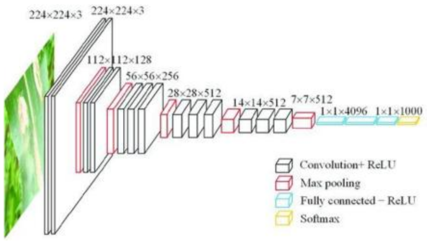
Transfer Learning

Problem Framing:

1. **Classify Rare Water Animals:**
   - We are task with building a classifier for rare underwater animals.
   - The dataset includes only a few hundred images for ten classes (species), which is problematic because:
     - Data is scarce and hard to acquire.
     - Training a deep model from scratch would lead to overfitting.

2. **Naive Solution Training a CNN on Limited data:**
   - Directly training a CNN (e.g., VGG16) from scratch on a small, rare dataset of underwater animal images. Why this fails:
     - Too many parameters: VGG16 has ~134 million parameters. The fully connected layer alone has 4096 neurons.
     - Too few data samples: With only a few hundred training images, the model quickly overfits — meaning it memorizes the training set but performs poorly on unseen images.
   - Thus, even tricks like data augmentation (e.g., rotation, flipping) won't produce new semantically meaningful information, just more of the same distribution, and can't combat the severe data scarcity.



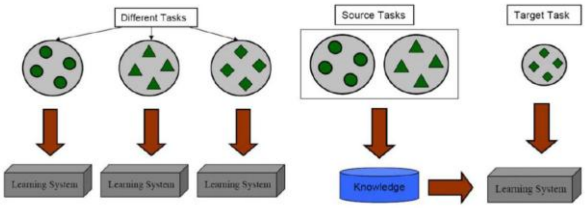
1. **Potential Outcome of Naive Training:**
   - High parameter count (134M) + low data availability = poor generalization.
   - You can't expect to train a model like VGG16 from scratch unless you have a massive dataset.
   - It's very complicated to tune 134 million parameters using hundreds of samples... so, we need something smarter.

Introduction To Transfer Learning

1. **A Smarter Strategy:** reusing a model trained on a large dataset and adapting it to your task.
   - Key Idea: Take the knowledge (weights of the neural network) acquired from one task and reuse it for another.
   - Use a CNN pretrained on ImageNet (a large dataset with 1M+ images and 1000 classes).
   - Either freeze the early layers and train a new classifier (feature extraction), or fine-tune all or part of the model.

2. **Formal Definition of Transfer Learning:**
   - Domain: $\mathcal{D} = \{X, P(X)\}$:
     - X: Feature space.
     - $P(X)$: Marginal probability of distribution over those features.
   - Task:$\mathcal{T} = \{Y, f(\cdot)\}$:
     - Y: Label space (digits).
     - $f(\cdot)$: predictive function learned from training data not directly observed.
   - A transfer learning scenario involves:
     - A source domain/task : $(\mathcal{D}_S, \mathcal{T}_S)$.
     - A target domain/task: $(\mathcal{D}_T, \mathcal{T}_T)$.
   - Different combinations of similarities/differences among domains and tasks lead to different transfer learning settings:
     - Same domain, different task: MNIST used for digit classification vs digit segmentation.
     - Different domain, same task: X-ray vs MRI images for lung cancer detection image looks different but the task (classifying cancer is same).
     - Both different: Most complex, transfer from English sentiment classification to French sarcasm detection.

3. **Transfer learning Vs. Traditional ML:**
    - Traditional ML: Tasks are learned independently. Each model is trained from scratch.
    - Transfer Learning: you learn from a source task and carry over the knowledge to a target task.
    - The knowledge refers to the pretrained weights.



(a) Traditional Machine Learning (b) Transfer Learning

Transfer Learning Scenarios

1. **Different Feature Space**

$$X_S \neq X_T$$

- The features space (the type and structure of input data) is different between source and target.
- Example:
    - Source: Document A in English.
    - Target: Document B in French.
    - Event though both documents deal with similar contents, the language (feature representation) is different.
- Task:
    - Train a model to recognize phonemes or linguistic patterns in one language and apply it to another language.
    - This is a form of cross-lingual transfer, common in NLP tasks.
- Another Example from professor: Imagine tissue images acquired with different devices—one being optical microscopy, the other atomic force microscopy. Though they represent the same tissue, the feature spaces are fundamentally different.

2. **Same Feature Space, Different Distributions**

$$P(X_S) \neq P(X_T)$$

- The input data is of the same type, but its distribution differs between source and target. This is often referred as domain shift.
- Examples:
    - MRI vs CT scans of lungs: both produce medical images, but the texture, resolution, and visual patterns vary.
    - MNIST vs SVHN: both are digits datasets, but MNIST is grayscale handwritten digits, while SVHN colored street house numbers.
- Task: Use a model trained on one distribution (MRI) to classify on another (CT scans).
- Links to domain adaptation: The images might look the same dimensionally, but $P(x)$ (how features are distributed) differs. That's why CT and MRI may require different visual understanding, even though the pixels are both RGB.

3. **Different Label Space**

$$Y_S \neq Y_T$$

- The tasks are different, even if domain remain the same. We're asking the model to learn something new.
- Examples:
    - ImageNet with 1000 natural categories.
    - Target: A new dataset with farm animals.
- Task: we're using the visual knowledge of ImageNet but teaching the model to classify new labels.

4. **Different Conditional Distribution**

$$P(\, Y_S \mid X_S \,) \neq P(\, Y_T \mid X_T \,)$$

- The input-output relationship changes. The same input might correspond to different output probabilities, often due to the class imbalance or concept drift.
- Use a source-trained model, but expect its internal class likelihoods will not generalize correctly due to shifts in $P(Y|X)$.
- This is subtle but critical. We can have the same feature space and the same label space, but with different frequency of class appearances differs, our model can fail to generalize.

Key Aspects of Transfer Learning

Transfer Learning is not a one-size-fit-all approach. Successful applications depend on three interconnected questions, what, when, and how to transfer.

1. **What to Transfer?**
   - Determines which component of the learned knowledge from the source task are generalizable and valuable for target task:
     - Low-level features: edges and textures leaned from early layers of CNNs.
     - Mid-level representation: Object shapes or parts.
     - Weights of pretrained models.
   - The knowledge, very basically, is just weights of neural network which has been trained on something; and in some ways, we exploit these weights to get more information on another task.
   - Understanding what is domain/task specific (final layers trained on a specific labels) versus domain-invariant (edge detectors or generic patterns) is central to deciding what to use.

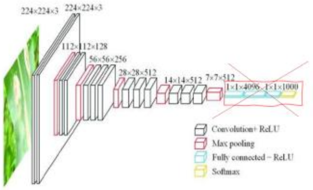
2. **When to transfer:**
   - Decide whether transfer learning will help or potentially hurt the model's performance. This is essential to avoid negative transfer, where transferring knowledge degrades the performance.
   - Situations to consider:
     - Data Scarcity: if the target task how few labeled examples.
     - Domain Similarity: if source and target domains are similar, transfer is more likely to succeed.
     - Label Space Overlap: Helps guide whether feature reuse or full fine-tuning is appropriate.
   - The "When" isn't just about data size, it's also about data quality, distribution, and representational alignment.

3. **How to Transfer:**
   - Choose the appropriate method or strategy to transfer knowledge from source to target. This involves algorithmic design and practical decisions such as:
     - Feature Extraction: Freeze the pretrained model and use it as feature encoder.
     - Fine-tuning: Retrain some or all layers of the pretrained model on the new dataset.
     - Partial Transfer: Fine-tune only the later layers.
     - Knowledge Distillation: Transfer knowledge from a complex teacher model to a simpler student model.
   - For fine-tuning we should use a small learning rate otherwise, we may erase previous learned representations.

Transfer Learning Strategies - CNN codes

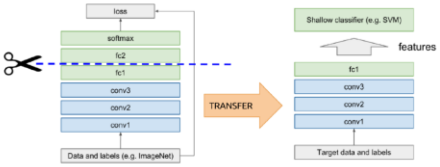
1. **Why not Train From Scratch?**
   - Training a CNN from scratch is impractical due to:
     - Lack of annotated data.
     - Computational Cost.
     - Rist of Overfitting on small dataset.
   - Hence, the community standard is to rely on pretrained CNNs (typically trained on large datasets like ImageNet with 1.2M images across 1000 categories).

2. **Strategy 1: ConvNet as Fixed Feature Extractor:**



   - Pretrained Model: start with a CNN trained on ImageNet.
   - Remove the final layer: discard the final fully connected (FC) and softmax layers designed to predict 1000 classes.
   - Use intermediate Activations: Use the output of the last hidden layer as a feature representation for your new images.
   - Train a new Classfier: on the top of these extracted features, train a simple classifier (logistic regression or a new softmax) for your target classes.
   - The idea is to extract generic features and only train a classifier on top, especially when your dataset is small and similar to ImageNet, this avoid overfitting.

3. **Strategy 2: Fine-tuning the network:**



   - Rather than freezing the pre-trained layers, you allow (some of) them to be updated via backpropagation during the training on the target dataset.
   - Two main configurations:
     - Partial fine-tuning: Only freeze last few layers to adapt high-level features to the new task.
     - Full fine-tuning: Unfreeze and retrain the entire network.
   - Why not always fine-tune everything?
     - Because with small datasets, full fine-tuning may lead to catastrophic forgetting and overfitting.

4. **Four practical fine-tuning Scenarios**

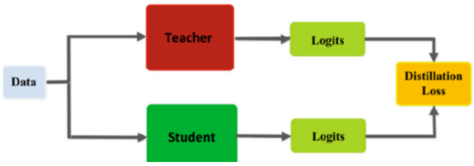
Target Dataset	**Similarity to Source**	**Recommended Strategy**	**Remarks**
Small	Similar	Feature extraction	Avoid fine-tuning to prevent overfitting
Large	Similar	Fine-tuning	Can fine-tune whole or partial layers
Small	Dissimilar	Partial Feature extraction (mid-layers)	Use early/mid features, freeze rest
Large	Dissimilar	Fine-tuning or train from scratch	Start from pretrained, then retrain

   - If you fine-tune with too high a learning rate, you risk *catastrophic forgetting* — overwriting good weights.

Knowledge Distillation

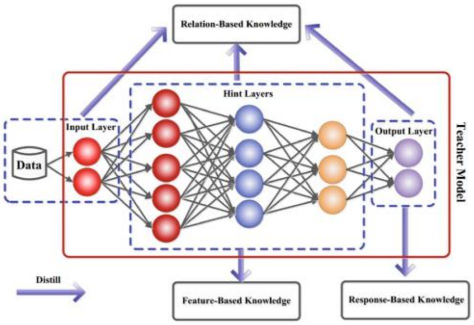
1. **What is Knowledge Distillation KD:** KD is a model compression method, but it's also a form of transfer learning.

- A large model (teacher), trained on a task, passes its knowledge to a smaller model (student).
- The student is trained not just on raw label, but to mimic the teacher's behavior, typically using the soft class probabilities (logits).
- How it works:
  - Feed the same input data to both teacher and student.
  - Compare the logits of teacher and student.
  - Minimize the KL divergence between them.
- This forces the student to approximate the same decision boundaries as the teacher.
- It's not just about compressing; it's also about helping the smaller model generalize better by following the teacher's output distribution.



2. **Types of Distillation Knowledge**

- Response-based Knowledge:
  - Refers to soft outputs (logits or probabilities) of the teacher.
  - This is the most common type, in classic KD setups.
- Feature-based Knowledge:
  - Use intermediate representation (feature maps) from inside the teacher model.
  - The student is encouraged to replicate these hidden-layer outputs, not just the logits.
- Relation-based Knowledge:
  - Captures the structural relationship between activities (similarity between pairs of instance).
  - Enables the student to understand not just what teacher knows, but how it relates instances internally.



Self-Supervised Learning

SSL Prelude

1. **Motivation: Deep Learning and Supervised Learning are Powerful Tools.**

- Faster R-CNN enables object detection by identifying multiple classes (e.g., cars, people) in real-world scenes.
- ResNet revolutionized image classification by introducing residual connections that mitigate vanishing gradients, enabling deeper networks.
- Mask R-CNN extends object detection to instance segmentation, producing pixel-wise masks.
- Human Pose Estimation captures spatial joint structures, supporting sports analytics or AR.
- Visual Question Answering demonstrates multi-modal learning, where the model understands images and answers natural language queries.

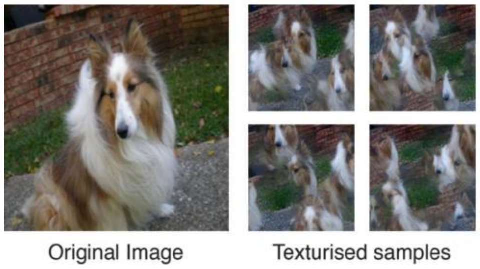
2. **Typical Supervised Learning Framework**

- Begin with a predefined set of classes (e.g., airplane, cat, dog, etc.).
- Collect a large and diverse dataset for each class.
- Feed this dataset into a CNN.
- Training is then performed to minimize classification error. The model learns to extract hierarchical features, progressing from low-level edges to high-level semantic concepts.
- However, this process assumes the availability of large labeled datasets, which is often not feasible.

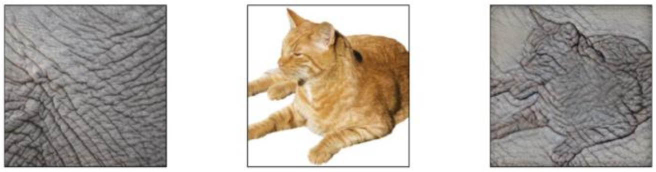
3. **Real-World Constraints: Cons of Supervised Learning**
   - Constructing datasets like ImageNet is labor-intensive, requiring:
     - Annotation.
     - Data cleaning.
     - Quality Control.
   - It’s time-consuming, expensive, and prone to human error.
   - Beyond effort and cost, there's an even deeper problem: data distribution shifts.
     - Fashion trends from the 1980s exemplify this — the world is dynamic, and models trained on past data may become obsolete.
     - Launching frequent large-scale annotation campaigns to keep up is infeasible.
   - Supervised learning models lack adaptability to such non-stationary environments.
   - This motivates the search for label-free alternatives that can evolve with data.

4. **Alternative: Exploiting Raw Data**
   - Raw, unlabeled images are abundant and easy to collect.
   - However, supervised models can’t make use of them without labels.
   - This disconnect paves the way for self-supervised learning, which can leverage such data.
   - Self-supervised approaches aim to extract useful features from raw data by creating pretext tasks—artificial learning objectives that don't require manual labels.

5. **Labeling Bias and Its Consequences:** another issue is bias in labeled datasets, especially due to over-reliance on texture rather than the shape:
   - The experiment shows how VGG-16, trained on ImageNet, misclassifies textured samples.
   - An original image of a collie is altered through texture-based transformations.
   - Despite retraining structural information, prediction changes drastically.
   - This reveals that supervised models may not generalize well outside the training distribution, specially if they latch onto superficial cues like texture.



6. **Texture-shape Cue Conflict:**
   - A patch of elephant skin texture is classified as an “Indian elephant” with high confidence.
   - A clean image of a tabby cat is correctly classified.
   - But when the cat’s texture is transformed with elephant skin, the model once again predicts Indian elephant.
   - Texture-shape cue conflict illustrates that CNNs prioritize local texture over global shapes; a limitation of current supervised training models. It also highlights the lack of robust feature representation; models don’t always see objects like human do.



(a) Texture image
81.4% **Indian elephant**
10.3% indri
8.2% black swan

(b) Content image
71.1% **tabby cat**
17.3% grey fox
3.3% Siamese cat

(c) Texture-shape cue conflict
63.9% **Indian elephant**
26.4% indri
9.6% black swan

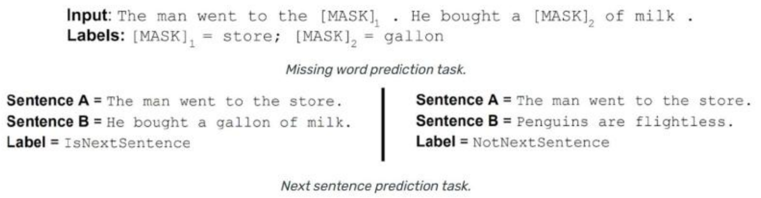
SSL General Concept

1. **What is SSL:**
   - SSL is presented as a subset of unsupervised learning; the key distinction being that the data itself generates the supervisory signal.
   - Instead of relying on human-labeled examples, SSL uses pretext tasks, designed to force the model to learn meaningful features.

2. **Key elements:**
   - Supervised without labels: the network learns to solve a synthetic task generated from raw data.
   - Pretext task: A surrogate task that doesn't reflect the final goal but encourages the model to learn useful representations.
   - Down Stream Task: After pretext task, the learned features are transferred and fine-tuned.
   - The only important thing for pretext task is that it forces the network to learn visual features.

3. **Main technique behind SSL:**
   - Hide part of the data (crop a patch, remove a word, mask a region).
   - Ask the network to predict it based on the visible portion.
   - By solving this auxiliary task, the model is essential learning to understand structure and semantics.
   - The goal is to define a task which in order to be solved, requires semantic understanding of the input.

4. **Example from NLP (word2vec):**
   - Mask word prediction: the model predicts the missing word in a sentences. This forces understanding of syntactic and semantic context.
   - Next sentence prediction: the model decides if one sentences logically follows another, which requires understanding of meaning and sequence.



Input: The man went to the [MASK]$_1$. He bought a [MASK]$_2$ of milk .
Labels: [MASK]$_1$ = store; [MASK]$_2$ = gallon

Missing word prediction task.

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

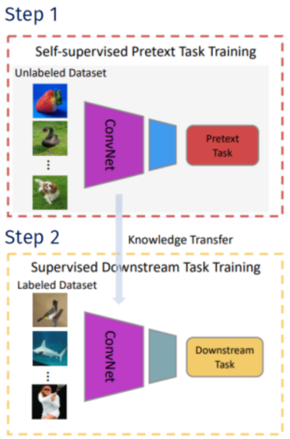
Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence

Next sentence prediction task.

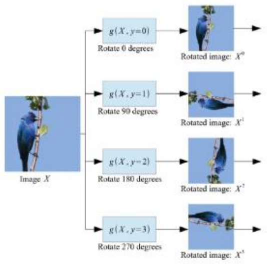
   - The missing word, or the sentence becomes a label; but a label automatically generate from structure.

5. **What exactly is SSL?**
   - The signal for learning in SSL comes internally from the data.
   - The network isn't just making a random prediction it's solving a semantically meaningful task.
   - Even if we choose a task which is very different, it will help us with our downstream task as long as it forces the model to learn useful features.

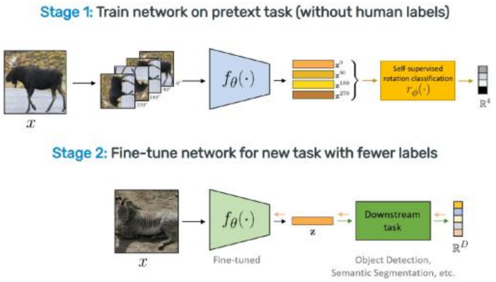
6. **SSL Pipeline:**
    - Pretext Task Training:
        - Uses Unlabeled data.
        - The network is trained to solve a pretext task.
        - No human supervision is involved.
        - The goal is feature extraction.
    - Downstream Task Fine-tuning:
        - A smaller labeled dataset is used.
        - Either the encoder is fine-tuned, or its features are frozen and passed to a new head.
        - The downstream task benefits from the generalized features learn in Pretext training.
    - You train on a large unlabeled set, then throw away the task head and retain the encoder for your real task.



7. **Example Pretext Task – Rotation Prediction:**
    - Take an image and rotate it by {0°, 90°, 180°, 270°}.
    - The model's job is to predict rotation angle.
    - The task works because solving it helps model to generalize and understanding global features of the image.
    - The rotation itself is the label; but a synthetic one. This is not data augmentation. In augmentation label is fixed. Here, the label changes with transformation.



8. **Alternate view of SSL Pipeline:**
    - Stage 1: the model is trained to predict the dotation angle of various augmented views.
    - Stage 2: the learn decoder f_θ is then fine-tuned for real-world tasks like classification.
    - Requires no labels for pretraining.
    - Use a small label set for final adaptation.
    - Offers a path to reduce annotation costs while still maintaining performance.
    - You pretrain a good f, through away the pretext classifier, and fine-tune f on the downstream task. It's very similar to transfer learning, except the pretraining doesn't need labels.

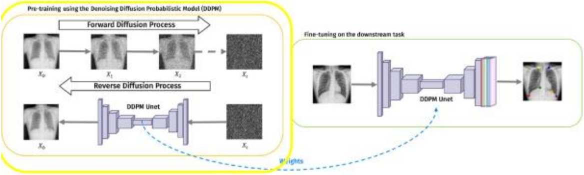

Stage 1: Train network on pretext task (without human labels)
x
$f_\theta(\cdot)$
z^0
z^{90}
z^{180}
z^{270}
Self-supervised rotation classification $r_\phi(\cdot)$
$\mathbb{R}^4$
Stage 2: Fine-tune network for new task with fewer labels
x
$f_\theta(\cdot)$
Fine-tuned
z
Downstream task
Object Detection, Semantic Segmentation, etc.
$\mathbb{R}^D$

Generation-Based Pretext Tasks

1. **Autoencoder (AE):**
    - In SSL, we discard the decoder after training. The encoder alone (the learned feature extractor) is reused in the downstream task.

2. **Generative Adversarial Network (GAN):**
    - The discriminator learns rich visual features to distinguish fake; this feature extractor (or parts of it) can be repurposed in downstream tasks.
    - You can take the discriminator encoder; it must learn strong visual features to tell fake from real. Then use it for the actual Task.

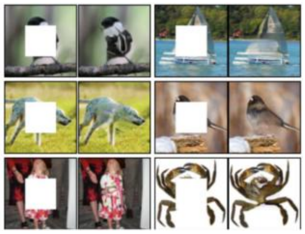
3. **Diffusion Models:**
    - A model (U-Net) learns robust features during this process and can be fine-tuned for downstream task like landmark direction.
    - We use diffusions to reconstruct X-ray images, then transfer the encoder to detect anatomical landmarks with few labels.


Pre-training using the Denoising Diffusion Probabilistic Model (DDPM)
Forward Diffusion Process
X_0
X_1
X_2
X_t
Reverse Diffusion Process
DDPM Unet
X_0
X_t
Weights
Fine-tuning on the downstream task
DDPM Unet

4. **Image Colorization:**
   - The model receives a grayscale image and must predict its color channels.
   - This task forces the model to learn high-level understanding (grass is green, sky is blue, …).
   - Color distributions (from ImageNet) are used to re-weight loss contributions, since some colors are rare.
   - This forces the model to capture semantics, object boundaries, and context; making it a rich supervisory signal.

5. **GAN Super Resolution:**
   - The generator takes a low-resolution image and reconstructs a high-resolution version.
   - The discriminator evaluates image realism.
   - This setup forces the model to learn textures, edges, and fine structures.

6. **GAN Inpainting:**
   - A model fills missing patches in an image.
   - As shown visually, the model reconstructs occluded areas; requiring it to understand context, continuity, and semantics.
   - They are perfect pretext task by trying to restore or generate, the model learns useful visual features.



7. **Video:**
   - VideoGAN (Temporal Continuity): generates video sequences from noise.
   - Video colorization (Object tracking): predict color across frames.
   - Feature frame prediction (motion dynamics): given a set of past frames, generate the next one.

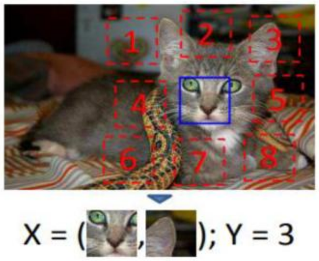
8. **Why generation-based pretext tasks work:**
   - They create supervision signals from data itself, by framing generation or restoration as a task.
   - They force the model to learn meaningful representations without manual labels.
   - The learned representation (encoder) can be reused in the downstream tasks like classification, detection, and segmentation.

Pretext Task	What It Learns	Downstream Utility
Autoencoder	Data compression & reconstruction	General feature extractor
GAN (Discriminator)	Visual realism, texture, structure	Object recognition, classification
Diffusion Models	Progressive denoising, image structure	Medical imaging, few-shot learning
Colorization	Semantic color understanding	Object and scene understanding
Super Resolution	Detail recovery, fine structure	High-resolution classification, editing
Inpainting	Spatial context, semantic filling	Object boundaries, segmentation
Video Prediction	Motion, continuity, causality	Action recognition, tracking

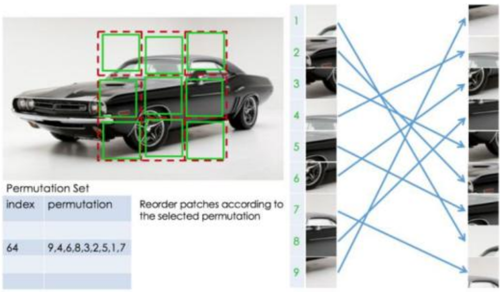
Context-Based Pretext Task

1. **Geometric Transformations: Image Rotation Prediction:**
   - The Idea: randomly rotate an image by 0°, 90°, 180°, or 270°, and train the model to predict the rotation angle.
   - Why it works: to correctly identify the rotation, the network must understand the semantic orientation of objects(birds stand upright).
   - Architecturally, it becomes a 4-class classification task with synthetic labels (0–3)
   - Even though rotation is unrelated to down-stream task, it teaches the network to extract meaningful visual features, like object boundaries, shapes, and orientation.

2. **Relative Patch Position Prediction:**
   - Given a center patch, the network is shown a second path and asked to predict its relative position (top-left, right, bottom).
   - A CNN is used to extract features from both patches, which are passed to a fully connected classifier that predicts one of the 8 relative positions.
   - In order for the model to solve the task, it must learn to understand spatial layout and object consistency (cat eyes are above noise).
   - It captures mid-level features like symmetry, pose, and part-whole relationship.



3. **Jigsaw Puzzle Solving:**
   - An image is divided into 9 tile and permuted.
   - To model is trained to predict the correct permutation index from a fixed set of arrangements.
   - Model must learn object geometry and spatial dependencies.
   - Encourages global scene understanding, which is critical for detection and segmentation.
   - We want the model to learn 'visual common sense'—understanding that wheels belong under cars, heads above shoulders, etc.



4. **Frame Order Prediction (Video Context):**
   - Frame Order recognition:
     - Randomly shuffle frames from a video and train a model to predict their correct order (identifying right permutation).
     - Helps model temporal dynamics and causality.
   - Frame Order Verification:
     - Simpler task: given a sequence, decide if it's the correct or incorrect order.
     - Lexx complex but still forces a temporal understanding.

5. **Arrow of Time:**
   - Given a sequence of frames, the model predicts if the video is played forward or backward.
   - Solving this requires understanding motion consistency (how objects move, fall, and interact).
   - This taps both low level physics and high-level reasoning.

6. **Why Context-based Pretext task work:**
   - Goal: Teach models "visual common sense" via spatial/temporal tasks (rotation, jigsaw, ordering).
   - These tasks demand semantic understanding even without labels.
   - We care less about pretext performance and more about the transferability of learned features.
   - This insight motivates the shift to contrastive methods, which learn from similarity/dissimilarity and tend to yield more domain-agnostic features.

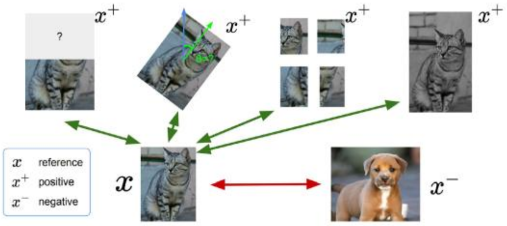
Pretext Task	What It Learns	Challenge Type
Rotation Prediction	Global object orientation	Geometric
Relative Patch Prediction	Local spatial relationships	Spatial context
Jigsaw Puzzle	Part-whole reasoning, object structure	Spatial rearrangement
Frame Order Prediction/Verification	Temporal coherence, motion understanding	Temporal ordering
Arrow of Time	Motion causality, direction detection	Temporal reasoning

Contrastive Learning

1. **Limitations of Pretext tasks from Image Transformation:**
   - The features are often task-specific.
   - Rotation focuses on low-level cues like edges.
   - Representation may not generalize well to the downstream task.
   - We have to engineer every pretext task. But worse; each task might only capture one aspect of visual structure.

2. **Towards Generality: what is the core issue?**
   - The problem is that multiple pretext views of the same object are actually different augmentations of the same semantic entity. Yet these tasks treat them as separate prediction problems.
   - Instead of predicting something task-specific, why not bring together different view of the same object.

3. **The Shift: from task specific to general Representation Learning**
   - All transformation of the same object (rotated, colored, masked) are positive pairs.
   - Images of different objects (cat vs. dog) form negative pairs.
   - So instead of solving a fixed puzzle, the model goal's becomes:
     - Attract embeddings of similar images.
     - Repel embeddings of dissimilar images.



4. **Self-Supervised Contrastive Learning:**
   - x: anchor image.
   - x^+: positive sample (an augmented version of the same image).
   - x^-: negative sample (different image).
   - Objective: learn an encoder $f(.)$ such that:

$$\text{score}\big(f(x).\,f(x^+)\big) \gg \text{score}\big(f(x).\,f(x^-)\big)$$

   - This scoring function (often cosine similarity) evaluate the closeness in representation space.
   - This structure allows the model to learn without labels, by relaying on data augmentations to create new positive examples.

5. **Contrastive Learning:**
   - Definition: is a self-supervised learning method, where the model learns the features based on the differences and similarities between the data points.
     - It's like telling the model:
       - This is the different view of the same object – learn to treat them as the same.
       - This is a different object – learn to treat them as different.
     - we use contrastive learning to generalize the features and make them task-agnostic.

6. **How contrastive learning work:**
   - Start with an image
   - Apply two different random transformation (cropping and coloring) to get two different views.
   - These two views are called positive pairs.
   - Other images in the batch are called negative pairs.
   - The model learns to pull together the representation of positive pairs and push apart negative pairs.
   - This is done using contrastive loss, often InfoNCE loss.

7. **InfoNCE: Contrastive Loss Function:**

$$L = -E_X\left[\log \frac{\exp\Big(s\big(f(x), f(x^+)\big)\Big)}{\exp\Big(s\big(f(x), f(x^+)\big)\Big) + \sum_{j=1}^{N-1} \exp\Big(s\Big(f(x), f\big(x_j^-\big)\Big)\Big)}\right]$$

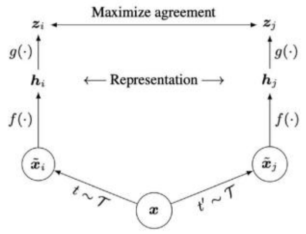
   - x : Anchor sample (original image or input)
   - x^+: Positive sample (augmented view of the same image as x)

- x_j^-: j-th negative sample (augmented view of a different image)
- $f(.)$: encoder network mapping an input to a latent vector (representation)
- $s(u, v)$: Similarity score between vector u and v.
- $exp(.)$ exponential function
- log: Natural logarithm
- E_X: Expectation over all anchor samples x in a batch.
- N: number of total samples (1 positive and N-1 negative)
- For each anchor x we want the similarity between $f(x)$ and $f(x^+)$ to be much higher than the similarity between $f(x)$ and $f\left(x_j^-\right)$ for all the negative samples.
- This is essentially a softmax loss, where:
  - The numerator is the similarity with the positive.
  - The denominator is the sum of similarities with all the samples (positive and negative).

SimCLR

1. **Objective and Framework Overview**
   - SimCLR is a self-supervised learning framework designed to learn meaningful representations of data without requiring manual annotations.
   - The key idea is to maximize the agreement between augmented views of the same data sample and repel other samples using contrastive learning loss function.

2. **Core Components:**
   - Data Augmentation $t \sim \mathcal{T}$: each image is augmented twice to produce a positive pair. These augmentations include:
     - Random cropping
     - Color distortion
     - Gaussian blur
   - Based encoder $f(.)$: A deep CNN (e.g., ResNet) maps the augmented images into a representation space, yielding feature vectors$h_i = f(x_i)$
   - Projection Head $g(.)$: A small MLP applied on the top of the encoder to map feature representations to the contrastive space. This improves contrastive performance.



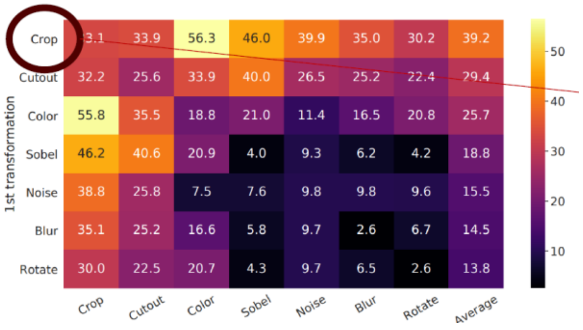
3. **Visualize the pipeline:**
   - A sample x undergoes two different augmentations $t, t' \rightarrow \widetilde{x_i}, \widetilde{x_j}$.
   - Passed through encoder $f(.) \rightarrow h_i, h_j$.
   - Then through projection head $g(.) \rightarrow z_i, z_j$.
   - The model maximizes agreement between z_i, z_j.

4. **Similarity Metric and Contrastive Loss (InfoNCE):**
   - Cosine Similarity $s(u, v) = \frac{u^\top v}{\tau|u||v|}$: Measures the directional similarity between two projected vectors.
   - SimCLR employs the InfoNCE loss, where the goal is to classify the positive pair (z_i, z_j).

$$\mathrm{l}_{i,j} = -\log \frac{\exp\left(\mathrm{sim}\left(z_i, z_j\right)/\tau\right)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\mathrm{sim}(z_i, z_k)/\tau)}$$

   - Here, τ is a temperature parameter that controls the sharpness of the distribution.
   - the denominator represents negative examples, which are all other augmented samples in the batch. Therefore, a large batch size is essential to provide many negative samples and a tighter mutual information bound.
   - Small batch size weakens the performance due to a poor approximation of the negative sample space.

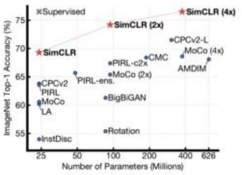
5. **Data Augmentations' Heatmap:**
   - Augmentation Heatmap shows different combinations and their impact on representation quality.
   - The conclusion is that no single augmentation is suffice, but composition(crop+color distortion) significantly improves contrastive prediction performance.
   - Why: since combining two different augmentation helps for a more generalize representation since some of the augmentations (e.g. cropping) help with local feature extraction while some other ones (e.g. color distortion) exploit global feature extraction.



1st transformation	Crop	Cutout	Color	Sobel	Noise	Blur	Rotate	Average
Crop	3.1	33.9	56.3	46.0	39.9	35.0	30.2	39.2
Cutout	32.2	25.6	33.9	40.0	26.5	25.2	22.4	29.4
Color	55.8	35.5	18.8	21.0	11.4	16.5	20.8	25.7
Sobel	46.2	40.6	20.9	4.0	9.3	6.2	4.2	18.8
Noise	38.8	25.8	7.5	7.6	9.8	9.8	9.6	15.5
Blur	35.1	25.2	16.6	5.8	9.7	2.6	6.7	14.5
Rotate	30.0	22.5	20.7	4.3	9.7	6.5	2.6	13.8

6. **Algorithm Details:**
   - For each image x_k, two augmentations are applied, producing a positive pair $(\widetilde{x_{2k-1}}, \widetilde{x_{2k}})$
   - Each is passed through f and g to obtain representation z, which are then used to compute the loss.
   - The final loss is averaged over both $l(2k-1,2k)$ and $l(2k,2k-1)$, meaning each positive pair is considered in both directions, doubling the effective supervision.
   - This bidirectional consideration enhances stability and symmetry, and that the network is trained using gradient descent on this loss.

7. **Limitations and Extensions:**
   - Dependence on large batch size: more negative helps bound the mutual information tighter, but computational cost rise.
   - No explicit alignment for semantically similar but different images (e.g., two different cats). All non-augmented pairs are treated as negatives, which might not reflect semantic closeness.



8. **Performance and Impact:**
   - SimCLR significantly improves ImageNet Top-1 accuracy, and with larger models (e.g., SimCLR 4x), it even surpasses some supervised baselines.

Momentum Contrast (MoCo)

1. **Core Intuition: Dictionary Look-up Framing**
    - MoCo reframes contrastive learning as a dictionary look-up task
    - We have an encoded query vector q and a dictionary of encoded keys $\{k_0, k_1, \dots, k_K\}$.
    - Among these, only one have k^+ corresponds to a positive sample (it is an augmented view of the same image as the query), while the rest acts as negatives.
    - The goal is to make q similar to k^+ and dissimilar to all k^-.

$$\mathcal{L}_q = -\log \frac{\exp(q \cdot k^+/\tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)}$$

    - τ is the temperature parameter to sharpen or soften the distribution.
    - Unlike SimCLR, which requires many negatives per batch, MoCo introduces a key idea to decouple dictionary size from batch size.

2. **Key Architecture Innovation: Queue and Momentum Encoder**
    - MoCo maintains a dynamic dictionary (queue) of negative keys.
    - Each mini-batch's encoded features are enqueued into the dictionary.
    - Simultaneously, the oldest entries are dequeued, maintaining a fixed-size FIFO queue.
    - MoCo uses a momentum encoder, a slowly evolving version of the query encoder.
    - The encoder produces the keys of dictionary, but its parameters are not updated via backpropagation.
    - Instead, they are updated using exponential moving average:

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q$$

3. **Why the Momentum Encoder?**
    - If we allowed full backpropagation though the queue, updating the key encoder would be computationally infeasible, since the dictionary can grow large.
    - Thus by freezing gradients and using momentum updates, MoCo ensures that:
        - The keys remain consistent and stable.
        - The dictionary can span a rich and diverse set of negatives, improving representation learning.
    - This design choice lets MoCo support many more negative samples than what's feasible in SimCLR, even with small mini-batch sizes.

4. **Practical Comparison to SimCLR**

Component	SimCLR	MoCo
Negative Samples	From within the same batch	From a **FIFO queue** across batches
Gradient Flow	Through entire architecture	**No gradient to momentum encoder**
Batch Size Requirement	Needs large batch sizes	Works well with small batches
Encoder Synchronization	Single encoder for all views	Uses both **query** and **momentum encoders**

- This design decouples negative pool size from batch size, solving one of SimCLR's biggest limitations.

5. **MoCo: Momentum contrast :** MoCo solves the SimCLR problem with large batch size requirements.
    - Uses a dictionary of keys (embedding of negative samples) maintained across batches.
    - Introduces two encoders:
        - Query encoder f_q: learns normally via backpropagation
        - Key encoder f_k: updates slowly via momentum

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q$$

        - θ_k: parameter of key encoder
        - θ_q: parameter of query encoder
        - m: momentum coefficient
6. **MoCo Training:**
    - Augmentation: two augmented versions of each x are generated x_q and x_k.
    - Encoding:
        - $x_q \rightarrow q = f_q(x_q)$: processed by the main encoder with backprop.
        - $x_k \rightarrow k = f_k(x_k)$: encoded by the momentum encoder (detached from gradient).
    - Similarity Computation:
        - Positive pair: Similarity between q and k.
        - Negative pairs: similarities between q and all entries in the queue.
    - Loss: Contrastive loss calculated using positive and negative logits.
    - Backpropagation: Smooth update of momentum encoder.
    - Queue Maintenance: Append current keys, drop oldest.
    - This allows MoCo to build long term memory of past negatives while keeping computation tractable.

7. **Theoretical Motivation and Practical Advantages**
    - MoCo's large, stable dictionary improves the mutual information bound of InfoNCE loss by effectively simulating more negatives.
    - This enhances contrastive discrimination and improves downstream generalization — especially when labeled data is scarce.

Domain Adaptation

Domain Adaptation Prelude

1. **"Is There a Bird?" – Domain Shift and Perceptual Change**
    - These images illustrate a domain shift in the visual context. Even though both domains (photos and sketches) contain the same semantic category (birds), their visual appearances differ drastically — color, texture, detail level, and overall representation.
    - A classifier trained on photos (source domain) will likely perform poorly on sketches (target domain) because visual distribution changes, even though the label space is the same.
    - Domain adaptation is needed when appearance changes prevent standard models from generalizing.

2. **Real-World Example – COVID-19 Ultrasound Classification**
    - Problem: The same label/task is used (presence of COVID markers), but sensor differences and imaging protocols across hospitals cause a domain shift.
    - Observation: Accuracy is ≥85% when train/test data come from the same hospital, but performance drops up to 20% when using data from a different hospital.
    - Challenge: Annotated data isn't always available for all hospitals, making unsupervised domain adaptation essential.

3. **Domain Shift – Definition and Example**
    - In visual systems, this often means changes in appearance between domains.
    - Example: Training on colored photos and testing on black-and-white sketches of the same categories. The model performs well in the source domain (85%) but poorly in the target domain (55%).
    - Domain adaptation is introduced as a remedy to learn representations invariant to such shifts while maintaining class discriminability.

4. **Why Domain Shift Matters**
    - Using applications like autonomous vehicles and drones
    - Urban environments captured in RGB vs. IR (thermal) imagery represent different modalities.

5. **Domain Shift is Ubiquitous**
    - Time & Environmental Changes: Day vs. night or weather changes.
    - Different Modalities: RGB vs. IR cameras.
    - Synthetic to Real: Simulated environments vs. real-world images.

9. **Dataset Bias and Domain Shift**
    - Different datasets (e.g., Caltech101, SUN, ImageNet) have differing styles, lighting, object positions, and backgrounds, even for the same class like "car."
    - The distribution of features learned on each dataset varies, causing poor transfer across datasets.
    - Domain adaptation methods seek to align these feature distributions to ensure consistent performance.

Notations

1. **Domain and Task:**

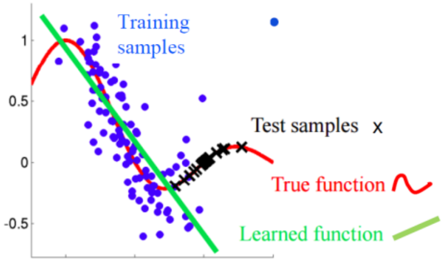
$$D = \{\mathcal{X}, P(X)\}$$

$$T = \{\mathcal{Y}, P(Y \mid X)\}$$

    - $\mathcal{X}$: input space.
    - $\mathcal{Y}$: output space.
    - $X \in x$: a specific input
    - $Y = \mathcal{Y}$: a specific label
    - D: Domain, defined by the input space and its distribution.
    - Task T, defined by the label space and the conditional probability of labels given inputs.
- This notation helps to distinguish between two kinds of changes:
    - Those in the input data distribution $P(X)$.
    - Thos in the conditional label distribution $P(Y|X)$.
- In Domain adaptation we usually assume the task remains the same while the domain changes.

1. **The domain Shift Problem:**
    - $D^s = \{\mathcal{X}^s, P(X^s)\}$
    - $D^t = \{\mathcal{X}^t, P(X^t)\}$
    - $T^s = \{\mathcal{Y}^s, P(Y^s \mid X^s)\}$
    - $T^t = \{\mathcal{Y}^t, P(Y^t \mid X^t)\}$
    - Domain Adaptation Assumption:
        - $D^t \neq D^s$: Domains are different.
        - $T^s = T^t$: The tasks are the same.
    - The input distribution differs $P(D^t) \neq P(D^s)$, possibly due to lighting, background, resolution, or modality changes.
    - possibly due to lighting, background, resolution, or modality changes. $P(Y|X)$.
    - DA is only valid when the task remains the same across domains.
    - If the label space differs, we are dealing with generic transfer learning, not Domain Adaptation.

2. **Covariant Shift Assumption:**
    - The model trained on source data generalizes poorly on the test set due to input distribution shift even though the underlying conditional distribution $P(Y|X)$, is the same.
    - even though the underlying conditional distribution $P(D^t) \neq P(D^s)$ but $P(Y|X)$ remains invariant.
    - If we align the feature representations across source and target (domain invariant embedding) the source-trained model can generalize better.
    - If I cannot discriminate between source features and target features, and if $P(Y|X)$ is the same across domains, then the classifier trained on source should generalize well on target." This is the central intuition of unsupervised domain adaptation.



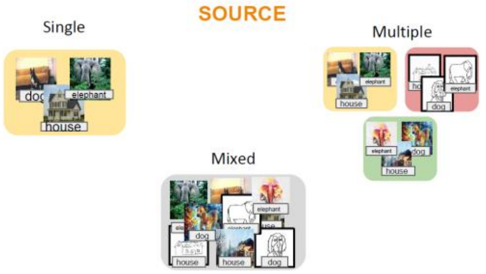
6. **What is domain adaptation:** Domain Adaptation is a special case of Transfer Learning where:
    - The task remains the same across the source and target domain.
    - The domain differs due the changes in sensors, light, style, etc., …
    - The goal is to train a model on the source domain and generalize well on the target model.

7. **Unsupervised Domain Adaptation (UDA):** in this case the target domain data is unlabeled, thus we need to learn domain-invariant features, so that the model trained on the sources generalizes well on the target.

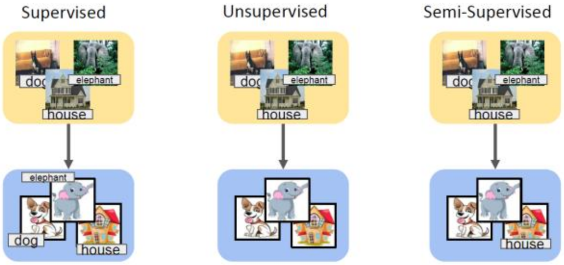
8. **Types of Domain Adaptation:**
    - Discrepancy based method: Minimizes the difference (discrepancy) between feature distribution of the source and target domains
    - Adversarial based: Using domain discriminators to encourage domain confusion through an adversarial objective.
    - Reconstruction based: using data reconstruction as an auxiliary task to ensure feature invariance.

Domain Adaptation Scenarios

1. Number of Source Domains:
    - Single Source DA:
        - One label source domain is used.
        - Classic Scenario.
    - Multi-Source DA:
        - Multiple labeled source domains are available.
        - These domains might vary by sensor, style, background.
        - Multi-source DA is important because it encourages the model to learn more generalized features by combining diverse but related domains.
    - Mixed Source DA:
        - All domains are merged into one dataset without domain tags.
        - Assumes a shared representation without needing domain-specific adaptation modules.

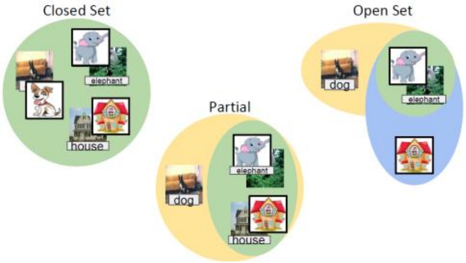


2. **Target Labels:**



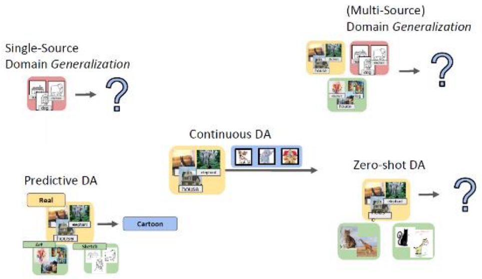
   - Supervised DA:
     - Full labels are available in both source and target.
     - This isn't common in real-world DA use cases; it becomes a multi-domain supervised learning problem rather than true DA.
   - Unsupervised DA:
     - Most common in practice.
     - Source domain is labeled; target domain is unlabeled.
     - Goal: transfer knowledge by learning domain-invariant representations.
     - This is the focus of most of the lecture and methods like DANN, MMD, etc.
   - Semi-Supervised DA:
     - A small fraction of target domain samples is labeled.
     - This helps guide the alignment between domains with a few anchor points.
     - pseudo-labeling techniques can be thought of as approximating this setup dynamically (by labeling the target domain in a self-supervised loop).
3. **Unsupervised DA:**
   - Source domain: labeled real-world images of "dog," "house," and "elephant."
   - Target domain: unlabeled cartoon versions of the same categories.
   - Unsupervised DA means you don't have target labels, but you assume the tasks are the same. The main challenge is that $P(X)$ shifts while $P(Y|X)$ is stable. So all the adaptation method focus on aligning feature space across domains.

4. **The Label Space:**



   - Closed Set DA:
     - Label sets are identical across source and target.
     - Standard assumption for many methods: $\mathcal{Y}^s = \mathcal{Y}^t$.
   - Partial DA:
     - Target label space is a subset of the source label space.
     - Risk: the model may waste capacity adapting to classes that don't appear in the target.
   - Open Set DA:
     - Target label space contains unknown classes not present in source.
     - Additional challenge: model must detect unknown target samples and not force classification into known source classes.
     - Requires open-set recognition or novelty detection strategies.

5. **Without Target Data! (Domain Generalization):**
   - Single-Source Domain Generalization: One labeled domain used; generalization is expected to unseen domains.
   - Multi-Source Domain Generalization:
     - Multiple labeled domains used to learn generalizable representations.
     - Key idea: train on diverse enough data so the model generalizes out-of-the-box to any new domain.
   - Predictive DA:
     - Based on metadata or descriptions (e.g., "you will get cartoon images").
     - Uses auxiliary information like style tags or domain embeddings to estimate target properties.
   - Continuous DA:
     - The target domain shifts gradually over time.
     - Requires models that can adapt incrementally.
   - Zero-Shot DA:
     - No target data, no label overlap, no metadata.
     - Must generalize purely from structural priors or semantic embeddings (like word vectors).
   - Domain generalization is even stricter than DA. It assumes you don't know the target at all. It's very realistic: in the real world, we don't know what domain we'll encounter next.



6. **Deep Domain Adaptation – Three Main Categories**
   - Discrepancy-Based Methods:
     - Align feature distributions using metrics like MMD (Maximum Mean Discrepancy) or CORAL (CORrelation ALignment).
     - focus on minimizing distance between feature representations across domains.
   - Adversarial-Based Methods:
     - Introduce a domain discriminator to classify source vs. target features.
     - Use a gradient reversal layer (GRL) to fool the discriminator and make features domain-invariant.
     - Key example: DANN (Domain-Adversarial Neural Network)
   - Reconstruction-Based Methods:
     - Use an autoencoder-style setup to ensure shared features are rich enough to reconstruct target samples.
     - Enforce semantic consistency without access to target labels.
   - These three categories are different strategies, but the objective is always the same: make the features domain-invariant, while preserving label-discriminative information from the source.

DA Aspect	Type / Concept	Description
Number of source domains	Single / Multi / Mixed	Number and combination of source datasets
Target supervision	Supervised / Unsupervised / Semi-Supervised	Whether labels are available in the target domain
Label space relation	Closed / Partial / Open Set	Whether source and target classes are aligned or disjoint
Target data availability	Source-Free / Domain Generalization / Zero-shot	Whether any target data is known or accessible during training
Adaptation strategy	Discrepancy / Adversarial / Reconstruction	How the model achieves domain alignment

Discrepancy based method

1. **General Setting for Discrepancy-Based DA**
   - Source Dataset: $\mathcal{D}_s = \{(\boldsymbol{x}_i^s, y_i^s)\}_{i=1}^{n_s}$
   - Target Dataset (unlabeled): $\mathcal{D}_t = \{\boldsymbol{x}_j^t\}_{j=1}^{n_t}$
   - Train a deep network that learn a representation that minimizes target risk:

$$\epsilon_t(\theta) = \Pr_{(x,y)\sim q}[\theta(x) \neq y]$$

   - Since we don't have y^t, we:
     - Train using source labels.
     - Align the feature distribution of source and target using a discrepancy measure (MMD).
   - Since we can't fine-tuned on target labels, we rely on minimizing the discrepancy between the two domains in the feature space.

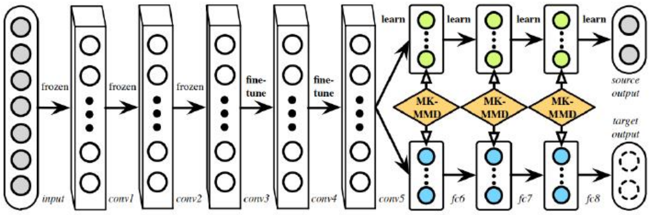
2. **Deep Adaptation Network (DAN):**
   - Maximize the Mean Discrepancy (MMD) between the source and target distribution at different layers (fc6-fc8) of a CNN.

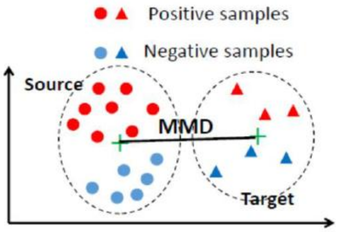
$$d_k^2(p,q) = |E_p[\phi(x^s)] - E_q[\phi(x^t)]|_{\mathcal{H}_k}^2$$

     - Where $\phi(.)$ maps the features into kernel space.
   - The feature extractor (e.g. AlexNet) is trained on source data.
     - Bottom layers are frozen.
     - Top layers are fine-tuned using MMD.
   - The total Loss:

$$\min_{\theta} \frac{1}{n_s} \sum_{i=1}^{n_s} \mathcal{L}(\theta(x_i^s), y_i^s) + \lambda \sum_{l=l_1}^{l_2} d_k^2(\mathcal{D}_s^\ell, \mathcal{D}_t^\ell)$$

   - We add the MMD loss to the usual cross-entropy, but only on layers closer to the output. Those layers are more domain-specific and need adaptation.





   - Feature space becomes more aligned and discriminative with DAN than with simpler method like DDC.
   - Target features (colors/shapes) are more clustered and aligned with the source.

2. **Maximum Mean Discrepancy (MMD)**: MMD is a statistical test for checking if two distribution (target, source) are the same using kernel embeddings.
   - Used in DAN (Deep Adaptation Network) to align features in hidden layers.
   - MK-MMD: uses multiple kernel to better estimate the discrepancy.
   - What makes DAN effective:
     - CNN backbone: Extracts hierarchical feature representations
     - Supervised Loss: Ensures correct source label classification
     - MMD Loss: Aligns the distribution of the source and target features.
     - MK-MMD: improves the flexibility of feature matching.
     - Frozen layers: keeps general features untouched.
     - Fine-tuned layers: learns domain-invariant high-level representation.

3. **Weighted Domain Adaptation Network (WDAN):**
    - Standard DAN assumes balanced class distribution. In practice (e.g., MNIST, USPS, SVHN), class priors differ → biased adaptation.
    - Solution: Introduce per-class $\alpha_{y_i^s}$ weights MMD:

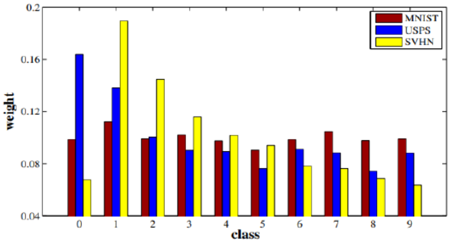
$$\mathrm{MMD}_w^2(\mathcal{D}_s, \mathcal{D}_t) = |\frac{1}{\sum \alpha} \sum \alpha_{y_i^s}\ \phi(x_i^s) - \frac{1}{N} \sum \phi(x_j^t)\ |_{\mathcal{H}}^2$$

        - Source classification loss.
        - Pseudo-labeling for target.
        - Weighted MMD across layers:

$$\min_{W, \widehat{y_j^t}, \alpha} \left[\frac{1}{M} \sum \mathrm{l}(x_i^s, y_i^s) + \gamma \frac{1}{N} \sum \mathrm{l}\left(x_j^t, \widehat{y_j^t}\right) + \lambda \sum_{\mathrm{l}} \mathrm{MMD}_{\mathrm{l},w}^2 \right]$$

    - This modification helps prevent dominant classes in the source from skewing the adaptation. It is specifically useful when target labels are estimated via pseudo-labeling.

3. **WDAN:** the original DAN (Domain Adaptation Network) uses MMD to align feature distribution across source and target domains. However, it assumes class distributions (priors) are similar across domains which is often false.
    - When class priors differ between source and target domains, aligning global distribution (via MMD) can actually hurt the performance. That's called class prior mismatch.
4. **Motivation behind WDAN:**
    - The bar chart shows class prior distribution of digits in 3 different datasets:
        - MNIST, USPS, SVHN
    - Example in SVHN, class 1 and 2 are more frequent that others. In USPS, classes 0 and 1 are dominant.
    - This shows class distribution varies a lot across domains which lead to bias during MMD-based alignment.



5. **WDAN solution:** introduce a class specific weighting into the MMD calculations:
    - Each class gets an auxiliary weight based on how frequent it is.
    - These weights are used to balance the contribution of each class when aligning distribution.

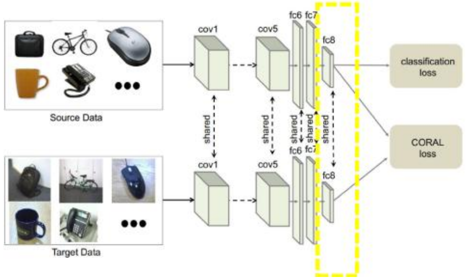
6. **Weighted MMD Formula:**

$$\mathrm{MMD}_{\mathrm{w}}^2(\mathcal{D}_s, \mathcal{D}_t) = |\frac{1}{\sum_{i=1}^{M} \alpha_{y_i^s}} \sum_{i=1}^{M} \alpha_{y_i^s} \phi(x_i^s) - \frac{1}{N} \sum_{j=1}^{N} \phi(x_j^t)\ |_{\mathcal{H}}^2$$

    - $\phi(x)$: feature embedding of input x.
    - $\alpha_{y_i^s}$: weight assigned to class y_i^s of source sample x_i^s
    - M, N: number of source and target samples
    - $\mathcal{H}$: RKHS (feature space)
    - This reweights source embedding to neutralize class imbalance in source and make alignment fairer.

7. **CORAL (CORelation Alighment):**

- Core Idea: Instead of aligning means (as MMD does), CORAL aligns the second-order statistics — i.e., covariances of source and target features.
- Minimize the distance between second order statistics covariance of the source and target feature distribution.
- This encourages the network to learn domain-invariance feature, improving generalization to the target domain.
- The CNN is trained simultaneously on source and target data.
- Feature extractors (e.g., from conv1 to fc8) are shared for both domains.
- Two branches:
  - Source: goes to classification head: produces classification loss.
  - Target: has no labels : used in CORAL loss only.
- CORAL loss is applied between source and target features at a specific layer (fc8) pushing their covariance matrices to match.



- CORAL loss equation:

$$\mathrm{l}_{\mathrm{CORAL}} = \frac{1}{4d^2} |C_S - C_T|_F^2$$

  - C_S, C_T: covariance matrices of source and target features.
  - d: feature dimension (output of fc8)
- This equation penalizes the difference in correlation structure between source and target.
- Covariance Matrix Computation: Given a matrix of feature vectors $D \in R^{n \times d}$

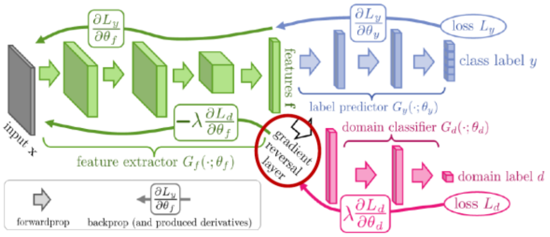
$$C_S = \frac{1}{n_S - 1}\left(D_S^\top D_S - \frac{1}{n_S}(1^\top D_S)^\top (1^\top D_S)\right)$$

- Why use Covariance:
  - Matching Means (as in MMD) may not be enough. Distribution can have different shapes or spreads.
  - Covariance captures relationships between features.
  - CORAL aligns these relationships: a stronger match that just aligning averages.
- CORAL works well because it uses a simple statistical assumption — it's light compared to MMD but still reduces domain shift effectively

Method	Core Mechanism	Key Feature	Notes
DAN	MMD	Aligns feature means in RKHS	Fine-tunes fc layers only
WDAN	Weighted MMD	Adjusts for class imbalance	Includes pseudo-labeling
CORAL	Covariance alignment	Aligns second-order stats (covariance)	Simpler but effective

Domain Adversarial Neural Networks (DANN)

1. **Component Overview**
    - Feature Extractor $G_f(x)$: A CNN backbone that maps the input x into feature space f. This component is shared across both the label predictor and the domain classifier.
    - Label Predictor $G_y(x)$: A classifier trained only on labeled source data to predict the class label y via cross-entropy loss L_y.
    - Domain Classifier $G_d(f)$: A binary classifier trained to distinguish weather a feature f come from source or the target domain, using domain labels $d \in \{0,1\}$.
    - DANN is generic; it can be plugged into a feedforward network. The key trick is to add a discriminator on domain identity and reverse its gradient to learn domain-invariant features.


$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features f
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
input x
gradient reversal layer
domain classifier $G_d(\cdot;\theta_d)$
feature extractor $G_f(\cdot;\theta_f)$
domain label d
loss L_d
$\lambda \frac{\partial L_d}{\partial \theta_d}$
forwardprop
backprop (and produced derivatives)

2. **Adversarial Strategy and Gradient Reversal**
    - To align source and target features, DANN introduces a Gradient Reversal Layer (GRL) between the feature extractor and the domain classifier.
    - In forward pass, GRL acts as the identity function.
    - In backward pass, it reverses the gradient:

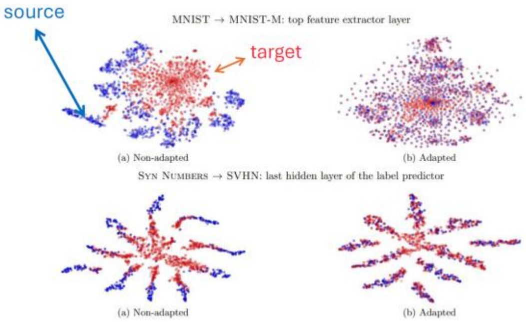
$$\frac{\partial \mathcal{L}_d}{\partial \theta_f} \leftarrow -\lambda \cdot \frac{\partial \mathcal{L}_d}{\partial \theta_f}$$

        - Where λ is a hyperparameter controlling adversarial strength.
    - The domain Classifier learns to distinguish source from target (minimizing L_d).
    - But the feature extractor learns to fool the domain classifier by maximizing L_d.
    - Resulting in domain-invariant features.
    - Objective function:

$$\min_{\theta_f,\theta_y} \max_{\theta_d} \mathcal{L}_y\left(y, G_y\left(G_f(x)\right)\right) - \lambda\, \mathcal{L}_d\left(d, G_d\left(G_f(x)\right)\right)$$

    - This setup guarantees that the features are discriminative for the label predictor, but not for the domain — which is exactly what we want in domain adaptation.

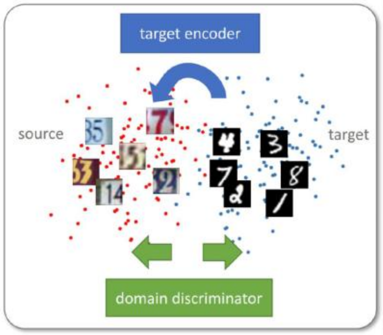
3. **Domain Alignment Visualization with DANN**
    - The figure shows the effect of DANN using t-SNE plots before and after adaptation.
    - Non-adapted (left):Features are clearly separated by domain; blue (source) and red (target) clusters are apart.
    - Adapted with DANN (right):Features are intermixed; domain identity is blurred, but class-level structure is preserved.
    - Effective domain confusion, as the domain classifier can no longer separate source and target.
    - Discriminative class features are maintained (red and blue points with similar class semantics remain close).
    - When you can't distinguish features by domain anymore, and class information is still preserved — that's successful adaptation.



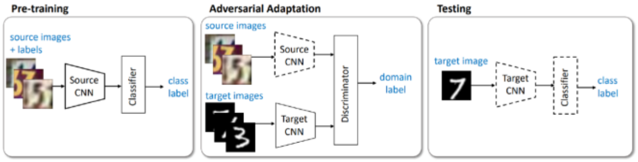
Adversarial Discriminative Domain Adaptation (ADDA)

1. **GANs Analogy for Domain Adaptation:**
   - Generator (G) = Learns to map target features to the source-like domain.
   - Discriminator (D) = Learns to distinguish between source domain and target domain features.
   - Objective = Fool the discriminator into classifying target domain features as source-like.
   - Think of it like counterfeiting. The target encoder keeps adjusting until its output features are mistaken for source domain features by the discriminator.

2. **Feature Space Alignment in ADDA:**
   - Red = Source domain features (e.g., SVHN).
   - Blue = Target domain features (e.g., MNIST).
   - The domain discriminator learns to separate the two.
   - The target encoder is trained adversarially to map target features closer to source.
   - Make the distribution of encoded target features align with those of source features, without changing source representation.



3. **ADDA Architecture – Training Phases**
   - Pre-training:
     - Train source encoder + classifier using labeled source data.
     - Source encoder becomes fixed afterwards.
   - Adversarial Adaptation:
     - Train a target encoder to fool the domain discriminator.
     - Domain discriminator is training to distinguish source vs. target features.
   - Testing: Apply target encoder and the frozen classifier to unseen target data.
   - Why freeze source encoder?
     - Prevents "catastrophic forgetting" of well-trained source features.
     - Ensures that target encoder aligns itself to the already learned source space.
   - This decoupling makes optimization easier and more stable compared to DANN. You optimize the source encoder first, then focus solely on aligning the target.



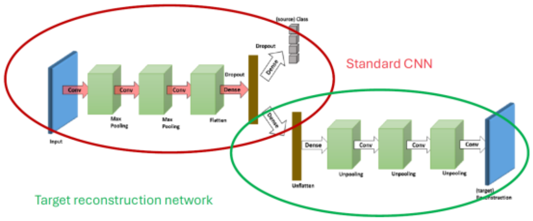
4. **ADDA vs DANN:**

Aspect	DANN	ADDA
Domain classifier	Shared with feature extractor	Applied only during target encoder training
Gradient reversal	Used	Not used (explicit adversarial training)
Training phases	End-to-end, joint	Sequential (pre-train, adapt, test)
Flexibility	Less modular	More modular: separate encoders, losses
Optimization stability	Can be harder	Easier due to stepwise structure

- Adversarial DA is inspired by GANs but focuses not on generating images, but on generating features that are indistinguishable across domains. Whether you reverse gradients (DANN) or use separate networks (ADDA), the goal is the same: remove domain information while preserving class semantics.

Reconstruction-based domain adaptation (with Image Translation)

1. **Dual-Pipeline Architecture of Deep Reconstruction Classification Network (DRCN):**



   - Standard CNN for classification (red circle):
     - Input flows through convolutional and dense layers to produce source-domain class predictions.
     - Trained only on labeled source data.
   - Reconstruction Network for target data (green circle):
     - Shares encoder with classifier.
     - Then passes through mirrored decoder layers (unpooling + deconvolution) to reconstruct the input.
     - Trained only on unlabeled target data.
   - The same encoder is shared between both tasks. The classification pipeline makes it label-discriminative, while the reconstruction pipeline ensures domain alignment.

2. **Functional Breakdown:**
   - Classification function:

$$f_c(x) = (g_{\text{lab}} \circ g_{\text{enc}})(x)$$

     - g_{enc}: shared encoder.
     - g_{lab}: classifier head (Dense Layer).
   - Reconstruction Function:

$$f_r(x) = (g_{\text{dec}} \circ g_{\text{enc}})(x)$$

     - g_{dec}: decoder network (used only with target data).
   - Reconstruction forces the encoder to retain fine-grained details about target domain inputs, preventing it from overfitting to the source domain.

3. **Loss Functions and Optimization Objective**
   - DRCN optimizes a joint loss:

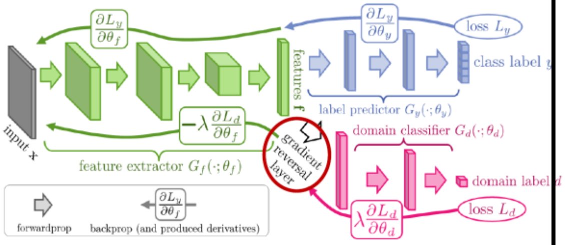
$$\min \lambda \mathcal{L}_c^{n_s}(\Theta_{\text{enc}}, \Theta_{\text{lab}}) + (1-\lambda)\mathcal{L}_r^{n_t}(\Theta_{\text{enc}}, \Theta_{\text{dec}})$$

   - $\mathcal{L}_c^{n_s}$: cross-entropy classification loss (on source).
   - $\mathcal{L}_r^{n_t}$: Reconstruction loss (MSE) on target.
   - λ: weighting factor balancing the two.
   - Start with a strong encoder via classification.
   - Let reconstruction regularize the encoder to make it generalize across domains.
   - If we only optimized for source classification, the encoder would ignore target distribution nuances. Reconstruction acts like unsupervised fine-tuning.

4. **Training Algorithm: This algorithm alternates between two stages per epoch:**
   - Supervised training on labeled source data.
   - Unsupervised training on target data for reconstruction.
   - This loop continues until convergence.

Adversarial based

1. **Why Adversarial Domain Adaptation:** For successful domain transfer, a model should make predictions based on features that are shared between domains therefore, the ideal feature representation is one that:
    - Clearly separates class labels(cats vs dogs)
    - But doesn't reveal whether the sample comes from target or source domain.
    - So the key idea is to use a domain classifier to distinguish between source and target domain features and train the feature extractor to confuse it.
        - This forces the network to learn domain-invariance features.
2. Domain Adversarial Neural Networks (DANN):
    - Feature extractor $G_f(x) - green$: maps input to a latent feature space.
    - Label predictor $G_y(f) - blue$: classifies samples into their respective classes (only on source data).
    - Domain classifier $G_d(f) - Pink$: Predicts whether a feature comes from the source or target domain.
    - Gradient Reversal Layer (GRL)
        - A crucial trick to make adversarial training easy.
        - During forward pass: does nothing.
        - During backpropagation: multiplies gradient by $-\lambda$, flipping the sign.



$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features $\mathbf{f}$
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
domain classifier $G_d(\cdot;\theta_d)$
input x
gradient reversal layer
feature extractor $G_f(\cdot;\theta_f)$
domain label d
loss L_d
$\lambda \frac{\partial L_d}{\partial \theta_d}$
forwardprop
backprop (and produced derivatives)

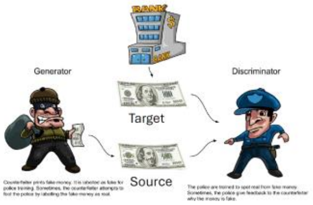
    - This means:
        - The domain classifier tries to minimize domain classification loss.
        - The feature extractor receives reversed gradients: it tries maximize the domain confusion.
    - Objective functions
        - L_y: classification loss on source labels.
        - L_d: domain classification loss (source vs. target).
        - Then the training optimizes:

$$\min_{\theta_f,\theta_y} L_y - \lambda L_d \quad \text{(feature extractor + label predictor)}$$

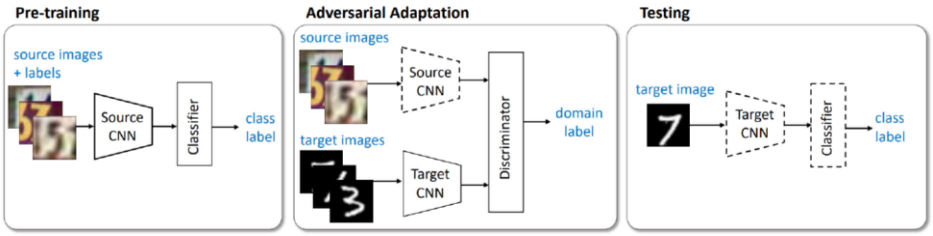
$$\min_{\theta_d} L_d \quad \text{(domain classifier)}$$

        - θ_f: parameters of feature extractor
        - θ_y: label classifier
        - θ_d: domain classifier
    - When to use DANN:
        - When we have labeled source data and unlabeled target.
        - You want to learn robust features that generalizes across domains with different distributions.

3. **Adversarial Discriminative Domain Adaption (ADDA)**
    - Generator: tries to make fake (target) money that looks like real (source) money.
    - Discriminator: tries to distinguish real from fake.
    - The Generator learns from the feedback to generate better fakes.
    - The target encoder trained adversarially to produce features indistinguishable from source feature.
    - The discriminator tries to tell if a feature comes from source or target
    - This leads to aligned feature spaces just like GAN, but the fake here is unaligned target domain.
    - ADDA Framework:



BANK
Generator
Discriminator
Target
Source
Counterfeiter prints fake money. It is labelled as fake for police training. Sometimes, the counterfeiter attempts to fool the police by labelling the fake money as real.
The police are trained to spot real from fake money. Sometimes, the police give feedback to the counterfeiter why the money is fake.

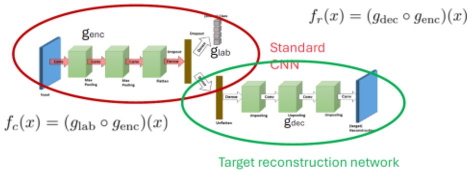


- Pretraining:
    - Training a source CNN on labeled source data to learn a classifier.
    - Only the source encoder and classifier are updated.
- Adversarial Adaption:
    - Freeze the source encoder
    - Train a target encoder on unlabeled target data to fool a domain discriminator
    - The discriminator tries to distinguish source from target.
    - The target encoder tries to fool the discriminator (make features look line source ones)
- Testing:
    - Discard the discriminator
    - Use the target encoder + source classifier to predict the target labels.
- The source and target encoders now map to a common feature space.

4. Domain Reconstruction classification Network (DRCN)

- DRCN is a multi-task learning approach combining:
    - Source classification: standard supervised learning on labeled source data.
    - Target Reconstruction: unsupervised reconstruction of target data.
- By training a shared encoder to work well for both tasks the learned features are useful for both classification and generalize well to the target domain.

- DRCN Architecture
- Standard CNN for Classification
    - Trained on source labeled data
    - Output class predictions
    - Classic structure: conv -> pooling -> dense -> softmax



- Target Reconstruction Network
    - Take some encoder as classification model
    - Adds decoder to reconstruct the target image
    - Use operations like unflattening and unpooling
- Both networks share the encoder; this is the key transferring knowledge from source to target.
- $g_{enc}(x)$: shared encoder that maps the input to feature space
- g_{lab}: classifier head (for predicting labels from from source features)
- g_{dec}: decoder (reconstructs the image from encoded features)
- Classification Function:

$$f_c(x) = (g_{\text{lab}} \circ g_{\text{enc}})(x)$$

- Reconstruction Function:

$$f_r(x) = (g_{\text{dec}} \circ g_{\text{enc}})(x)$$

- Supervised Classification Loss:

$$\mathcal{L}_c^{n_s} = \sum_{i=1}^{n_s} \mathrm{l}_c(f_c(x_i^s), y_i^s)$$

    - On source samples x_i^s with labels y_i^s.
    - Typically cross-entropy loss.
- Unsupervised Reconstruction Loss:

$$\mathcal{L}_r^{n_t} = \sum_{j=1}^{n_t} \mathrm{l}_r(f_r(x_j^t), x_j^t)$$

    - On target samples x_j^t with no labels.
    - Loss is L2 between input and reconstruction
- Final Objective:

$$\min \lambda \mathcal{L}_c + (1 - \lambda)\mathcal{L}_r$$

    - A weighted combination
    - λ balances classification vs. reconstruction

Source-Free domain Adaptation (SFDA) and Domain Generalization (DG)

1. **Conceptual Understanding:**
   - SFDA is a special case of Unsupervised Domain Adaptation where:
     - You don't have access to source data during adaption (only pre-trained model).
     - You only use target unlabeled data and adapt the model to the new domain.
   - DG is even stricter:
     - You don't know what the target domain will be at train time.
     - You train a model that generalizes unknow and unseen domains.
   - SFDA is more realistic for practical scenarios (due to privacy, storage, legal issues)
   - You cannot compute discrepancies (like MMD) directly between source and target, so methods need to:
     - Use the target data structure itself.
     - Leverage self-training, pseudo-labeling, or neighborhood-based reasoning.

2. **Data-Centric and Model-Centric Approaches for SFDA:**
   - Data-centric:
     - Domain-based reconstruction: Split or reconstruct target domain to mimic source features.
     - Neighborhood clustering: Exploits the natural structure in the target data (assumes similar samples cluster together).
   - Model-centric:
     - Pseudo-labeling: Use the source model to label confident target samples and iteratively improve.
   - Key ideas:
     - You must adapt using only the target data and source model.
     - Use the structure in target data (data-centric).
     - use the model's predictions as a guide (model-centric).

3. Domain Generalization:
   - Domain Generalization is the task of learning a model that generalizes to unseen domains during the training, using only labeled data from multiple source domains during training.
   - This is different from UDA and SFDA because:
     - You never have access to the target domain during training (not even unlabeled).
     - You do have multiple source domains (with labels) which introduce diversity.
     - The model is trained to capture generalizable features across the variability in source domains.
   - Why is this hard?
     - Each domain has a different distribution (different styles, lighting, texture, background, etc.).
     - The model must extract features that are domain-invariant, so it doesn't overfit to any one domain's specific style.
     - Generalization is even harder than adaptation, because you can't tune anything with target data.

Unsupervised Learning

Introduction to Unsupervised Learning

1. **Types of Learning:** with or without teacher

	With Teacher	Without Teacher
Active	RL/Active Learning	Intrinsic Motivation/Exploration
Passive	Supervised Learning	Unsupervised Learning

   - Passive + Teacher = Supervised Learning (model passively learns from labeled data)
   - Passive + No Teacher = Unsupervised Learning (No guidance, model finds structure in raw data)
   - Active + Teacher = Active Learning / RL (Feedback loop with human or environment)
   - Active + No Teacher = Exploration (agent-drive exploration)

2. **Why learn without a Teacher?** Three motivations
   - Label scarcity: labels are hard to define or expensive to obtain.
   - Biological plausibility: Unsupervised learning mimics human intuition closely
   - Generalization: learning representations without predefined tasks encourages general-purpose models adaptable to new tasks.

3. **Limitations of Transfer Learning:** transfer learning assumes knowledge from one domain generalizes to another.
   - Works to a degree (speech models transferring across languages).
   - But transfer is often limited due to task-specific representations.
   - Hypothesis: supervised targets/rewards don't carry enough information for effective transfer.

4. The cake analogy – Yann LeCun's insight: If intelligence was a cake, unsupervised learning would be the cake, supervised learning the icing, and reinforcement learning the cherry on top
   - Supervised targets are low in information content.
   - RL rewards are often less informative.
   - Input data itself holds vast, untapped knowledge – idea for unsupervised learning.

5. **Illustrative Example: ImageNet**
   - ImageNet ≈ 1.28M images, 1000 labels.
   - Label entropy: ≈ 12.8 Mbits.
   - Raw image data (128×128 uncompressed): ≈ 500 Gbits.
   - Conclusion: the label contains 4-5 orders of magnitude less information than the input.

6. **Unsupervised Learning: Core Challenge**
   - We have : $D = \{x_1, x_2, \dots, x_n\}$
   - But no explicit task => what is $l(D)$?
   - Core question: can we define a task implicitly that:
     - Doesn't require supervision.
     - Yields features useful across different tasks

7. Density Modeling as one solution: Use maximum likelihood estimation (MLE):

$$L(\mathcal{D}) = \sum_{x \in \mathcal{D}} - \log p\,(x)$$

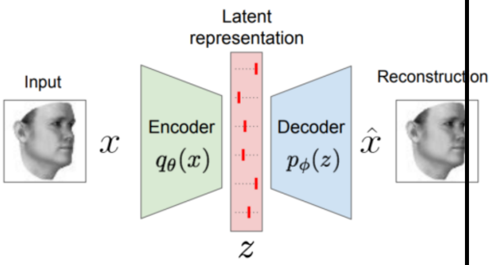
   - Try to learn the distribution from which the data was sampled.
   - Example: VAE, Normalizing flow, diffusion models
   - But density modeling is not the only way to learn representations

8. Generative models are unsupervised: Generative models such as Diffusions, GAN, VAE can model the distribution $p(x)$ of the data directly.
   - Why they are unsupervised:
     - They don't require labels.

    - Their goal is to generate new samples from a learned distribution.
    - When trained properly, they encode the structure of the data in their latent space.
- Benefits:
    - Sampling lets us visualize what the model has learned.
    - We can inspect model failures (if some classes are missing or distorted)
- Example:
    - The image grid shows generated samples.
    - If one class "cake" looks weird while others (dog, panda, …) looks fine, this signals bias or poor coverage in the data distribution.

Unsupervised Representations

1. **Autoencoders**

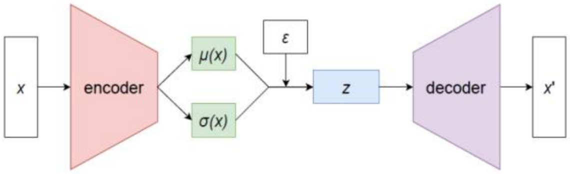


- Core Concept: An autoencoder learns a compact representation z by minimizing the reconstruction error between input x and output $\hat{x}$.

$$x \to q_\theta(x) = z \to p_\phi(z) = \hat{x}$$

- Purpose of Unsupervised Learning:
    - Learns a latent space that captures underlying data structure.
    - No label required.
    - Often used as a feature extractor for downstream tasks like clustering or classification.
- Limitation:
    - The latent space is not guaranteed to be well-structured (continuity, disentanglement)
    - Similar images might not be map close together.

2. **Variational Autoencoders (VAEs)**



$$x \xrightarrow{Encoder} (\mu(x), \sigma(x)) \xrightarrow{Sampling} z \xrightarrow{Decoder} x'$$

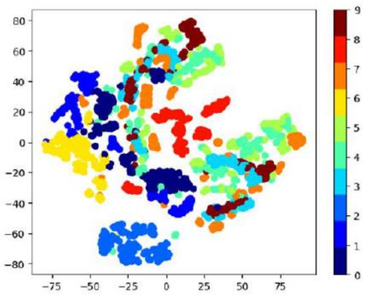
- Encoder learns a distribution over latent space (Usually Gaussian).
- Sampling trick ensures differentiability during training.
- Loss function combines reconstruction error and KL divergence (regularization):

$$\mathcal{L} = E_{q(z|x)}[\log p\,(x|z)] - \mathrm{KL}\big(q(z|x)\,|\,p(z)\big)$$

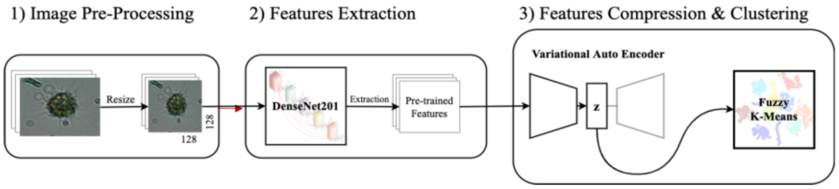
- Advantages Over AE:
    - The latent space becomes smooth and structured.
    - Great for clustering and generation (can sample new data).
    - Regularity ensures generalization across input.

3. Real world Scenarios

- Case: Plankton Images
  - Real-world data is fine-grained and highly variable.
  - Unlabeled biological samples require meaningful, discriminative embeddings.
- Issue: Even with generative models, latent representation might not be discriminative enough for tasks like clustering or anomaly detection.
- Visual Insight:
  - The scatter plot shows embedding clustering in a latent space.
  - Ideally, different classes (even if unknown) should form a well-separated compact clusters.



4. Transfer Learning + VAE

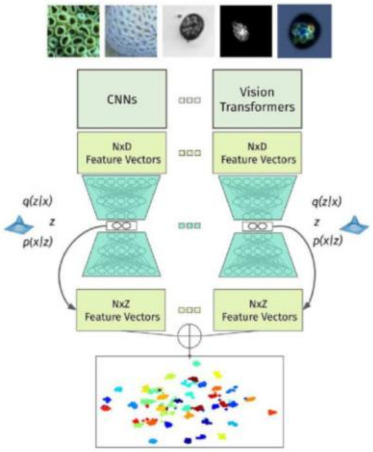


- Preprocessing Images (resize, normalization)
- Use a pretrained model (DenseNet101) to extract deep features.
- Feed these features to a VAE to compress them.
- Use the compressed features n clustering (Fuzzy, K-means).

- Key insights:
  - Pretrained models like DenseNet are trained on a supervised data (ImageNet) and can capture high-level semantic features.
  - VAE regularize and compress these features into a usable latent space.
- Benefits:
  - You obtain a latent space that is both compact and informative.
  - Fuzzy K-means helps account for ambiguity in cluster boundaries.

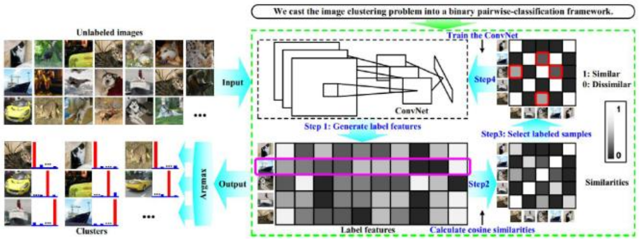
5. Unsupervised Representation (5): Ensembles of VAEs

- Problem: Even pretrained models have inductive biases (CNNs focus on local patterns, transformers on global context)
- Solution: Use an ensemble:
  - Extract features from different models (CNNs, ViTs).
  - Train individual VAEs on each.
  - Concatenate (or fuse) the resulting latent vectors.
  - Cluster or analyze combined features.
- Diagram Breakdown:
  - Each model -> feature extractor -> VAE -> Latent *z*
  - Combine the various *z* spaces into a meta-representation.
  - Final output: a richer, multi-view embedding idea for clustering or transfer.



Deep Adaptive Image Clustering (DAIC)

1. **Overview of Deep Adaptive Image Clustering (DAIC):**
    - DAIC is a single-stage ConvNet method.
    - Instead of direct clustering, it reframes the task as a pairwise similarity classification:
        - Does image x_i belongs to the same cluster as image x_j?
    - Architecture:
        - Unlabeled Images
        - CNN (ConvNet)
        - Feature embeddings
        - Compare pairs
        - Learn similarity
    - This approach bypasses the need for explicit labels and directly emphasizes relational knowledge.


We cast the image clustering problem into a binary pairwise-classification framework.
Unlabeled images
Input
Train the ConvNet
ConvNet
Step4
1: Similar
0: Dissimilar
Step 1: Generate label features
Step3: Select labeled samples
1
0
Argmax
Output
Step2
Similarities
Clusters
Label features
Calculate cosine similarities

2. **Mathematical Formulation of DAIC:**
    - Dataset Setup:
    - Input images : $\mathcal{X} = \{x_i\}_{i=1}^{n}$
    - Training set: $\mathcal{D} = \{(x_i, x_j, r_{ij})\}$
    - $r_{ij} = 1$ if same cluster
    - $r_{ij} = 0$ if different
    - But r_{ij} is unknown, we estimate it via similarity score
    - Objective Function

$$\min_{w} E(\boldsymbol{w}) = \sum_{i,j} L\left(r_{ij}, g(x_i, x_j; \boldsymbol{w})\right)$$

        - $g(.)$: similarity function
        - L: Binary cross-entropy loss
    - Loss function expanded:

$$L\left(r_{ij}, g(x_i, x_j; w)\right) = -r_{ij} \log g\,(x_i, x_j) - (1 - r_{ij}) \log\left(1 - g(x_i, x_j)\right)$$

3. **Cosine Similarity in latent space**
    - Feature construction
        - Label features : $\mathcal{L} = \{l_i \in R^k\}$
        - Extract from the last FC layer with k neurons (clusters).
    - Constraints:

$$|l_i|_2 = 1, \quad l_{ih} \geq 0 \quad \forall h = 1, \ldots, k$$

    - Similarity Function
        - Use cosine similarity: $g(x_i, x_j; w) = l_i \cdot l_j$
    - New Objective with Constraints:

$$\min_{w} \sum_{i,j} L(r_{ij}, l_i \cdot l_j) \quad \text{s.t.} \quad |l_i|_2 = 1,\ l_{ih} \geq 0$$

4. **Threshold for pair selection**: to decide if $r_{ij} = 1$ or $r_{ij} = 0$, define a threshold:

$$r_{ij} = \begin{cases} 1 & if\ l_i . l_j \geq \mu(\lambda) \\ 0 & if\ l_i . l_j < l(\lambda) \\ none & otherwise \end{cases}$$

    - $\mu(\lambda)$: upper threshold
    - $l(\lambda)$: lower threshold
    - Only confident pairs (above/below thresholds) contribute to training.
    - This is adaptive and improves over time.

5. **Curriculum learning and optimization**
    - As training progress, model becomes more confident.
    - Gradually tighten thresholds, (increasing training samples).
    - Penalty Term: Penalty $= u(\lambda) - l(\lambda)$
        - Smaller gap : more samples used.
    - Final objective:

$$\min_{w,\lambda} \sum_{i,j} v_{ij} \mathcal{L}\left(r_{ij}, l_i \cdot l_j\right) + u(\lambda) - l(\lambda)$$

    Where :

    - $v_{ij} = 1$ if $r_{ij} \in \{0,1\}$, else 0.

6. **Alternating Optimization Strategy:** Alternating Minimization

    - Fix λ, and update the weights w:

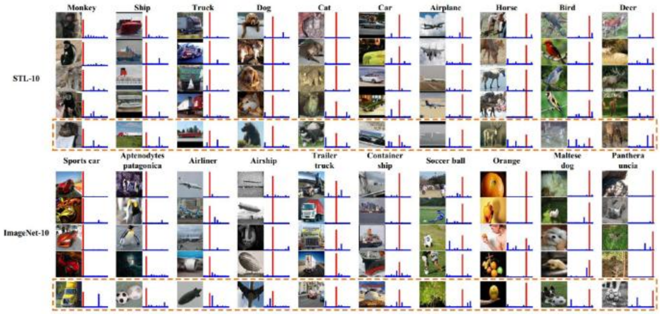
$$\min_{w} \sum_{i,j} v_{ij} \mathcal{L}\left(r_{ij}, f(x_i) \cdot f(x_j)\right)$$

    - Fix W, and updating the λ:

$$\min_{\lambda} u(\lambda) - l(\lambda) \quad \Rightarrow \quad \lambda \coloneqq \lambda - \eta \frac{\partial E(\lambda)}{\partial \lambda}$$

    - This encourages the gradual expansion of training coverage, helping the network avoiding early overfitting.

7. Results – STL-10 & ImageNet-10

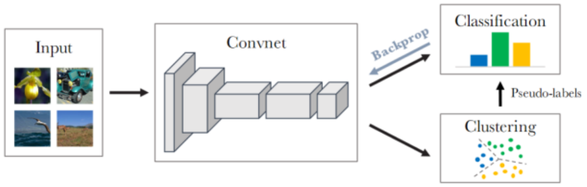


    - Images grouped by predicted clusters.
    - Bars indicates confidence or activation across soft cluster dimensions.
    - High confidence (red) aligns with clean separation.
    - Visually verifies meaningful and interpretable clusters.
    - It demonstrates DAIC can extract semantically meaningful groups from unlabeled data.

Deep Clusters

1. **What it solves:**
    - Manual labeling is expensive, slow, and error-prone.
    - Pretext tasks in self-supervised learning (colorization, rotation, …) are domain-dependent and manually crafted.
    - DeepCluster eliminates the need for pretext tasks by using clustering to generate pseudo-labels in a fully unsupervised manner.

2. **How DeepCluster works: High-level pipeline:**
    - Input: Unlabeled Images (ImageNet)
    - ConvNet: Feature extractor (AlexNet, VGG)
    - Clustering: Apply K-means to extracted features
    - Pseudo-labels: Assign cluster IDs to images
    - Classification: Train Network to predict pseudo-labels.
    - Repeat: Alternate clustering and classification in a loop.



3. **Iterative Optimization: This forms a closed training loop**
    - Extract features -> Clusters -> assign pseudo labels.
    - Use Pseudo-labels to train the CNN (Like a supervised classifier)
    - Repeat: as CNN improves, so does the quality of the clustering.

4. **Detailed Pipeline Breakdown:**
    - ConvNet + Preprocessing:
        - Starts with a ConvNet (AlexNet) initialized randomly.
        - Extract features from FC2 layer (dim = 4096).
        - Apply PCA to reduce to 256 dimensions.
        - Whitening and L2 normalization to standardize features.
        - Apply K-means clustering to generate pseudo-labels.
    - Data Augmentation – Two Cases

Purpose	Augmentation Strategy
For Clustering	Simple resize + center crop (224× 224), normalization
For training CNN	Random crop, resize, flip, normalize + sobel filter for contrast

5. **Clustering to Generate Labels**
    - K-means assigns each image to a cluster.
    - These assignments treated as pseudo-labels.
    - The network is then trained using cross-entropy loss, just like n supervised learning.

6. **Architecture Details: AlexNet (Used in DeepCluster):**
    - 5 Conv layers + 3 FC layers.
    - Remove local response normalization (LRN).
    - Add BatchNorm and Dropout for regularization.
    - Output of FC2 is used as a feature vector for clustering.

7. **Why many clusters?**
    - Using more clusters = better granularity.
    - Instead of 2 clusters (cats vs. dogs), using 4 splits the data into breeds.
    - Paper uses 10000 clusters n ImageNet (which has 1000 classes) to capture fine semantic differences.

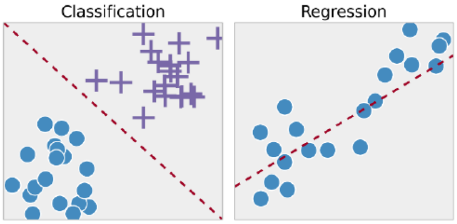
8. **DeepCluster Loop**
    - Initialize CNN weights (random or pretrained)
    - For each epoch:
        - Extract features from FC2 layer
        - Apply PCA -> L2 norm -> K-means
        - Assign cluster indices.
        - Train CNN to classify clusters using cross-entropy
    - After each epoch, repeat the clustering on new features.
    - Continue for a fixed number of epochs (eg 500)

9. **Quantitative Evaluation**
    - Metrics:
        - NMI (Normalized Mutal Information): $\mathrm{NMI}(A;B) = \frac{I(A;B)}{\sqrt{H(A)H(B)}}$
        - Measures how similar the cluster assignment A is to ground truth B.
        - Ranges from 0 (independent) to 1 (perfect match).
        - mAP (mean average Precision): Measures precision across ranked predictions.
    - Observations:
        - NMI improves over time.
        - Cluster reassignment becomes more stable across epochs.
        - Choosing $k = 10^4$ clusters yield the best mAP (fine-grained grouping)

Semi-Supervised

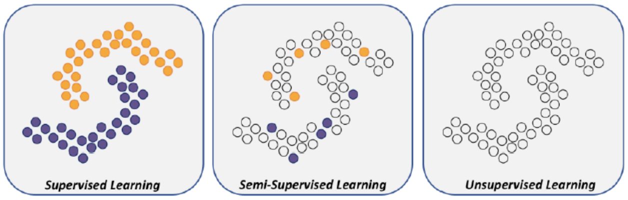
Difference with Supervised and Unsupervised

1. **Supervised Learning:**
    - Definition: is a machine learning paradigm where the algorithm trained on labeled dataset. Each input point has a corresponding output label. The model goal's is to learn a mapping from inputs X to output y, and to generalize the mapping to predict labels for unseen data.
    - Key tasks:
        - Classification: algorithm predicts discrete labels. For example, email spam detection, diseases diagnosis.
        - Regression: the algorithm predicts the continues outputs. For example predicting house price.



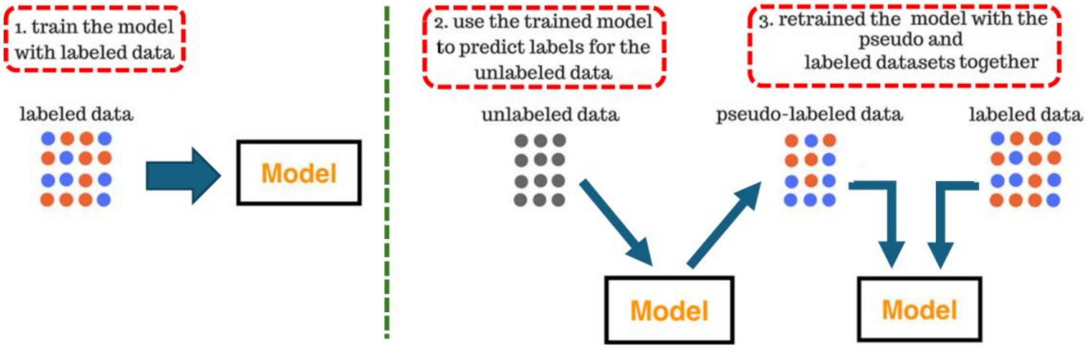
2. **Unsupervised Learning:**
    - Definition: the algorithm is given unlabeled data and must find the hidden patterns or intrinsic structures without any explicit instructions or known outputs.
    - Core Concepts: there is no predefined target variable. The model must infer patterns purely from the structure of the input data.
    - Primary tasks:
        - Clustering: grouping data based on similarity.
        - Association: discovering interesting relationship between features in the data. (people who buy bread also buy butter)

3. **Semi-Supervised Learning (SSL):**
    - Definition: SSL is a hybrid approach that lies between supervised and unsupervised learning. It leverages small amount of data along with a large amount of unlabeled data to improve model performance.
    - Concept illustration:
        - In supervised learning, all data is labeled.
        - In unsupervised learning no data is labeled.
        - In SSL a few points are labeled the rest are not.
    - SSL aims to generalize better by incorporating the structure from unlabeled data.
    - Why SSL matters?
        - Cost-Efficiency: labeling data is expensive and time-consuming.
        - Abundance of unlabeled data: in most applications, unlabeled data is plentiful (web data, sensor streams, etc.)
        - Better Generalization: SSL can leverage structure in unlabeled data to improve decision boundaries, reducing overfitting to the small labeled set.



Pseudo-Labeling

1. **Definition:** Pseudo-labeling is a straightforward yet effective semi-supervised technique proposed in 2013. It leverages a small labeled dataset and a larger unlabeled dataset. By assigning "Pseudo-labels" to the unlabeled data using current model's predictions.


1. train the model with labeled data
labeled data
Model
2. use the trained model to predict labels for the unlabeled data
unlabeled data
3. retrained the model with the pseudo and labeled datasets together
pseudo-labeled data
labeled data
Model
Model

2. **Three step work-flow:**
    - Train the initial model using the labeled data only. (standard supervised learning on small dataset)
    - Predict pseudo-labels for the unlabeled data. (the trained model is used to assign labels to the unlabeled data by selecting the class with the highest predicted probability)
    - Retrain the model using the both real and pseudo-labeled data. (This model is fine-tuned or retrained on the combination of actual labeled data and the newly pseudo-labeled data)
    - This process allows the model to leverage the structure in the unlabeled data while being anchored by ground-truth labels.

3. **Mathematical Formulation of Pseudo-labeling.**
    - Pseudo-labeling generation: left $f_i(x)$ be the model's output score for class i on input x. The pseudo-labeling y_i' is assigned as:

$$y_i' = \begin{cases} 1 & if\ i = argmax f_i(x) \\ 0 & otherwise \end{cases}$$

        - This simply means the models picks the class with the highest probability and assigns it as the label.

4. **Loss Function Design**: To combine the contribution from both labeled and pseudo-labeled data, the total loss function is a weight sum:

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{C} L(y_i^m, f_i^m) + \alpha(t) \cdot \frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} L(y_i'^m, f_i'^m)$$

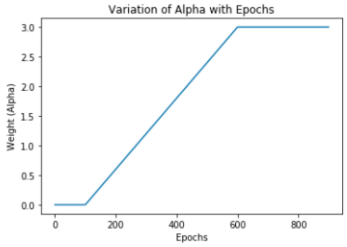
- The first term is the standard supervised loss on n labeled examples.
- The second term is the unsupervised loss on n' pseudo-labeled examples.
- $\alpha(t)$ is a dynamic weight that grows over time to increase the influence of the pseudo-labeled data.
- Simplified Interpretation:

$$Loss\ per\ Batch = Labeled\ Loss + Weight * Unlabeled\ Loss$$

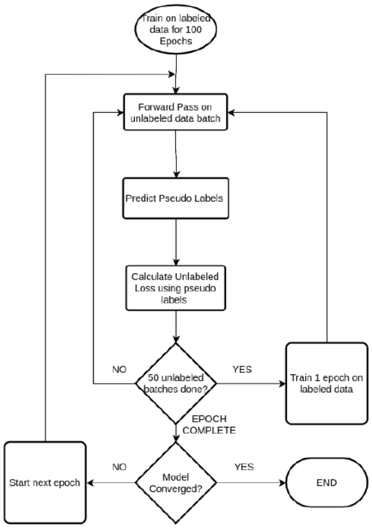
5. **Dynamic Weighting with $\boldsymbol{\alpha(t)}$:** To prevent early-stage noisy pseudo-labeling from harming training, $\alpha(t)$ increases gradually.

$$\alpha(t) = \begin{cases} 0 & t < T_1 \\ \dfrac{t - T_1}{T_2 - T_1} & T_1 \le t < T_2 \\ \alpha_f & T_{2 \le t} \end{cases}$$

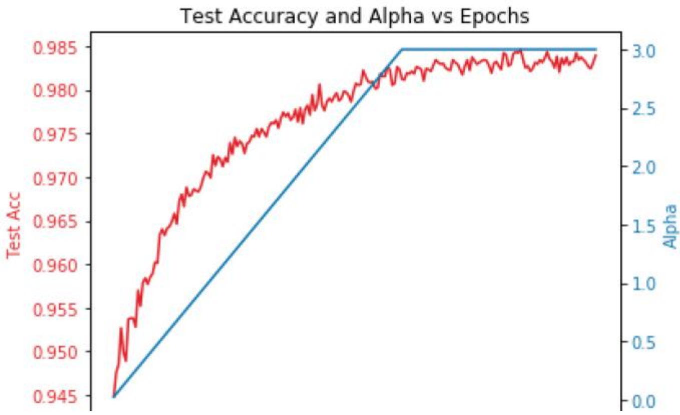
- T_1: Initial phase where only labeled data is used (first 100 epochs).
- T_2: Final phase where full weight α_f is applied.
- The strategy helps gradually incorporate unlabeled data and improve generalization.
- The graph of alpha vs. epochs shows a linear ramp-up until it saturates aligning with the increased test accuracy curve.



6. **Training Loop and Implementation logic**
    - Initially, train on only labeled data for T_1 epochs.
    - Then, alternate between:
        - Forward pass on unlabeled data(generate pseudo labels): compute unsupervised loss.
        - Occasional epochs of supervised training on labeled data.
    - If model converges (or after a fixed number of iterations), the training ends.
    - Advantages of alternating scheme:
        - Reduces overfitting by not solely relying on the limited data.
        - Increases training efficiency, as it requires only one forward pass for unlabeled data rather than two.

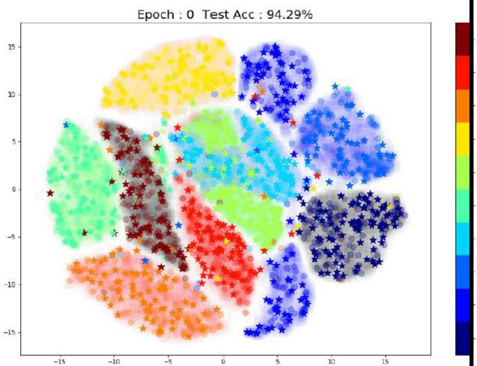

Train on labeled data for 100 Epochs
Forward Pass on unlabeled data batch
Predict Pseudo Labels
Calculate Unlabeled Loss using pseudo labels
NO
50 unlabeled batches done?
YES
Train 1 epoch on labeled data
EPOCH COMPLETE
Start next epoch
NO
Model Converged?
YES
END

7. **Empirical Results on MNIST**
    - Labeled: 1000 class-balanced images
    - Unlabeled: 59000 images
    - Test: 10000 images
    - Supervised baseline:
        - Trained on labeled data only
        - Test Accuracy: 95.57%
        - Test Loss: 0.233
    - Training configuration:
        - 100 epochs on unlabeled data
        - 170 epochs on semi-supervised setup
    - Best result on epoch on 168:
        - Test Accuracy: 98.46%
        - Test Loss: 0.075
        - Alpha Weight: 3
    - Alpha Vs. Accuracy Plot:
        - As α increases, more emphasis is place on pseudo-labeled data.
        - Test accuracy gradually improves, then saturates, indicating the model has extracted most of the useful structure from unlabeled data.



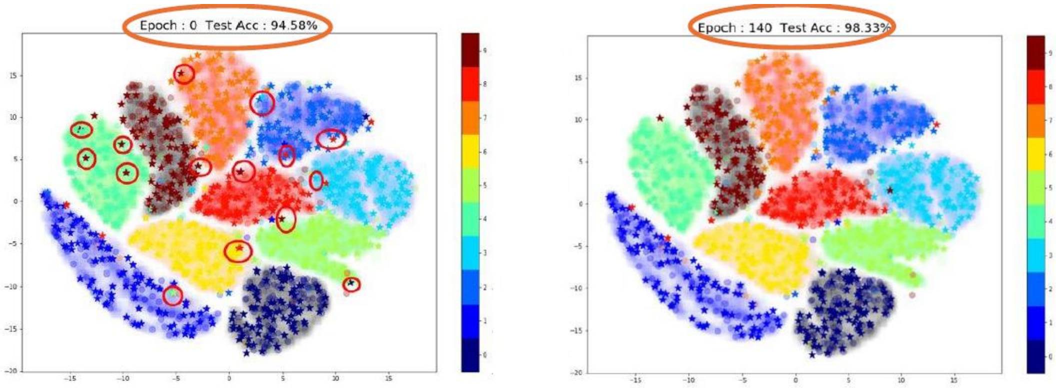
T-SNE Visualization of Pseudo-Labeling Dynamics

1. Introduction:
    - What is T-SNE? t-Distributed Stochastic Neighbor Embedding is a dimensionality reduction technique used for visualization of high-dimensional data. In the context of Pseudo-Labeling, it helps us see how the model's latent feature space evolves over training.
    - Visualization Breakdown: The plot shows the structure of model predictions at Epoch 0 with three components:
        - Background color: Represents the true label of all 60,000 training images.
        - Small Circles: Represent the 1,000 labeled examples used during supervised training. These act as the "anchor points" to initiate learning.
        - Small Stars: Represent 750 pseudo-labeled points, i.e., predictions made by the model for randomly selected unlabeled images at this epoch.



Circles = Labeled Data
Stars = Unlabeled Data

2. Evolution Across Epochs

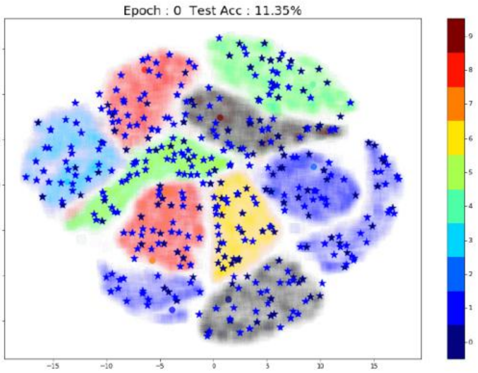


- Red circles highlight improved predictions: stars that were previously in wrong clusters have moved closer to their correct cluster.
- The pseudo-labeling accuracy increases over time, validating that the model’s internal representations improve with training.

3. Why Does Pseudo-Labeling Work?
    - Continuity Assumption (Smoothness):
        - Principle: points that are close in feature space are likely to share the same label.
        - Effect: small perturbations (minor rotations or shift) shouldn’t change the predicted label.
        - Use in Pseudo-labeling: allows the model to generalize pseudo-labels from confidently labeled data to nearby unlabeled data.
    - Cluster Assumption:
        - Principle: data tends to form discrete clusters, and points within the same cluster are likely to have the same label.
        - Implication: Decision boundaries should pass though low-density regions (between clusters)
        - Effect: the classifier avoids making sharp transitions withing a dense cluster, reducing the misclassification.

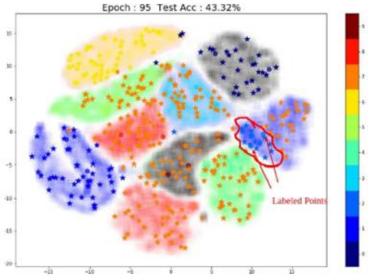
Why does Pseudo-labeling Fail?

1. Case 1: Too few labeled points
    - Experiment 1: Only 10 Labeled Points (1 per class)
    - Outcome: Test accuracy ~11.35%, nearly random chance for 10 classes.
    - Why it failed: one label point per class is not enough to define the structure of each class's cluster.
    - T-SNE Insight: Pseudo-labels (stars) are scattered across incorrect clusters, showing no alignment.

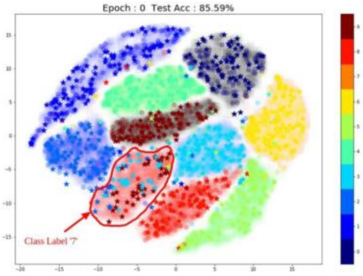


    - Experiment 2: 20 Labeled Points (2 per class)
    - Outcome: Test accuracy rises modestly (~43.32% after 95 epochs).
    - Observation: Pseudo-labels are more accurate near labeled points (especially when two are close by).
    - T-SNE Insight: Some clusters become more defined, but others remain ambiguous.

2. Case 2: Missing classes in labeled set
    - Scenario: One class (e.g., digit '7') is absent from labeled data but present in the unlabeled set.
    - Before SSL: Model reaches ~85.6% accuracy.
    - After SSL: Slight improvement to 87.98%, but fails to learn class '7'.
    - T-SNE Insight: Cluster for '7' (outlined) never gains proper labeling, since the model has never seen it labeled.



3. Model Complexity Limits
    - Issue: Some models (e.g., SVMs or logistic regression) may not benefit from more unlabeled data.
    - Why: These algorithms lack the capacity to exploit fine-grained representations or benefit from large-scale structure.
    - Solution: Larger deep networks (as shown in Andrew Ng's "Scale drives performance" graph) can learn richer hierarchies and generalize better with more data.
    - SSL works best with expressive models (deep neural networks) that can extract latent representations and adjust decision boundaries meaningfully as more data is seen.



Final Reflection and Conclusion:

1. Pseudo-labeling is simple, intuitive, and popular due to its plug-and-play nature.
2. It is often a strong baseline for SSL, but:
    - Relies heavily on early model's accuracy.
    - Sensitive to initial labeled data quality.
    - Struggles with missing classes or low confidence predictions.

Diffusions

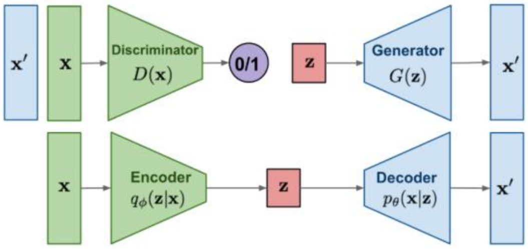
Introduction to Deep Generative Learning

1. **Main Goal:** Learning to generate data that resembles samples from real world.
   - The image with many cats represents real-world data, it's a data distribution, denoted as $p_{data}(x)$. The key point here is:
     - You don't know it; you can't formalize it. But can only sample from it.
   - This process involves training a neural network that captures the structure of an unknown distribution. During the training the models learns to understand the characteristics of real cats.
   - After training the model can generate new samples that resembles the original distribution.

2. **Undrestanding the Density Estimation:** As we discussed earlier the goal of deep generative models are to learn the undrelying data distribution $p_{data}(x)$ so that we can sample a similar distribution $p_{model}(x)$.
   - Training data $p_{data}(x)$: the unkown but real real-world distribution.
   - Model's goal: Approximate this with $p_{model}(x)$, such that new samples from the model look like the training data.
   - Two types of density estimation:
     - Explicit Density Estimation:
       1. The model explicitly defines a probability form for the $p_{model}(x)$ and optimize it.
       2. Example: VAEs, Normalizing flows
     - Implicit Density Estimation:
       1. The model doesn't define specific form for the probability distribution, instead it just learns how to sample from it.
       2. Example: GANs. They generate realistic data, but they don't give us mathematical form of $p(x)$.

3. **Model categories: GAN, Diffusions, VAEs:**
   - Explicit Density Estimation:
     - VAE (Variational Auto Encoder):
       1. Assumes a specific latent distribution (Gaussian).
       2. Learns to encode data into latent space and decode it back while approximate true data distribution.
       3. Gives a tracable, differentiable expression for $p_{model}(x)$.
     - Diffusion model:
       1. A new class that gradually learns reverse noise processing.
       2. Unline VAE, they don't assume a simple structure like Gaussian, they progressively denoise data in a multi-step proces.
       3. Diffusion is not a one-step process like VAE and GAN.
   - Implicit Density Estimation: GAN
     - Learns to generate data via game between generator and discriminator.
     - Doesn't defin or compute a probability density function.
     - Samples come from transforming noise through a neural network.
     - It can sample directly from noise without assumption thus its implicit.

4. **Architecture GAN vs VAE:**
    - GAN pipeline:
        - Generator $G(z)$ creates fake images x' from noise z.
        - Discriminator $D(x)$ tries to distinguish between real and fake.
        - Training is adversarial: the generator learns to fool the discriminator.
        - Problems: Vanishing gradient and mode collapse



    - VAE pipeline:
        - Encoder $q_\emptyset(z|x)$: Maps the input x to the latent space z.
        - Decoder $p_\theta(x|z)$: Recosntruct x from z.
        - Loss envolves reconstruction error + regularization (KL divergence).
        - Training is stable but outputs can be blurry due to the likelihood formula.

DDPM (Denoising Diffusion Probabilistic) Overview

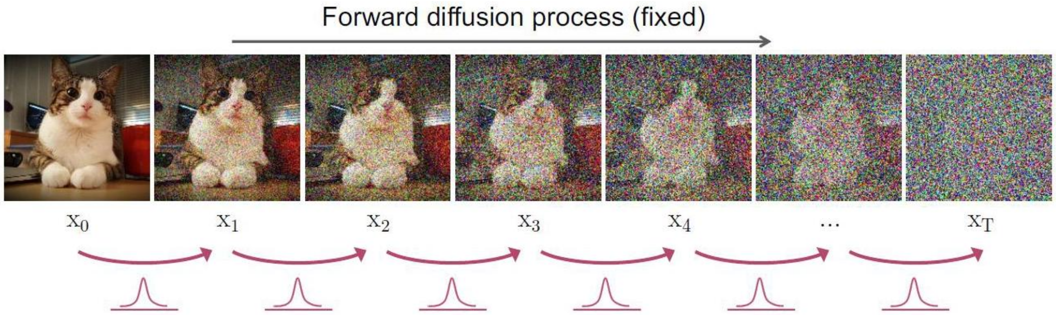
1. The central idea of DDPM is to rather than generating samples in a single step, DDPMs use a multiple-iterative process based on a Markov chain to gradually move from noise to data.
    - Two Key phases:
        - Forward Process (diffusion): add small amount of Gaussian noise to real data at each step until you reach pure noise.
        - Reverse Process (denoising): learn how to reverse this corruption using neural network.
    - Why this is done: Because smaller steps are easier to model. Instead of trying to jump from noise to image in one go (like GANs), DDPM learns many small transitions that are each tractable.
    - The entire process is Markovian, meaning each state only depends on the previous one. This makes the math and modeling more manageable.
2. VAE and Reparameterization Trick.
    - Loss Function of a VAE:

$$\mathcal{L}_{\mathcal{VAE}} = -\log \mathrm{p}_\theta(\mathrm{x}) + \mathrm{D}_{\mathrm{KL}}\left(\mathrm{q}_\phi(\mathrm{z} \mid \mathrm{x}) \mid \mathrm{p}_\theta(\mathrm{z} \mid \mathrm{x})\right)$$

        - The first term is the reconstruction likelihood: how well the decoder reproduces the input.
        - The second is the KL divergence: A penalty if the latent distribution diverges from desired prior.
    - The Problem: we need to sample from $z \sim q_\emptyset(z|x)$ but sampling is stochastic, so the gradient cannot backpropagate.
    - Solution: Reparameterization Trick
        - Instead of: $z \sim \mathcal{N}(\mu, \sigma^2 I)$
        - We rewrite: $z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$
        - Now the gradient can flow through μ and σ, enabling end-to-end training. We just use a normal standard distribution, sample it and pass it through the parameters learned by the encoder.

Forward Diffusion Process

1. Core forward process of DDMPs visual.



- Starting from a clean image x_0, we add Gaussian noise incrementally though T time steps. $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_T$
- Each transition: $q(x_t \mid x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t} x_{t-1}, \beta_t I\right)$
  - β_t: controls how much noise is added.
  - Over time, the image becomes indistinguishable from pure Gaussian noise.
- This is called forward process and is fixed no learnings happen here. (The forward process randomly adds noise depending on time step)

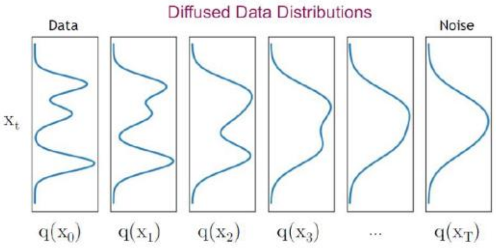
2. Key Simplification: instead of modeling every transition one by one, we can express x_t directly from x_0 using a reparametrized form.

$$x_t = \sqrt{\overline{\alpha}_t} x_0 + \sqrt{1-\overline{\alpha}_t}\,\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

- $\alpha_t = 1 - \beta_t$
- $\overline{\alpha_t} = \prod_{i=1}^{t} \alpha_i$
- So rather than computing each x_t via many steps, we can sample any x_t given x_0 in closed form. This is powerful and used during training to create noisy examples.

3. Understanding the Marginal Distribution of x_t:

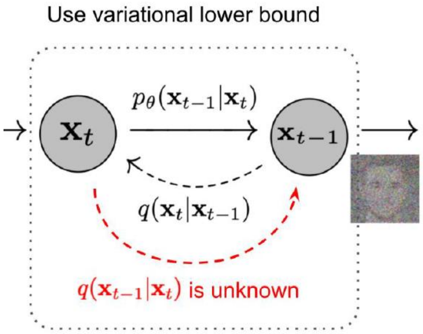
$$q(x_t) = \int q(x_0)\; q(x_t \mid x_0)\, dx_0$$



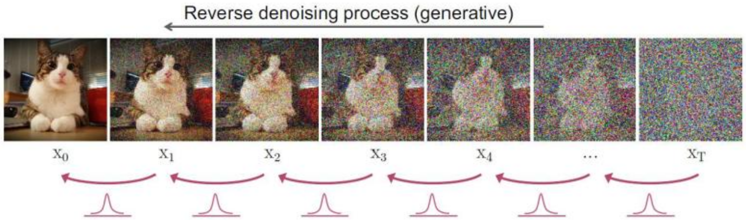
- This tells us how the data distribution evolves over time when noise is added.
- The visual graph of $q(x_t)$ changing over time shows the data becoming more and more like a standard Gaussian; the end result being pure noise.
- Key insight:
  - Sampling $x_0 \sim q(x_0)$
  - Then sampling $x_t \sim q(x_t | x_0)$
- This is known as ancestral sampling and simplifies data generation during training.

Reverse Diffusion Process

1. Main Goal: The key goal of reverse process is to go from pure noise $x_T \sim \mathcal{N}(0, I)$ to a real image x_0 by progressively removing noise.
   - Reverse Challenge:
     - We want to go backward: $x_T \to x_{T-1} \to \cdots \to x_0$.
     - But $q(\, x_{t-1} \mid x_t \,)$ is intractable (not known analytically).
     - So we learn a model $p_\theta(\, x_{t-1} \mid x_t \,)$ to approximate it.
   - We don't know the true reverse distribution $q(\, x_{t-1} \mid x_t \,)$, so we train a neural network to learn it. We use variational lower bond to do that.


Use variational lower bound
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is unknown

2. Summary of reverse process:



   - Assume:

$$p_\theta(\, x_{t-1} \mid x_t \,) = \mathcal{N}(x_{t-1};\, \mu_\theta(x_t, t),\, \sigma_t^2 I)$$

     - μ_θ: is predicted by the network.
     - σ_t^2: is either learned or fixed.
   - Then we form the full joint distribution:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(\, x_{t-1} \mid x_t \,)$$

   - This model learns to denoise from one time step to the previous, starting from pure Gaussian noise.
   - Once we predict the noise with a neural network, we can subtract it to be denoised, and step-by-step reach something close to the original image.

Learning A Denoising Model

1. We define a proper loss function to train this reverse process:
   - We want to maximize the likelihood of real data under our model: $log p_\theta(x_0)$.
   - That's intractable, so we use variational lower bound (VLB):

$$-\log p_\theta\,(x_0) \le E_q\left[\log \frac{q(\, x_{1:T} \mid x_0 \,)}{p_\theta(x_{0:T})}\right]$$

- Let this expectation be L_{VLB}, this is our loss:
    - It expresses how good the generation is. The KL divergence term shows how close the model's distribution is to the real process.

2. Now we break down the VLB into tractable terms:

$$\mathcal{L}_{\mathcal{VLB}} = E_q\left[D_{KL}(q(x_T \mid x_0) \mid p(x_T)) + \sum_{t=2}^{T} D_{KL}(q(x_{t-1} \mid x_t, x_0) \mid p_\theta(x_{t-1} \mid x_t)) - \log p_\theta(x_0 \mid x_1)\right]$$

- $q(x_T \mid x_0) \mid p(x_T)$: the end result of forward process (should be noise)
- $q(x_{t-1} \mid x_t, x_0)$: tractable posteriors (can be computed in closed form)
- $p_\theta(x_{t-1} \mid x_t)$: learned model we're training
- The key trick is:
    - We can compute the KL between Gaussians if both are Gaussian and have the same fixed variance which they do.
    - When both distributions are Gaussian with the same variance, KL becomes the squared difference between means.

Noise-Prediction Parametrization

1. Most efficient way to train the model: Instead of learning $\mu_\theta(x_t, t)$ Directly, we instead train the network to predict the noise ϵ added to get x_t:

$$x_t = \sqrt{\overline{\alpha}_t}x_0 + \sqrt{(1-\overline{\alpha_t)}\epsilon}$$

- Then the denoising model's mean becomes:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$$

- We add noise $\epsilon \sim \mathcal{N}(0, I)$ to x_0 get x_t
- Train the network to predict ϵ
- Loss is:

$$\mathcal{L}_{\text{sml}} = E[|\epsilon - \epsilon_\theta(x_t, t)|^2]$$

- In the final loss we only need to remember this: we compute x_t from x_0, add noise ϵ, train the network to predict the noise.

Training Objective Simplification

1. The loss function from earlier included a real-time dependent weight λ_t:

$$\lambda_t = \frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\overline{\alpha_t})}$$

- This weight scales the training loss at each time step.
- Key insight from Ho et al., NeurIPS 2020 article:

    - You can drop λ_t and instead use a simplified objective that ignores this weight leading to a better sample quality.
- This gives us the final practical loss used in most DDPMs:

$$\mathcal{L}_{\text{sml}} = E_{x_{\mathbb{0}},\epsilon,t}[|\epsilon - \epsilon_\theta(x_t,t)|^2]$$

    - This is the loss Pastore wants us to remember predicting the noise at any timestep and minimizing the MSE.
- So despite the complex derivation, we train the model by:
    - Sampling noise $\epsilon \sim \mathcal{N}(0,I)$.
    - Adding it to a clean image x_0 at timestep t.
    - Training the neural net to predict that noise from the noisy version x_t.

Summary Algorithms 1&2

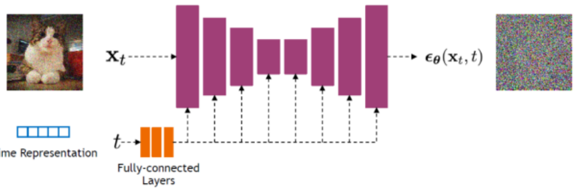
1. Algorithm 1: Training
    1) Sample a data point x_0.
    2) Choose a timestep $t{\sim}u(1,T)$.
    3) Sample noise $\epsilon \sim \mathcal{N}(0,I)$.
    4) Compute noisy image: $x_t = \sqrt{\overline{\alpha_t}}x_0 + \sqrt{1-\overline{\alpha_t}}$
    5) Use neural net $\epsilon_\theta(x_0,t)$ to predict ϵ.
    6) Minimize $||\epsilon{-}\epsilon_\theta(x_0.t)||^2$
    - The noise is predicted because we added it ourselves so we train the network to reverse it.
2. Algorithm 2: Sampling
    1) Start from pure noise $x_T{\sim}\mathcal{N}(0,I)$.
    2) Iteratively denoise from $x_T \to x_0$.

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha_t}}}\epsilon_\theta(x_t,t)\right) + \sigma_t z$$

    3) Add noise $z{\sim}\mathcal{N}(0,I)$ only if $t > 1$
    - For each image, I'm subtracting the predicted noise to move backward through the Markov chain.

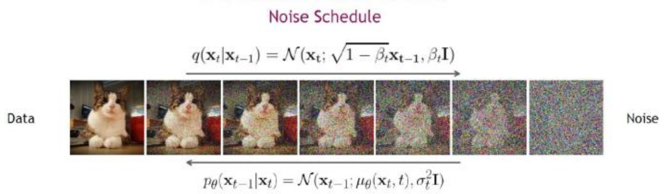
Implementation: U-Net Architecture

1. Architecture Highlights:
    - Input: Noisy Image x_t and timestep t
    - Time encoding: Positional Embedding (sinusoidal or Fourier)
    - Backbone: U-Net with ResNet and self-attention blocks
    - Output: predicted noise ϵ



2. This U-Net architecture allows:
    - Multi-scale feature extraction.
    - Rich detail reconstruction.
    - Explicit conditioning on diffusion timestep t.

Diffusion Parameters - Noise Scheduler


Noise Schedule
$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$
Data
Noise
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})$

1. Here we look at β_t and σ_t^2 which control:
   - Forward noise: how much noise is added at each step.
   - Reverse Uncertainty: variance of denoising model.
2. Typical Choices:
   - β_t: linearly increasing from small to large values.
   - σ_t^2: Often equals β_t though more advance versions learn it.
3. Recent work (Kingma et al. 2022, Bao et al. 2022) proposes:
   - Learning schedular based on SNR.
   - Analytic representation for stable training.

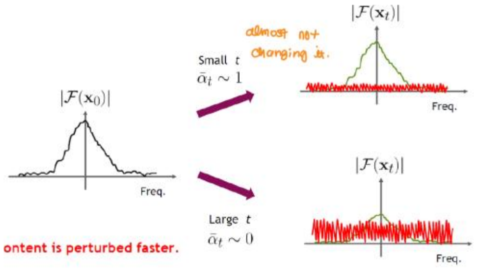
What Happens to Images in Forward Diffusion?

1. How different frequencies behave over time.

$$x_t = \sqrt{\overline{\alpha_t}}x_0 + \sqrt{1-\overline{\alpha_t}}$$

   - Taking Fourier Transform:

$$\mathcal{F}(x_t) = \sqrt{\overline{\alpha}_t}\,\mathcal{F}(x_0) + \sqrt{1-\overline{\alpha}_t}\,\mathcal{F}(\epsilon)$$



ontent is perturbed faster.

2. Interpretation:
   - At early timesteps: $\overline{\alpha_t} \approx 1$ so signal dominates.
   - At later timesteps: $\overline{\alpha_t}$—> 0, so noise is dominant.
3. Results:
   - High-frequency components (fine details) vanish faster.
   - Early x_t keeps structure, late x_t is almost pure noise.
4. First, we lose details, then broad content. In the reverse, we recover low-frequency (structure) first, the high-frequency (details).

Content Detail Trade-off

1. The denoising model first reconstructs low-frequency (coarse shapes).
2. Them, over many steps, it adds high-frequency details (texture, edges, sharpness).