Transfer Learning

Ramendra Kumar

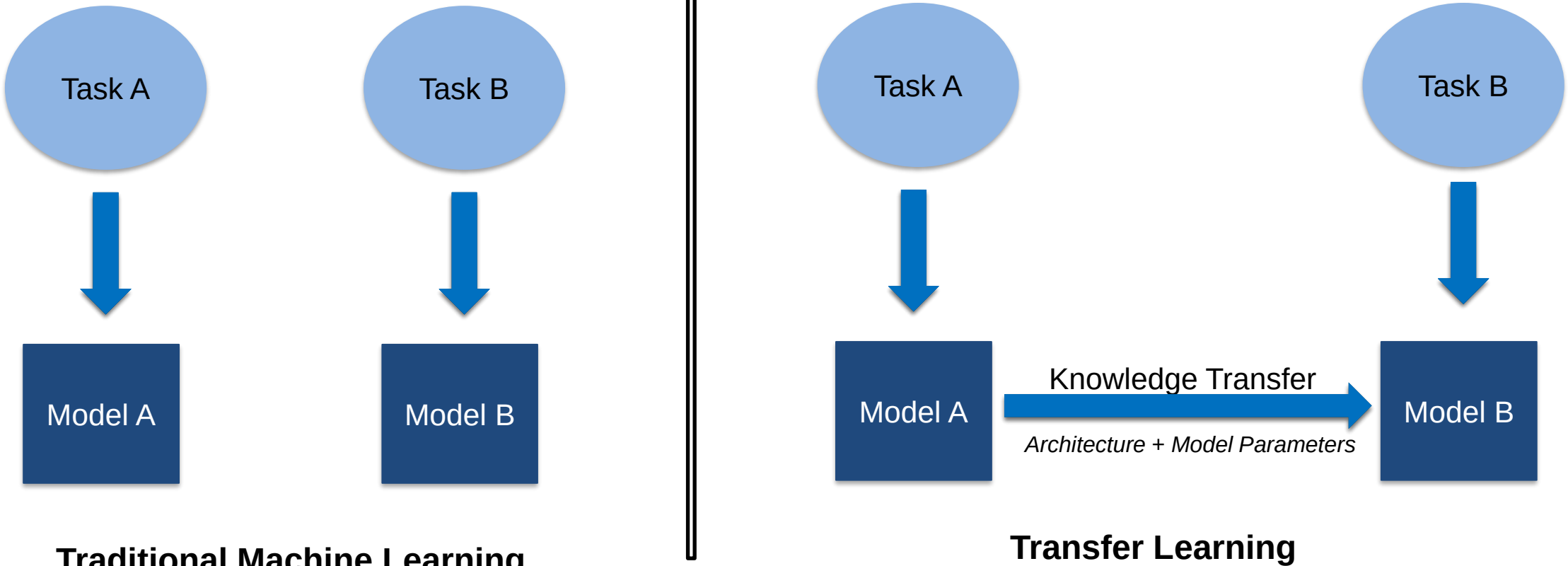
What is Transfer Learning

Improving a model

Methods to improve a model	What does it do?
More data	Model gets more chance to learn pattern between samples.
Better data	Not all data samples are created equally. Removing bad samples or adding more better samples can improve the model's performance
Augmentation	Increasing diversity of training dataset without collecting more data: performing different transformation operation.
Using Transfer Learning	Take an existing model's pre-learned patterns from one problem and use it for different one by just a little bit of tweaking

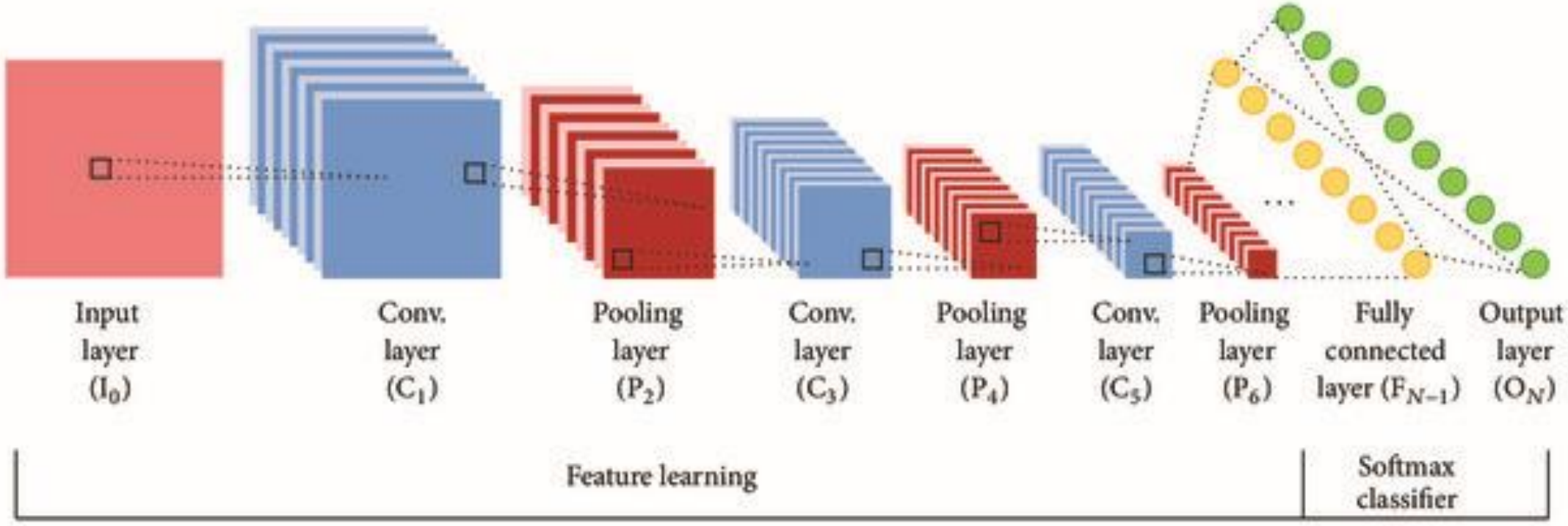
Transferring the knowledge of one model to perform a new task.

How it works


Task A
Task B
Model A
Model B
Traditional Machine Learning
Task A
Task B
Model A
Knowledge Transfer
Architecture + Model Parameters
Model B
Transfer Learning

A CNN typically consists of a:

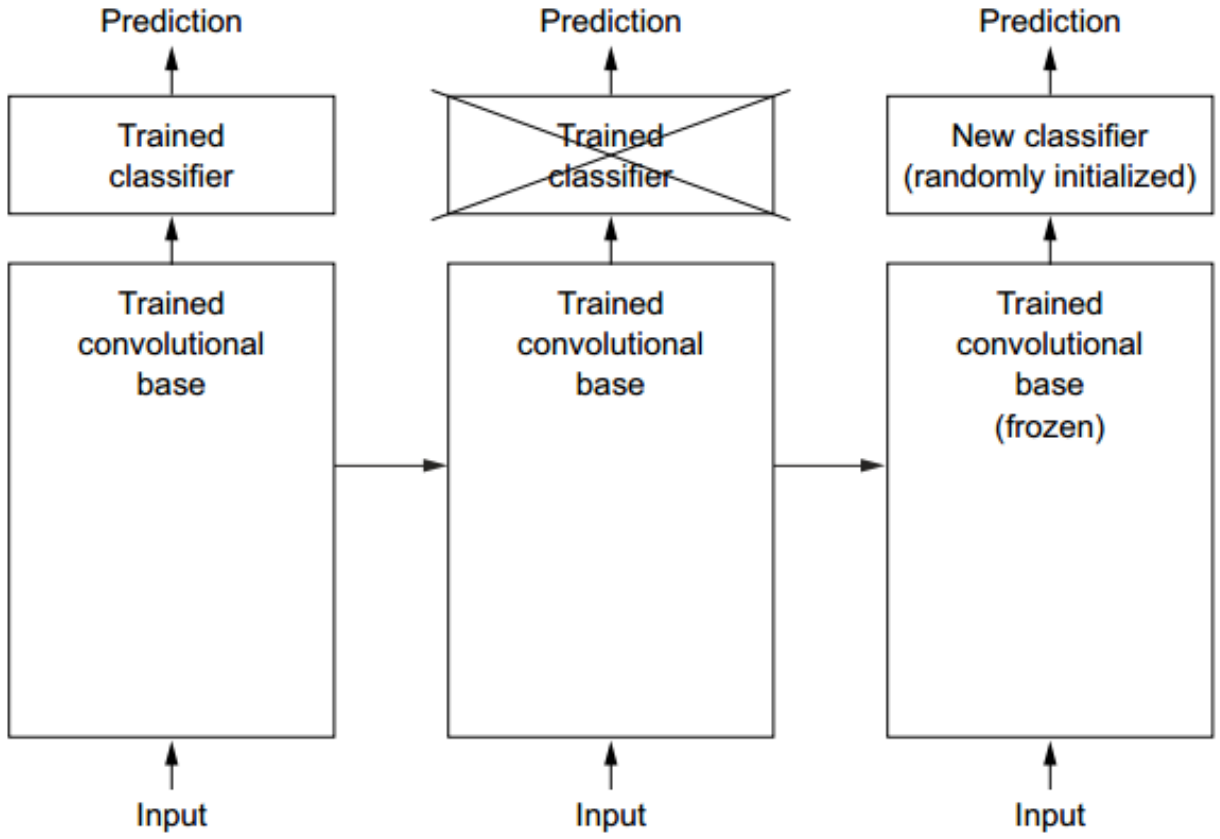
- Convolutional base
- Densely connected classifier



Feature extraction

A CNN typically consists of a:

* Convolutional base
* Densely connected classifier



Key Idea -

* Features are learned by the convolutional base. So reuse it.
* Train a new classifier for your problem

Why To Use

- Unavailability of data
- Long training time to train deep learning models
- More Computational resources

A common and highly effective approach to deep learning on small image datasets is to use a pre-trained model.

If the original dataset is large enough and general enough, the spatial hierarchy of features learned by the pre-trained model can effectively act as a generic model of the visual world, and hence, its features can prove useful for many different computer vision problems even though these new problems may involve completely different classes than those of the original task.

Step involved while using a pre-trained model:

- ❖ Feature extraction
- ❖ Fine-tuning

Transfer Learning Process

1. Start with pre-trained network
2. Partition network into:
    1. Featurizers : identify which layers to keep
    2. Classifiers : identify which layers to replace
3. Re-train classifier layers with new data
4. Unfreeze weights of the later layers and fine-tune the network

Which layers to re-train?

- Depends on the domain
- Start by re-training the last layers (last full-connected and last convolutional)
    - work backwards if performance is not satisfactory

Ways to Fine tune the model

Dataset Size	Dataset Similarity	Recommendation
Large	Very different	Train the model from scratch
Large	Similar	Retain the architecture and initial weights. Customize only last layer for number of classes
Small	Very different	Customize earlier layers and train the classifier
Small	Similar	Don't fine-tune (overfitting)

Pre-trained Models

Different Pre-trained Models

Few …

Network	Year	Parameters	Top-5 accuracy
AlexNet	2012	62M	84.70%
VGGNet	2014	138M	92.30%
Inception	2014	6.4M	93.30%
ResNet-152	2015	60.3M	95.51%

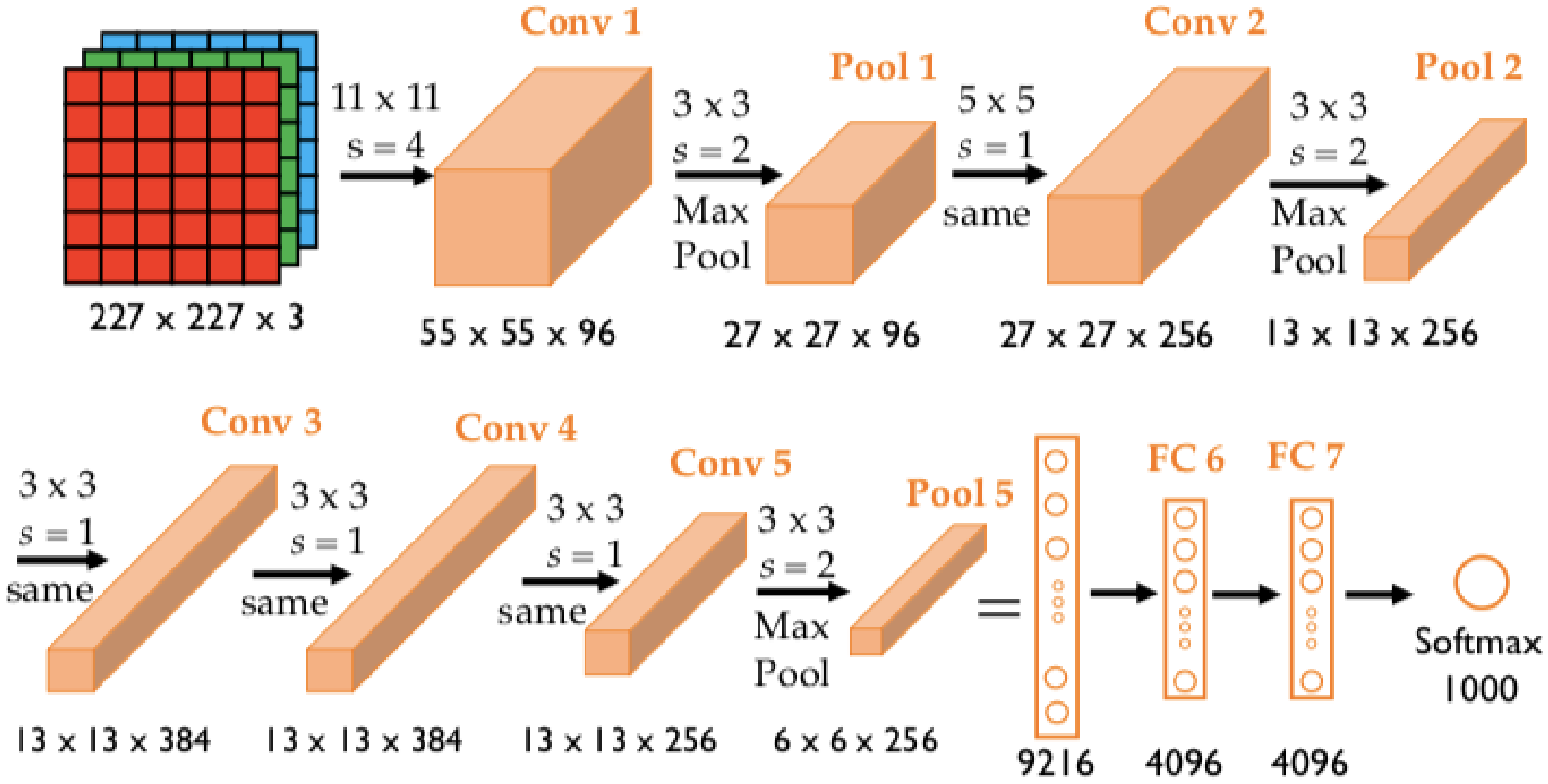
Link for Pre-trained models

→ https://pytorch.org/vision/stable/models.html

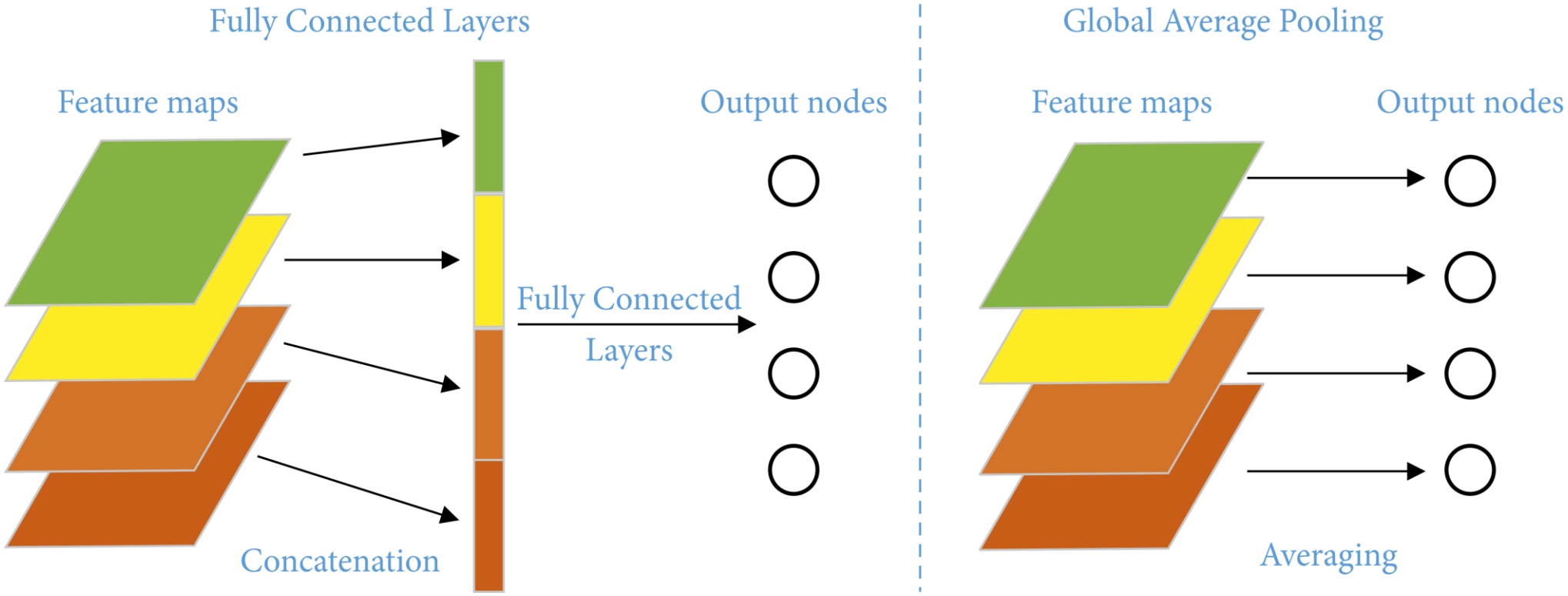
→ https://huggingface.co/models

→ https://paperswithcode.com/sota

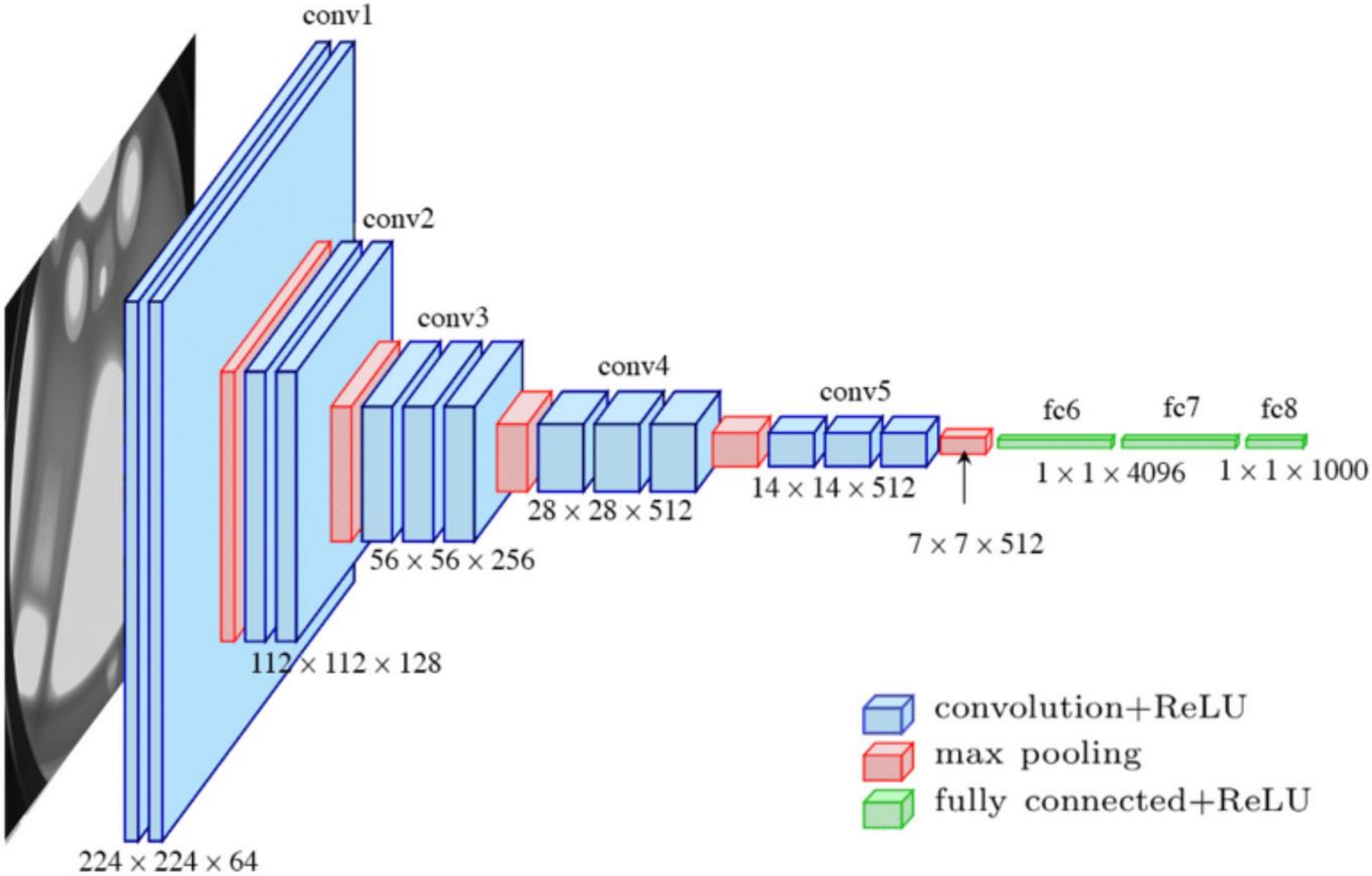
AlexNet (2012)



Concept of Global Average Pooling at last Convolution Layer



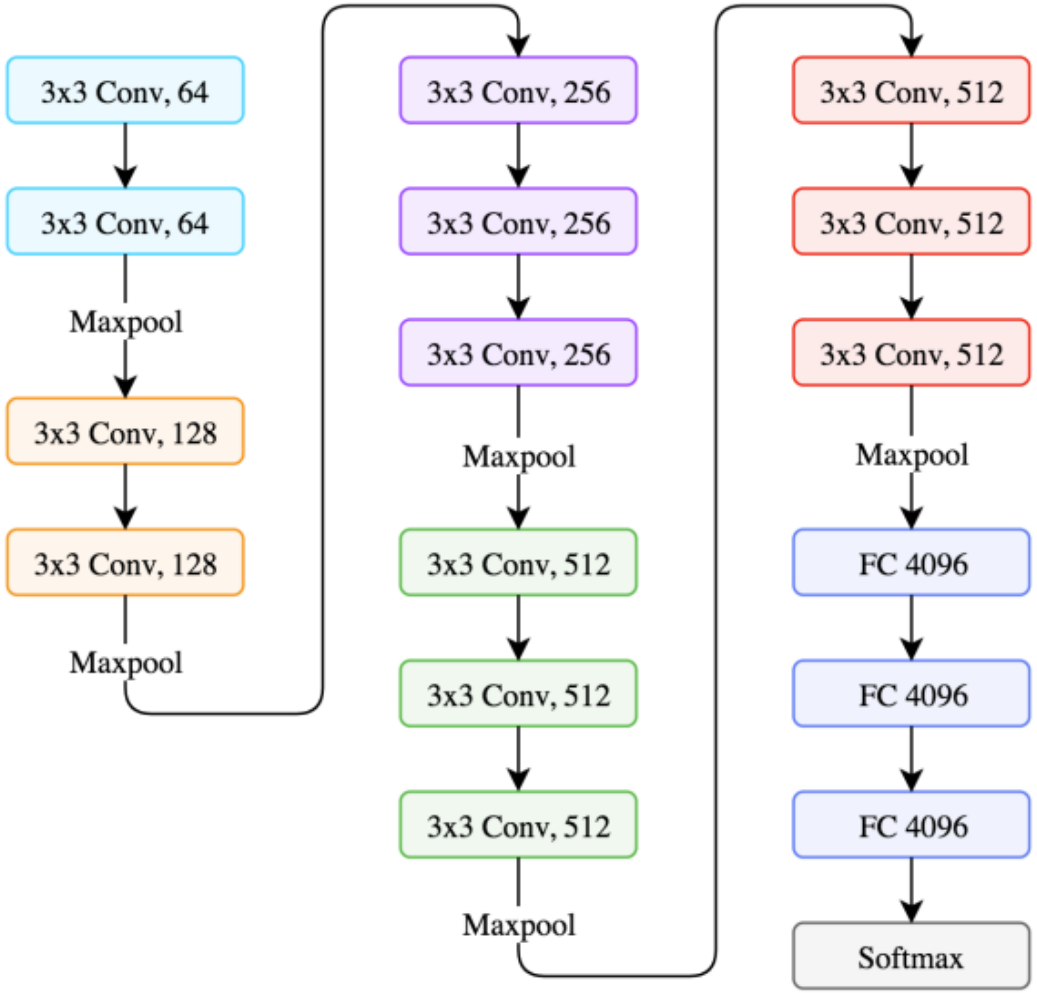
VGG16 (2014)


conv1
conv2
conv3
conv4
conv5
fc6
fc7
fc8
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU

VGG16

Characteristics

- Input size: 224 × 224
- Filter sizes: 3 × 3
- Convolution stride: 1
- Padding: 1
- Max pooling: 2 × 2 with a stride of 2
- ReLU activations
- No fancy input normalizations
- No Local Response Normalizations
- Although deeper, number of weights is not exploding

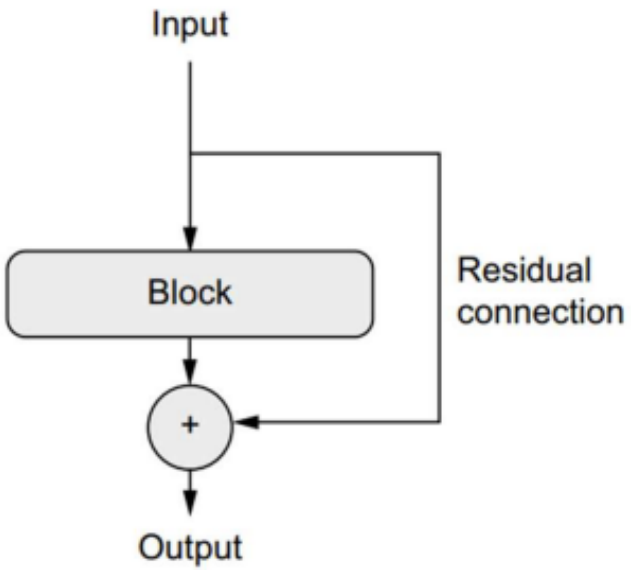


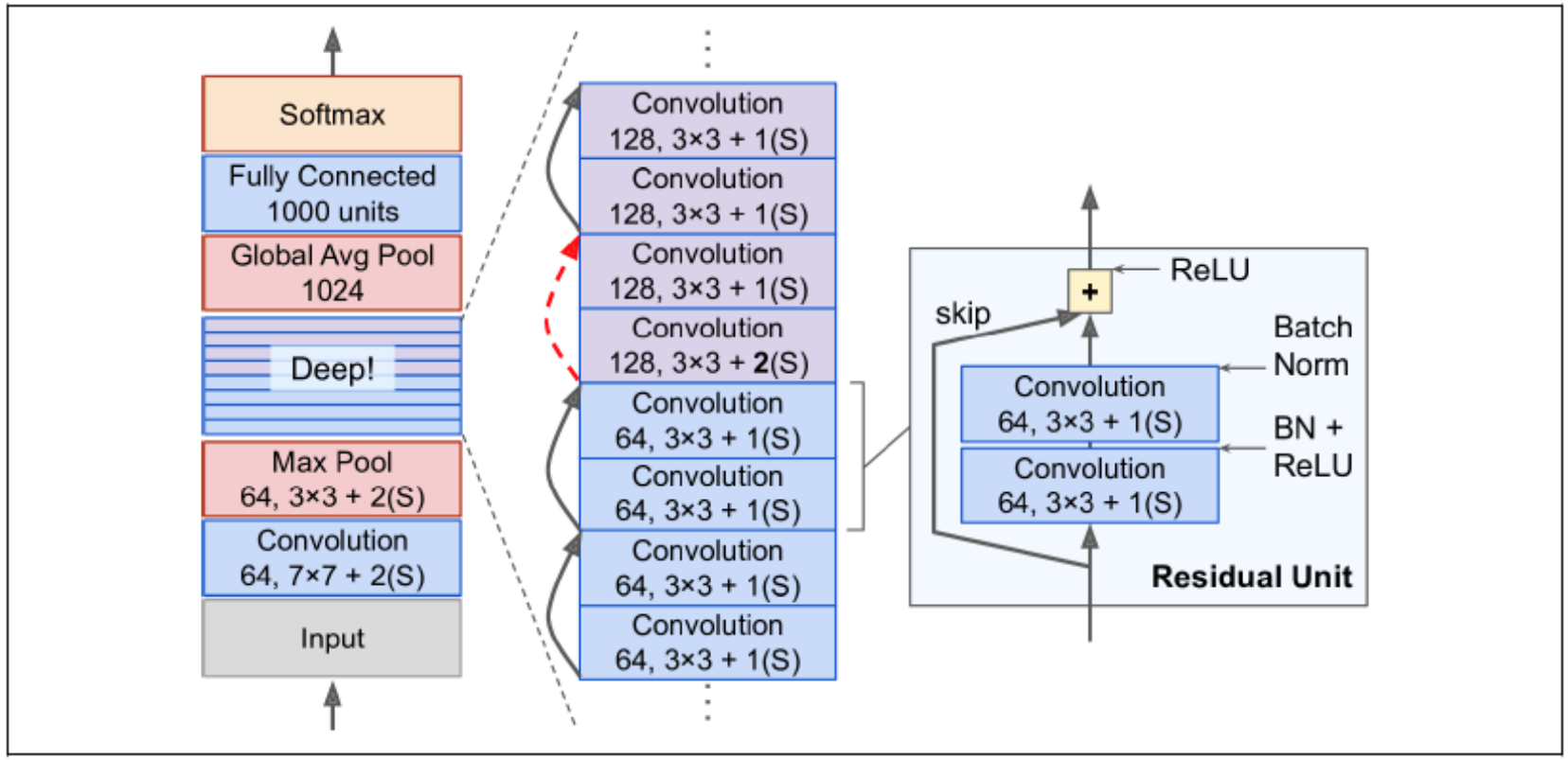
https://uvadlc.github.io/lectures/apr2019/lecture5-modern_convnets.pdf

ResNet (2015)

Deeper hierarchies are intrinsically good because they encourage feature reuse, and therefore abstraction. In general, a deep stack of narrow layers performs better than a shallow stack of large layers. **However, there's a limit to how deep you can stack layers, due to the problem of vanishing gradients.**

→ Residual connection





(a)

34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Avg pool
fc 1000

(b)

Demo Experiment

Classifying Cats and Dogs using AlexNet and VGG-16 models

- Note: When using a pre-trained model, the data that you pass through it should be transformed in the same way that the data the model was trained on.

Updating Entire Classifier Layer

https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html

```
model.classifier =  torch.nn.Sequential (
                                        )

...
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

Augmentation

The small dataset can cause a high variance estimation of model performance.

To avoid this problem altogether by artificially (and cleverly) producing new data from the already available data.
We perform ****data augmentation****.

Data augmentation takes the approach of generating more training data from existing training samples by augmenting the samples via a number of random transformations that yield a believable-looking image. Common transformations include:

* Flipping the image
* Rotating the image
* Zooming in/out of the image

See some sample images below after augmentation:

Thanks

Questions