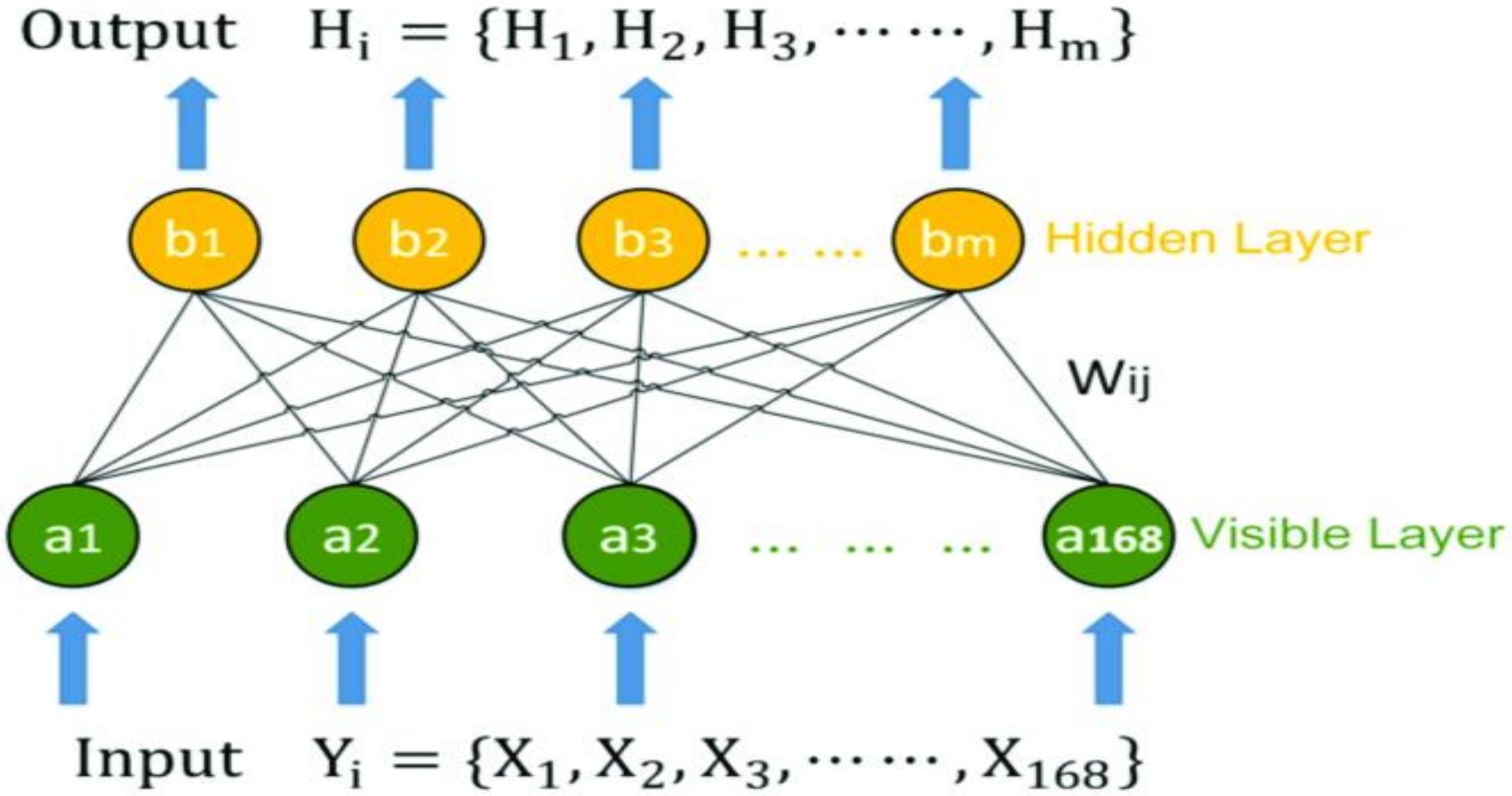
Unit V -Representation Learning

Greedy Layerwise Pre-training

Greedy Layerwise Pre-training

- New approach to boost the performance and efficiency of neural networks.
- A method for layer-by-layer training deep neural networks is called greedy layer-wise pre-training.
- It includes training each layer of a neural network individually in a greedy way, starting with the first layer and advancing to the last layer.
- Idea behind ,is that it allows each layer to learn features that are useful for the next layer, thereby improving the overall performance of the network.

- Using a <u>**restricted Boltzmann machine (RBM)**</u> an unsupervised neural network, first trained on the input data in greedy layer-wise pre-training.
- This unsupervised pre-training stage is used to initialize the weights of the neural network.
- After initializing the weights, the network can be adjusted using supervised learning to enhance its performance on a particular task.
- Once a layer’s weights have been mastered, they are frozen and the following layer is trained using those weights.
- Up till all the levels have received training, this process is repeated.
- RBM consists of two layers of neurons –
- <u>**visible layer and hidden layer.**</u>
- The visible layer represents the input data, while the hidden layer represents a set of features that are learned by the network.


Output $H_i = \{H_1, H_2, H_3, \cdots\cdots, H_m\}$
b1
b2
b3
... ...
bm
Hidden Layer
Wij
a1
a2
a3
... ...
a168
Visible Layer
Input $Y_i = \{X_1, X_2, X_3, \cdots\cdots, X_{168}\}$

Benefits of Greedy Layer-wise Pre-training

- It resolves the vanishing gradients issue that deep neural networks sometimes experience.
- The network can learn better representations of the input data by initializing the weights.
- Helps to reduce overfitting in deep neural networks.
- The network can learn features that are helpful for the next layer.

Representation learning

- *Representation learning is a technique that allows a machine to automatically discover the features or representations needed for a specific task from raw data.*
- It can reduce the need for manual feature engineering and make the machine learning process more efficient and effective.
- Representation learning can be applied to different types of data, such as images, text, audio, video, etc.

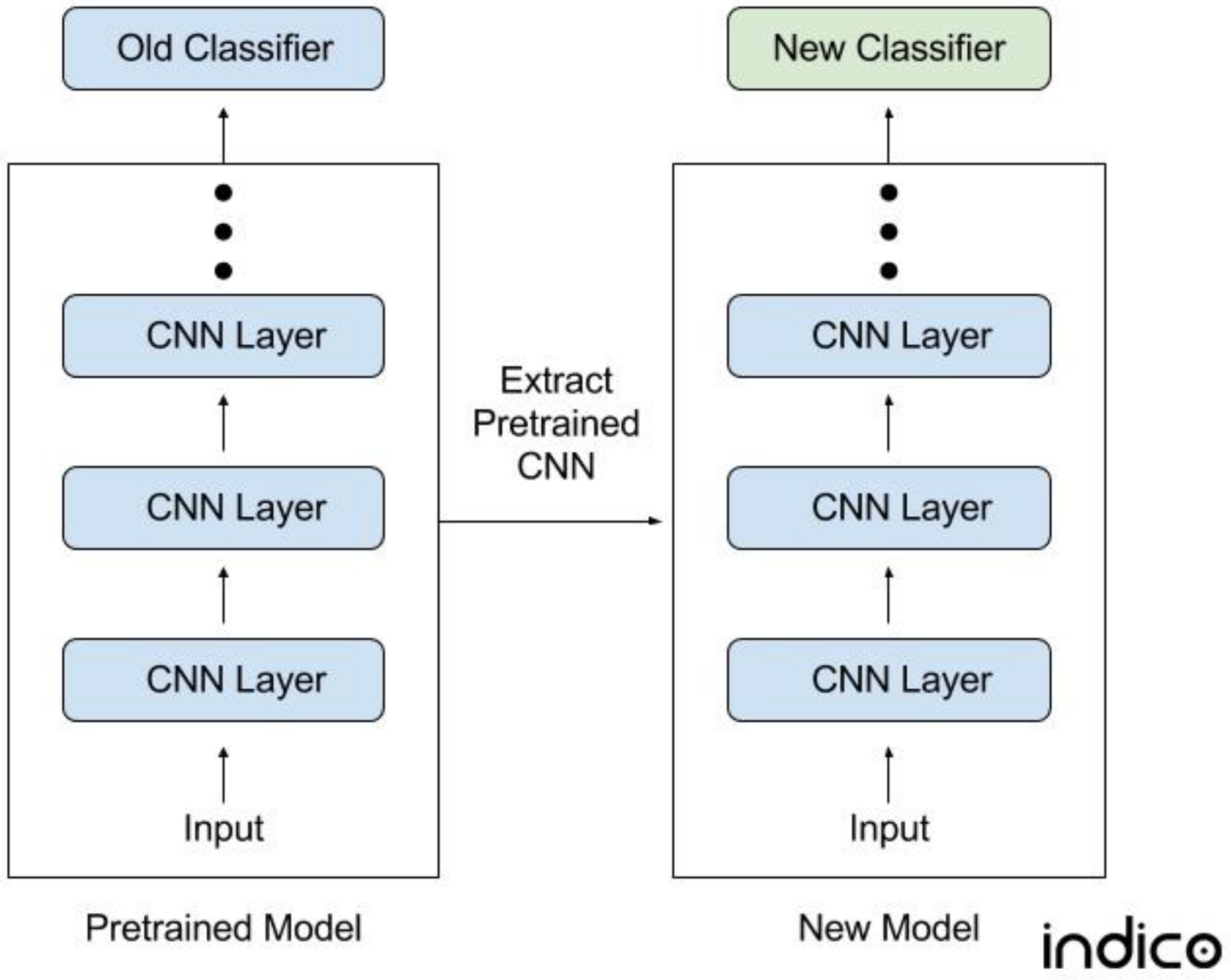
- Some examples of representation learning methods are:
- **Autoencoders**.
- **Word embeddings**.
- These are vector representations of words that capture their semantic and syntactic similarities and relations.
- They can improve the performance of natural language processing tasks such as sentiment analysis, machine translation, text classification, etc.
- **Convolutional neural networks (CNNs).**

Transfer learning

- Transfer learning, used in machine learning, is the reuse of a pre-trained model on a new problem.
- In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another.
- For example, in training a classifier to predict whether an image contains food, you could use the knowledge it gained during training to recognize drinks.
- The general idea is to use the knowledge a model has learned from a task with a lot of available labeled training data in a new task that doesn't have much data.

- Transfer learning is mostly used in computer vision and natural language processing tasks like sentiment analysis.
- huge amount of computational power required.
- **How Transfer Learning Works?**
- In computer vision, for example
- Neural networks usually try to detect edges in the earlier layers, shapes in the middle layer and some task-specific features in the later layers.
- In transfer learning, the early and middle layers are used and we only retrain the latter layers.
- It helps leverage the labeled data of the task it was initially trained on.
- Transfer learning has several benefits, saving training time, better performance of neural networks (in most cases), and not needing a lot of data.

- In transfer learning, we try to transfer as much knowledge as possible from the previous task the model was trained on to the new task at hand.

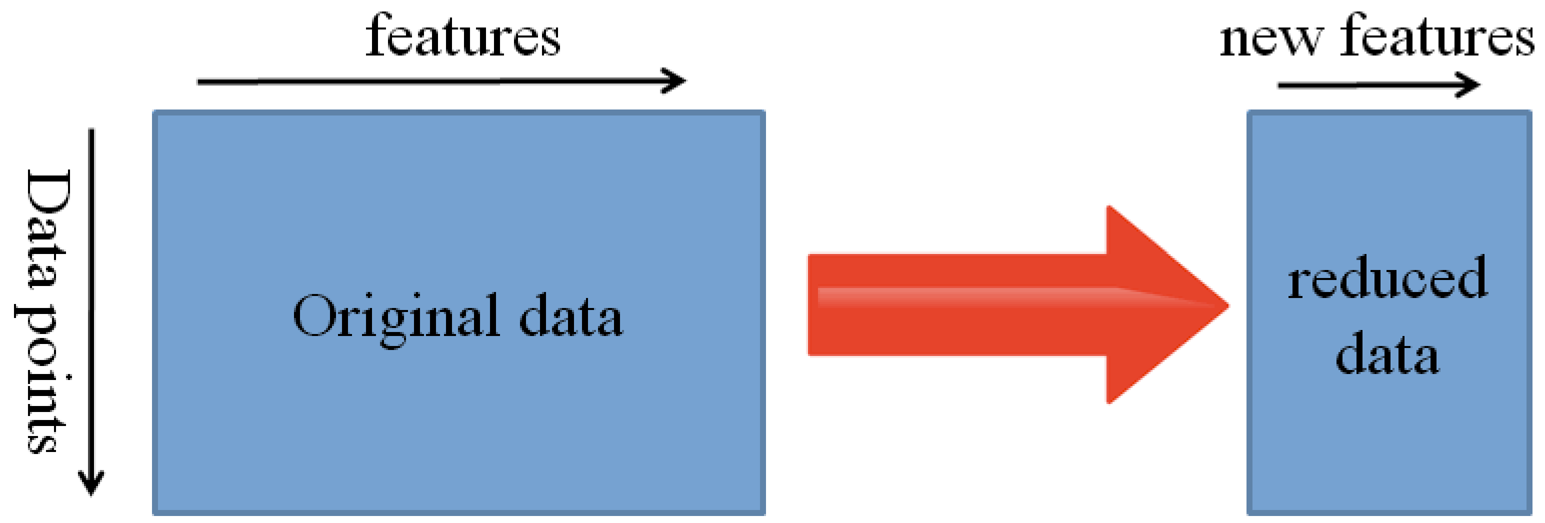


Approaches to Transfer Learning

- **1. TRAINING A MODEL TO REUSE IT**
- To solve task A but don’t have enough data to train a deep neural network.
- One way around this is to find a related task B with an abundance of data.
- Train the deep neural network on task B and use the model as a starting point for solving task A.
- Use the whole model or only a few layers depends on the problem.
- Same input in both tasks, possibly reusing the model and making predictions for new input is an option.

- **2. USING A PRE-TRAINED MODEL**
- Use an already pre-trained model.
- There are a lot of these models out there, so make sure to do a little research.
- How many layers to reuse and how many to retrain depends on the problem.
- Keras, provides numerous pre-trained models that can be used for transfer learning, prediction, feature extraction and fine-tuning.
- There are many research institutions that release trained models.
- This type of transfer learning is most commonly used throughout deep learning.

- **3. FEATURE EXTRACTION**

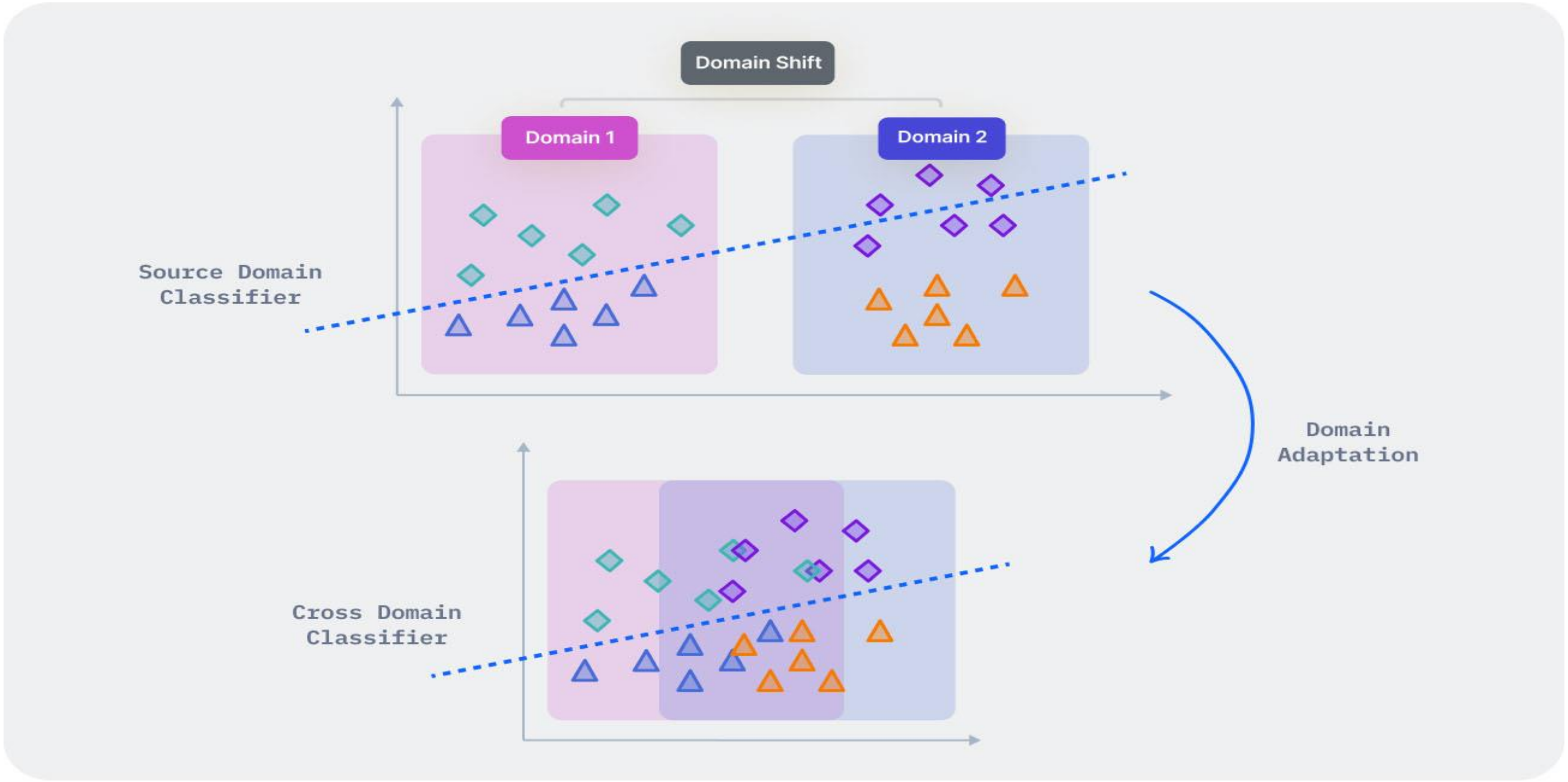


- **POPULAR PRE-TRAINED MODELS**
- Inception-v3 model, which was trained for the ImageNet “Large Visual Recognition Challenge.”
- In this challenge, participants had to classify images into 1,000 classes like “zebra,” “Dalmatian” and “dishwasher.”
- Microsoft offers MicrosoftML R package and the Microsoftml Python package.
- Other popular models are ResNet and AlexNet.

Domain Adaptation

- *Domain adaptation is a specific form of transfer learning that deals with the challenge of adapting a model from one source domain to another target domain, where the data distributions may be different.*
- This is crucial when the training data and the real-world data different.
- Domain adaptation aims to bridge this gap and improve the model's performance on the target domain.
- **Key Concepts in Domain Adaptation:**
- **Source Domain:** This is the domain where the model is initially trained.
- It has a labeled dataset, and the model learns to make predictions based on this data.

- **Target Domain:** This is the domain where you want the model to perform well.
- It may have a labeled or unlabeled dataset, but it's different from the source domain, either due to variations in data distribution or other factors.
- **Domain Shift:** The term "domain shift" refers to the differences in data distribution between the source and target domains.
- These differences can include variations in data statistics, feature representations, or label distributions.
- **Adaptation:** Domain adaptation techniques aim to adapt the model's knowledge from the source domain to the target domain, effectively reducing the impact of domain shift.


Domain Shift
Domain 1
Domain 2
Source Domain Classifier
Domain Adaptation
Cross Domain Classifier

Common Types of Domain Adaptation

- **Instance-based Domain Adaptation:**
- **Instance Weighting:** In this approach, instances (data points) from the source domain are assigned weights to emphasize or de-emphasize their importance during model training on the target domain.
- **Instance Selection:** Only a subset of instances from the source domain that are considered most relevant to the target domain is used for training.
- **Feature-based Domain Adaptation:**
- **Feature Replication:** Features that are common between the source and target domains are replicated or directly used for the target domain.

- **Feature Selection:** Relevant features are selected or transformed to align the feature spaces of both domains.
- **Principal Component Analysis (PCA):** PCA is applied to reduce dimensionality and align feature distributions between domains.
- **Canonical Correlation Analysis (CCA):** CCA identifies linear combinations of features in both domains that have maximum correlation, helping in domain alignment.
- **Model-based Domain Adaptation:**
- **Adaptive Layers:** Additional layers are added to the model to adapt it to the target domain.
- These layers may be domain-specific or learn to adapt features.

- **Domain Adversarial Training:** A domain classifier is added to the model, and the model is trained to confuse the domain classifier while performing well on the main task.
- This encourages the learning of domain-invariant features.
- **Fine-tuning:** Pre-trained models on the source domain are fine-tuned on the target domain data to adapt to the new distribution.
- **<u>Domain Adversarial Methods:</u>**
- **Domain Adversarial Neural Networks (DANN):** DANNs include a domain classifier that distinguishes between source and target domains.
- The model is trained to minimize the domain classification loss while maximizing the main task's performance.
- **Gradient Reversal Layer (GRL):** GRL is used to reverse the gradients during training, making the model focus on domain-invariant features.

- **Unsupervised, Semi-supervised, and Supervised:**
- Domain adaptation approaches can also be categorized based on the availability of labeled data in the target domain.
- Unsupervised domain adaptation uses no labeled data in the target domain.
- Semi-supervised uses limited labeled data.
- and Supervised domain adaptation leverages ample labeled data.

Unsupervised domain adaptation

- The primary goal is to adapt a model from a source domain to a target domain without using any labeled data from the target domain.
- In UDA, you have a source domain with labeled data and a target domain with unlabeled data.
- **The Unsupervised Domain Adaptation Process:**
- The UDA process typically involves the following steps:
- **Source Domain Pre-training:**
- Pre-train a model on the labeled data from the source domain.
- This initial model captures knowledge about the source domain and the task associated with it.

- **Feature Alignment:**
- Modify the model or add components to align the feature representations of the source and target domains.
- This alignment aims to make the learned features more domain-invariant and reduces the impact of domain shift.
- Common techniques include domain adversarial training, feature-based alignment, or fine-tuning with domain-specific normalization layers.
- **Unlabeled Target Data:**
- Use the unlabeled data from the target domain to further refine the model.
- The model is fine-tuned on the target domain data, leveraging the domain-invariant features learned in the previous step.

- **Evaluation:**
- Evaluate the adapted model's performance on the target task using validation or test data from the target domain.
- The goal is to achieve good performance on the target domain, even though it had no labeled data during training.

Supervised Domain Adaptation

- The primary goal is to adapt a model trained on a labeled source domain to perform well on a related but different target domain, utilizing both labeled source data and limited labeled target data.
- SDA the advantage of having access to a small amount of labeled target domain data to fine-tune the model.
- **The Supervised Domain Adaptation Process:**
- The SDA process typically involves the following steps:

1.Source Domain Pre-training:

- Pre-train a model on the labeled data from the source domain.
- This initial model captures knowledge about the source domain and the task associated with it.

- **Feature Alignment:**
- Modify the model or add components to align the feature representations of the source and target domains.
- Feature alignment aims to reduce the impact of domain shift, making the learned features more domain-invariant.
- Common techniques include domain adversarial training, fine-tuning with domain-specific normalization layers, or explicit domain alignment modules.
- **Target Domain Fine-tuning:**
- Fine-tune the model using the limited labeled data from the target domain.
- This fine-tuning step helps the model adapt to the target domain and improve its performance.

- **Evaluation:**
- Evaluate the adapted model's performance on the target task using validation or test data from the target domain.
- The goal is to achieve good performance on the target domain, leveraging both source and target domain knowledge.

Semi-supervised adaptation

- The primary goal is to adapt a model from a labeled source domain to perform well on a related but different target domain, utilizing a small amount of labeled target data and a larger amount of unlabeled target data.
- **The Semi-supervised Domain Adaptation Process:**
- The SSDA process typically involves the following steps:

1. **Source Domain Pre-training:**
2. Pre-train a model on the labeled data from the source domain.
3. This initial model captures knowledge about the source domain and the task associated with it.

- **Feature Alignment:**
- Modify the model or add components to align the feature representations of the source and target domains.
- Feature alignment aims to reduce the impact of domain shift, making the learned features more domain-invariant.
- Common techniques include domain adversarial training, fine-tuning with domain-specific normalization layers, or explicit domain alignment modules.
- **Target Domain Fine-tuning with Labeled Data:**
- Fine-tune the model using the limited labeled data from the target domain.
- This fine-tuning step helps the model adapt to the target domain and improve its performance on the target task.

- **Unsupervised Feature Learning:**
- Utilize the large pool of unlabeled target domain data to learn domain-invariant features.
- This is often achieved by training the model to minimize the discrepancy between the source and target domain feature distributions using unsupervised learning techniques.
- **Joint Learning:**
- Some SSDA methods involve joint training, where the model simultaneously optimizes multiple objectives, including classification on the labeled target data and domain alignment on the unlabeled target data.

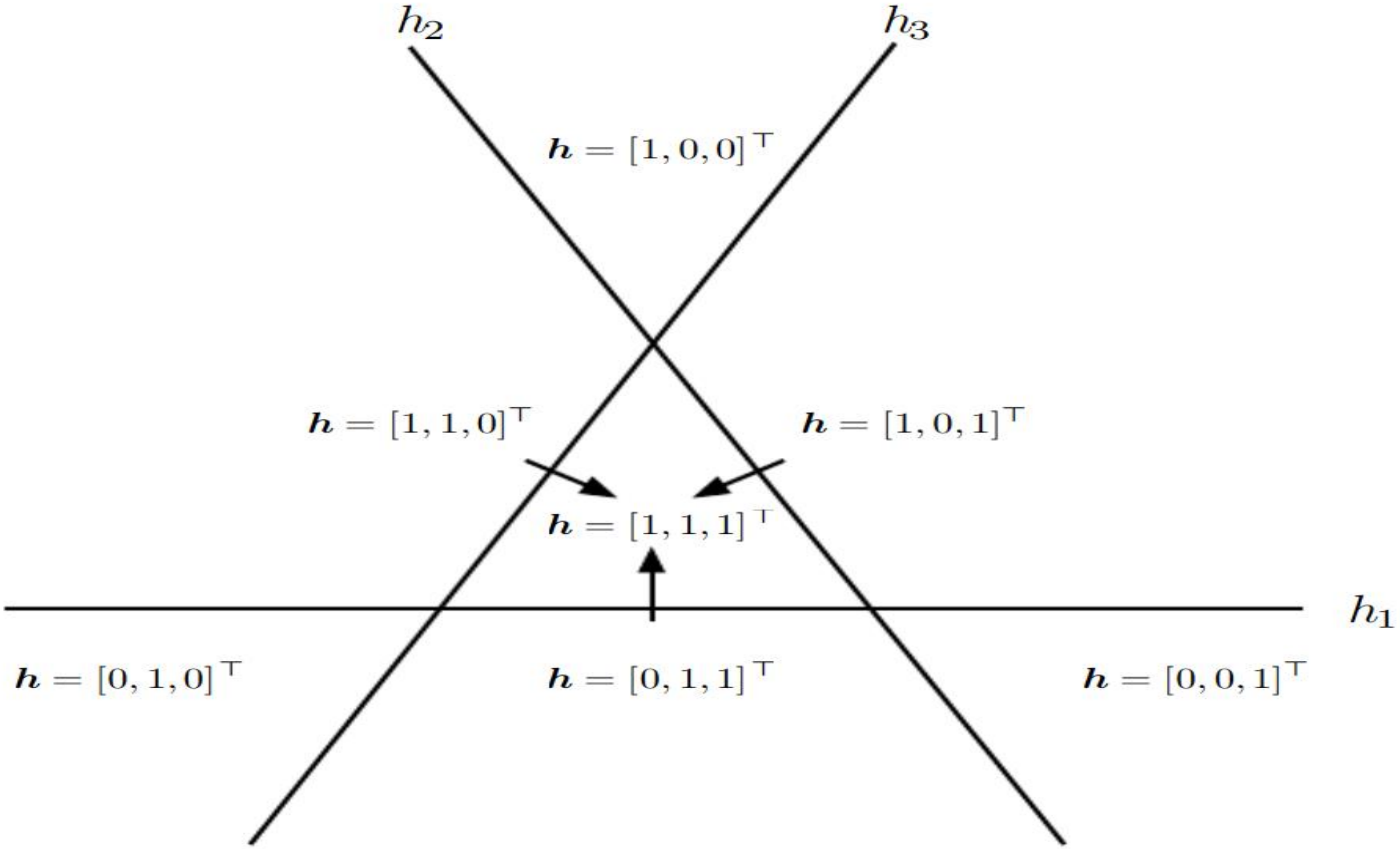
- **Evaluation:**
- Evaluate the adapted model's performance on the target task using validation or test data from the target domain.
- The goal is to achieve good performance on the target domain, leveraging both source and target domain knowledge.

Distributed Representation

- Distributed representation is a way of representing data or information by spreading it across multiple components or features.
- **Example:** Think of a picture puzzle. Instead of having one big piece that shows the entire picture, you have many smaller pieces, each showing a part of the picture.
- In distributed representation, information is broken down into smaller parts, and each part contributes to the overall representation.
- **Advantages:** Distributed representations are powerful because they can capture complex patterns and relationships in data.

- **Input Space:**
- The input space refers to the set of all possible inputs or examples that a machine learning model can receive.
- **Example:** Building a machine learning system to recognize animals, the input space would include all possible images of animals such as pictures of cats, dogs, birds, and more.
- **Distributed Representation and Input Space**
- Distributed representation can be especially useful when dealing with complex input spaces.
- For example, Working with images of animals in various poses and backgrounds (a complex input space), you can represent each image with distributed features.

- A learning algorithm based on a distributed representation breaks up the input space into regions.
- Three binary features *h*1,*h*2 and *h*3, each defined by thresholding the output of a learned linear transformation.
- Binary ,take on one of two values (0 or 1).
- Thresholding is a process where you compare the output of a transformation to a certain threshold value.
- If the output is greater than or equal to the threshold, the feature is set to 1; otherwise, it's set to 0.
- This results in binary features that are sensitive to specific conditions or patterns in the data.


h_2
h_3
$\boldsymbol{h} = [1,0,0]^\top$
$\boldsymbol{h} = [1,1,0]^\top$
$\boldsymbol{h} = [1,0,1]^\top$
$\boldsymbol{h} = [1,1,1]^\top$
h_1
$\boldsymbol{h} = [0,1,0]^\top$
$\boldsymbol{h} = [0,1,1]^\top$
$\boldsymbol{h} = [0,0,1]^\top$

- For each of these binary features (h1, h2, h3), there is a corresponding set of inputs for which the feature is equal to 1.
- This set of inputs is represented by hi+.
- Each binary feature defines a decision boundary in the input space.
- **Decision Boundaries with Arrows:**
- Each decision boundary, represented as a line, separates the input space into two half planes.
- An arrow associated with a specific binary feature (hi) points toward the side of the boundary where hi=1 (i.e., the hi+ side).
- This arrow indicates which region of the input space is associated with that feature being active.

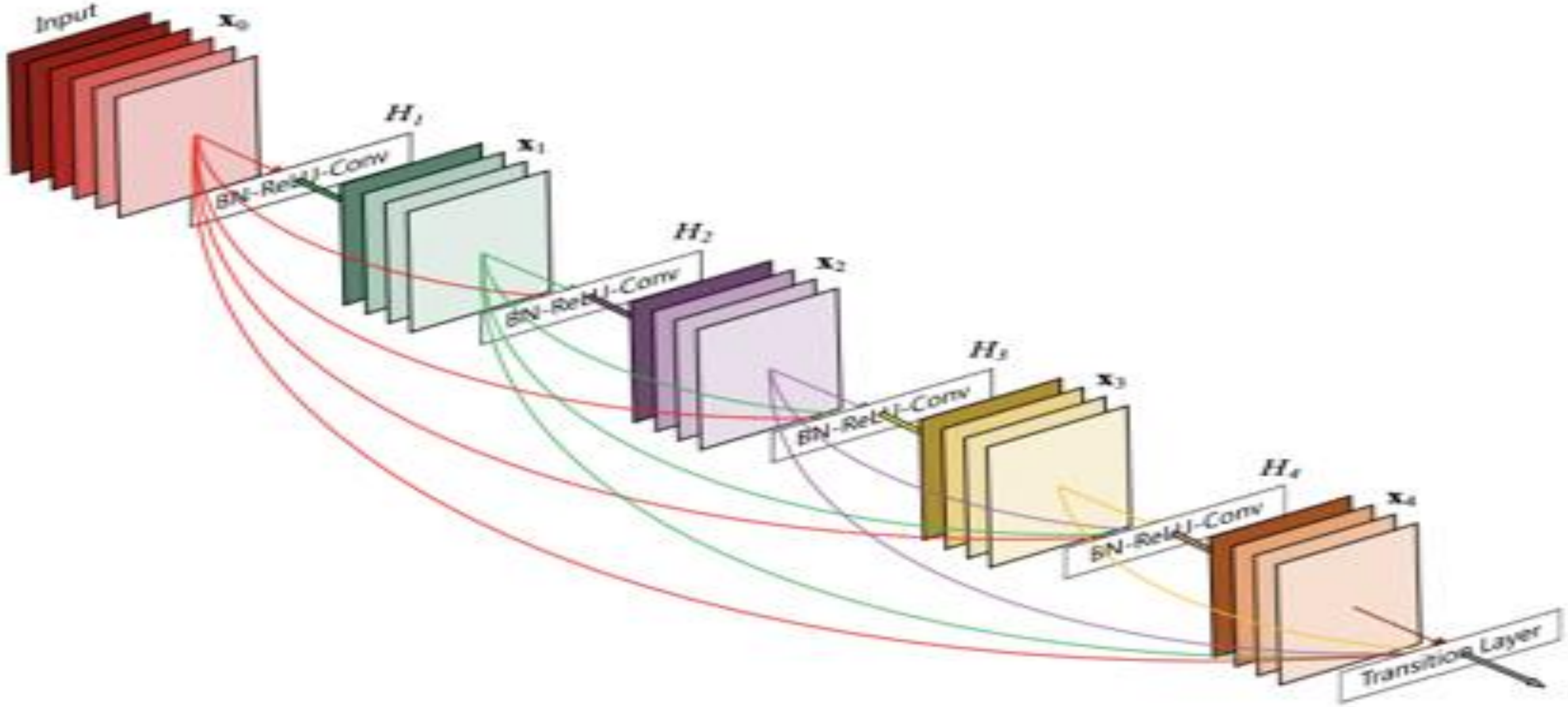
Variants of CNN: DenseNet

- DenseNets, short for Densely Connected Convolutional Networks, are a groundbreaking architecture in the field of deep learning, specifically designed for **image classification and computer vision tasks.**
- Introduced by Gao Huang, Zhuang Liu, and Laurens van der Maaten in their 2016 paper titled "Densely Connected Convolutional Networks,“.
- DenseNets have significantly improved the **efficiency, training speed, and overall performance** of convolutional neural networks (CNNs).
- Traditional CNN architectures typically consist of a sequence of convolutional layers, pooling layers, and fully connected layers.

Some limitations:

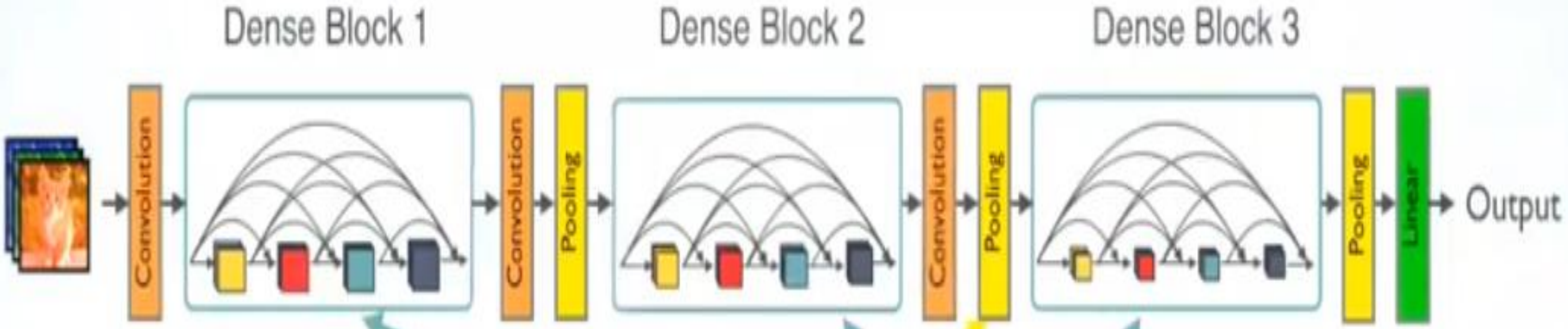
1. **Vanishing Gradient:** As networks become deeper, the gradients can vanish during backpropagation, making training difficult.
2. **Feature Redundancy:** Features learned by early layers of the network are not reused efficiently in later layers.

- **Dense Connectivity:**
- The core innovation behind DenseNets is dense connectivity.
- In a DenseNet architecture, every layer is connected to every other layer in a feedforward fashion.
- This dense connectivity encourages the reuse of features and enables efficient gradient flow.



DenseNet Structure

- **Dense Block:**
- DenseNets are divided into dense blocks, where each dense block consists of multiple convolutional layers.
- In a dense block, each layer receives feature maps from all previous layers within the same block.
- These feature maps are concatenated, increasing the depth and complexity of the network.
- Dense blocks have a powerful impact on feature learning, as they enable the network to make full use of feature maps generated at different network depths.



Pooling reduces
feature map sizes

Feature map sizes match
within each block

- **Transition Layers:**
- Transition layers are used to reduce the spatial dimensions (width and height) of feature maps.
- They also control the growth of model parameters, making DenseNets computationally efficient.
- Transition layers typically include a batch normalization layer, a 1x1 convolutional layer for dimensionality reduction, and a pooling layer.

Benefits of DenseNets:

1. **Improved Gradient Flow:** Dense connectivity ensures that gradients can flow directly and efficiently through the network, mitigating the vanishing gradient problem.
2. **Feature Reuse:** Feature maps are reused within dense blocks, leading to better feature extraction and improved model performance.
3. **Parameter Efficiency:** DenseNets achieve competitive accuracy with significantly fewer parameters compared to traditional architectures.

- **Applications:**
- DenseNets have been widely adopted and achieved state-of-the-art results in various computer vision tasks, including image classification, object detection, and image segmentation.
- **Variants:**
- Over time, several variants of DenseNets have been proposed, including DenseNet-BC (Bottleneck layers with Compression).
- Further reduces the model's computational requirements while maintaining performance.