



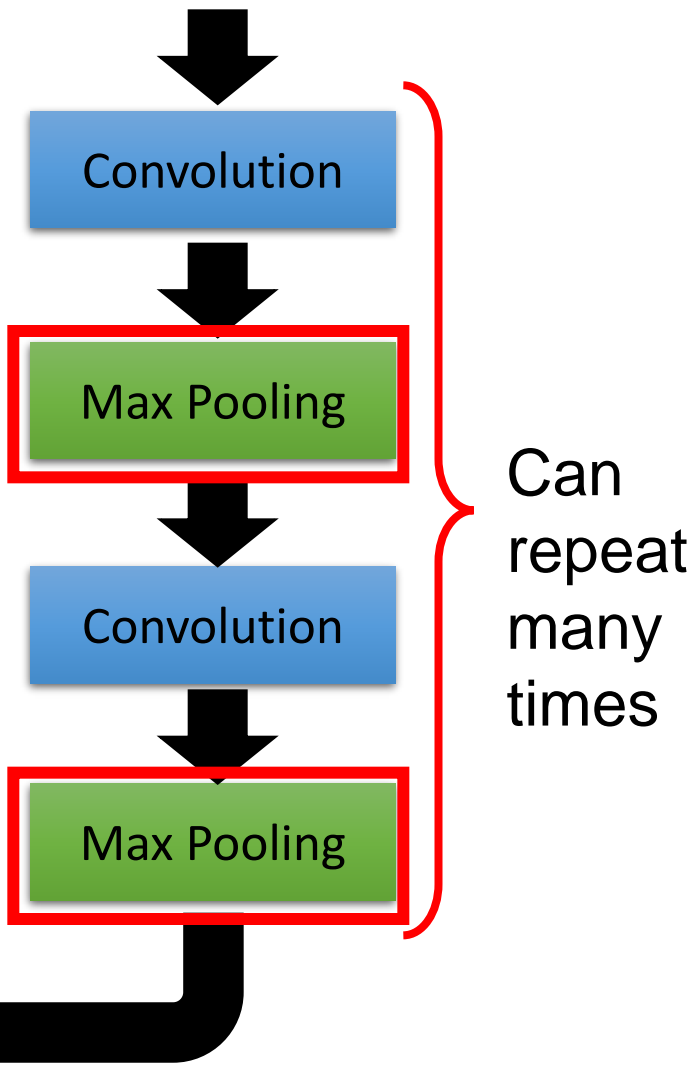
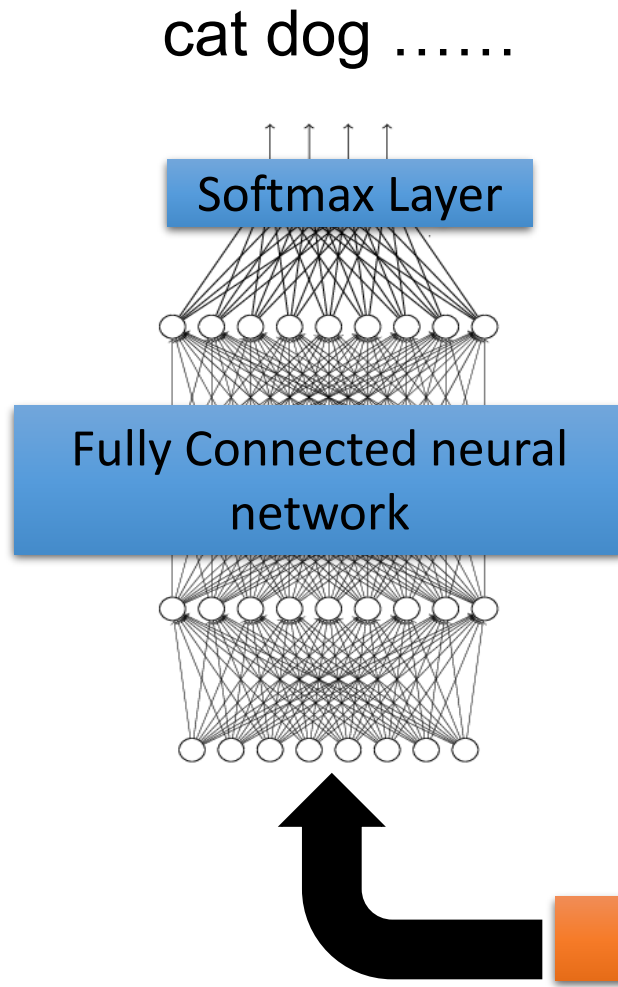
CNNs and Advanced DL methods

Dr. Huseyin Kusetogullari

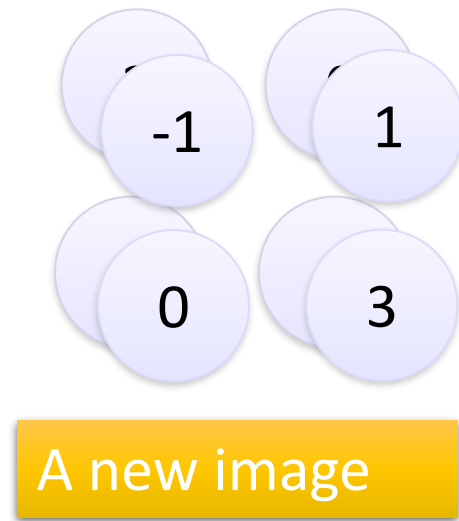
Assignments are available on Canvas.

Any question?

The whole CNN

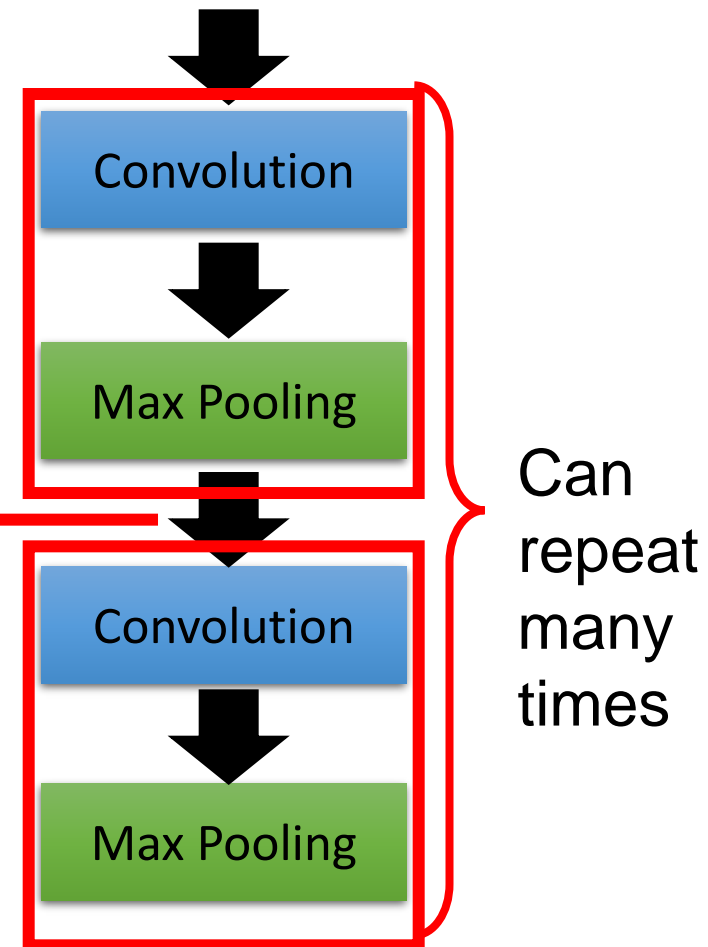


The whole CNN

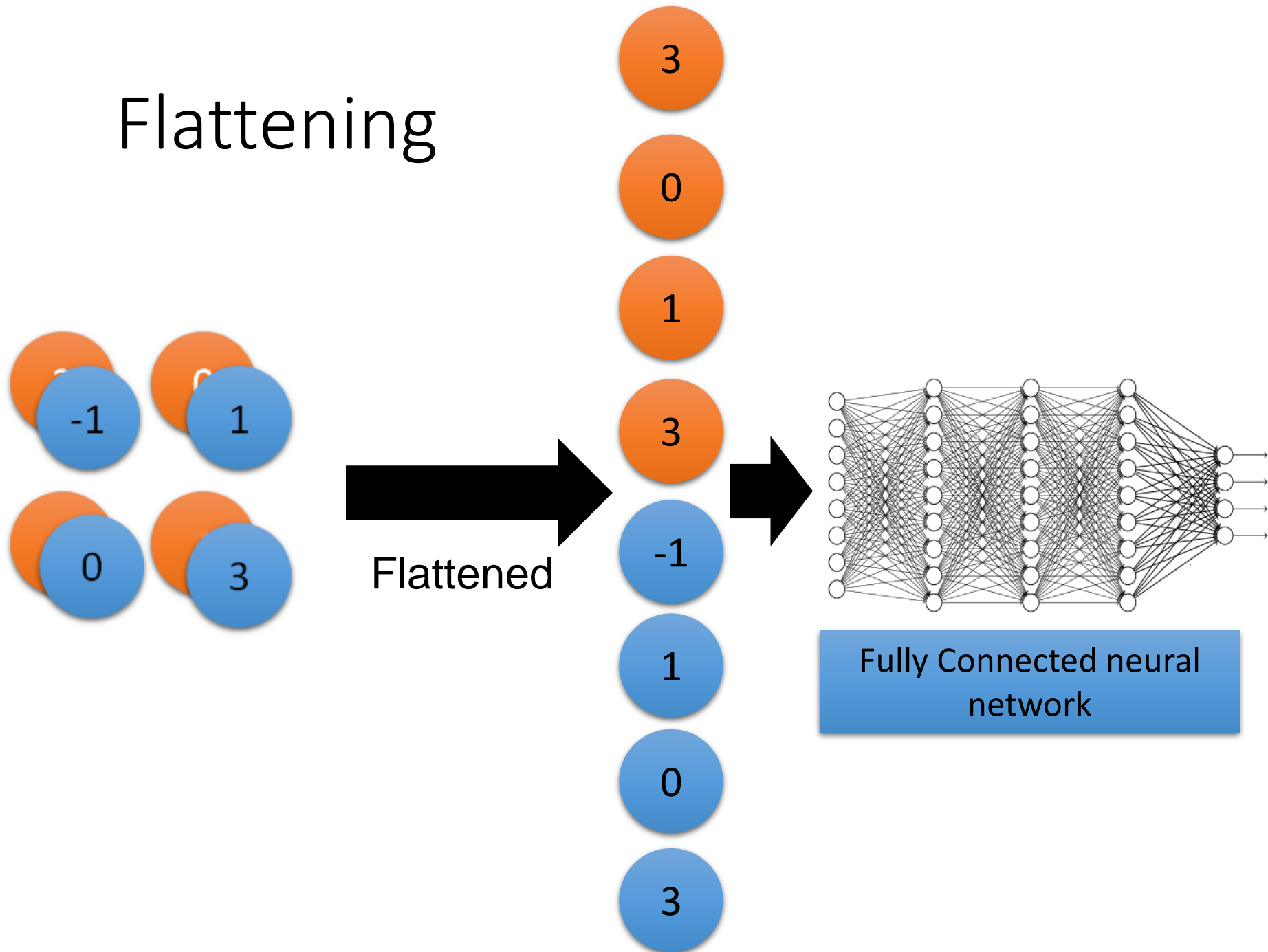


Smaller than the original image

The number of channels is the number of filters



Flattening



CNN in Keras

Only modified the *network structure* and *input format* (vector -> 3-D tensor)

```
model2.add( Convolution2D( 25, 3, 3,  
                           input_shape=(28, 28, 1)) )
```

1	-1	-1	1	-1
-1	1	-1	1	-1
-1	-1	-1	1	-1
		-1	1	-1

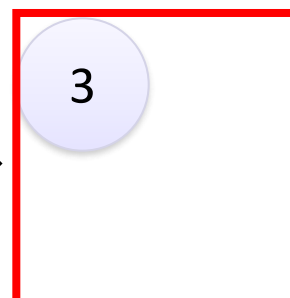
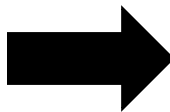
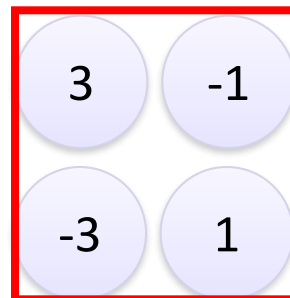
There are
25 3x3
filters.

Input_shape = (28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

```
model2.add(MaxPooling2D( (2, 2) ))
```



input

Convolution

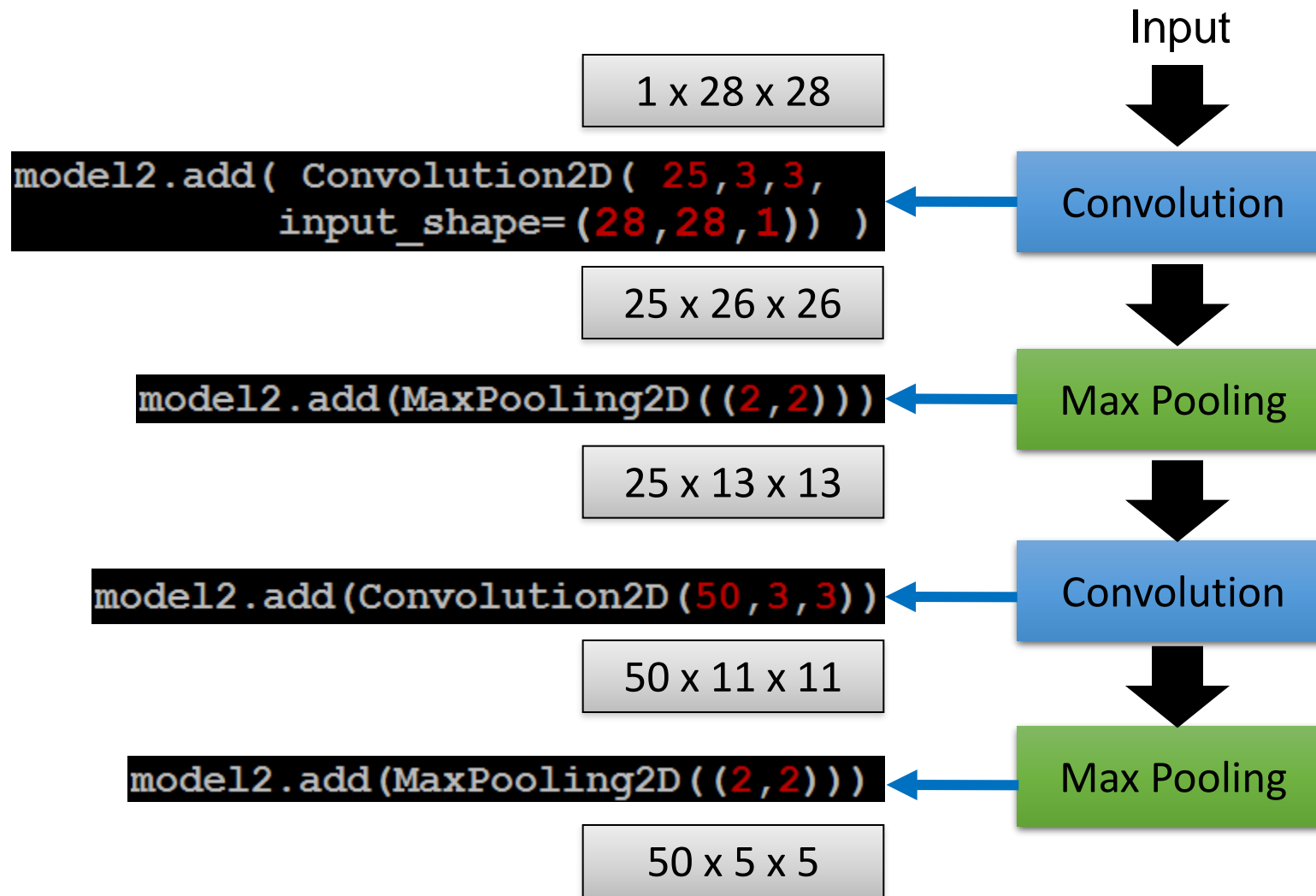
Max Pooling

Convolution

Max Pooling

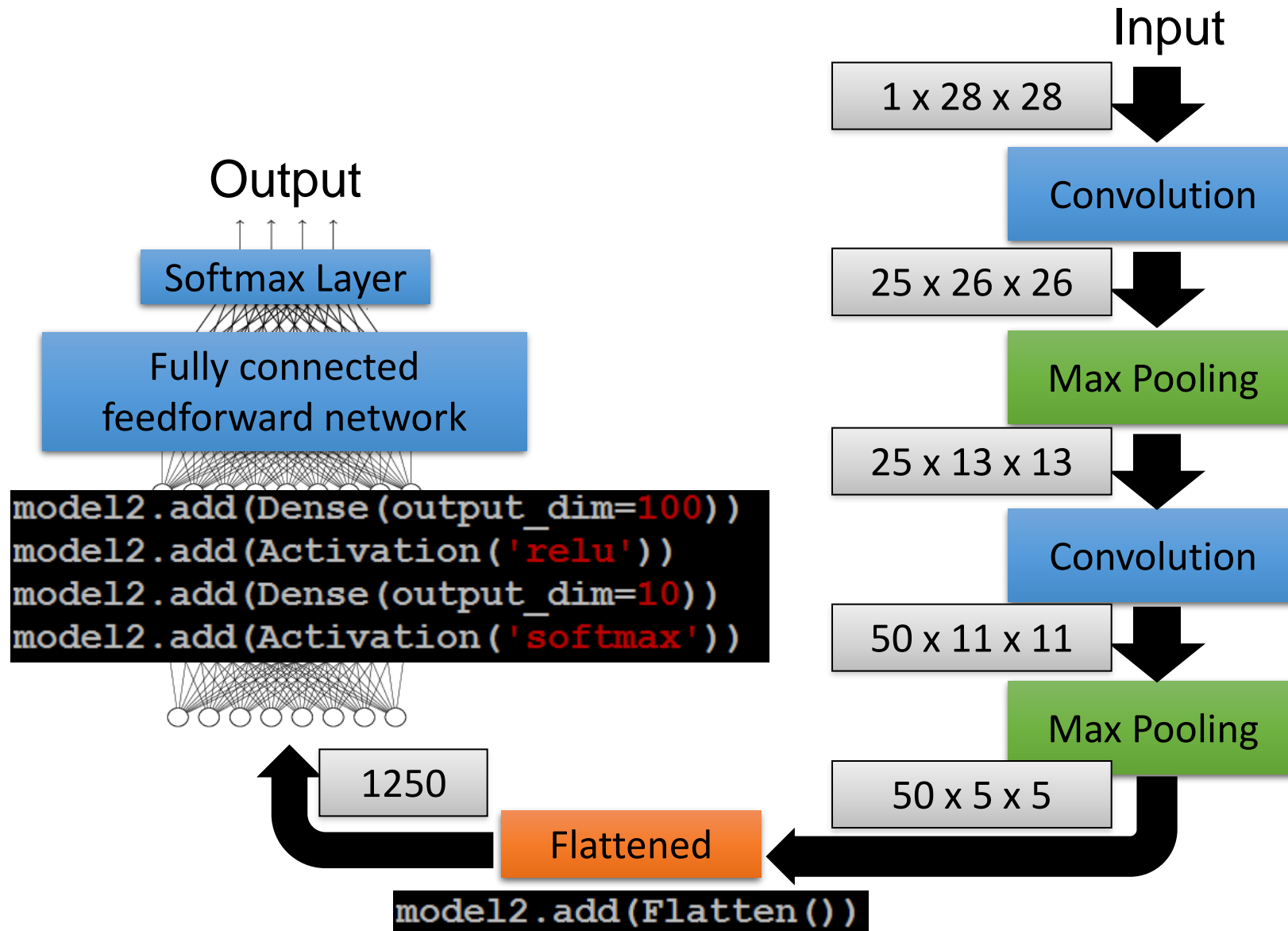
CNN in Keras

Only modified the *network structure* and *input format (vector -> 3-D array)*



CNN in Keras

Only modified the *network structure* and *input format (vector -> 3-D array)*



When to use Deep Learning?

- **Use Traditional Machine Learning when:**
- **1. Small to Medium-Sized Datasets**
- Algorithms like decision trees, SVMs, and logistic regression perform well with limited data.
- **2. Structured or Tabular Data**
- Traditional ML often shines with financial data, customer behavior logs, or medical records.
- Examples: Random Forest, XGBoost, etc.
- **3. Faster Training and Less Compute Power**
- Traditional models are lighter and easier to run on regular CPUs.
- Suitable for environments with limited computational resources.
- **4. Need for Interpretability**
- You often want to **understand the decision-making process** (e.g., in healthcare or finance).
- Traditional models are usually easier to explain (especially linear/logistic regression or decision trees).
-
-

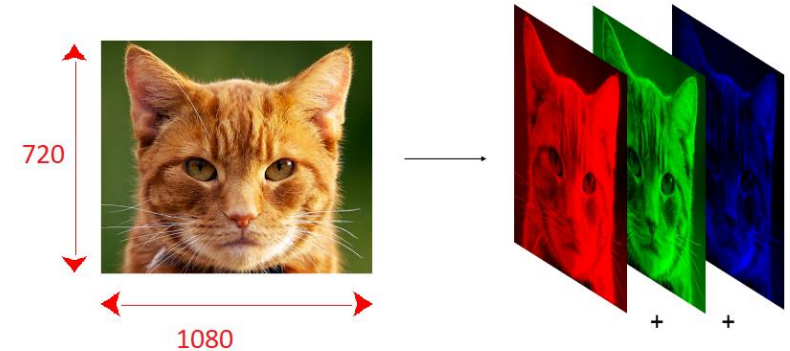
- **Use Deep Learning when:**
- **1. Large Datasets**
- DL models are performed **on huge amounts of data**.
- More data typically leads to better performance.
- **2. Unstructured Data**
- DL dominates in processing:
 - **Images** (CNNs)
 - **Audio/Speech** (RNNs, Transformers)
 - **Text/NLP** (Transformers like BERT, GPT)
- **3. Complex Pattern Recognition**
- When the problem requires learning **non-linear, abstract relationships**, DL can uncover deeper insights.
- Think facial recognition, autonomous driving, sentiment analysis, etc.
- **4. You Have Access to GPUs/TPUs**
- DL often requires specialized hardware to train efficiently.
- Tools like TensorFlow, PyTorch, and CUDA help here.

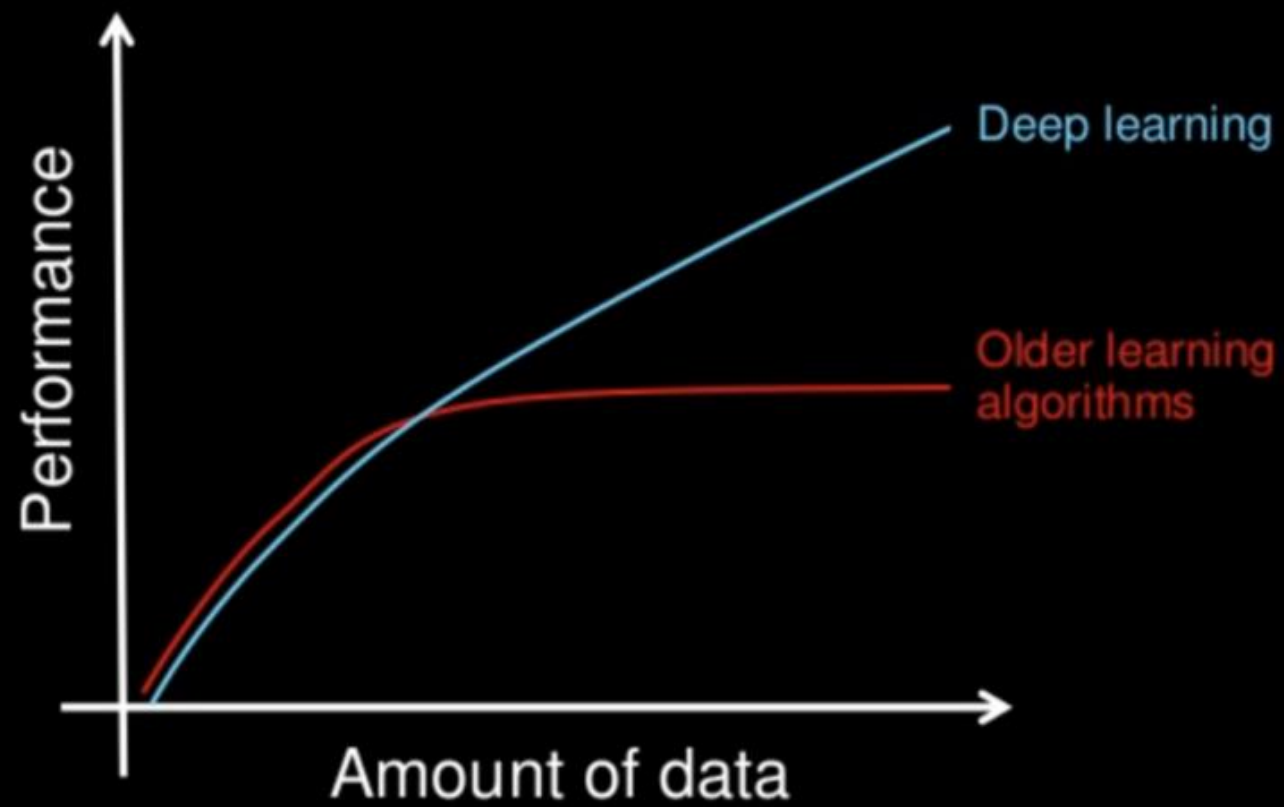
Cont.

Situation	Use Traditional ML	Use Deep Learning
Dataset size	Small/Medium	Large
Data type	Structured (tables)	Unstructured (images, text)
Training speed/efficiency	Faster, low compute	Slower, needs GPUs
Interpretability	Easy to interpret	Often a black box
Task complexity	Simple to moderate	Highly complex

Example

- For training purposes, we have taken 10000 images initially.
- So, total data points included in the training process is
- $1080 * 720 * 3 * 10000 = 2332800 * 10000 = \mathbf{23 \text{ billion data points}}$
- An algorithm with nearly 23 billion data points may take an enormous amount of time to train or may even fail due to huge dimensions of data.
- So, we can apply deep learning models to automatically extract useful features from these 23 billion data points which can help in classifying dog vs cat.
- We generally use Convolution Neural Networks or CNN for processing these huge features and finding only relevant distinctive features which can classify whether the image is of cat or a dog.





What is Image Classification problem?

- Image classification is a technique that involves analyzing the content of an image to attach tags or labels to it. This distributes the images into different classes. The purpose of this process is to digitally explain the contents of the image to a machine. Image classification is combined with image localization to enable an object detection system to find objects in images.

Single-label vs. multi-label classification

★ Single-label classification

airplane

automobile

bird

cat

deer

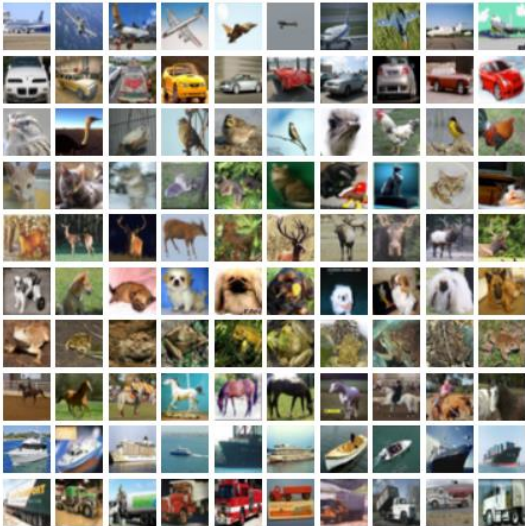
dog

frog

horse

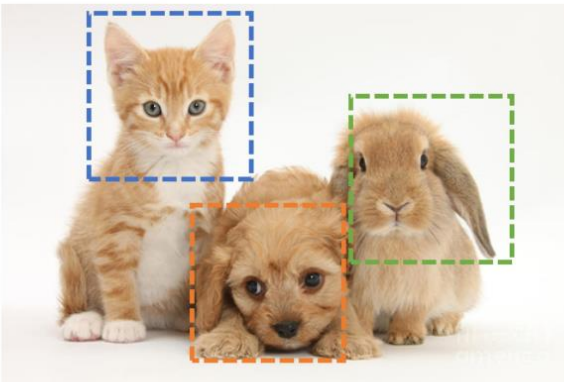
ship

truck



CIFAR-10 Dataset

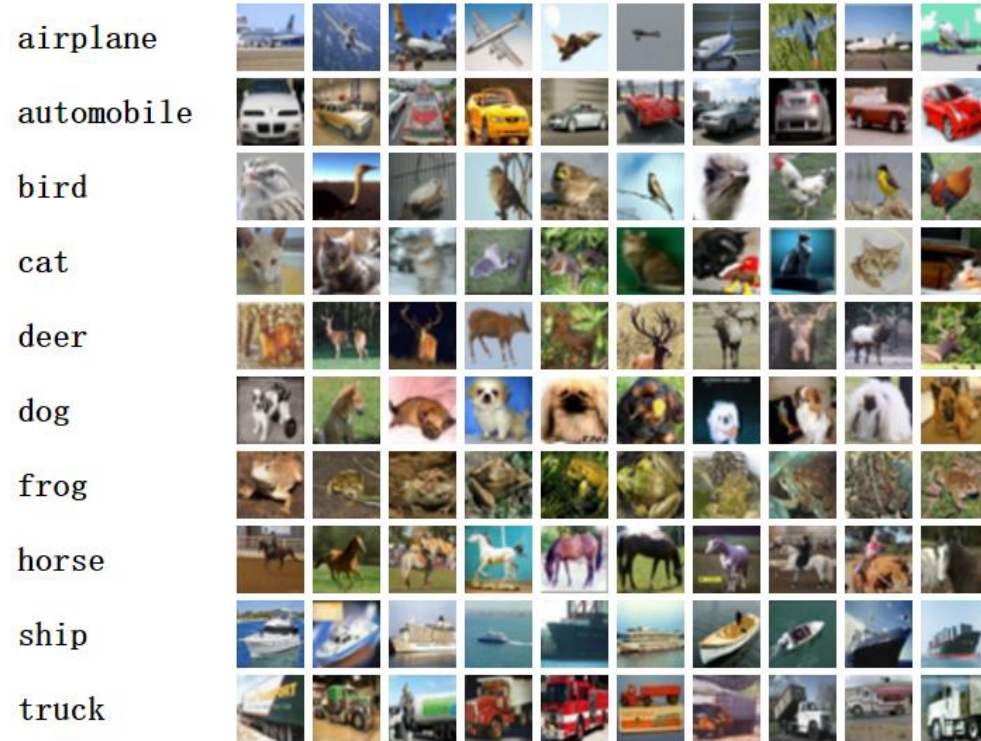
★ Multi-label classification



Cat, Dog, Rabbit

CIFAR10 dataset and state of the art

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.



Accuracy

Model	Acc.
VGG16	92.64%
ResNet18	93.02%
ResNet50	93.62%
ResNet101	93.75%
ResNeXt29(32x4d)	94.73%
ResNeXt29(2x64d)	94.82%
DenseNet121	95.04%
PreActResNet18	95.11%
DPN92	95.16%

Cont.



Single-label image classification is a traditional image classification problem where each image is associated with only one label or class. For instance, an image of a cat can be labeled as “cat” and nothing else. The task of the classifier is to predict the label for a given image.



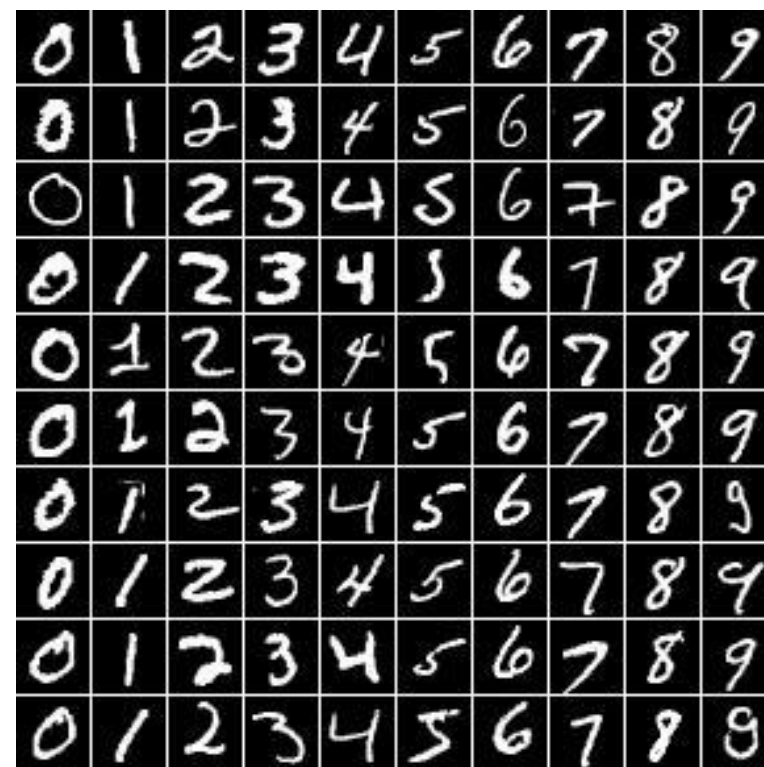
Multi-label image classification is an extension of the single-label image classification problem, where an image can have multiple labels or classes associated with it.



Multi-label image classification is a **more complex problem** than single-label image classification since it requires the classifier to identify multiple objects or features in an image and attach them with their corresponding labels.

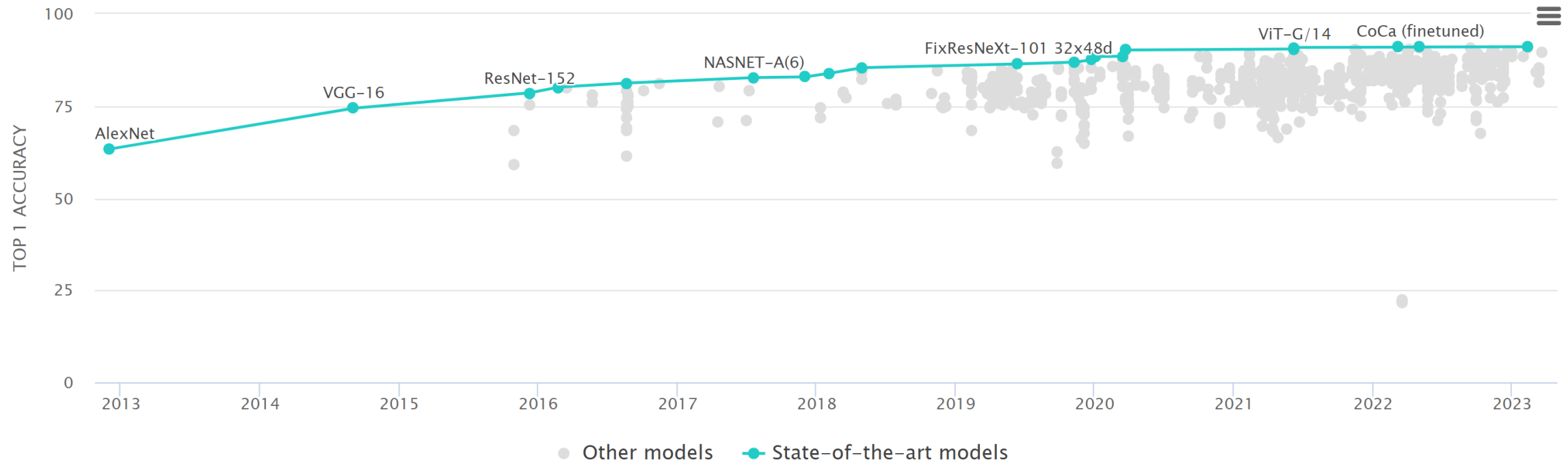
MNIST dataset

- The MNIST database of handwritten digits,
- available from this page,
- has a training set of 60,000 examples, and a test set of 10,000 examples.
- It is a subset of a larger set available from NIST.
- The digits have been size-normalized and centered in a fixed-size image.



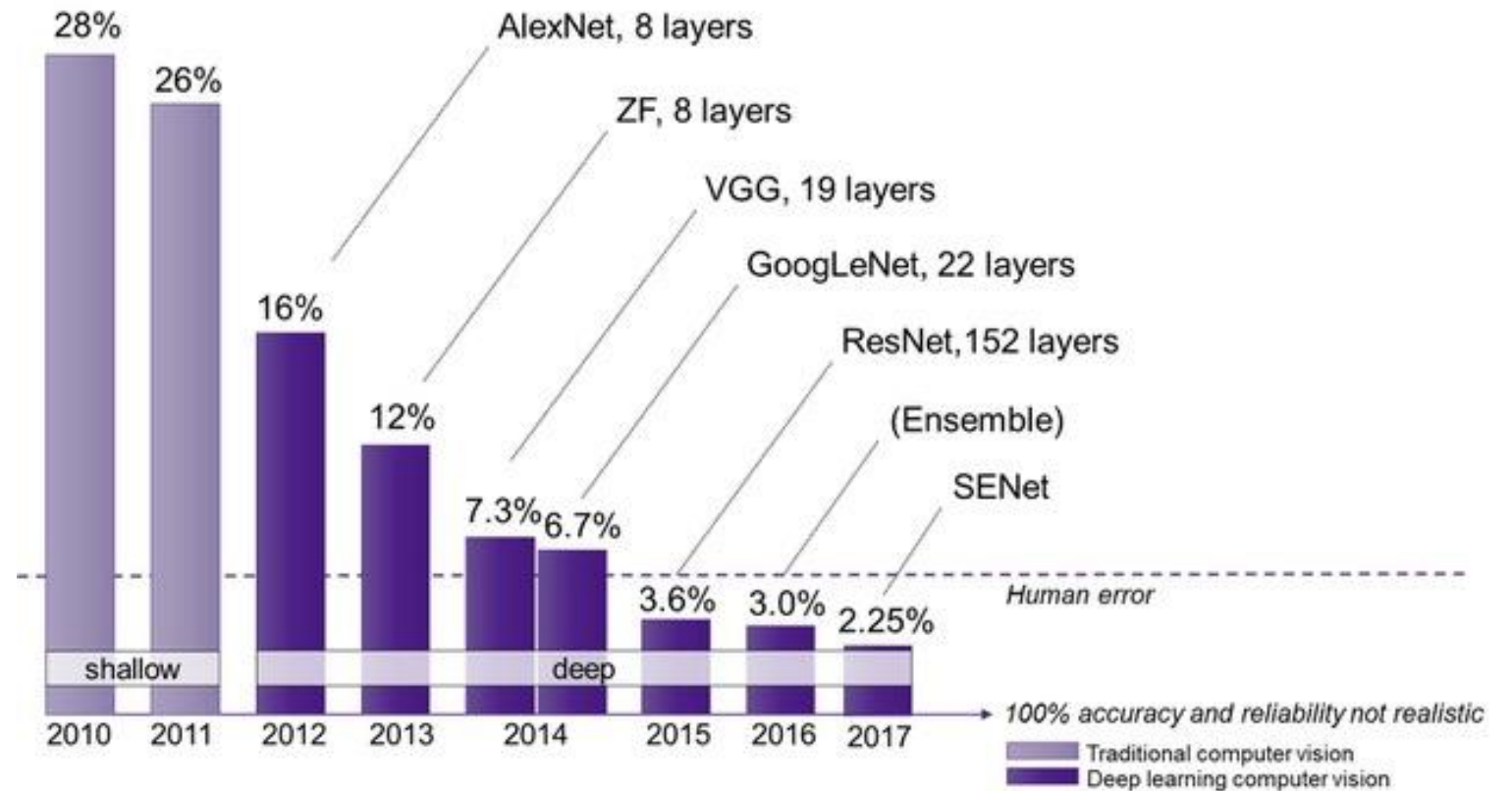
The best performing algorithms over the years for ImageNet Contest

<https://paperswithcode.com/sota/image-classification-on-imagenet>



Evolution of networks

Imagenet large scale visual recognition challenge winners



Case studies

AlexNet. The first work that popularized Convolutional Networks in Computer Vision was the [AlexNet](#), developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton.

The AlexNet was submitted to the [ImageNet ILSVRC challenge](#) in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error).

The Network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single CONV layer always immediately followed by a POOL layer).

Case studies

GoogLeNet. The ILSVRC 2014 winner was a Convolutional Network from [Szegedy et al.](#) from Google.

Its main contribution was the development of an **Inception Module** that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).

Additionally, this paper uses **Average Pooling instead** of Fully Connected layers at the top of the ConvNet, eliminating a large amount of parameters that do not seem to matter much.

There are also several follow-up versions to the GoogLeNet, most recently [Inception-v4](#).

Case studies

VGGNet. The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the [VGGNet](#).

Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.

Their [pretrained model](#) is available for plug and play use in Caffe. A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M).

Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

Image Classifiers

- VGG-16
- ResNet
- DenseNet

VGG (Visual Geometry Group)

- VGG16 is a Convolution Neural Network (CNN) architecture.
- Most unique thing about VGG16 is that it focuses on having convolution layers of 3x3 filter with a stride 1 and used same padding and maxpool layer of 2x2 filter of stride 2.
- In the end it has 3 FC (fully connected layers) followed by a softmax for output.
- This network has about 138 million (approx.) parameters.

1.Convolutional Layers: There are 13 convolutional layers in VGG-16. Each convolutional layer applies filters to the previous layer's output (or the input image for the first layer) to create feature maps.

2.Pooling Layers: There are 5 max-pooling layers in VGG-16, each following a block of convolutional layers. Pooling layers reduce the spatial size of the representation, which reduces the number of parameters and computation in the network.

3.Fully Connected Layers: After the convolutional and pooling layers, VGG-16 has 3 fully connected layers. The first two have 4096 channels each, and the third performs classification and has 1000 channels (for the 1000 classes of the ImageNet challenge).

•Convolutional Layers: $2 (64) + 2 (128) + 3 (256) + 3 (512) + 3 (512) = 13$

•Pooling Layers: There are 5 "pool/2" layers.

•Fully Connected Layers: There are 3, marked as "fc 4096", "fc 4096", and "fc 1000".

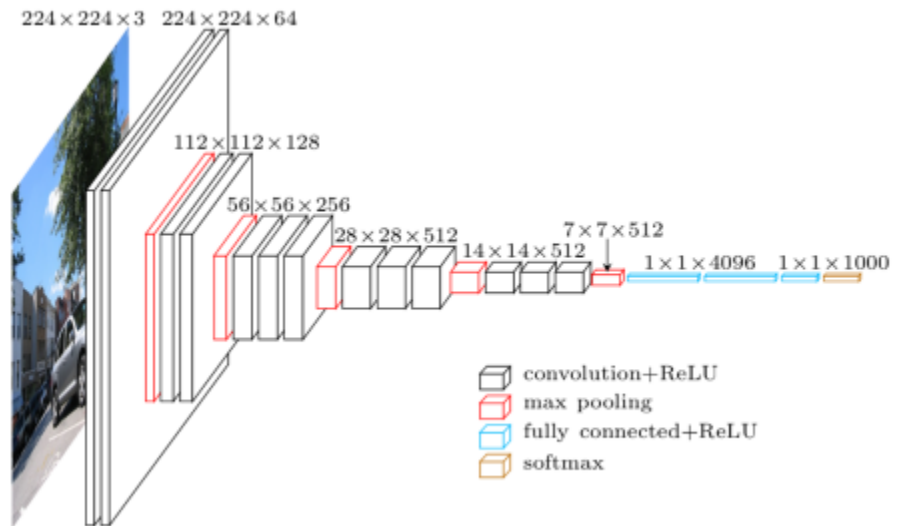
Adding these up gives us a total of $13 + 5 + 3 = 21$ layer



VGG-16

The VGG-16 network is named for having 16 weight layers, which includes 13 convolutional layers and 3 fully connected layers.

VGG- 16



1

```

model = Sequential()

model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=
(3,3),padding="same", activation="relu"))

model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",
activation="relu"))

model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

```

2

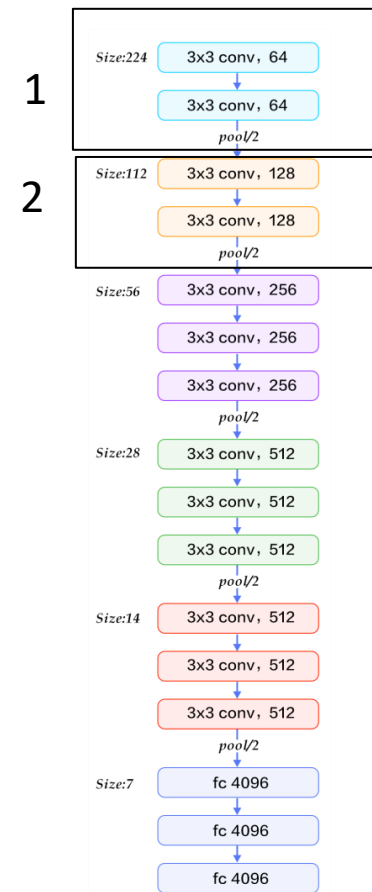
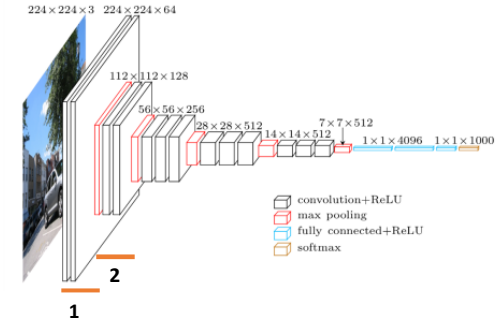
```

model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
activation="relu"))

model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
activation="relu"))

model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

```



3

```
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
activation="relu"))
```

```
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
activation="relu"))
```

```
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
activation="relu"))
```

```
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
```

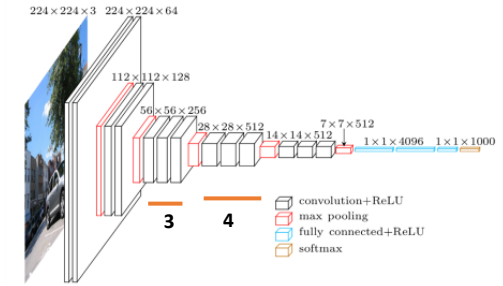
4

```
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
```

```
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
```

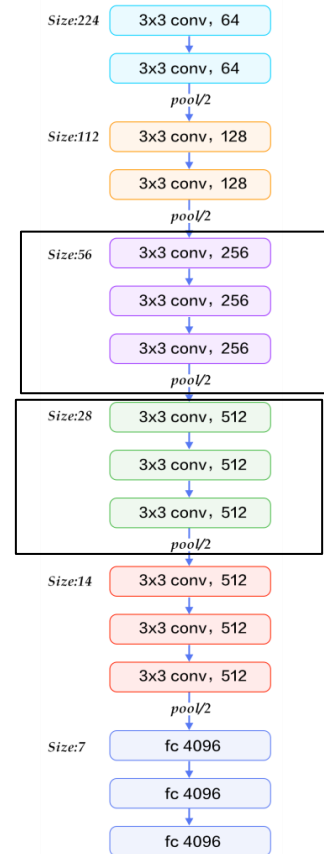
```
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
```

```
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
```



3

4



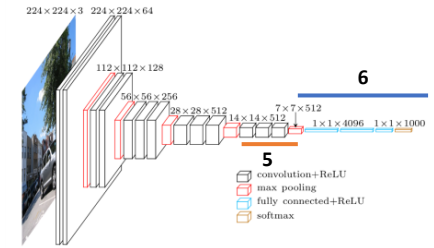
5

```
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
```

```
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
```

```
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
```

```
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
```



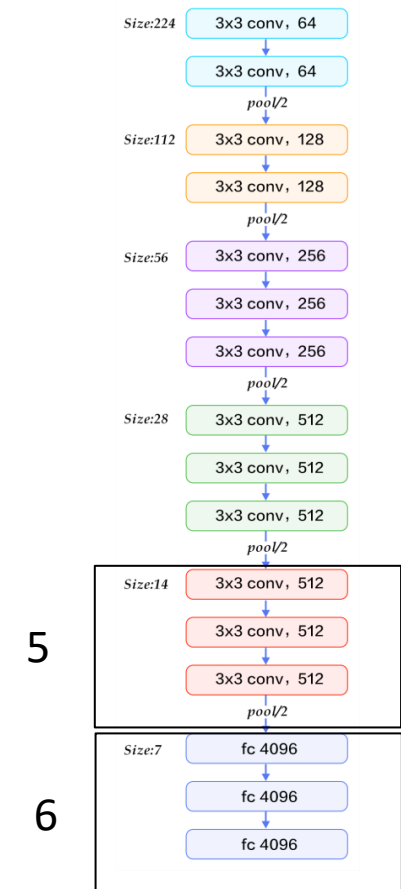
6

```
model.add(Flatten())
```

```
model.add(Dense(units=4096, activation="relu"))
```

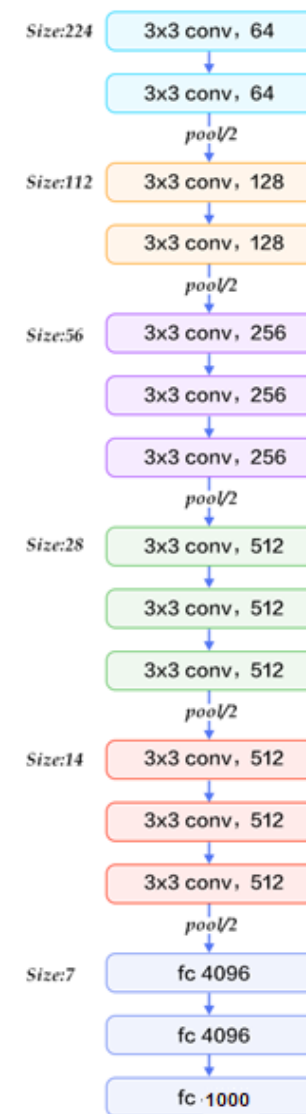
```
model.add(Dense(units=4096, activation="relu"))
```

```
model.add(Dense(units=1000, activation="softmax"))
```



Summary of the model

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 224, 64)	1792
conv2d_2 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_3 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_4 (Conv2D)	(None, 112, 112, 128)	147584
max_pooling2d_2 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_5 (Conv2D)	(None, 56, 56, 256)	295168
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_7 (Conv2D)	(None, 56, 56, 256)	590080
max_pooling2d_3 (MaxPooling2D)	(None, 28, 28, 256)	0
conv2d_8 (Conv2D)	(None, 28, 28, 512)	1180160
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2359808
conv2d_10 (Conv2D)	(None, 28, 28, 512)	2359808
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 512)	0
conv2d_11 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_13 (Conv2D)	(None, 14, 14, 512)	2359808
max_pooling2d_5 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 4096)	102764544
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
dropout_2 (Dropout)	(None, 4096)	0
dense_3 (Dense)	(None, 2)	8194
Total params: 134,268,738		
Trainable params: 134,268,738		
Non-trainable params: 0		



VGG

Advantages:

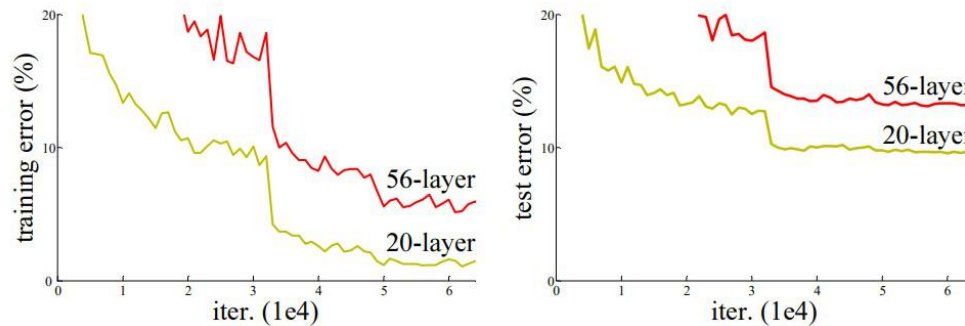
- 1.Simplicity:** VGG is straightforward and easy to understand, with a simple architecture consisting of repeated convolutional layers followed by pooling layers.
- 2.Flexibility:** It allows for easy modification of network depth and width by altering the number of layers and filter sizes.
- 3.State-of-the-Art Performance:** Despite its simplicity, VGG achieved competitive results on various image classification tasks during its time.

Disadvantages:

- 1.High Computational Cost:** VGG has a large number of parameters, making it computationally expensive, especially for training and inference on large datasets.
- 2.Memory Intensive:** The large number of parameters also requires a significant amount of memory, which can be a limitation for deployment on resource-constrained devices.
- 3.Overfitting:** Due to its large number of parameters, VGG is prone to overfitting, especially on smaller datasets.

Challenges and Problems

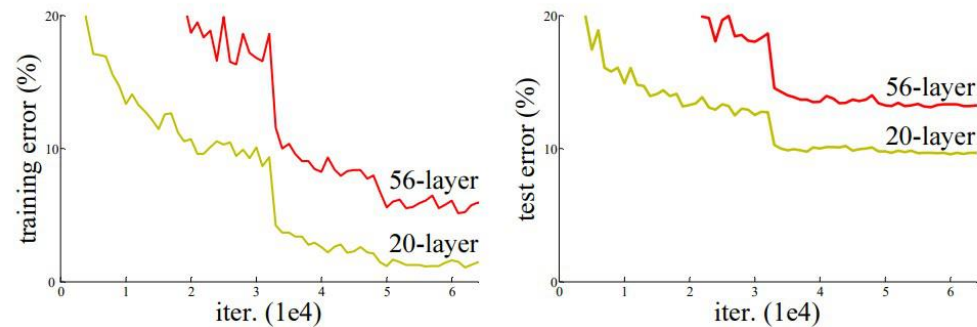
- After the first CNN-based architecture that win the ImageNet 2012 competition, every subsequent winning architecture uses more layers in a deep neural network to reduce the error rate.
- This causes the gradient to become 0 or too large.
- Thus when we increases number of layers, the training and test error rate also increases.



Example: we can observe that a 56-layer CNN gives more error rate on both training and testing dataset than a 20-layer CNN architecture.

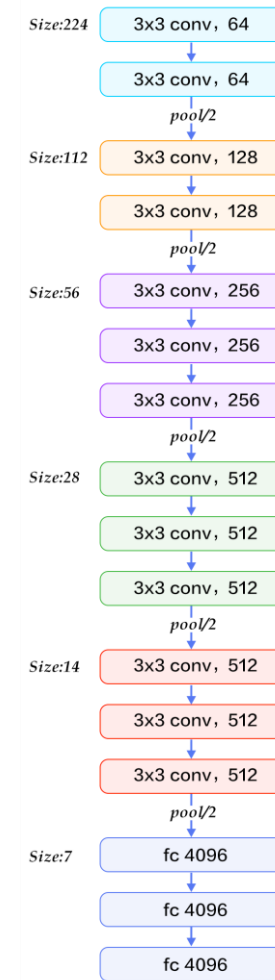
After analyzing more on error rate the we can reach a conclusion that it is caused by vanishing/exploding gradient.

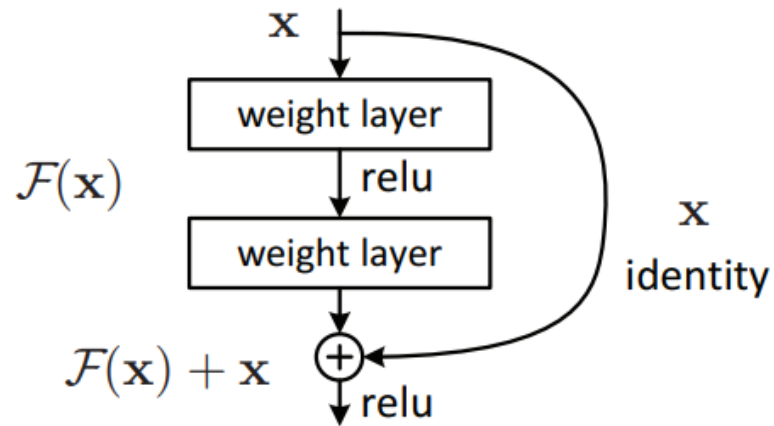
In a traditional deep neural network, each layer tries to **learn the direct mapping from its input to the output**. As the network depth increases, this can become increasingly difficult due to problems **like vanishing gradients**, where the gradients (used in training the network) become so small that the weights of the network stop updating effectively.



Example: we can observe that a 56-layer CNN gives more error rate on both training and testing dataset than a 20-layer CNN architecture. After analyzing more on error rate the we can reach a conclusion that it is caused by vanishing/exploding gradient.

Summary of the model





$$H(x) := F(x) + x.$$

Residual Networks (ResNet) – Deep Learning

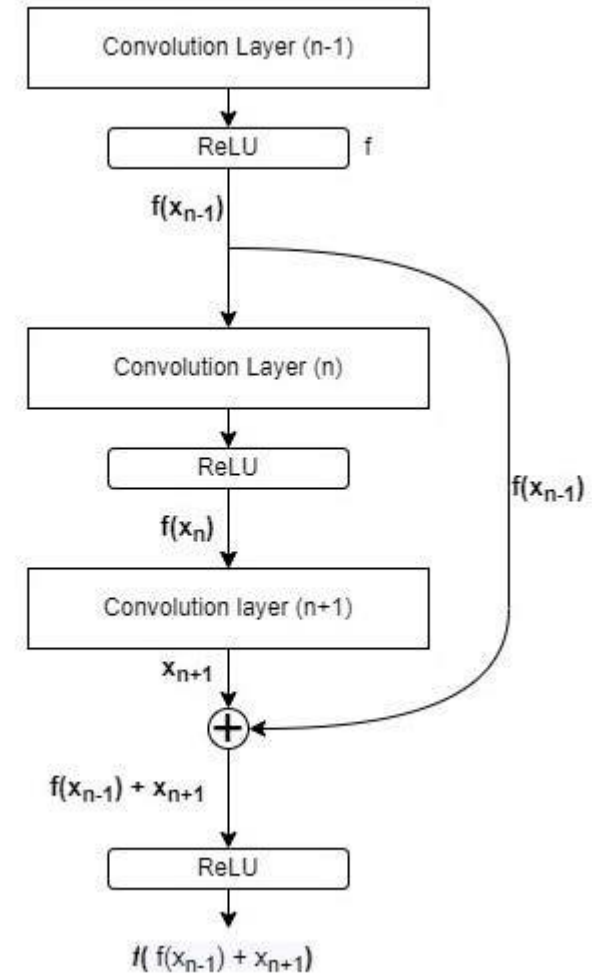
- In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Blocks.
- ResNet was developed by researchers from the Microsoft Research lab
- In this network, we use a technique called **skip connections**.
- The skip connection connects activations of a layer to further layers by skipping some layers in between.
- This forms a residual block.
- Resnets are made by stacking these residual blocks together.

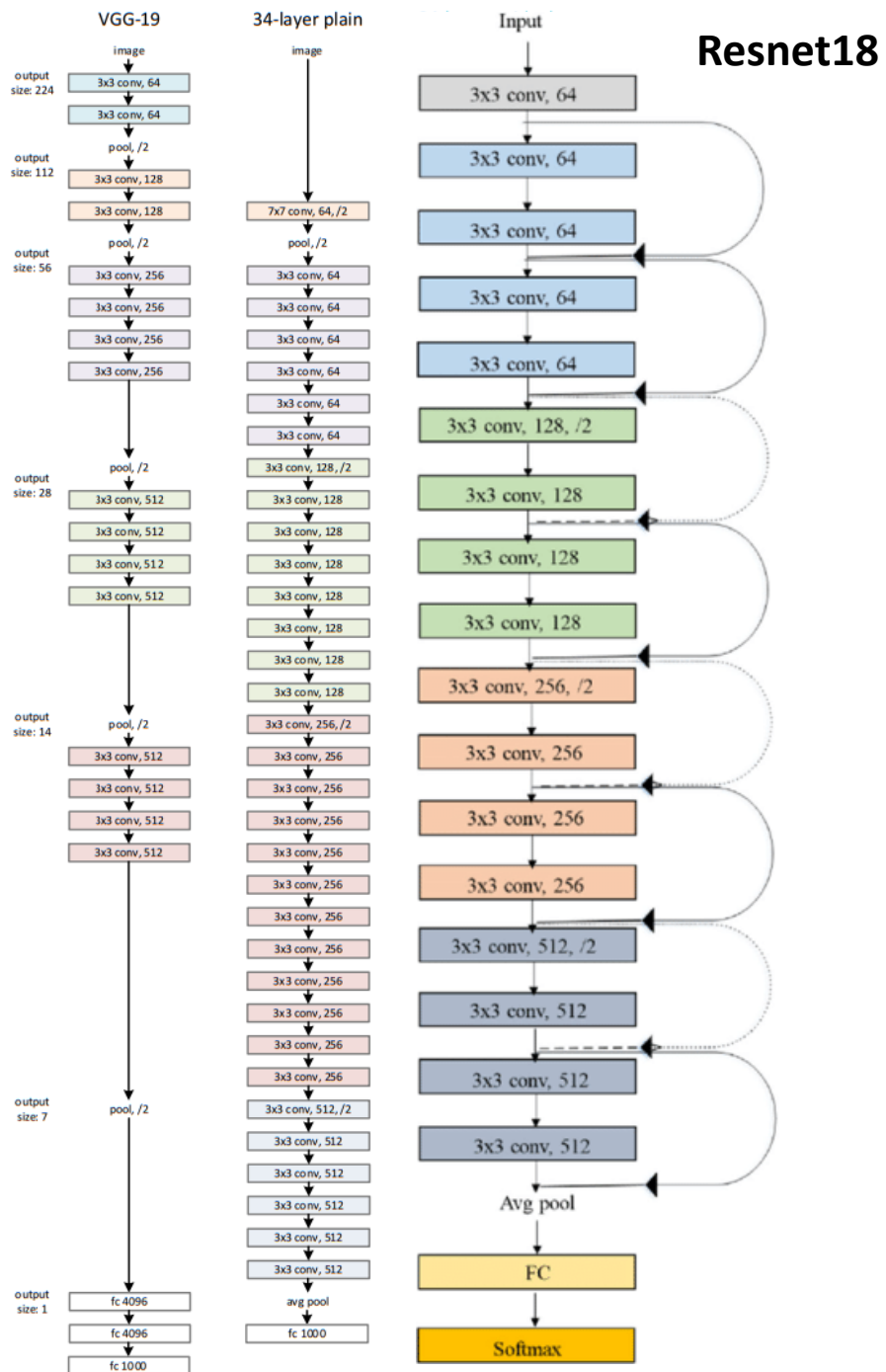
Residual Networks (ResNet) – Deep Learning

Shortcut Connections (Skip Connections): These are the core of ResNet architecture. Instead of trying to learn an end-to-end mapping directly, ResNet layers learn residual mappings.

These networks are designed to learn the differences or residuals between inputs and outputs, rather than attempting to learn the desired output directly from the input.

This approach is based on the insight that it can be easier for layers to learn adjustments to the identity function rather than to learn the mapping from inputs to outputs from scratch, especially as the network gets deeper.

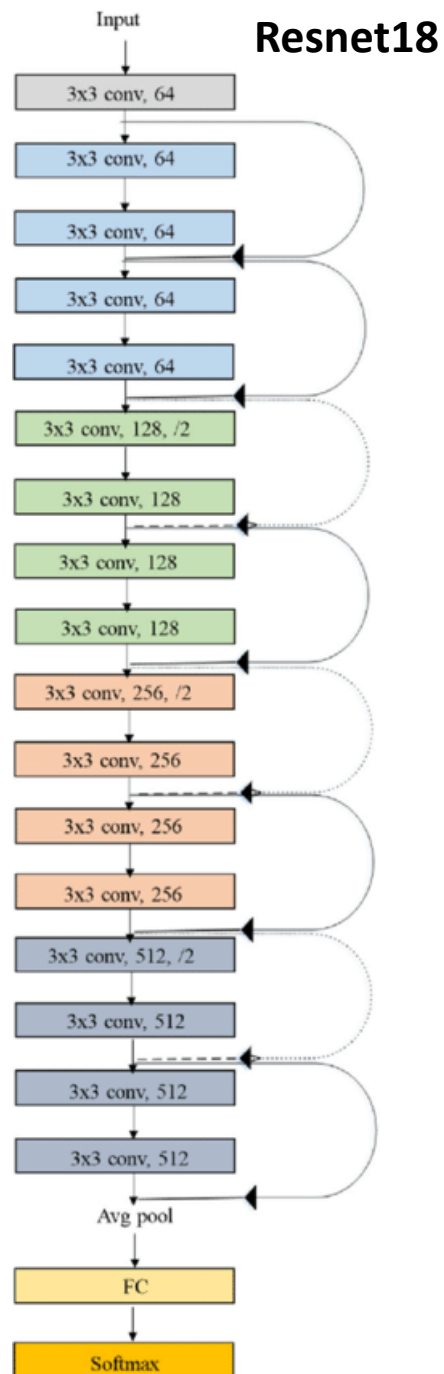




* **The ImageNet dataset:** The ImageNet project is a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided.

model	top-1 err.	top-5 err.
VGG-16	28.07	9.33
GoogLeNet	-	9.15
PRReLU-net	24.27	7.38
plain-34	28.54	10.02
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	21.43	5.71

top-1 and top-5 Error rate on ImageNet Validation Set.



1. **Input Layer:** This layer takes the input image.
2. **Convolutional Layer:** The first layer uses a 7x7 convolution with 64 filters, followed by batch normalization and a ReLU activation function.
3. **Max Pooling Layer:** A 3x3 max pooling layer with stride 2.
4. **Residual Blocks:**
 - **2 Blocks of BasicBlock:** Each block consists of two 3x3 convolutions, each with 64 filters. Between these convolutions, batch normalization and ReLU activation are applied. Each block has a shortcut connection that adds the input of the block to its output.
 - **2 Blocks of BasicBlock:** Similar structure as the first set, but with 128 filters and a downsampling layer at the beginning of the first block.
 - **2 Blocks of BasicBlock:** These blocks use 256 filters with a similar structure and include downsampling at the beginning of the first block.
 - **2 Blocks of BasicBlock:** Finally, these blocks use 512 filters. Downsampling is also applied at the start of the first block.
5. **Global Average Pooling:** A pooling layer that reduces each feature map to a single number.
6. **Fully Connected Layer:** This layer outputs the predictions for the classes.
7. **Softmax:** This layer converts the outputs to probability scores for each class.

* Each residual block features a shortcut connection that helps to alleviate the vanishing gradient problem in deep networks by allowing gradients to flow through the network more effectively during training.

* This architecture makes ResNet-18 particularly effective for deep learning tasks where capturing complex patterns in large datasets is necessary.

```
import torch
import torch.nn as nn
import torchvision.models.resnet as resnet
```

```
class BasicBlock(resnet.BasicBlock):
    expansion = 1
```

```
class ResNet18(nn.Module):
    def __init__(self, num_classes=1000):
        super(ResNet18, self).__init__()
        self.base_model = resnet.ResNet(BasicBlock, [2, 2, 2, 2], num_classes=num_classes)
        self.base_model.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.base_model.bn1 = nn.BatchNorm2d(64)
        self.base_model.relu = nn.ReLU(inplace=True)
        self.base_model.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.base_model.layer1 = self._make_layer(BasicBlock, 64, 2)
        self.base_model.layer2 = self._make_layer(BasicBlock, 128, 2, stride=2)
        self.base_model.layer3 = self._make_layer(BasicBlock, 256, 2, stride=2)
        self.base_model.layer4 = self._make_layer(BasicBlock, 512, 2, stride=2)
        self.base_model.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.base_model.fc = nn.Linear(512 * BasicBlock.expansion, num_classes)
```

```
def _make_layer(self, block, planes, blocks, stride=1):
    downsample = None
    if stride != 1 or self.base_model.inplanes != planes * block.expansion:
        downsample = nn.Sequential(
            nn.Conv2d(self.base_model.inplanes, planes * block.expansion,
                      kernel_size=1, stride=stride, bias=False),
            nn.BatchNorm2d(planes * block.expansion),
        )

    layers = []
    layers.append(block(self.base_model.inplanes, planes, stride, downsample))
    self.base_model.inplanes = planes * block.expansion
    for _ in range(1, blocks):
        layers.append(block(self.base_model.inplanes, planes))

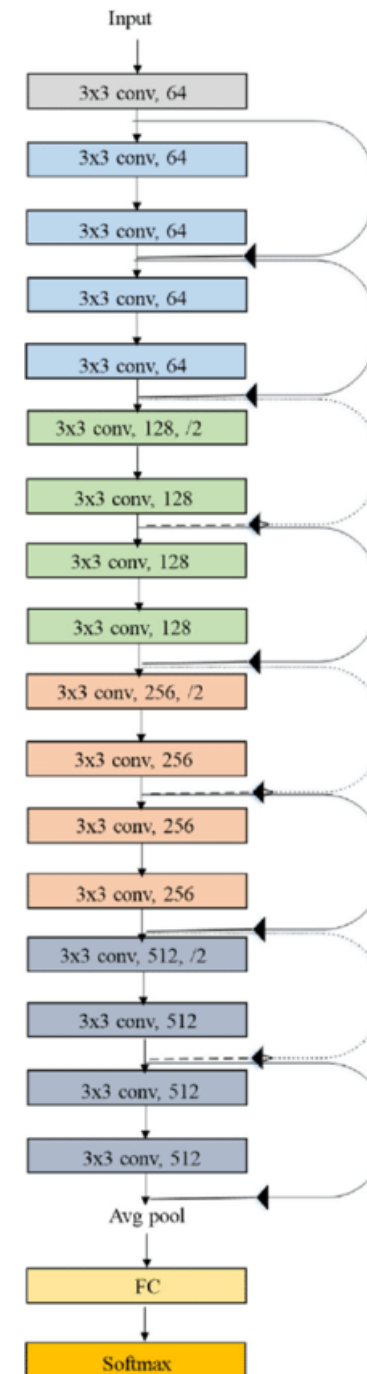
    return nn.Sequential(*layers)
```

```
layers = []
layers.append(block(self.base_model.inplanes, planes, stride, downsample))
self.base_model.inplanes = planes * block.expansion
for _ in range(1, blocks):
    layers.append(block(self.base_model.inplanes, planes))

return nn.Sequential(*layers)
```

```
def forward(self, x):
    return self.base_model(x)
```

```
# Initialize the model and print its architecture
model = ResNet18(num_classes=1000)
print(model)
```



* BasicBlock: This is the class that defines the basic residual block used by the ResNet model. In the case of ResNet-18, the basic block consists of two convolutional layers with batch normalization and ReLU activations, along with a skip connection that adds the input of the block to its output to form a residual connection.

* [2, 2, 2, 2]: This array defines the architecture of the ResNet-18 model, specifically how many BasicBlock units are in each of the four layers. For ResNet-18, there are 2 blocks in each of the four layers.

* num_classes=num_classes: This defines the number of output classes for the final fully connected layer of the network.

Residual Networks

Advantages:

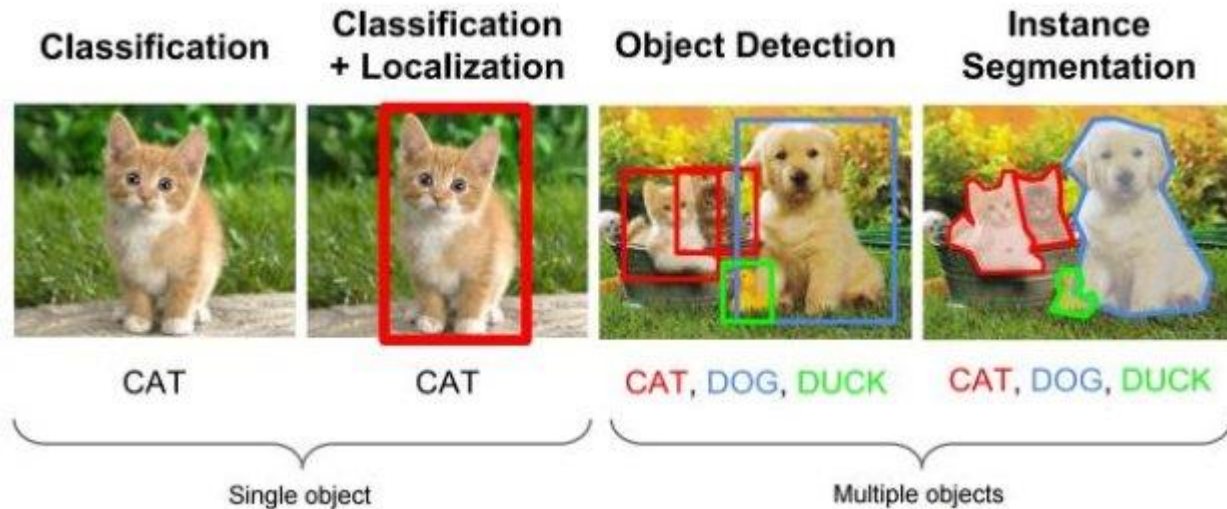
- 1.Ease of Training:** Residual connections enable the training of very deep networks (hundreds of layers) **without suffering from vanishing gradients, making them easier to train.**
- 2.Improved Gradient Flow:** Residual connections allow gradients to flow directly through the skip connections, **mitigating the vanishing-gradient problem and accelerating convergence.**
- 3.State-of-the-Art Performance:** ResNet architectures have consistently achieved state-of-the-art performance on various image recognition tasks, including image classification and object detection.

Disadvantages:

- 1.Increased Complexity:** The introduction of residual connections adds complexity to the network architecture, potentially making it harder to understand and implement.
- 2.Possible Degradation with Deepening:** Despite their success, very deep ResNet architectures (e.g., ResNet-1000) may suffer from **degradation problems**, where accuracy saturates and then degrades rapidly as depth increases.
- 3.Overfitting Risk:** ResNets with a large number of layers can be prone to overfitting, especially on smaller datasets, requiring careful regularization techniques.

What is YOLO?

- **YOLO** stands for **You Only Look Once**, and it's a real-time object detection algorithm. That means it can both **identify what an object is** (classification) and **where it is** (localization) in an image



- Instead of looking at parts of the image multiple times (like traditional methods), YOLO looks at the entire image once and predicts everything at once.
- It breaks the image into a grid and each grid cell predicts:
- Whether there's an object.
- The object's class (e.g., dog, car).
- The coordinates of a bounding box around it.

YOLOv1 (2016)

- Total layers:** ~24 convolutional + 2 fully connected layers
- Architecture:** Inspired by GoogLeNet (Inception-style), ends with 2 FC layers

YOLOv2 (YOLO9000, 2017)

- Total layers:** ~30 convolutional layers + 1 passthrough layer
- Backbone:** Darknet-19 (19 conv layers + 5 maxpool layers)
- Changes:** Removed fully connected layers; used anchor boxes

YOLOv3 (2018)

- Total layers:** ~106 layers
- Backbone:** Darknet-53 (53 conv layers, deeper than v2)
- Features:** Multi-scale prediction (3 scales)

YOLOv4 (2020)

- Total layers:** ~162 layers (varies based on configuration)
- Backbone:** CSPDarknet53
- Neck:** SPP + PAN
- Head:** YOLO detection layers

- YOLOv5 , YOLOv6 (by Meituan), YOLOv7 (2022)

YOLOv8 (2023, Ultralytics)

- Total layers:** Varies by model (n, s, m, l, x)
- Architecture:** Custom Ultralytics backbone (not Darknet-based anymore)
- Framework:** PyTorch
- Variants:**
 - YOLOv8n (Nano): ~168 layers
 - YOLOv8s (Small): ~225 layers
 - YOLOv8m (Medium): ~308 layers
 - YOLOv8l (Large): ~393 layers
 - YOLOv8x (X-Large): ~489 layers

How YOLO Works - Step by Step

Let's say we input an image of size **416x416 pixels**:

1. **Divide the image into a grid**, e.g., 13x13 cells.
2. Each grid cell is responsible for detecting objects **whose center falls inside it**.
3. Each cell outputs:
 - Bounding box coordinates: x, y, w, h
 - Confidence score (how sure it is there's an object)
 - Class probabilities (e.g., 90% dog, 5% cat)
4. All these predictions are combined, and non-max suppression is used to **remove overlapping boxes** and keep only the best ones.

YOLO was originally introduced by Joseph Redmon in 2016. Since then, we've had YOLOv2, YOLOv3, YOLOv4, YOLOv5 (by Ultralytics), YOLOv6, YOLOv7, YOLOv8... and each version gets faster and smarter!

How to collect a custom dataset and annotate it?

How to train a YOLO algorithm on the created custom dataset?

Start-Up Idea

**PROJECT: Automated Chicken Detection, Localization, and
Behavior Analysis in Farm Environments**

Annotating Images for YOLO

0 0.612264 0.463443 0.511792 0.879717

0 — Class ID (this refers to the object category; 0 is usually the first class in your list of labels)

0.612264 — x-center of the bounding box (relative to image width, so it's normalized between 0 and 1)

0.463443 — y-center of the bounding box (relative to image height)

0.511792 — width of the bounding box (again, relative to image width)

0.879717 — height of the bounding box (relative to image height)

Quiz

1. When should you consider using Deep Learning over traditional machine learning techniques?

- A. When the dataset is very small and features are well-defined
- B. When interpretability is more important than performance
- C. When you have a large amount of labeled data and high computational resources
- D. When the problem is linear and has low-dimensional input

C

2. What is the primary goal of image classification?

- A. Detect and localize objects in an image
- B. Cluster images based on similarity without labels
- C. Assign a single label to an entire image from a fixed set of classes
- D. Translate image text into another language

C

3. What is a key innovation in ResNet compared to traditional CNNs?

- A. Use of attention mechanisms
- B. Use of fully connected layers only
- C. Introduction of residual connections (skip connections)
- D. No convolutional layers

C

4. What is the purpose of annotation when preparing a dataset for YOLO training?

- A. To resize images uniformly
- B. To apply filters for preprocessing
- C. To label objects in images with bounding boxes and class names
- D. To normalize pixel values

C

Discussion

How does the choice between VGG16, ResNet, and YOLO depend on the type of problem you're solving (e.g., classification vs detection), and what trade-offs should be considered when training on a custom dataset?

VGG16 and ResNet are typically used for **image classification**, where the goal is to assign a single class to the entire image. ResNet is often preferred over VGG16 because it's more efficient and accurate due to residual connections. On the other hand, YOLO is designed for **object detection**, which involves both classifying and localizing multiple objects in an image.

When training on a custom dataset, one must consider trade-offs such as:

- **Data quality and quantity:** Detection models like YOLO require more detailed annotations and more data.
- **Computational resources:** ResNet and YOLO are more demanding than VGG16.
- **Inference speed:** YOLO is optimized for real-time applications.
- **Model complexity:** Deeper models like ResNet might overfit small datasets unless properly regularized.



Questions?