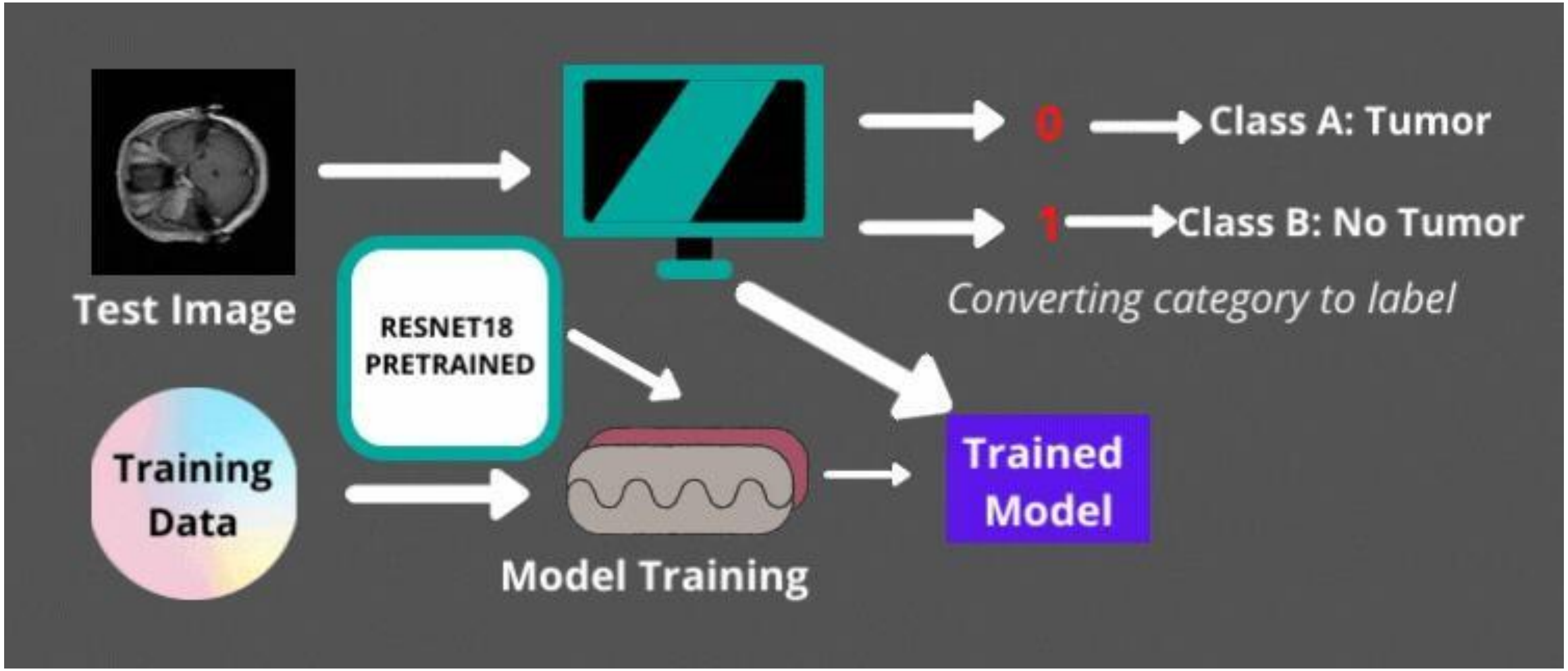
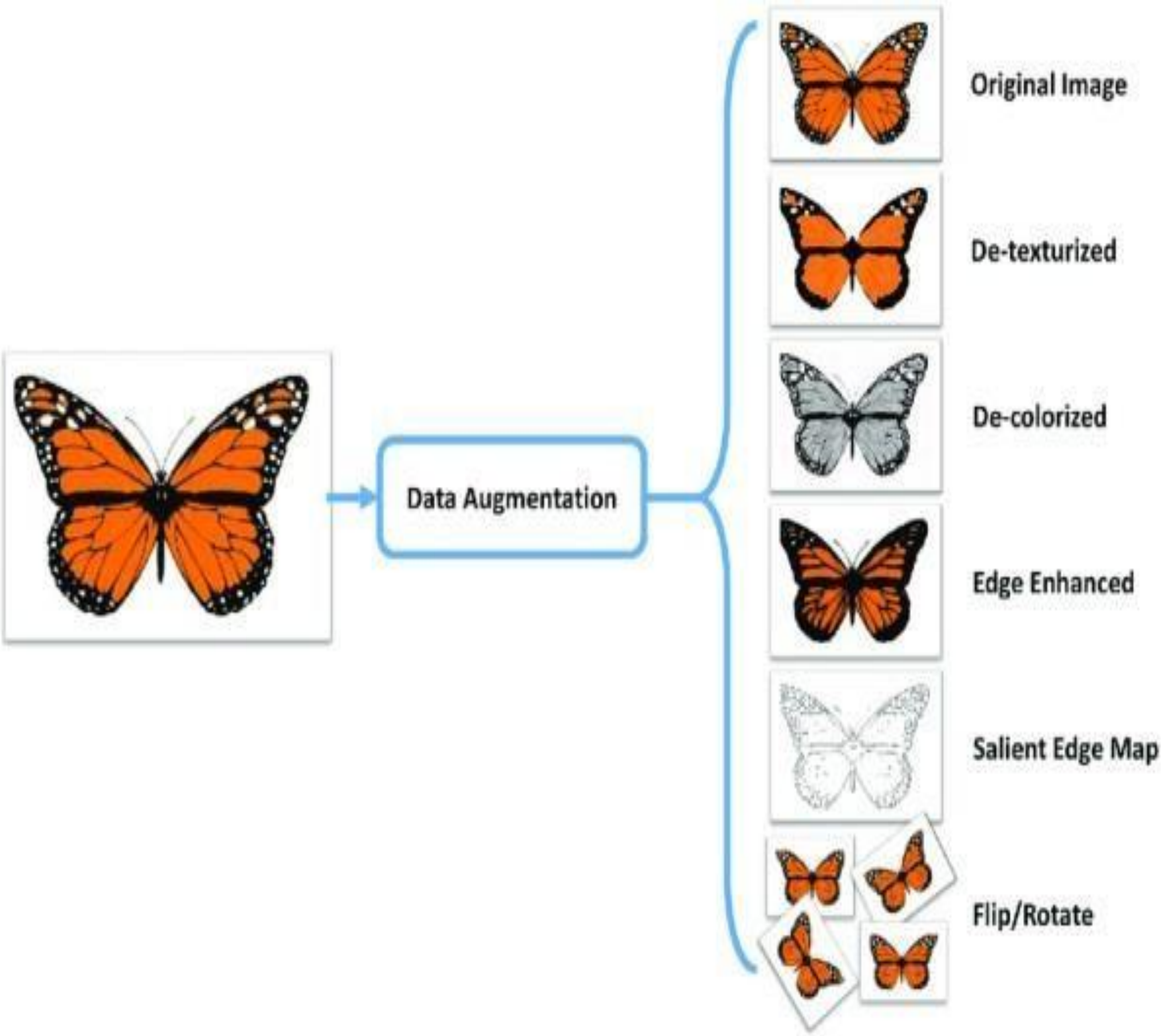
Data Augmentation and Transfer Learning



Data Augmentation

- Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data.
- It includes making minor changes to the dataset or using deep learning to generate new data points.
- Data augmentation is useful to improve the performance and outcomes of deep learning models by forming new and different examples to train datasets.

Purpose of Data Augmentation



- In computer vision, data augmentation aims to improve the downstream performance of the model.
- Data augmentation is done because augmenting the images will create a bigger dataset that will generalize in a better manner to situations that the model could encounter in production.
- The usefulness of various data augmentation varies in different situations.

Why is Data Augmentation Important

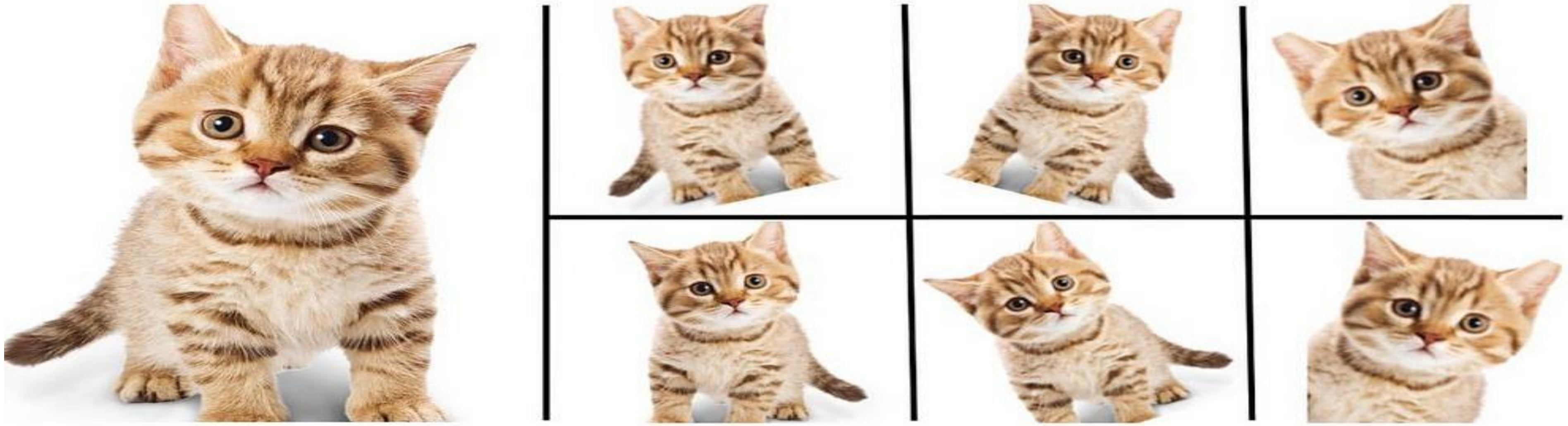
- The accuracy of predictions in supervised Deep Learning models depends to a large extent on the amount of data available to the model during training and the level of diversity in that data.
- You could literally consider data to be fuel for deep learning models. Greater volumes of diverse data lead to increasingly accurate predictions.
- But collecting data is not a cakewalk, and labeling it isn't easy either. It is a process that drains a lot of energy and money. And that is where data augmentation comes into the picture.
- Data augmentation techniques increase the precision and robustness of the deep learning models by creating variations of the data that the model might encounter in the real world.

Augmented vs. Synthetic data

- Augmented data is driven from original data with some minor changes. In the case of image augmentation, we make geometric and color space transformations (flipping, resizing, cropping, brightness, contrast) to increase the size and diversity of the training set.
- Synthetic data is generated artificially without using the original dataset. It often uses DNNs (Deep Neural Networks) and GANs (Generative Adversarial Networks) to generate synthetic data.

Data Augmentation in CNN?

- A convolutional neural network that can robustly classify objects even if its placed in different orientations is said to have the property called invariance.
- More specifically, a CNN can be invariant to translation, viewpoint, size or illumination (or a combination of the above).
- Data augmentation acts as a regularizer and assists in managing the overfitting of data.



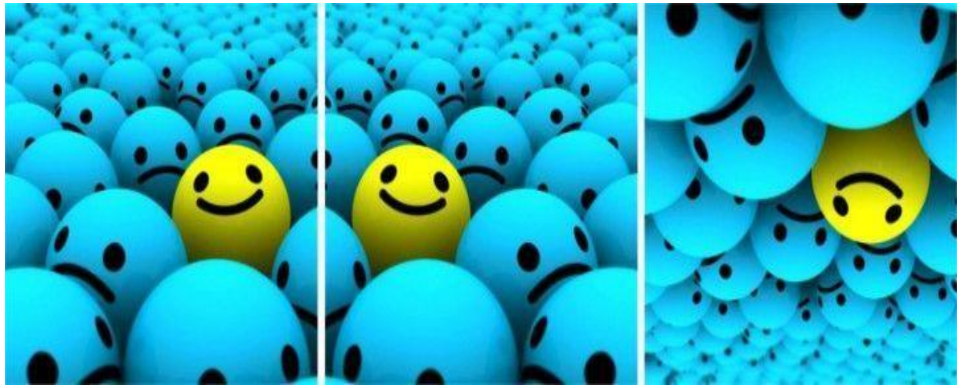
The Techniques Used in Data Augmentation?

1. Flip

- You can flip images horizontally and vertically

Data Augmentation Factor = 2 to 4x

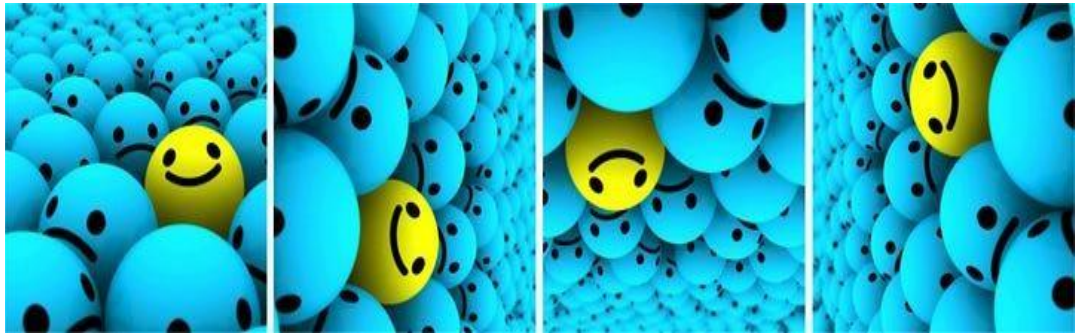
```
flip_1 = np.fliplr(img)
# TensorFlow. 'x' = A placeholder for an image. shape = [height, width, channels]
x = tf.placeholder(dtype = tf.float32, shape = shape)
flip_2 = tf.image.flip_up_down(x)
flip_3 = tf.image.flip_left_right(x)
flip_4 = tf.image.random_flip_up_down(x)
flip_5 = tf.image.random_flip_left_right(x)
```



2. Rotation

- One key thing to note about this operation is that image dimensions may not be preserved after rotation.

```
shape = [height, width, channels]
x = tf.placeholder(dtype = tf.float32, shape = shape)
rot_90 = tf.image.rot90(img, k=1)
rot_180 = tf.image.rot90(img, k=2)
shape = [batch, height, width, 3]
y = tf.placeholder(dtype = tf.float32, shape = shape)
rot_tf_180 = tf.contrib.image.rotate(y, angles=3.1415)
rot = skimage.transform.rotate(img, angle=45, mode='reflect')
```



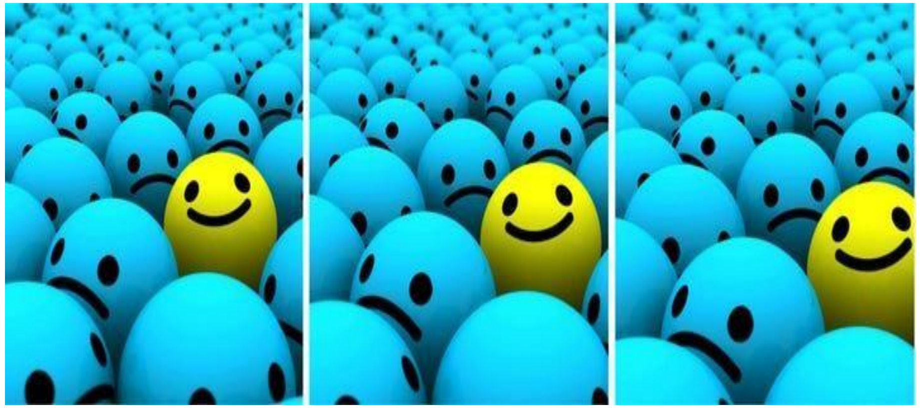
The Techniques Used in Data Augmentation?

3. Scale

- The image can be scaled outward or inward

Data Augmentation Factor = Arbitrary.

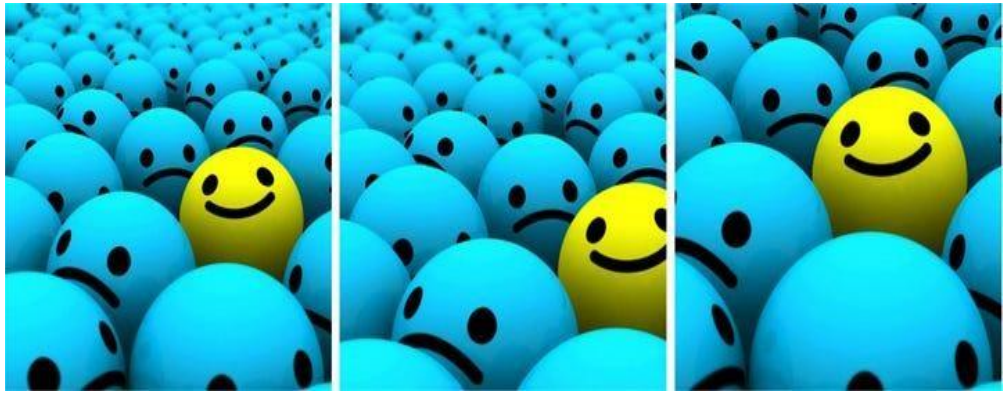
```
scale_out = skimage.transform.rescale(img, scale=2.0, mode='constant')
scale_in = skimage.transform.rescale(img, scale=0.5, mode='constant')
```



4. Crop

- Unlike scaling, we just randomly sample a section from the original image. We then resize this section to the original image size. This method is popularly known as random cropping.

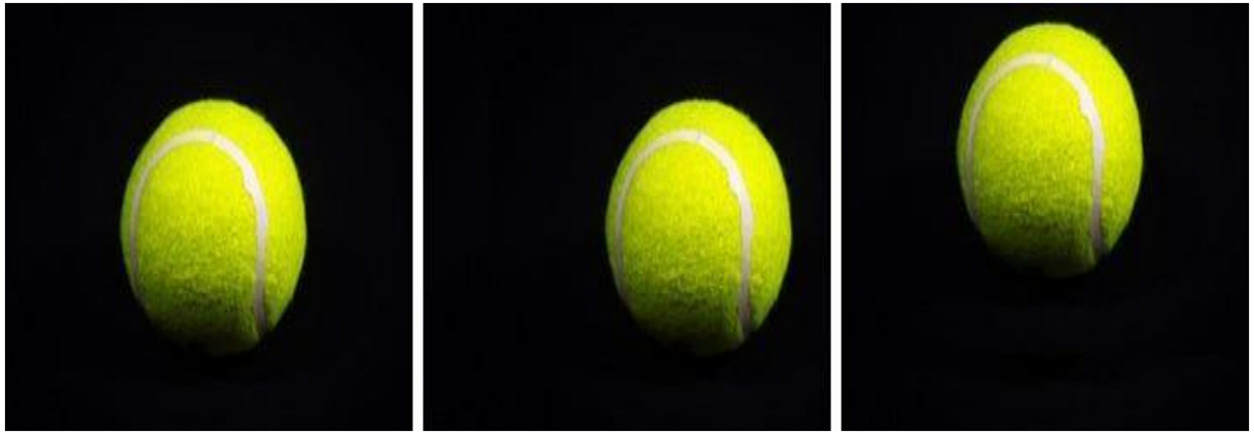
```
original_size = [height, width, channels]
x = tf.placeholder(dtype = tf.float32, shape = original_size)
crop_size = [new_height, new_width, channels]
seed = np.random.randint(1234)
x = tf.random_crop(x, size = crop_size, seed = seed)
output = tf.images.resize_images(x, size = original_size)
```



The Techniques Used in Data Augmentation?

5. Translation

- Translation just involves moving the image along the X or Y direction (or both). This method of augmentation is very useful as most objects can be located at almost anywhere in the image. This forces your convolutional neural network to look everywhere.



Data Augmentation Factor = Arbitrary.

```
shape = [batch, height, width, channels]
x = tf.placeholder(dtype = tf.float32, shape = shape)
x = tf.image.pad_to_bounding_box(x, pad_top, pad_left, height + pad_bottom + pad_top, width + pad_right +
pad_left)
output = tf.image.crop_to_bounding_box(x, pad_bottom, pad_right, height, width)
```

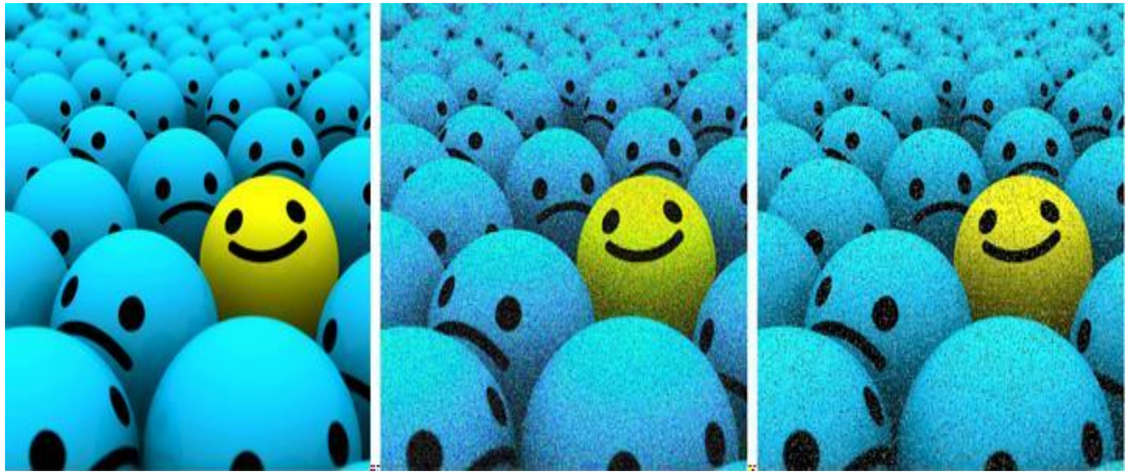
The Techniques Used in Data Augmentation?

6. Gaussian Noise

- Over-fitting usually happens when your neural network tries to learn high-frequency features (patterns that occur a lot) that may not be useful.
- Gaussian noise, which has zero mean, essentially has data points in all frequencies, effectively distorting the high-frequency features.
- This also means that lower frequency components (usually, your intended data) are also distorted, but your neural network can learn to look past that.
- Adding just the right amount of noise can enhance learning capability.

Data Augmentation Factor = 2x.

```
shape = [height, width, channels]
x = tf.placeholder(dtype = tf.float32, shape = shape)
# Adding Gaussian noise
noise = tf.random_normal(shape=tf.shape(x), mean=0.0, stddev=1.0,
dtype=tf.float32)
output = tf.add(x, noise)
```



Advanced Augmentation Techniques

Adversarial training

- Also known as adversarial machine learning. It generates adversarial examples. These examples disrupt machine learning models. Later it injects them into datasets.

GANs

- GANs expand as generative adversarial networks. These algorithms learn patterns from input datasets. It then creates new examples which resemble training data.

Neural style transfer

- The neural style transfer is a technique that blends content image and style image. It also performs a reverse function and separates style from content.

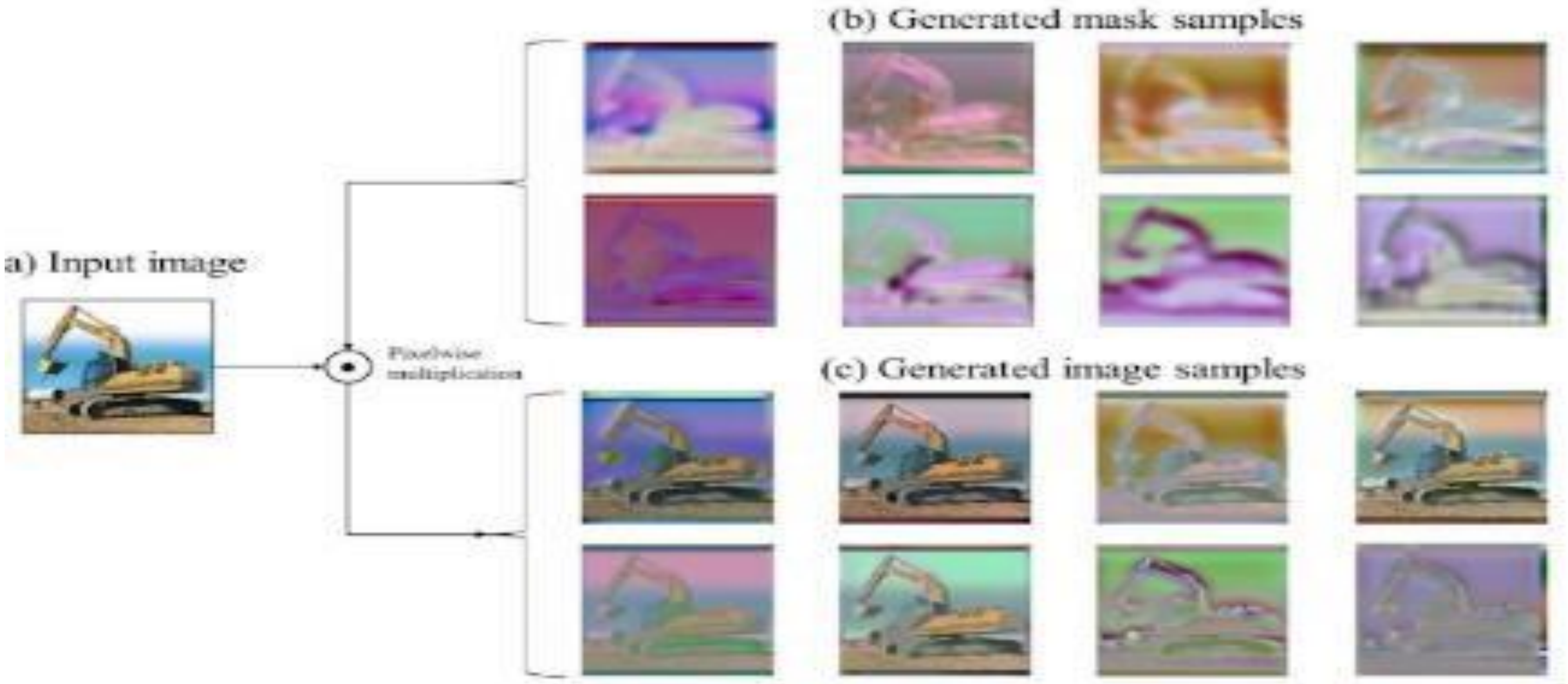
Reinforcement learning

- Reinforcement learning helps the software agents to make decisions in a virtual environment.

Advanced Augmentation Techniques

Adversarial training

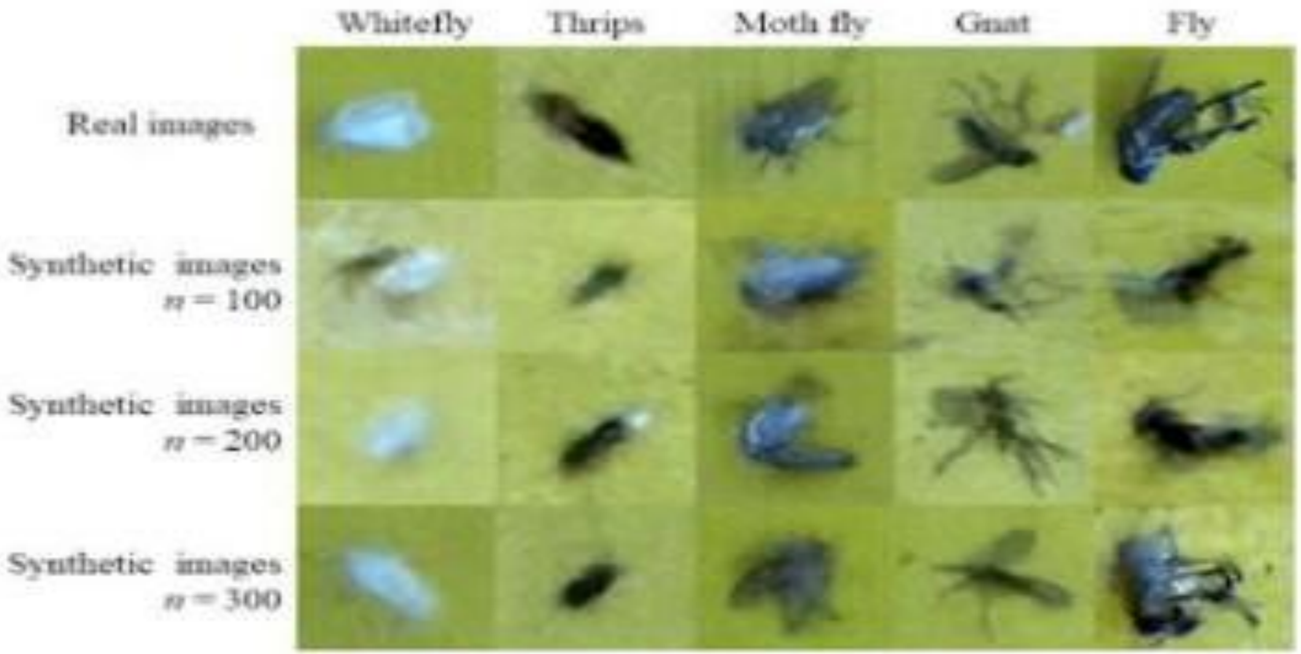
- The objective is to transform the images to deceive a deep-learning model to the extent that the model fails to correctly analyze it.
- Such transformed images can be used as training data to compensate for the weaknesses in the deep-learning model.



Advanced Augmentation Techniques

GAN based Augmentation

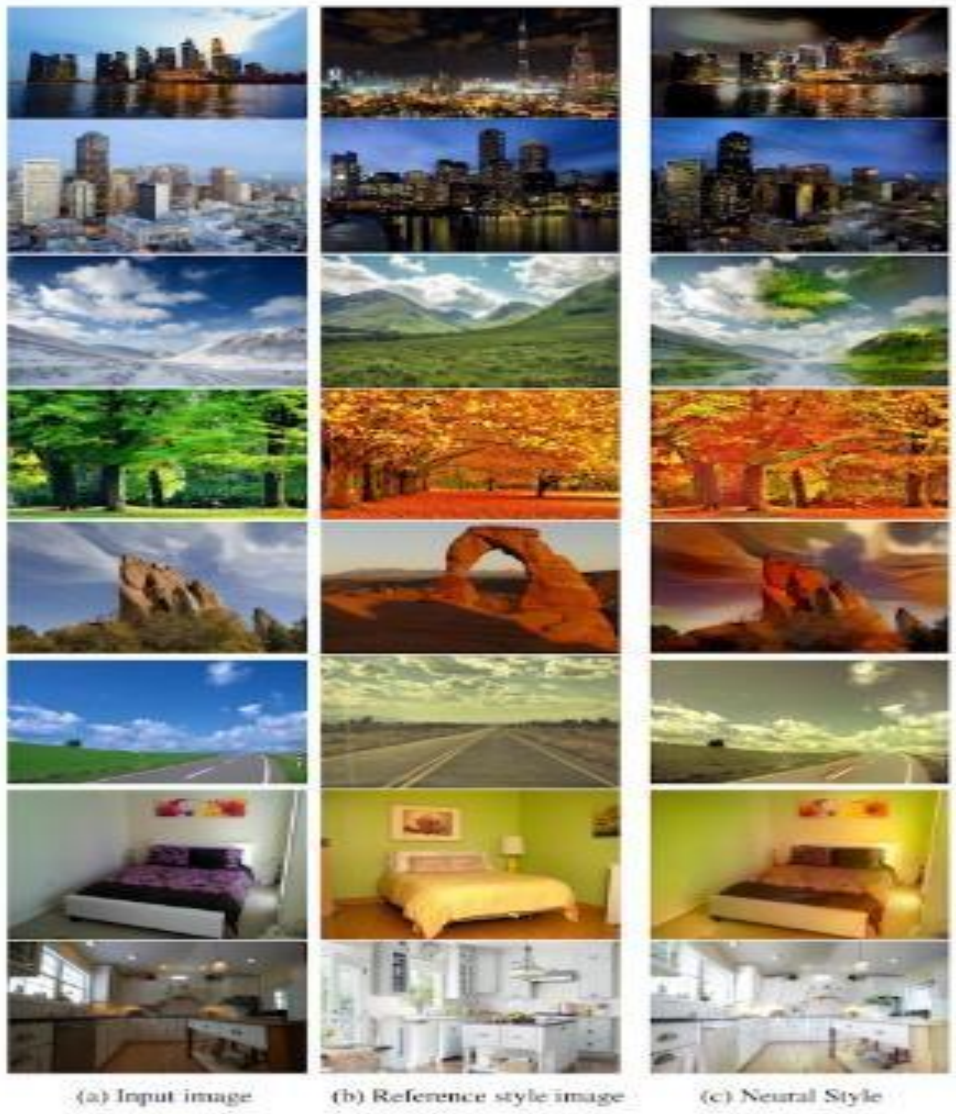
- It consists of two simultaneously trained neural networks:
- the generator and the discriminator.
- The goal of the generator is to generate fake images from the latent space and the goal of the discriminator is to distinguish the synthetic fake images from the real images.



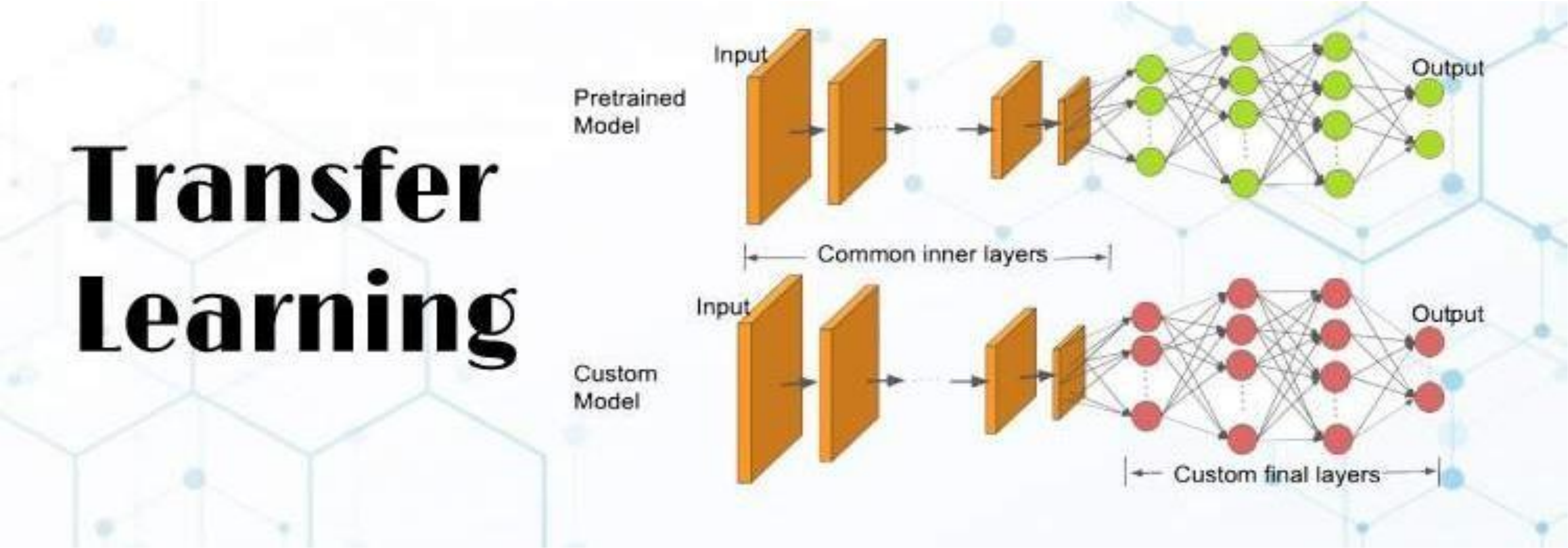
Advanced Augmentation Techniques

Neural Style Transfer based Augmentation

- In Neural style transfer, Deep Neural Networks are trained to extract the content from one image and style from another image and compose the augmented image using the extracted content and style.
- The augmented image is transformed to look like the input image, but "painted" in the style of the style image.

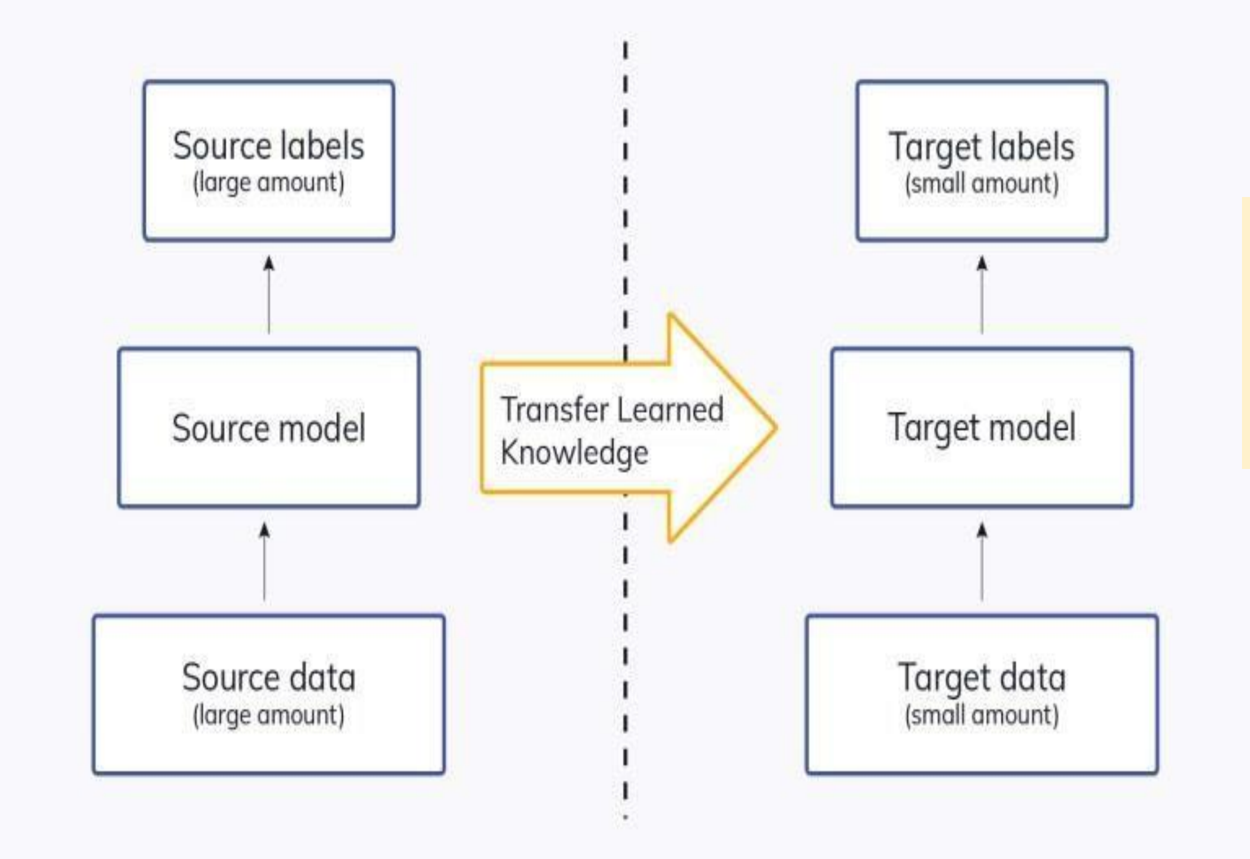


Transfer Learning



Transfer Learning

- Transfer learning is a machine learning method where we reuse a pre-trained model as the starting point for a model on a new task.
- Transfer learning is about leveraging feature representations from a pre-trained model, so you don't have to train a new model from scratch.

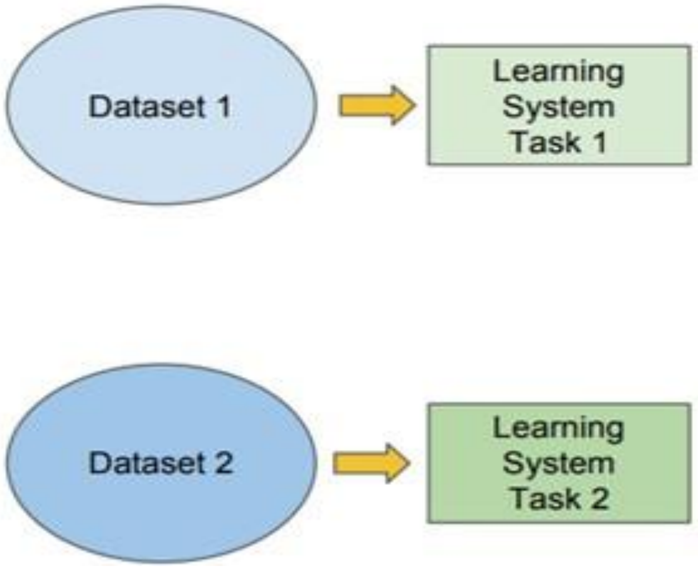


- By applying transfer learning to a new task, one can achieve significantly higher performance than training with only a small amount of data.

Machine Learning vs Transfer Learning

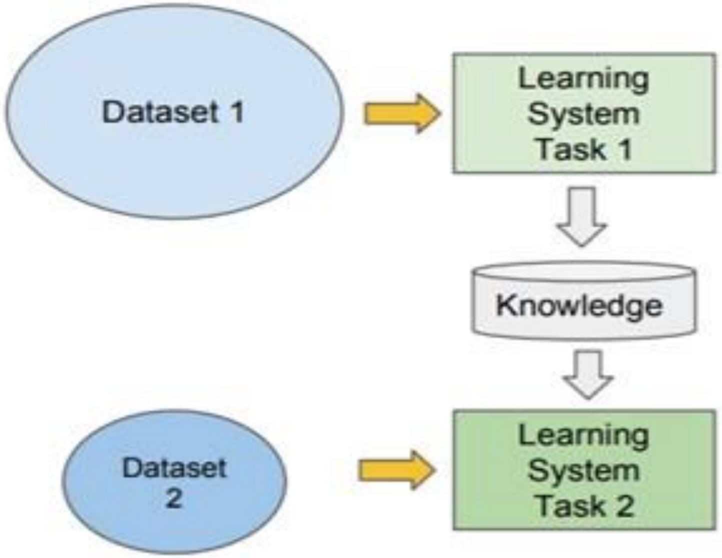
Traditional ML

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



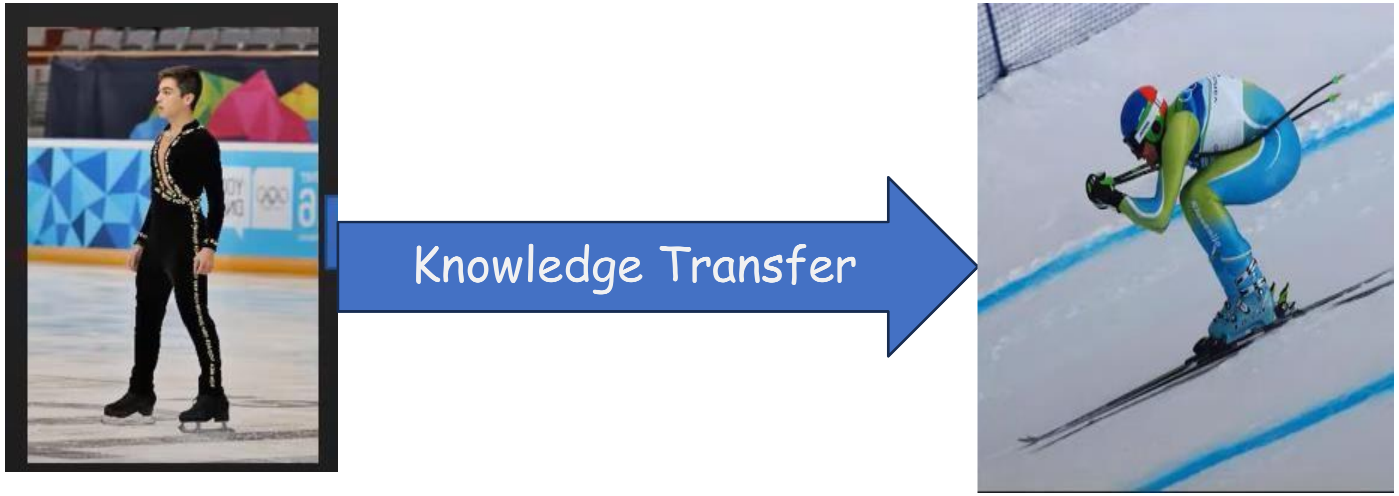
Transfer Learning

- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data

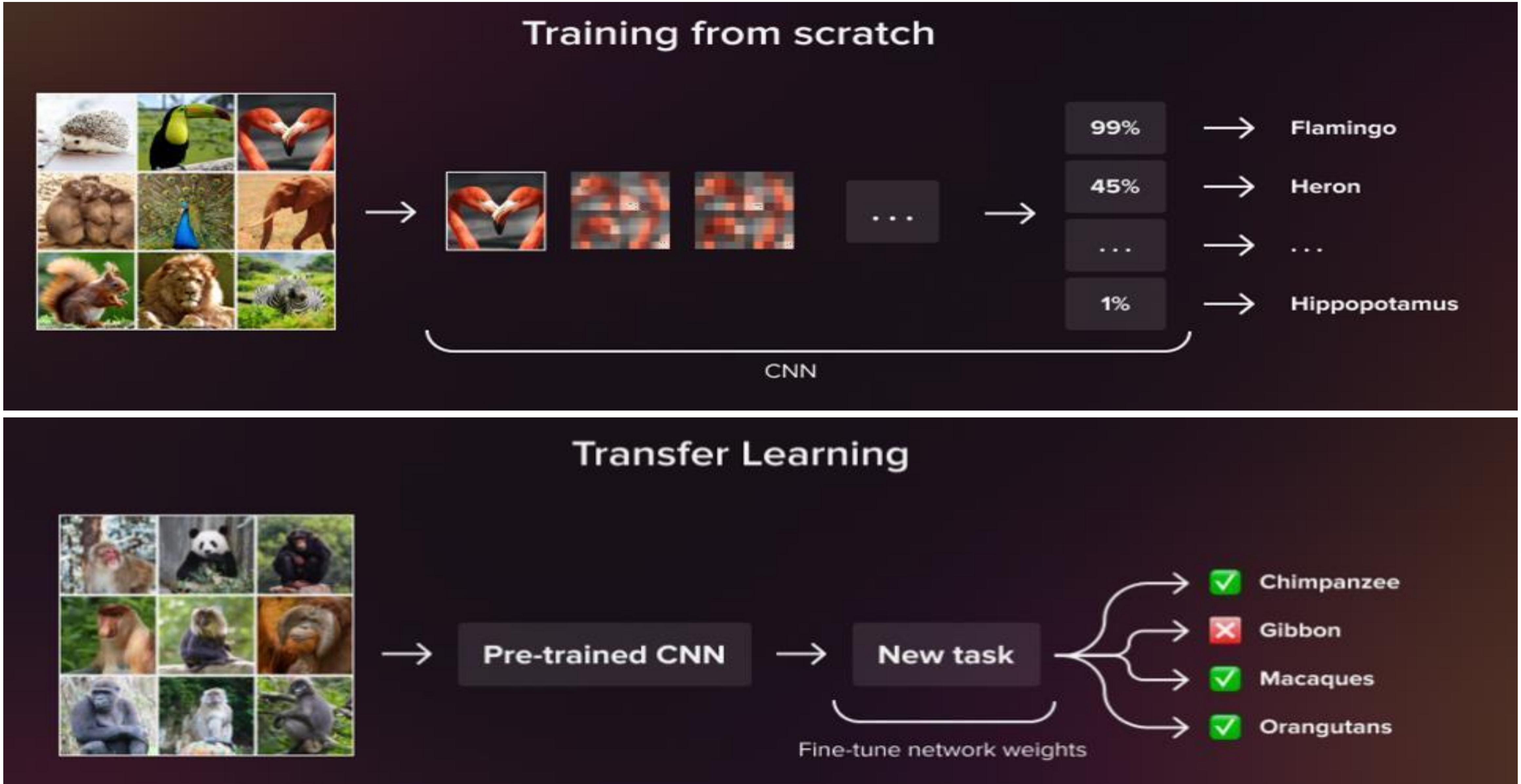


How Transfer Learning Works

- 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, try to transfer as much knowledge as possible from the previous task the model was trained on to the new task at hand. This knowledge can be in various forms depending on the problem and the data.



How Transfer Learning Works



How Transfer Learning Works

- Transfer learning should enable us to utilize knowledge from previously learned tasks and apply them to newer, related ones.

- If we have significantly more data for task ***T1***, we may utilize its learning, and generalize this knowledge (features, weights) for task ***T2*** (which has significantly less data).

- Certain low-level features, such as edges, shapes, corners and intensity, can be shared across tasks, and thus enable knowledge transfer among tasks!

Formal Definition

- A domain, D, is defined as a two-element tuple consisting of feature space, χ, and marginal probability, P(X), where X is a sample data point. Thus, we can represent the domain mathematically as D = {χ, P(X)}.
  - Feature space: χ
  - Marginal Distribution : P(X), X={x_1,x_2,….,x_n},x_i χ
  - Here x_i represents a specific vector as represented in the above depiction. A task, T, on the other hand, can be defined as a two-element tuple of the label space, γ, and objective function, η. The objective function can also be denoted as P(γ| X) from a probabilistic view point.

For a given domain **D**, a **Task** is defined by two components:

$$T = \{\mathcal{Y}, P(Y|X)\} = \{\mathcal{Y}, \eta\} \qquad Y = \{y_1, \cdots, y_n\}, y_i \in \mathcal{Y}$$

- A label space: $\mathcal{Y}$
- A predictive function η, learned from *feature vector/label* pairs, (x_i, y_i), $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$
- For each feature vector in the domain, η predicts its corresponding label: $\eta(x_i) = y_i$

Formal Definition

- Given a source domain D_S and learning task T_S, a target domain D_T and learning task T_T , transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$.
  - A domain is a pair $D = \{X , P(X)\}$. Thus the condition $D_S \neq D_T$ implies that either $X_S \neq X_T$ or $PS(X) \neq PT(X)$.

- Similarly, a task is defined as a pair $T = \{Y, P(Y|X)\}$. Thus the condition $T_S \neq T_T$ implies that either $Y_S \neq Y_T$ or $P(Y_S|X_S) \neq P(Y_T|X_T)$.

When the target and source domains are the same.

1. $X_S \neq X_T$ The feature spaces between the domains are different.
2. The feature spaces between the domains are the same but the marginal probability distributions between domain data are different $P(X_S) \neq P(X_T)$, where $X_{S_i} \in \mathcal{X}_S$ and $X_{T_i} \in \mathcal{X}_T$.
3. the label spaces between the domains are different $\mathcal{Y}_S \neq \mathcal{Y}_T$
4. The conditional probability distributions between the domains are different $P(Y_S|X_S) \neq P(Y_T|X_T)$,

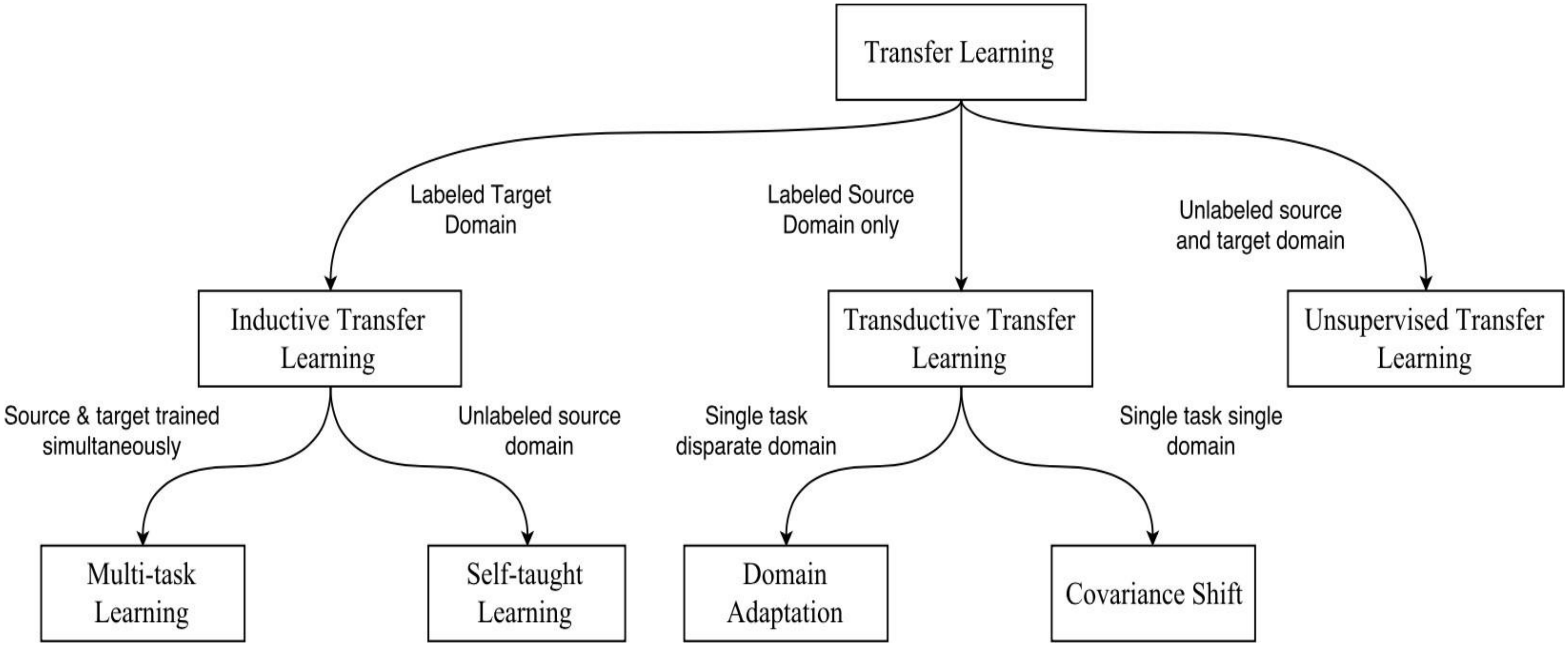
Classical Transfer Learning Strategies

- Different transfer learning strategies and techniques are applied based on the domain of the application, the task at hand, and the availability of data.

  - Inductive Transfer Learning
  - Transductive Transfer Learning
  - Unsupervised Transfer Learning

Type	Source Task ≠ Target Task?	Different Domains?	Labeled Data in Target?	Best Used For
Inductive	✅ Yes	❌ No	✅ Yes	Fine-tuning for new tasks (e.g., BERT for NLP, CNNs for medical images)
Transductive	❌ No	✅ Yes	❌ No (or very little)	Domain adaptation (e.g., different lighting, languages)
Unsupervised	✅ Yes	✅ Yes	❌ No	Self-supervised learning (e.g., anomaly detection, clustering)

Classical Transfer Learning Strategies



1. **Inductive Transfer Learning** (Labeled Target Domain)
    - **Multi-task Learning**: The source and target tasks are learned simultaneously.
    - **Self-taught Learning**: The source domain is unlabeled, but the target task has labels.
2. **Transductive Transfer Learning** (Labeled Source Domain Only)
    - **Domain Adaptation**: The source and target share the same task but have different data distributions.
    - **Covariance Shift**: The input distribution changes, but the conditional distribution remains the same.
3. **Unsupervised Transfer Learning** (Unlabeled Source & Target Domains)
    - Used when **no labels are available** in both domains.

Classical Transfer Learning Strategies

Inductive Transfer Learning

- Inductive Transfer Learning requires the source and target domains to be the same, though the specific tasks the model is working on are different.
- The algorithms try to use the knowledge from the source model and apply it to improve the target task. The pre-trained model already has expertise on the features of the domain and is at a better starting point than if we were to train it from scratch.
- Inductive transfer learning is further divided into two subcategories depending upon whether the source domain contains labeled data or not. These include multi-task learning and self-taught learning, respectively.

Classical Transfer Learning Strategies

Transductive Transfer Learning

- Scenarios where the domains of the source and target tasks are not exactly the same but interrelated uses the Transductive Transfer Learning strategy.
- One can derive similarities between the source and target tasks.
- These scenarios usually have a lot of labeled data in the source domain, while the target domain has only unlabeled data.

Classical Transfer Learning Strategies

Unsupervised Transfer Learning

- Unsupervised Transfer Learning is similar to Inductive Transfer learning.
- The only difference is that the algorithms focus on unsupervised tasks and involve unlabeled datasets both in the source and target tasks.

Common approaches to Transfer Learning

Homogeneous Transfer Learning:

- To handle situations where the domains are of the same feature space.
- In Homogeneous Transfer learning, domains have only a slight difference in marginal distributions.
- These approaches adapt the domains by correcting the sample selection bias or covariate shift.

1. Instance Transfer :

It covers a simple scenario in which there is a large amount of labeled data in the source domain and a limited number in the target domain.

Both the domains and feature spaces differ only in marginal distributions.

Example : Cancer for a specific region

Common approaches to Transfer Learning

2. Parameter transfer:

- This approach involves transferring knowledge through the shared parameters of the source and target domain learner models.
- One way to transfer the learned knowledge can be by creating multiple source learner models and optimally combining the re-weighted learners similar to ensemble learners to form an improved target learner.
- there are two ways to share the weights in deep learning models: soft weight sharing and hard weight sharing.
- Example : Fine tuning the pre trained network for a new domain

Common approaches to Transfer Learning

3. Feature-representation transfer

- Transform the original features to create a new feature representation.
- Asymmetric approaches transform the source features to match the target ones. In other words, we take the features from the source domain and fit them into the target feature space. There can be some information loss in this process due to the marginal difference in the feature distribution.
- Symmetric approaches find a common latent feature space and then transform both the source and the target features into this new feature representation.

Common Approaches to Transfer Learning

- **Relational-knowledge transfer**: Relational-based transfer learning approaches mainly focus on learning the relations between the source and a target domain and using this knowledge to derive past knowledge and use it in the current context.

- For example, if we learn the relationship between different elements of the speech in a male voice, it can help significantly to analyze the sentence in another voice.

Common approaches to Transfer Learning

- **Heterogeneous Transfer Learning:** Transfer learning involves deriving representations from a previous network to extract meaningful features from new samples for an inter-related task.

- This technique aims to solve the issue of source and target domains having differing feature spaces and other concerns like differing data distributions and label spaces.

- Heterogeneous Transfer Learning is applied in cross-domain tasks such as cross-language text categorization, text-to-image classification, and many others.

Feature	Homogeneous Transfer Learning	Heterogeneous Transfer Learning
Feature Space	Same	Different
Data Distribution	Different	Different
Techniques	Instance transfer, Parameter transfer, Feature representation, Relational knowledge	Feature space mapping, Cross-domain learning, Model transfer
Example	Fine-tuning ResNet for medical imaging	Using NLP knowledge for speech recognition

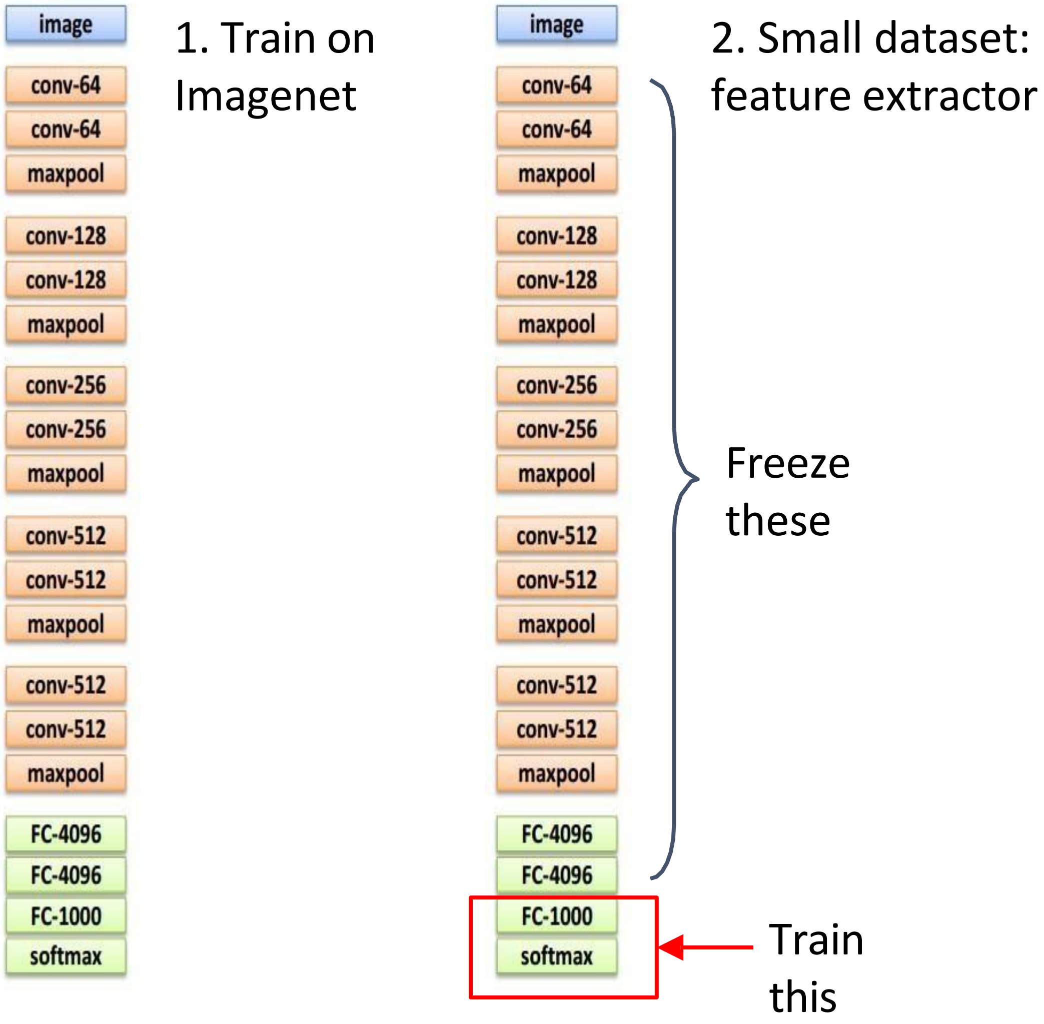
Category	Technique	Description	Example
Homogeneous Transfer Learning	**Instance Transfer**	Selects or reweights relevant instances from the source domain to match the target domain.	Adapting an email spam filter trained on Gmail data to Yahoo Mail.
	Parameter Transfer	Shares model parameters between source and target tasks to improve generalization.	Fine-tuning a ResNet model pre-trained on ImageNet for medical image classification.
	Feature Representation Transfer	Learns a common feature space between source and target data to make knowledge transferable.	Using CNN feature extractors trained on general object detection for crack detection in roads.
	Relational-Knowledge Transfer	Transfers structural relationships between data points in different domains.	Using user behavior patterns from an e-commerce platform to predict behavior in a different but similar platform.

Heterogeneous Transfer Learning	**Feature Space Mapping**	Uses transformation functions to align feature spaces between source and target domains.	Translating English text into Chinese before applying an NLP model trained in English.
	Cross-Domain Learning	Uses a shared latent space where both source and target data are projected for learning.	Mapping text and images into a common embedding space for text-to-image generation.
	Model Transfer Across Modalities	Transfers knowledge between different data modalities (e.g., image, text, audio).	Using an audio-based speech recognition model for lip-reading tasks.

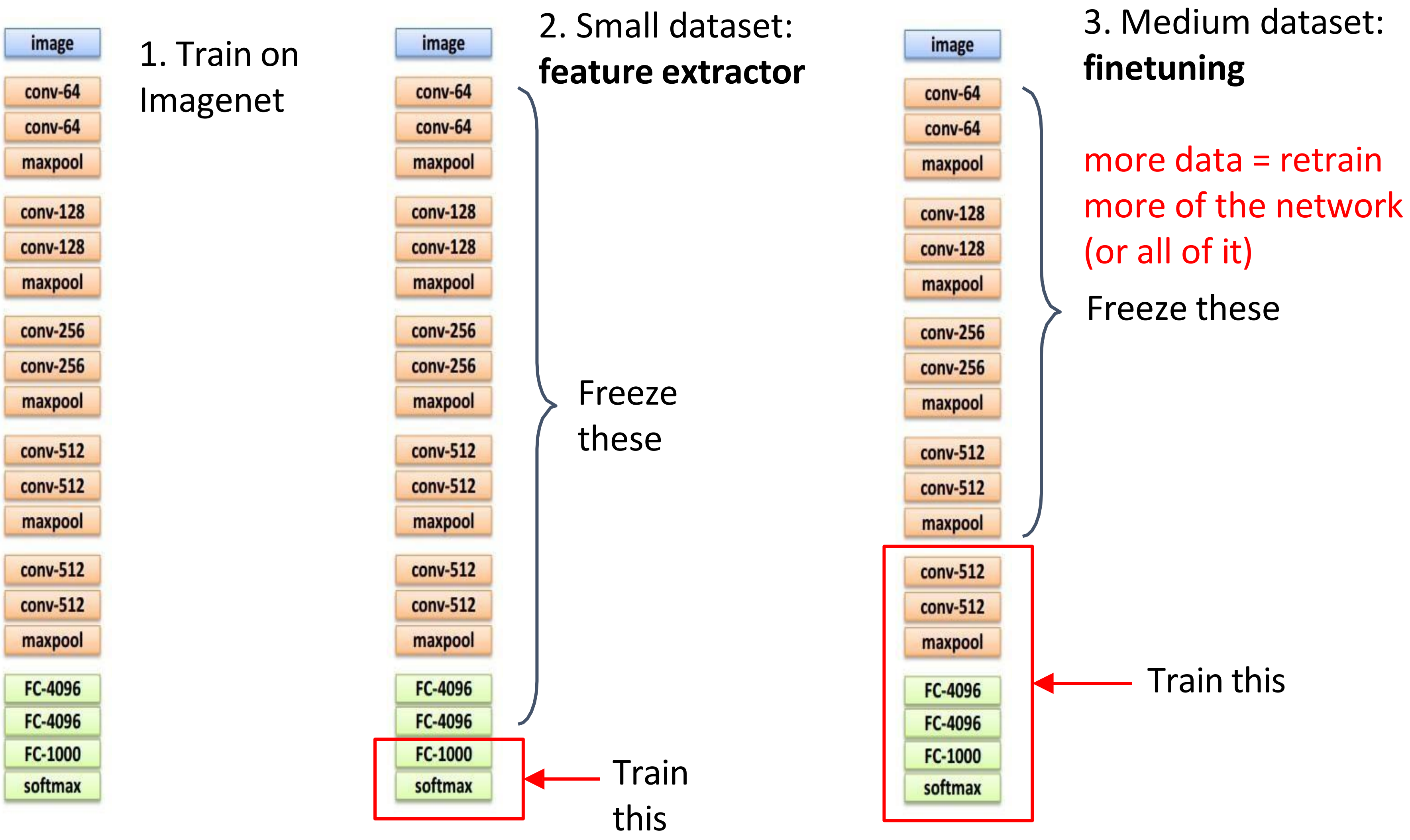
Transfer Learning with CNNs

1. Train on Imagenet

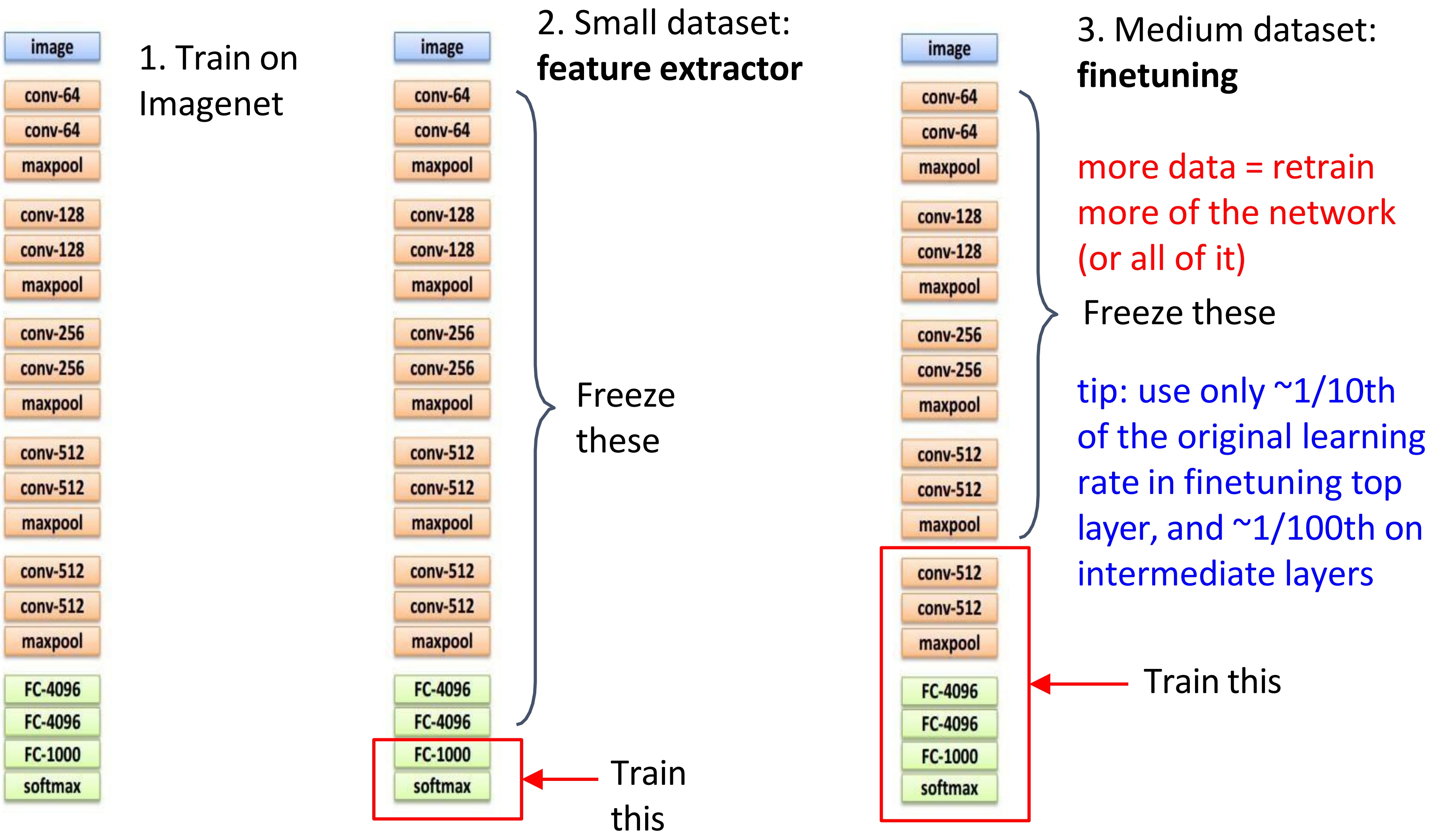
Transfer Learning with CNNs



Transfer Learning with CNNs



Transfer Learning with CNNs

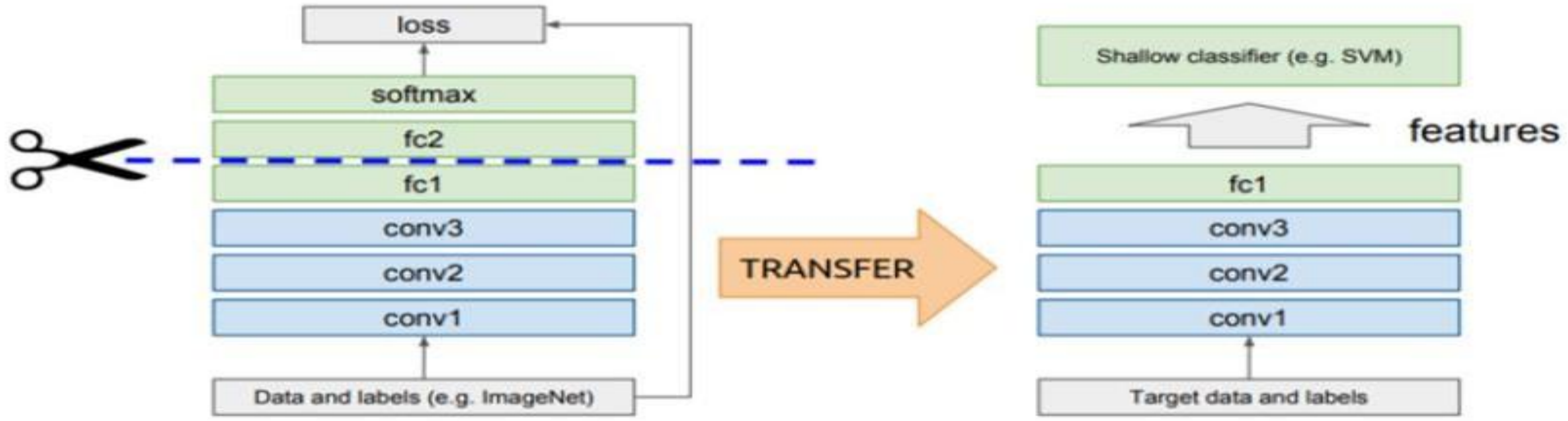


Transfer Learning for Deep Learning

- Deep learning systems are layered architectures that learn different features at different layers. Initial layers compile higher-level features that narrow down to fine-grained features as we go deeper into the network.

Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

Assumes that $D_S = D_T$

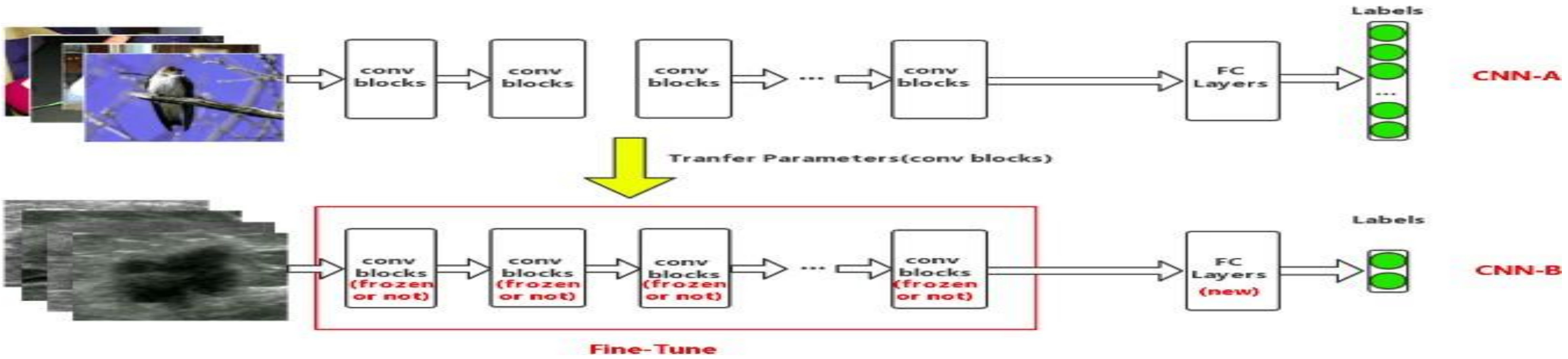


Transfer Learning with Pre-trained Deep Learning Models as Feature Extractors

Transfer Learning for Deep Learning

Fine Tuning Off-the-shelf Pre-trained Models

- **Fine-tuning** is an optional step in transfer learning. Fine-tuning will usually improve the performance of the model. However, since you have to retrain the entire model, you'll likely overfit.



- Overfitting is avoidable. Just retrain the model or part of it using a low learning rate. This is important because it prevents significant updates to the gradient. These updates result in poor performance. Using a callback to stop the training process when the model has stopped improving is also helpful.

Transfer Learning for Deep Learning

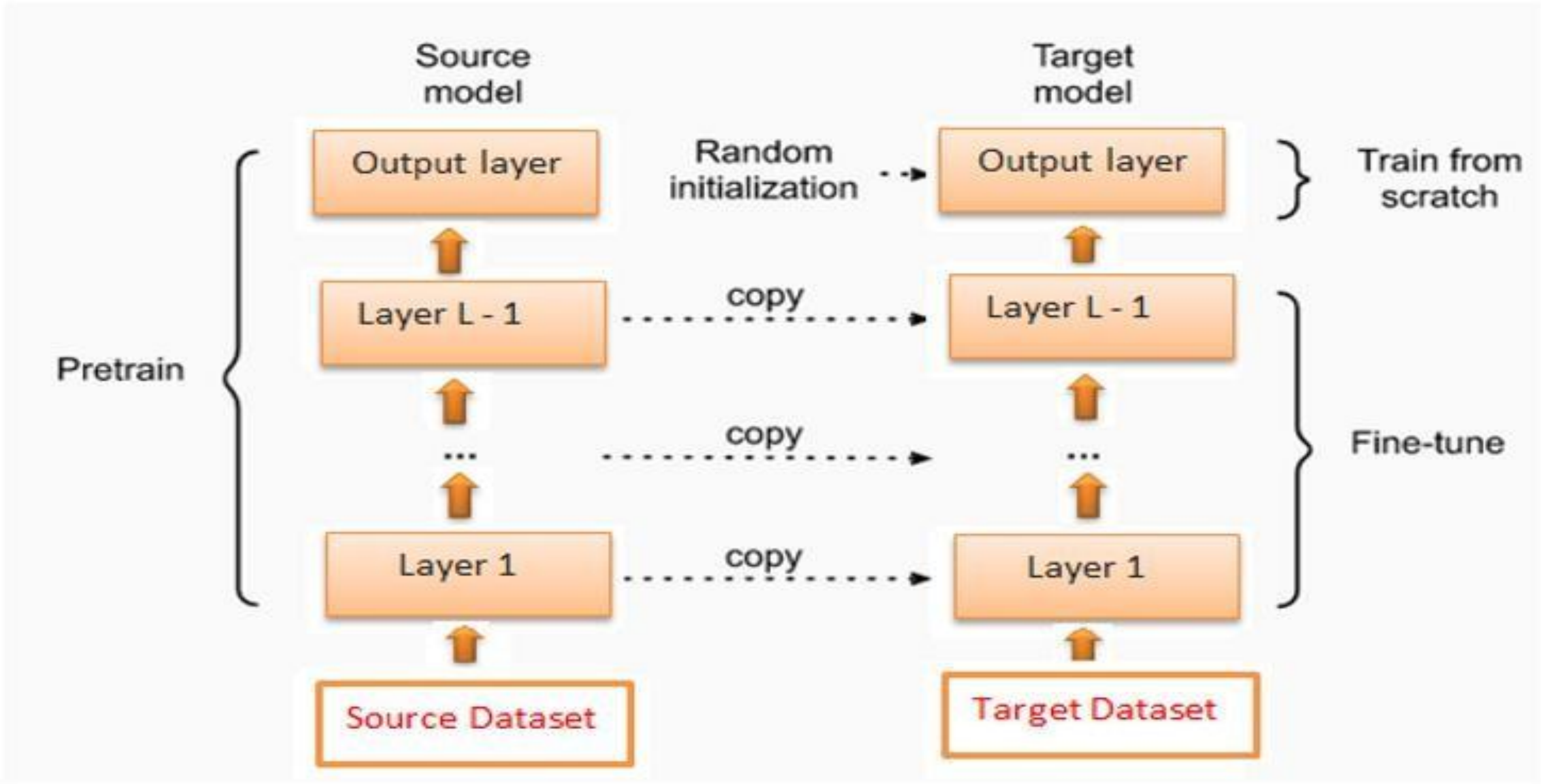
Fine Tuning Off-the-shelf Pre-trained Models

- Fine tuning is like optimization. We optimize the network to achieve optimal results. Maybe we can change the number of layers used, no of filters, and learning rate and we have many parameters of the model to optimize.
- Fine-tuning, in general, means making small adjustments to a process to achieve the desired output or performance.
- Tuning Machine Learning Model Is Like Rotating TV Switches and Knobs Until You Get A Clearer Signal.

  - Freeze the layers- Freezing a layer means the weights of that layer won't be updated. During training, we freeze the feature extraction layer i.e. these layers won't be trainable. Thus, higher accuracy can be achieved even for smaller datasets.

Transfer Learning for Deep Learning

Fine Tuning - How Does It Work



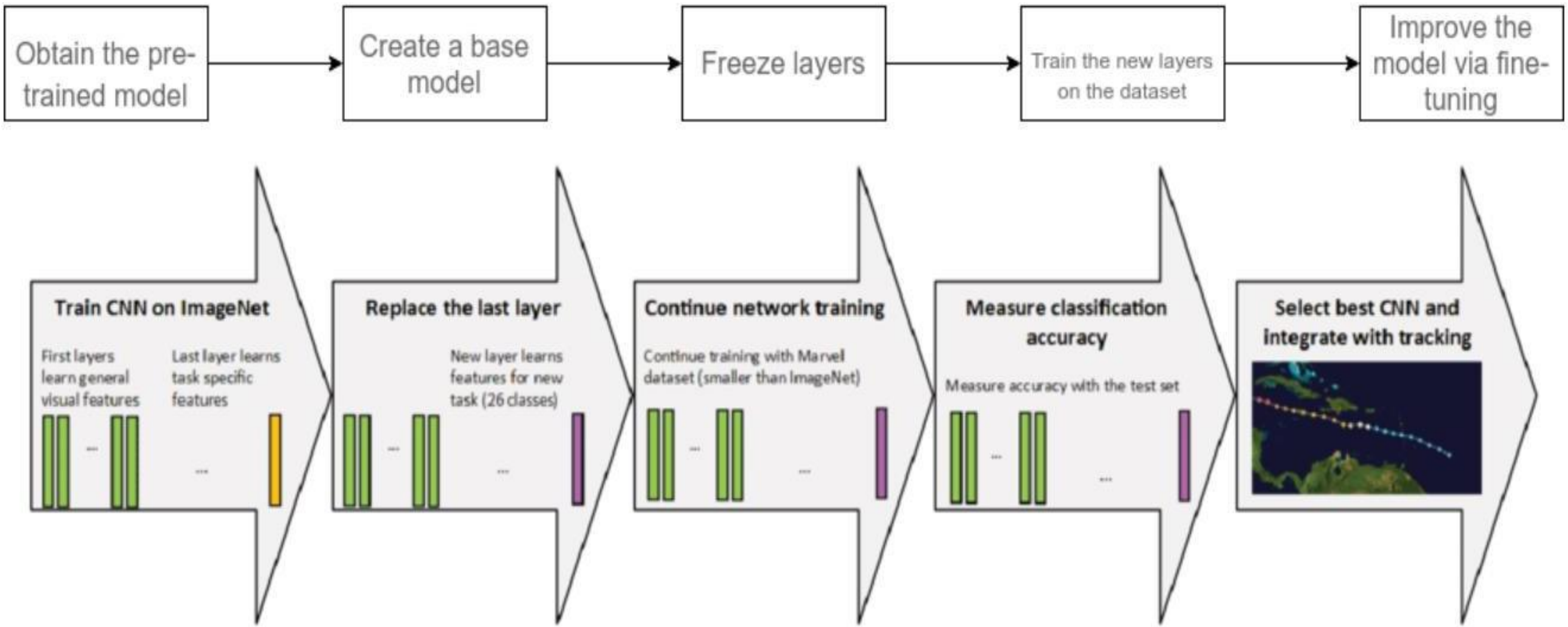
Transfer Learning for Deep Learning

When To Use Fine-tuning And Transfer Learning

- If there are similarities between the source and target model, there's no need to finetune the layers of the pre-trained model.
- When there are considerable differences between the source and target model, or training examples are abundant, we unfreeze several layers in the pre-trained model except the starting few layers which determine edges, corners, etc.
- When there are significant differences between the source and target model, we unfreeze and retrain the entire neural network called "full model fine-tuning", this type of transfer learning also requires a lot of training examples.
- When we are not provided with enough data.
- When we don't have sufficient computational power.

Transfer Learning for Deep Learning

Transfer Learning in 6 steps



Transfer Learning for Deep Learning

1. Obtain pre-trained model

- The first step is to choose the pre-trained model we would like to keep as the base of our training, depending on the task.

Here are some of the pre-trained models you can use:

For computer vision:

- VGG-16
- VGG-19
- Inception V3
- Xception
- ResNet-50

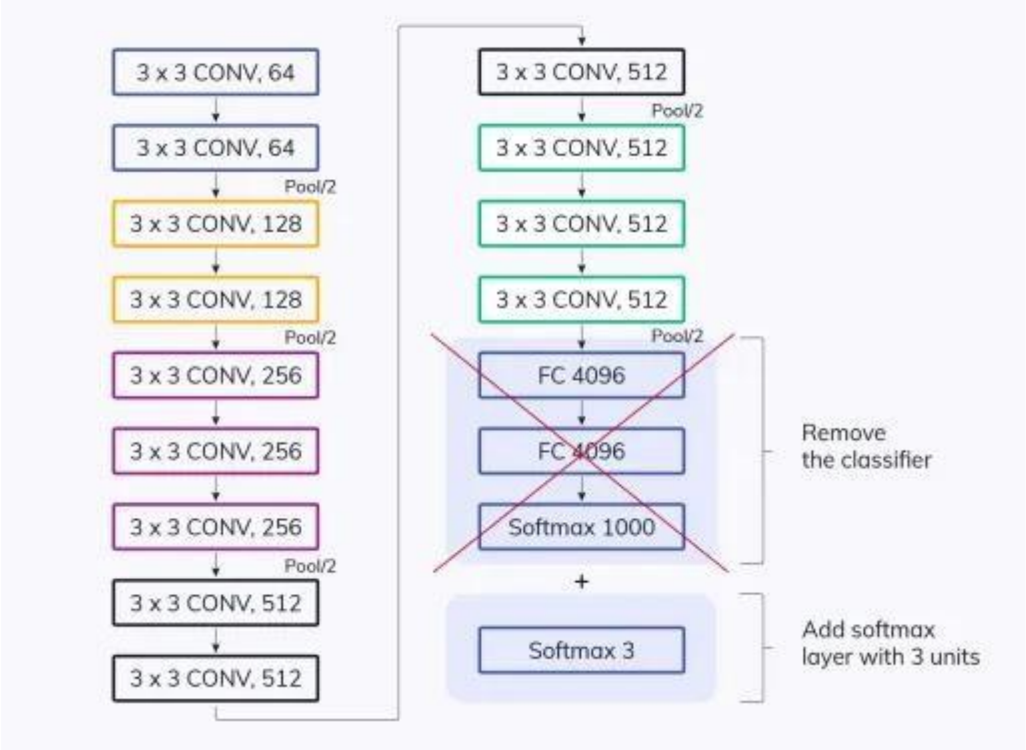
For NLP tasks:

- Word2Vec
- GloVe

Transfer Learning for Deep Learning

2. Create a base model

- The base model is one of the architectures such as ResNet or Xception which we have selected in the first step to be in close relation to our task.
- There can be a case where the base model will have more neurons in the final output layer than we require in our use case.



Transfer Learning for Deep Learning

3. Freeze layers

- Freezing the starting layers from the pre-trained model is essential to avoid the additional work of making the model learn the basic features.
- If we do not freeze the initial layers, we will lose all the learning that has already taken place. This will be no different from training the model from scratch and will be a loss of time, resources, etc..

3. Add new trainable layers

- The only knowledge we are reusing from the base model is the feature extraction layers. We need to add additional layers on top of them to predict the specialized tasks of the model. These are generally the final output layers.

Transfer Learning for Deep Learning

5. Train the new layers

- The pre-trained model's final output will most likely differ from the output we want for our model. For example, pre-trained models trained on the ImageNet dataset will output 1000 classes.
- However, we need our model to work for two classes. In this case, we have to train the model with a new output layer in place.

6. Fine-tune your model

- One method of improving the performance is fine-tuning.
- Fine-tuning involves unfreezing some part of the base model and training the entire model again on the whole dataset at a very low learning rate. The low learning rate will increase the performance of the model on the new dataset while preventing overfitting..