# Deep Neural Networks:
## Some Assorted Topics

CS771: Introduction to Machine Learning

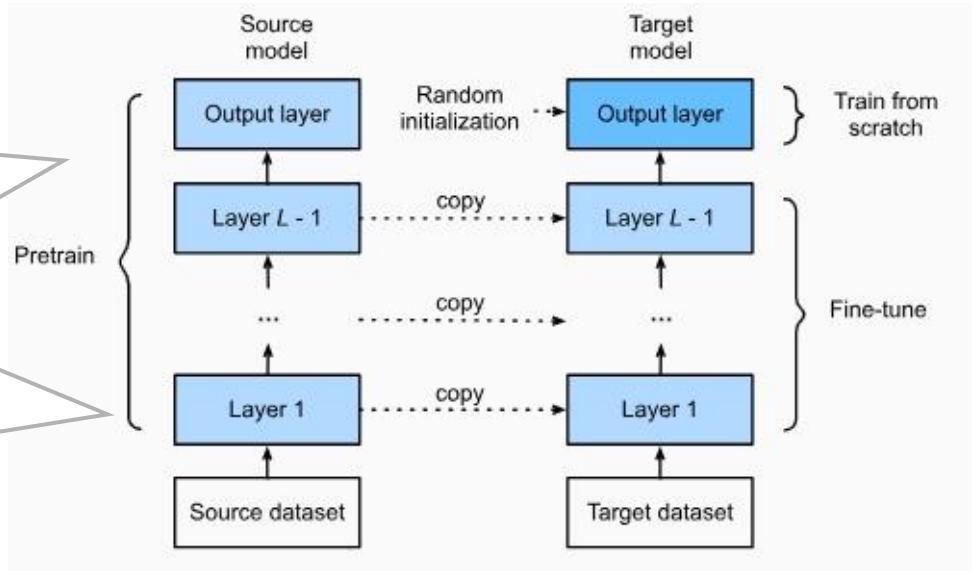
Piyush Rai

# Fine-tuning and Transfer Learning

- Deep neural networks trained on one dataset can be reused for another dataset
  - It is like transferring the knowledge of one learning task to another learning task
- This is typically done by "freezing" most of the lower layers and finetuning the output layer (or the top few layers) – this is known as "fine-tuning"

Initial model with frozen layers is called the "pre-trained" model and the updated model is called the "fine-tuned" model

This example is for an MLP like architecture but fine-tuning can be done for other architectures as well, such as RNN, CNN, transformers, etc




BERT (pre-trained in unsupervised manner) fine-tuned for a sentence classification task by adding a fully connect MLP to predict class-label of a sentence

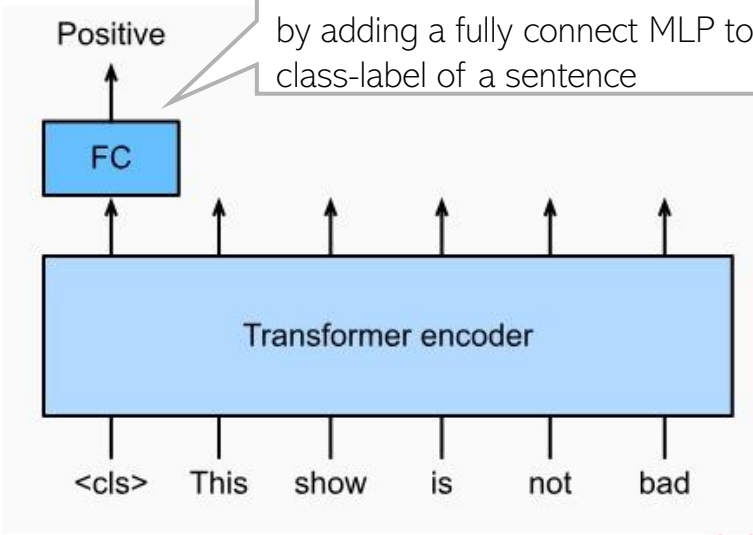
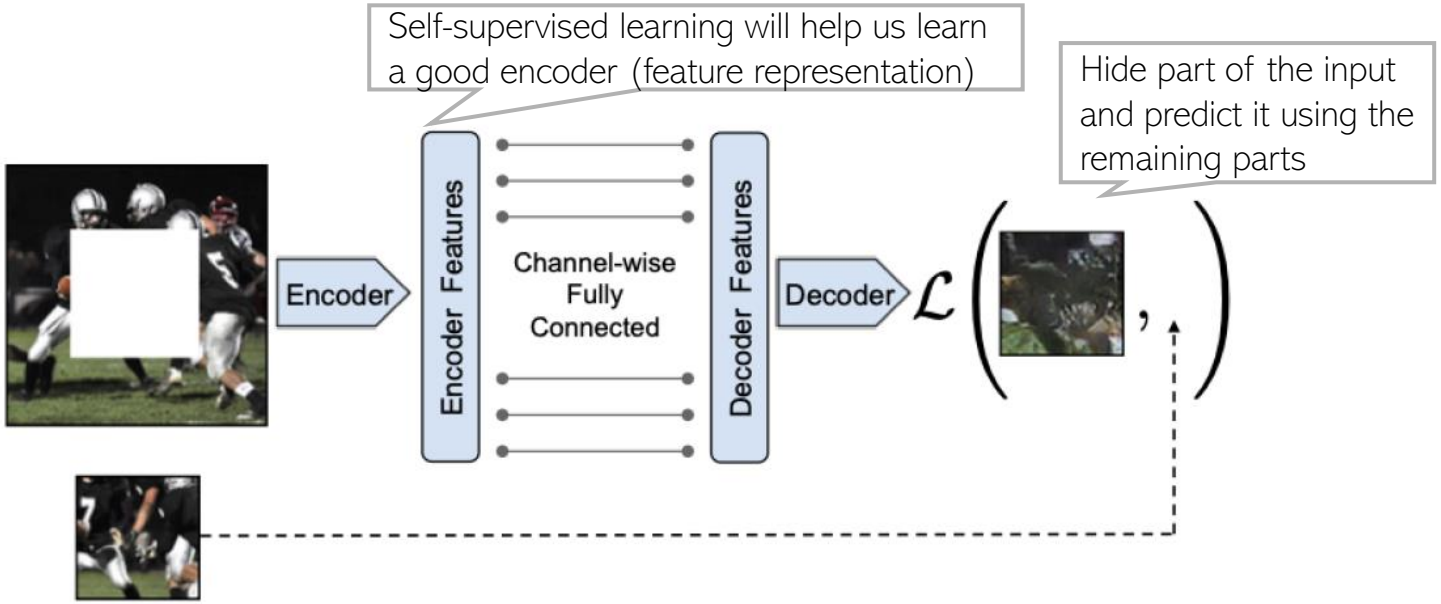
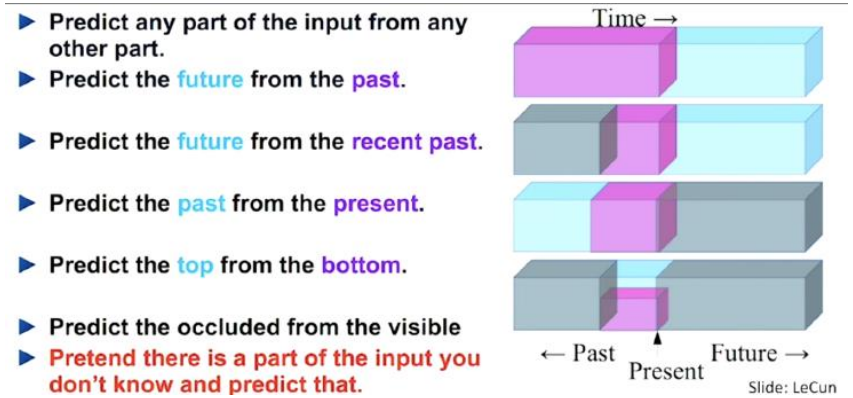



Figure source: Dive into Deep Learning (Zhang et al, 2023)

# Unsupervised Pre-training

- Self-supervised learning is a powerful idea to learn good representations unsupervisedly




- Self-supervised learning is key to unsupervised pre-training of deep learning models
  - Such pre-trained models can be fine-tuned for any new task given labelled data

- Models like BERT, GPT are usually pre-trained using self-supervised learning
  - Then we can finetune them further for a given task using labelled data for that task

# Auto-encoders

A special type of self-supervised learning: The whole input is being predicted by first compressing it and then uncompressing

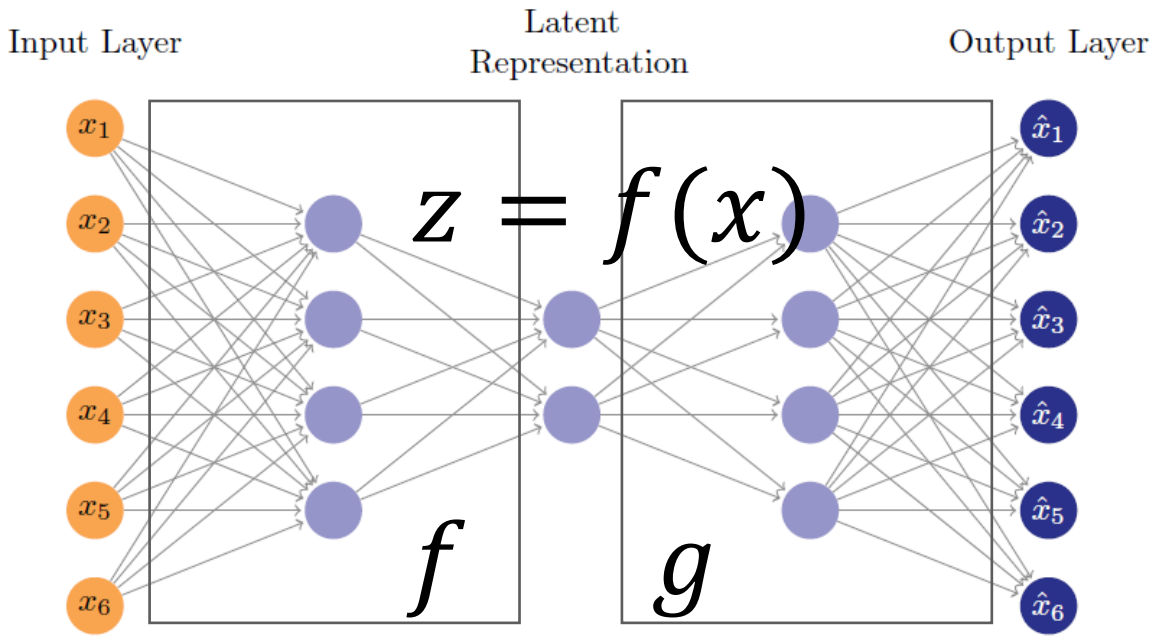
- Auto-encoders (AE) are used for unsupervised feature learning
- Consist of an encoder $f$ and a decoder $g$
  - $f$ and $g$ can be deep neural networks (MLP, RNN, CNN, etc)

Note: Usually only the encoder is of use after the AE has been trained (unless we want to use the decoder for reconstructing the inputs later)

If using a prior on $\boldsymbol{z}$, we can a probabilistic latent variable model called variational auto-encoder (VAE)

VAE can also generate synthetic data usings its decoder (standard AE's decoder can't generate "new" data)

$$\hat{x} = g(f(x))$$




Goal: Learn $f$ and $g$ s.t. $\|x - \hat{x}\|$ is small

Dimensionality of $\boldsymbol{z}$ can be chosen to be smaller or larger than that of $\boldsymbol{x}$

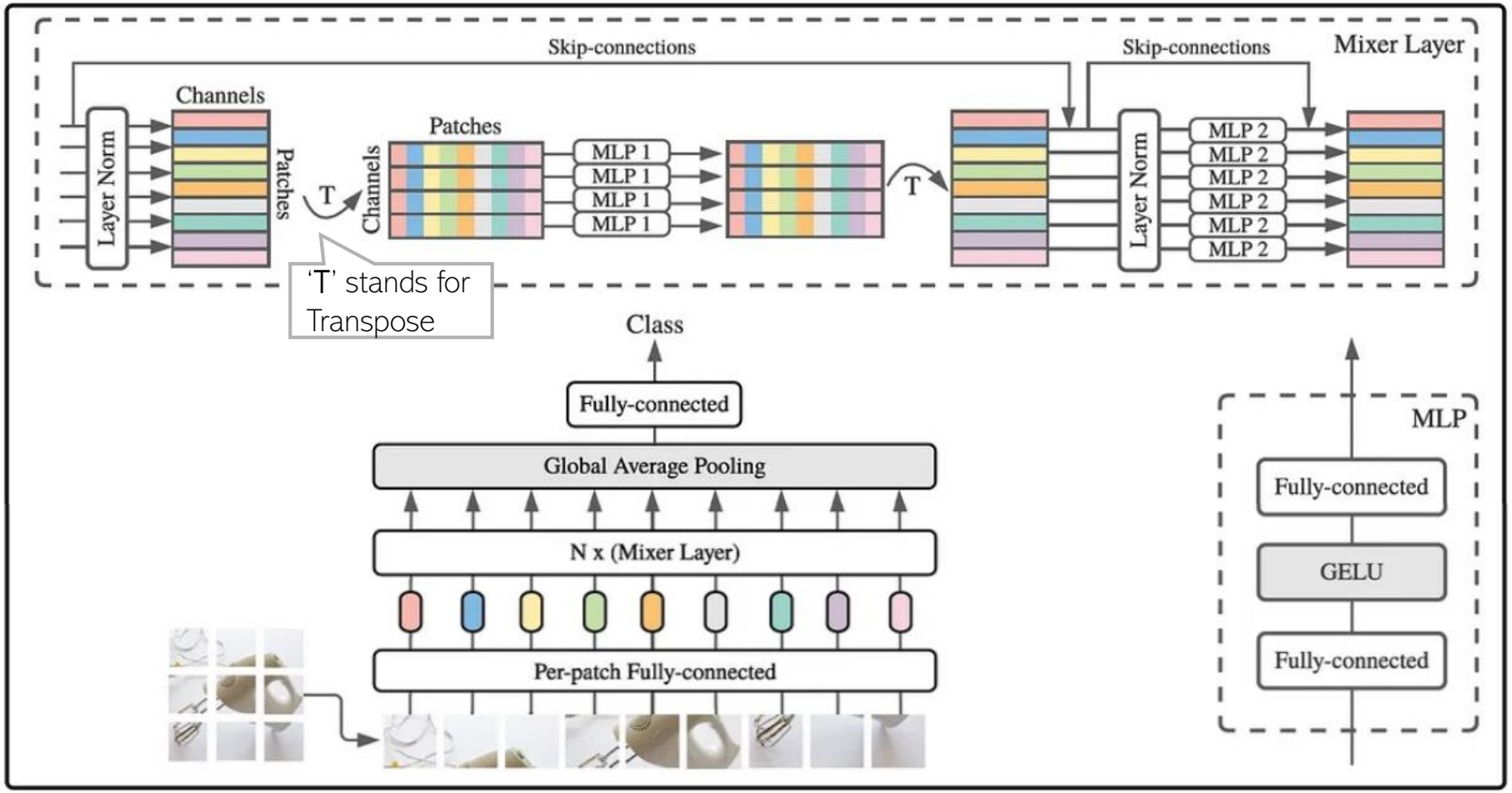
If using AE for dimensionality reduction

Sometimes we want to learn "overcomplete" feature representations of the input

In such cases, need to impose additional constraints on $f$ so that we don't learn an identify mapping from $x$ to $z$

# Convolution-less Models for Images: MLP-mixer

- Many MLPs can be mixed to construct more powerful deep models ("MLP-mixer")




MLP-Mixer: Network Architecture

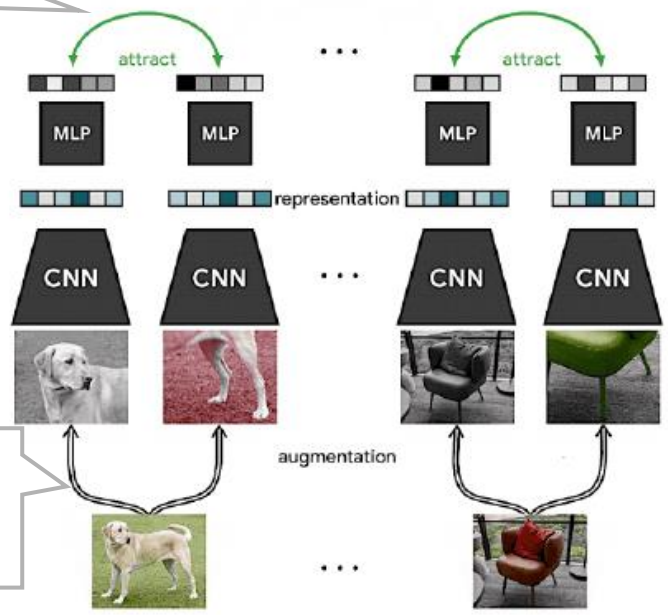
MLP-Mixer: An all-MLP Architecture for Vision (Tolstikhin et al, 2021)

# Contrastive Learning

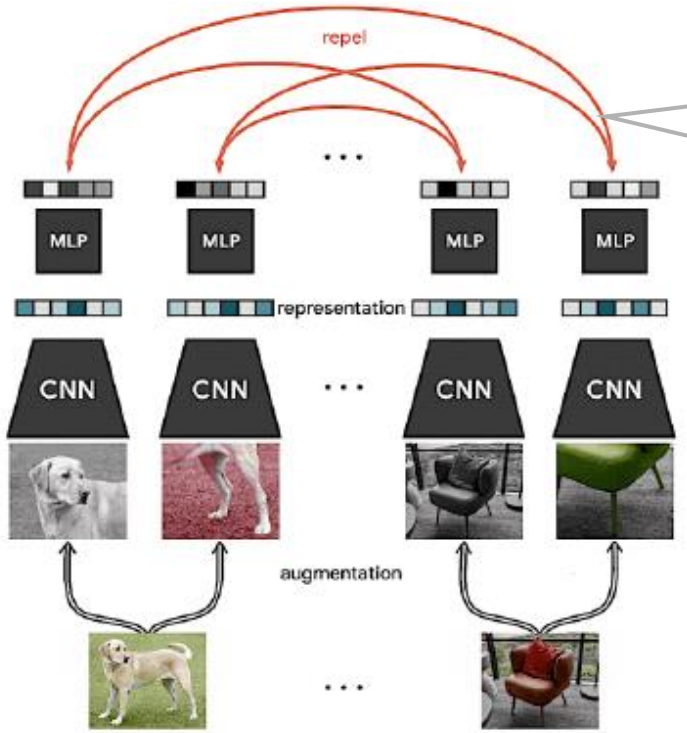
Or "triplets" (e.g., "cat" is more similar to "dog" than to a "table")

- Can learn good features by comparing/"contrasting" similar and dissimilar object pairs
- Such pairs can be provided by to the algorithm (as supervision), or the algorithm can generate such pairs by itself using "data augmentation" (as shown in example below)

The corresponding embeddings for such pairs must be close ("attract") to each other



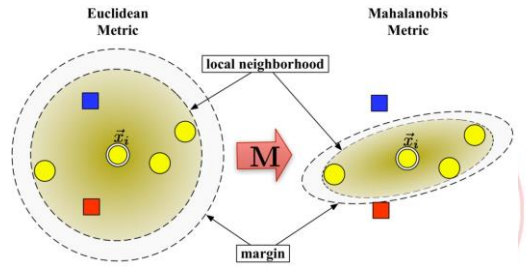

Augmentation by cropping and resizing. The class of the image remains unchanged in this augmentation




The embeddings for "non-match" pairs must be far away ("repel") from each other

Distance metric learning by learning $\mathbf{M}$ given similar/dissimilar pairs

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \mathbf{M} (\boldsymbol{x}_i - \boldsymbol{x}_j)}$$

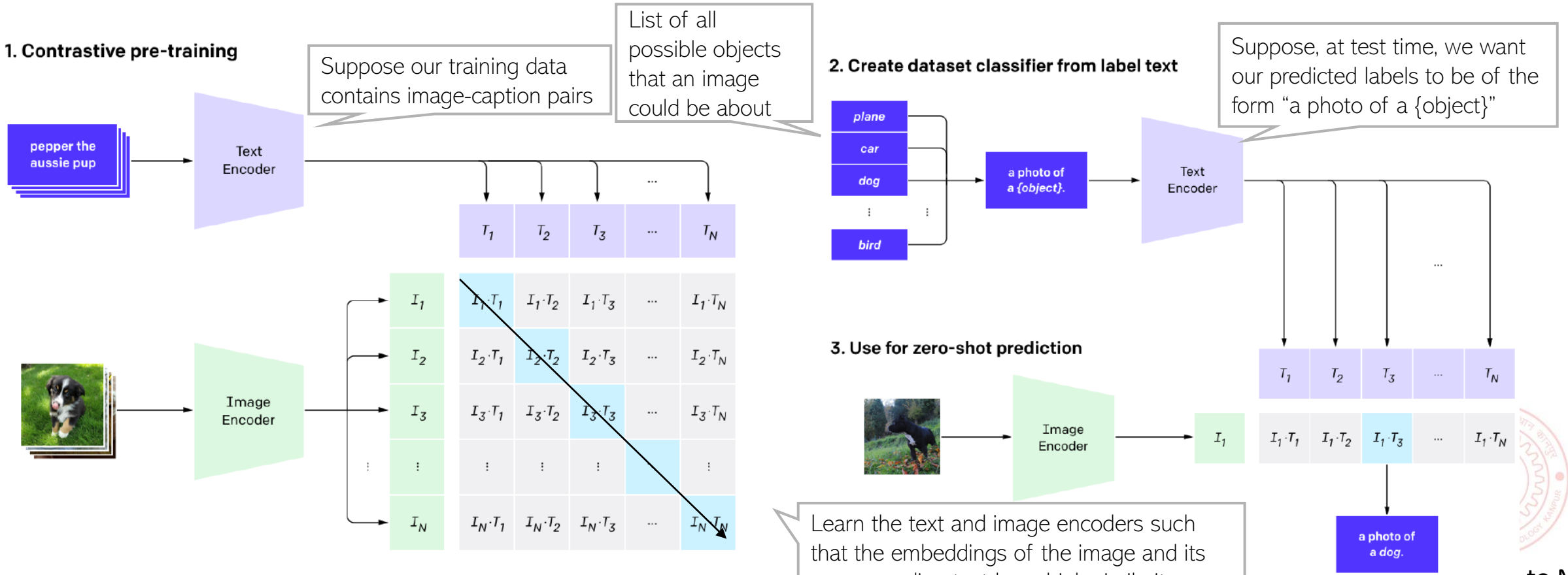



- Such "contrastive learning" of features is also related to "distance metric learning" algos

# Zero-Shot Learning and CLIP

- What if our training data doesn't have the test data classes?
- Several methods to solve ZSL using deep learning. CLIP is a recent approach

CLIP: "Contrastive Language-Image Pre-training" (Radford et al, 2021)



# Bias-Variance Trade-off

- Assume $\mathcal{F}$ to be a class of models (e.g., linear classifiers with some pre-defined features)
- Suppose we've learned a model $f \in \mathcal{F}$ learned using some (finite amount of) training data
- We can decompose the test error $\epsilon(f)$ of $f$ as follows

E.g., going from linear models to deep nets or by adding more features

Can bias reduce by making class $\mathcal{F}$ richer

$$\epsilon(f) = \underbrace{\left[\min_{f^* \in \mathcal{F}} \epsilon(f^*)\right]}_{\text{approximation error}} + \underbrace{\left[\epsilon(f) - \min_{f^* \in \mathcal{F}} \epsilon(f^*)\right]}_{\text{estimation error}}$$

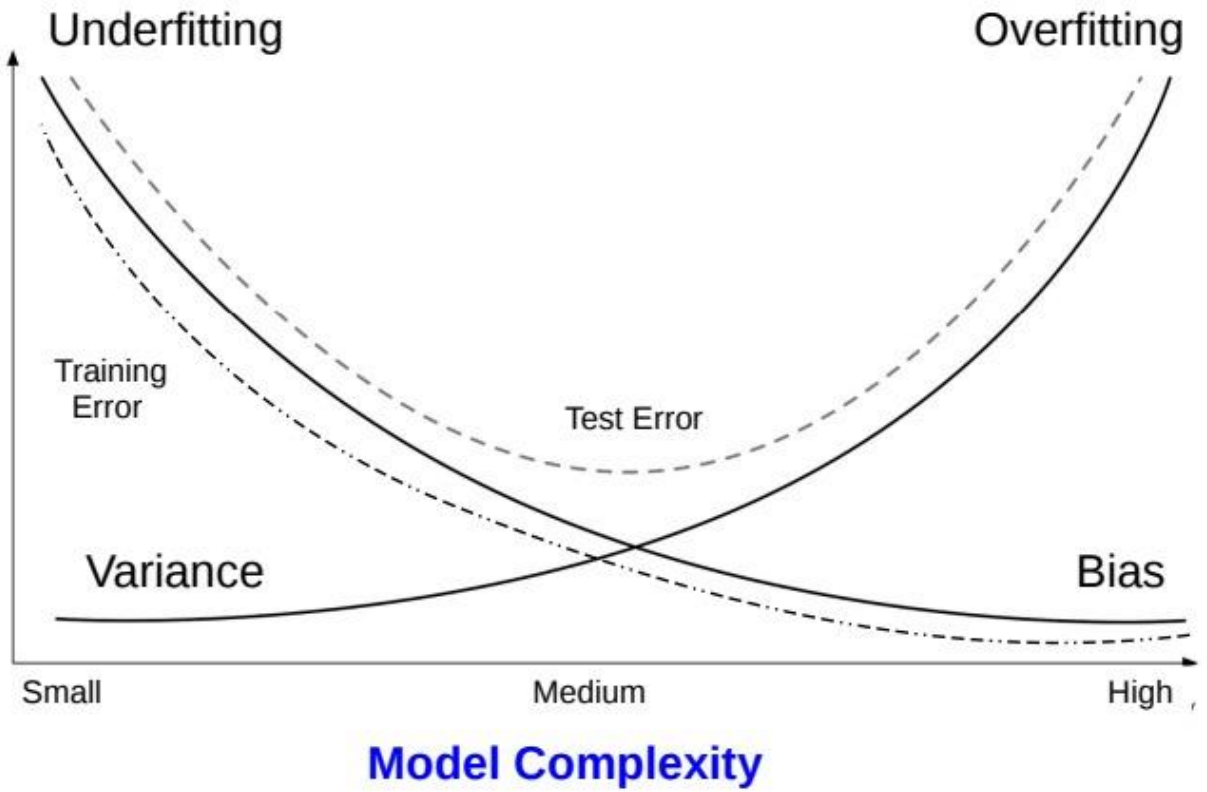
Reason: We are now learning a more complex model using the same amount of training data

Making $\mathcal{F}$ richer will also cause estimation error to increase

- Here $f^*$ is the best possible model in $\mathcal{F}$ assuming infinite amount of training data
- Approximation error: Error of $f^*$ because of model class $\mathcal{F}$ being too simple
  - Also known as "bias" (high if the model is simple)
- Estimation error: Error of $f$ (relative to $f^*$) because we only had finite training data
  - Also known as "variance" (high if the model is complex)
- Because we can't keep both low, this is known as the bias-variance trade-off

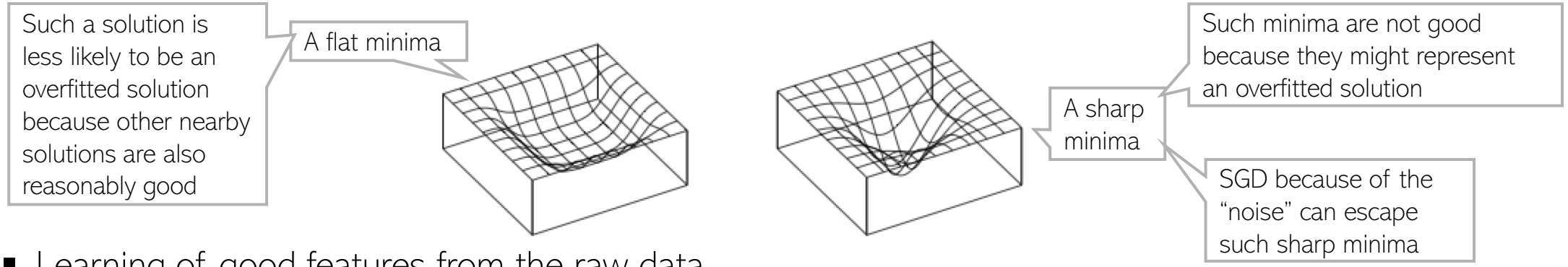
# Bias-Variance Trade-off

- Bias-variance trade-off implies how training/test losses vary as we increase model complexity



# Deep Neural Nets and Bias-Variance Trade-off

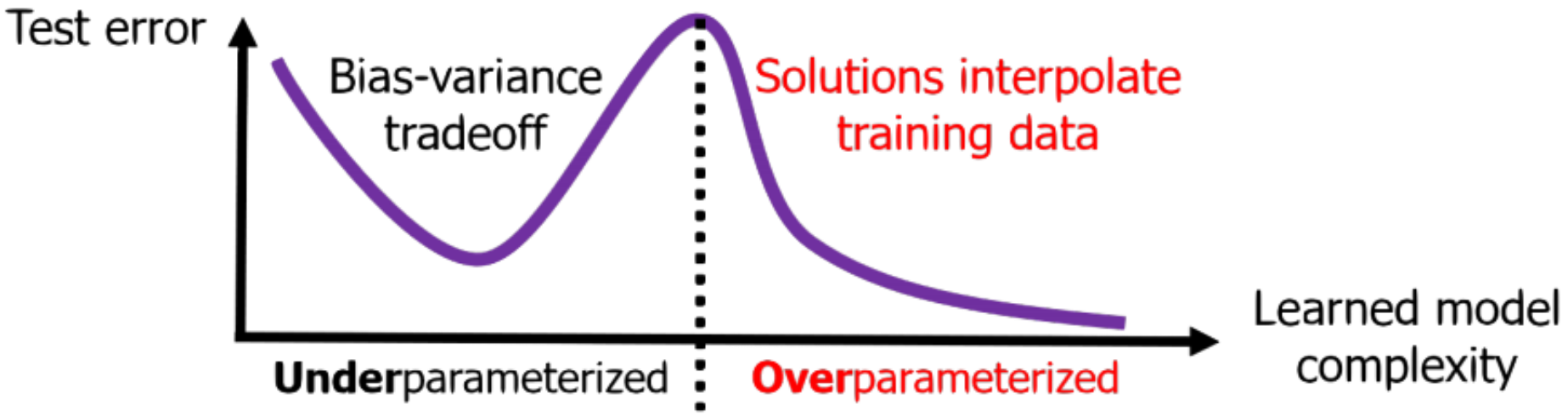
- Bias-variance trade-off doesn't explain well why deep neural networks work so well
  - They have very large model complexity (massive number of parameters – massively "overparametrized")
- Despite being massively overparametrized, deep neural nets still work well because
  - Implicit regularization: SGD has noise (randomly chosen minibatches) which performs regularization
  - These networks have many local minima and all of them are roughly equally good
  - SGD on overparametrized models usually converges to "flat" minima (less chance of overfitting)




  - Learning of good features from the raw data
  - Ensemble-like effect (a deep neural net is akin to an ensemble of many simpler models)
  - Trained on very large datasets

# Double Descent Phenomenon

- Overparametrized deep neural networks exhibit a "double descent" phenomenon



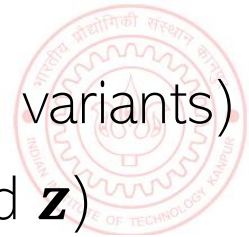

- Bias-variance trade-off seen only in the underparametrized regime
- Beyond a point (in the overparametrized regime), the test error starts decreasing once again even as the model gets more and more complex

Figure source: "A Farewell to the Bias-Variance Tradeoff? An Overview of the Theory of Overparameterized Machine Learning" (Dar et al, 2021)

# Deep Neural Networks: A Summary

# Common Types of Layers used in Deep Learning

- Linear layer: Have the form $\boldsymbol{W}^\top \boldsymbol{x}$ (used in fully connected networks like MLP and also in some parts of other type of models like CNN, RNN, transformers, etc)
- Nonlinearity: Activation functions (sigmoid, tanh, ReLU, etc)
  - Essential for any deep neural network (without them, deep nets can't learn nonlinear functions)
- Convolutional layer: Have the form $\boldsymbol{W} * \boldsymbol{x}$ (here * denotes the conv operation)
  - Usually used in conjunction with pooling layers (e.g., max pooling, average pooling)
- Residual or skip connections: Help when learning very deep networks (e.g., ResNets, transformers, etc) by avoiding vanishing/exploding gradients
- Normalization layer such as batch normalization and layer normalization
- Dropout layer: Helps to regularize the network
- Recurrent layer: Used in sequential data models such as RNNs and variants
- Attention layer: Used in encoder-decoder models like transformers (also in some RNN variants)
- Multiplicative layer: Have the form $\boldsymbol{x}^\top \boldsymbol{W} \boldsymbol{z}$ (used when each input has two parts $\boldsymbol{x}$ and $\boldsymbol{z}$)



# Popular Deep Learning Architectures

- MLP: Feedforward fully connected network
  - Not preferred when inputs have spatial/sequential structures (e.g., image, text)
  - Some variants of MLP (e.g., MLP-mix) perform very well on such data as well
- CNN: Feedforward but NOT fully connected (but last few layers, especially output, are)
- RNNs: Not feedforward (hidden state of one timestep connects with that of the next)
- Transformers: Very powerful models for sequential data
  - Unlike RNNs, can process inputs in parallel. Also uses (self) attention to better capture long range dependencies and context in the input sequence
- Graph Neural Networks: Used when inputs are graphs (e.g., molecules)
- Autoencoders and Deep Generative Models: For unsupervised representation learning and synthetic data generation tasks