perfect, D acc = 50%, D feedback gets less meaningful overtime, may collapse, sin ce GAN not stable, but fleeting convergence

**Minimax loss:** $\min_{\theta_g}\max_{\theta_d}\left[E_{X\sim P_{data}} \log D_{\theta_d}(c) + E_{Z\sim P(z)} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right]$
D output real | G output | z | D output fake

(CE loss btw real & fake dist)
- D max probability real & output prob fake low
- G no influence on 1st term, tries min 2nd term (1- ...) -> get high output for fake to fool D
- Can get stuck at early stages of training,
∴ use **Modified minimax loss:** $\max \log D(G(z))$ instead of $\min(1 - D(G(z)))$

**Deep Conv GANs:** Uses leaky ReLu (-ve slope before x = 0), sin ce ReLU is non-0 centered, non diff at 0 & dying problem

**Conditional GANs:** same feed label $Y$ **(paired data)**
Used for object transfiguration
**a) Pix2pix(cGAN** with original img as condition)
- Paired data, D compares $G(x)$ and $y$ given $x$ aka generate img in different style from original img
- To make img less blurry use L1/L2

$G\star = arg\min_G \max_D L_{cGAN}(G, D) + \lambda L_{L1}(G)$

**Cycle GAN (Unpaired data):** Domain transformation
- Uses adversarial loss + inverse mapping (sin ce mapping highly constrained). Introduces a cycle consistency loss to push $F(G(x)) \approx X$
- Not just img, e.g. waveform, but not spectrogram
- Used to convert accented speech