National Taiwan University of Science and Technology
Computer Science and Information Engineering

Text Mining Tutorial
Course ID: CS4813304

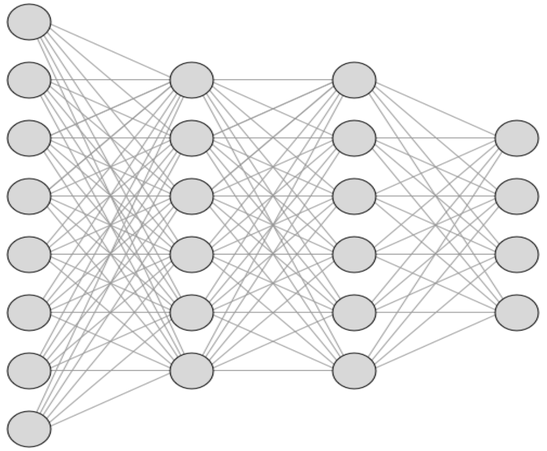
TUTORIAL 12: LARGE LANGUAGE MODELS (ADDITIONAL: CNN ARCHITECTURE)

Instructor: Prof. Hsing-Kuo Pao
TA: Zolnamar Dorjsembe (Zola)

Date: May 14, 2024

Contents

- Recap: Transformer
- Large language models
- CNN architecture
- Optimization functions



LET'S RECAP

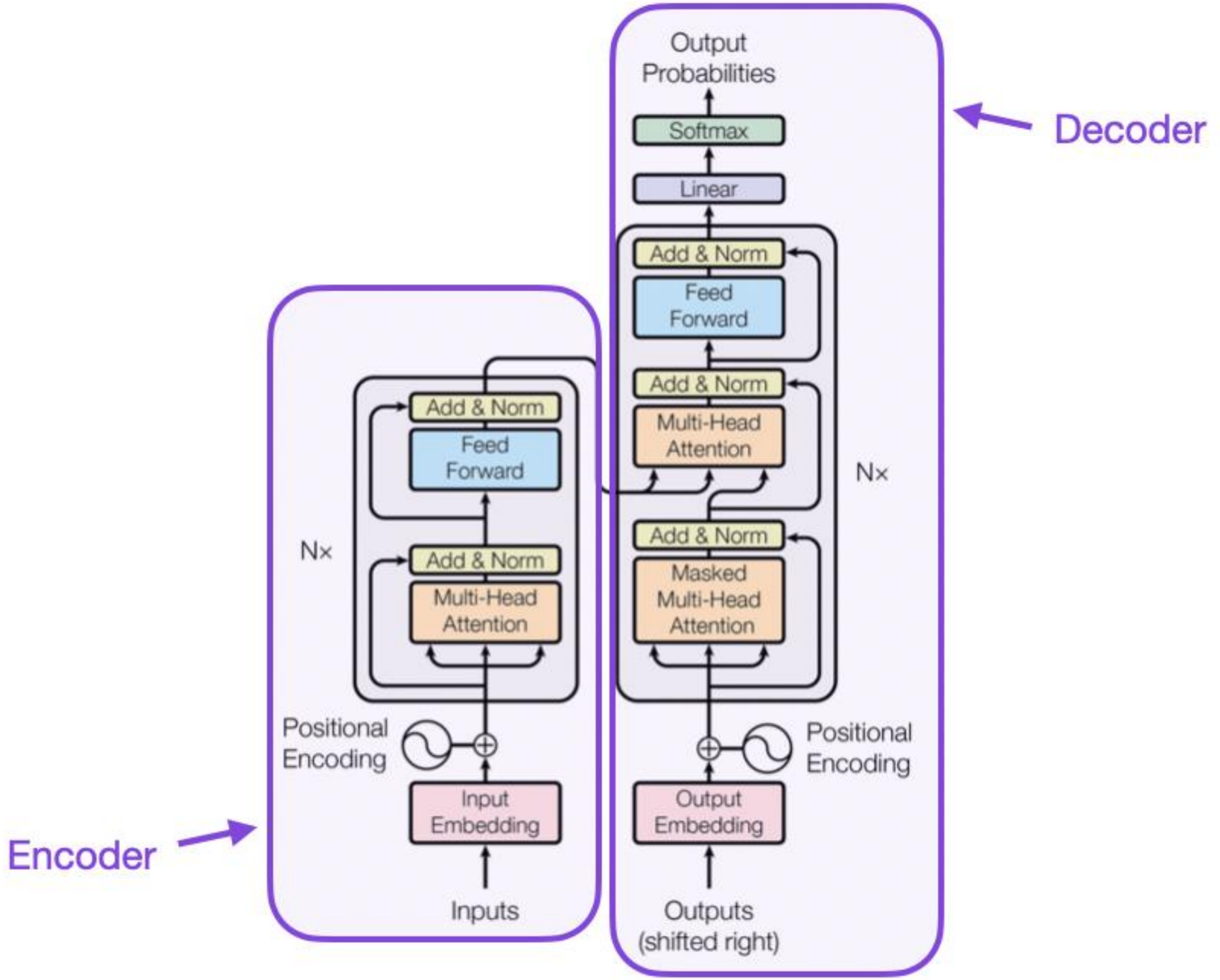
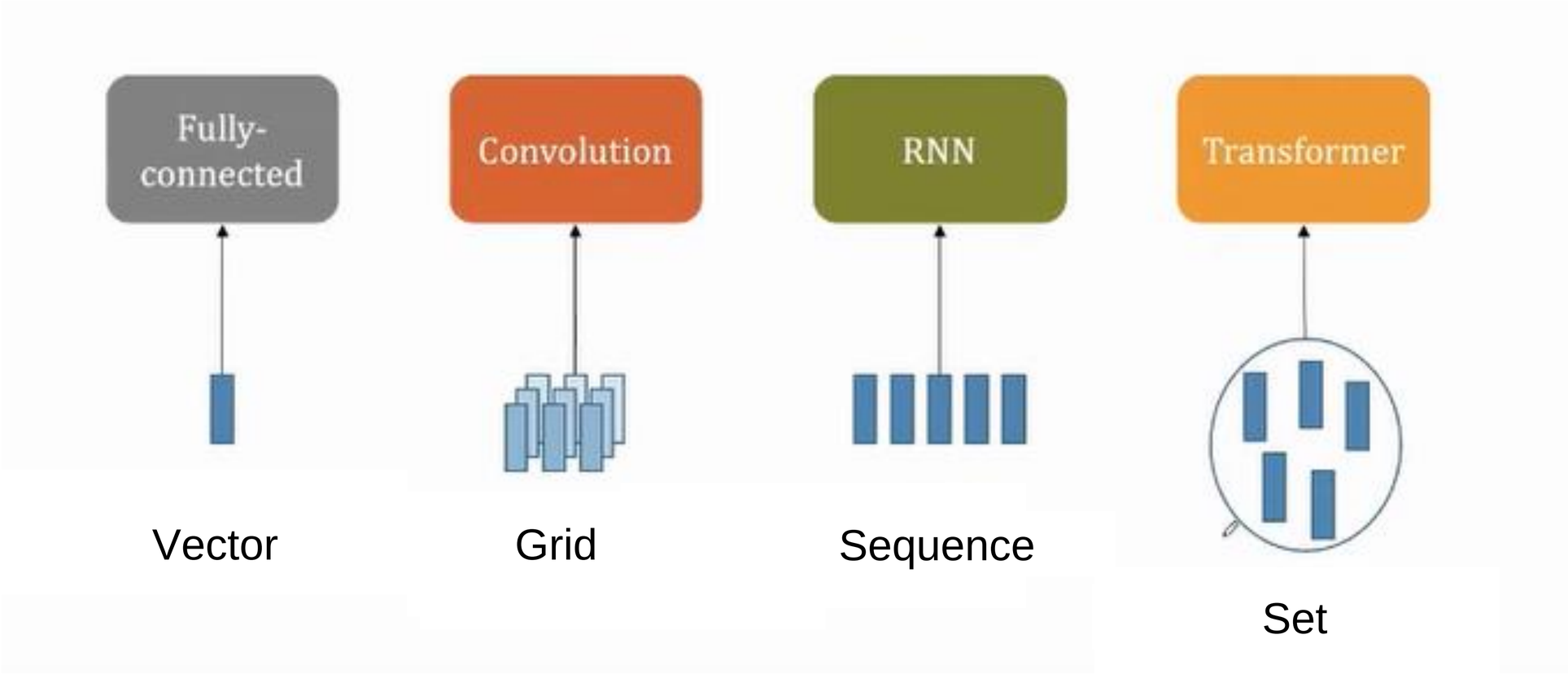


Figure 1: The Transformer - model architecture.

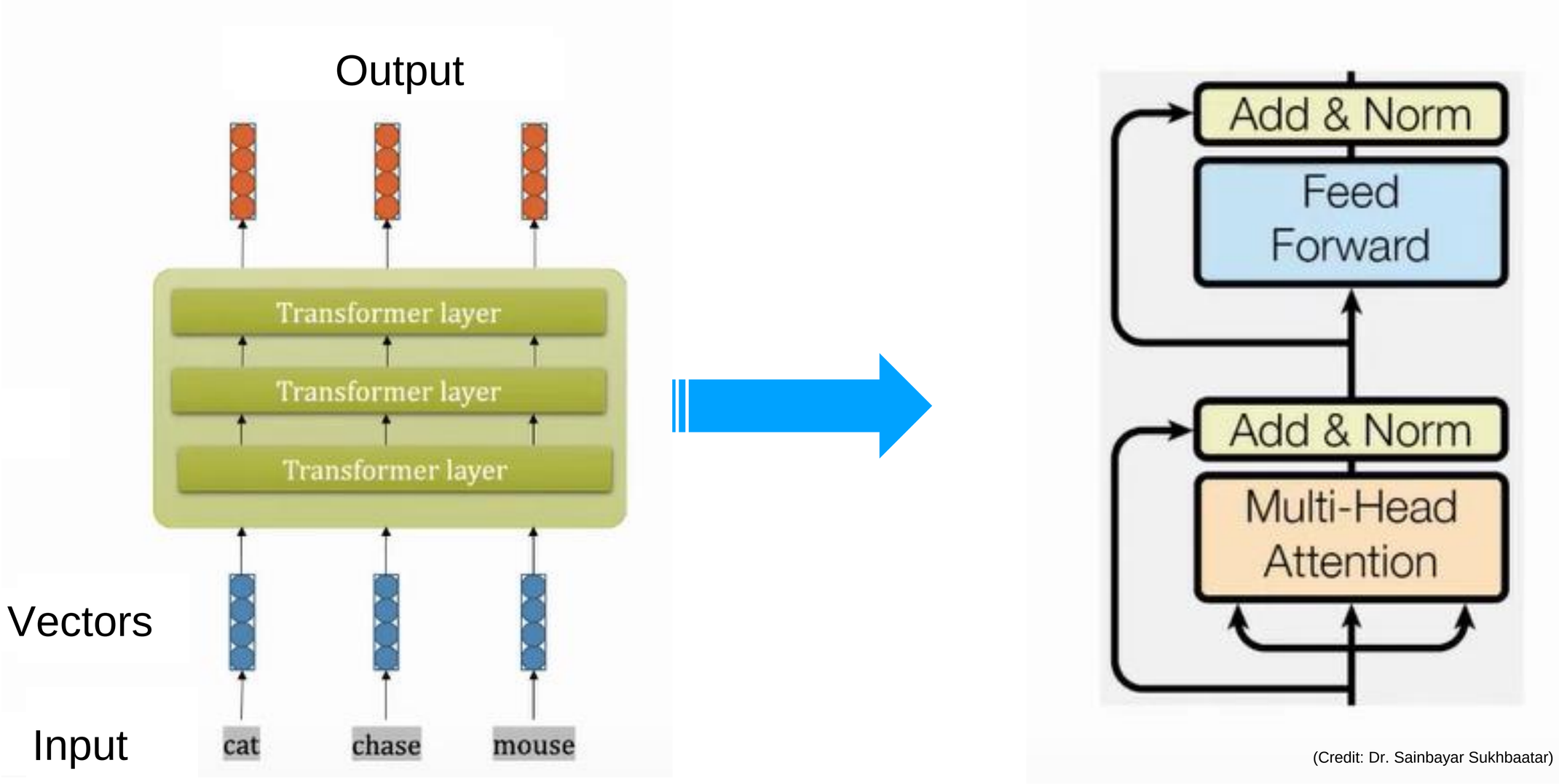
(Credit: sebastianraschka.com)

TRANSFORMER



(Credit: Dr. Sainbayar Sukhbaatar)

TRANSFORMER



(Credit: Dr. Sainbayar Sukhbaatar)

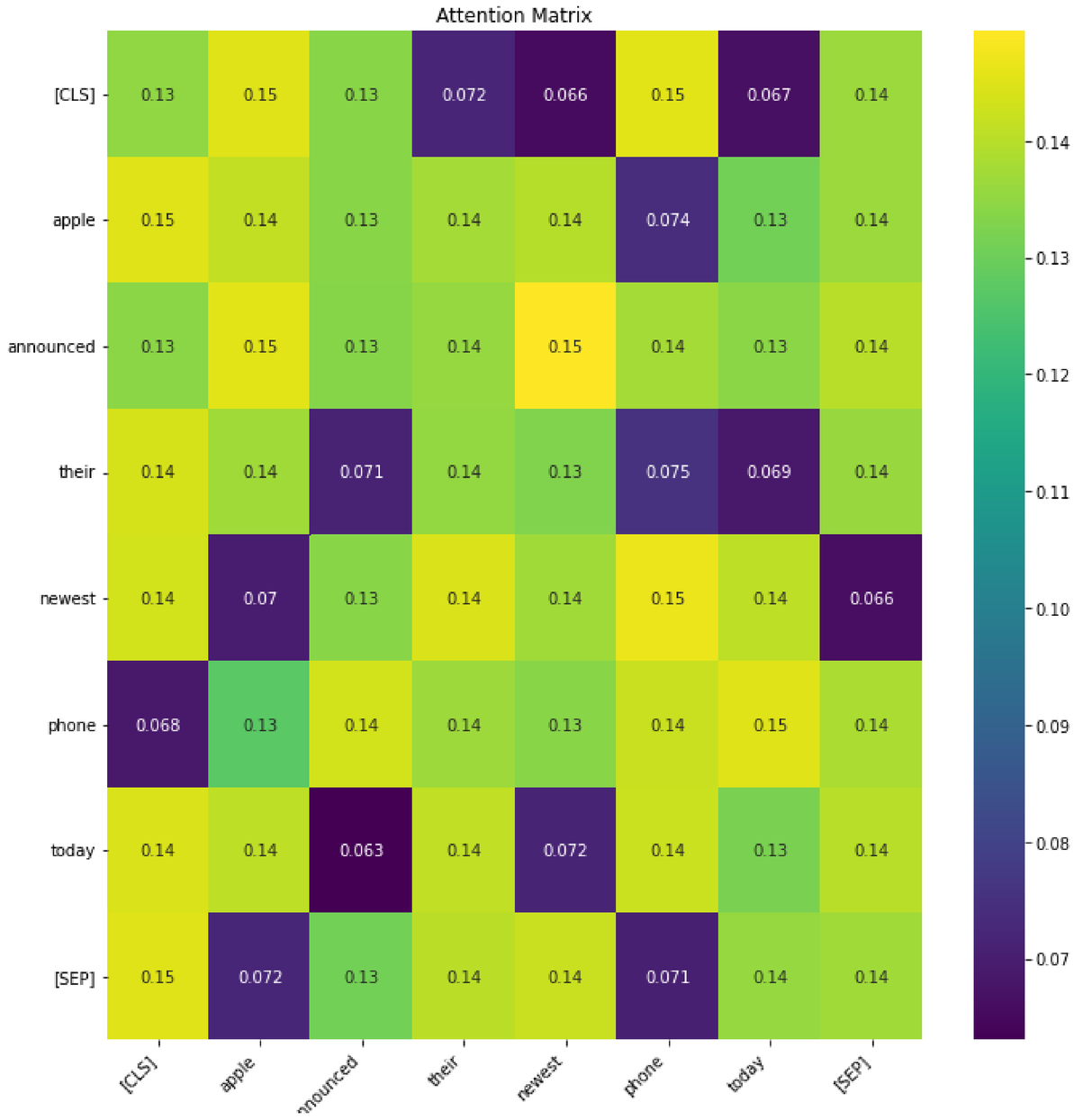
VISUALIZING ATTENTION IN TRANSFORMERS

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
from transformers import BertTokenizer, BertModel, BertConfig, AdamW
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Define a small dataset
class SmallDataset(Dataset):
    def __init__(self, sentences, tokenizer, max_len):
        self.sentences = sentences
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.sentences)

    def __getitem__(self, index):
        sentence = self.sentences[index]
        encoding = self.tokenizer.encode_plus(
            sentence,
            max_length=self.max_len,
            add_special_tokens=True,
            return_token_type_ids=False,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )
        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten()
        }

# Small dataset
sentences = [
    "Apple announced their newest phone today",
    "I love to eat a juicy apple pie"
]
```

```python
# Initialize tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Create dataset and dataloader
max_len = 10
dataset = SmallDataset(sentences, tokenizer, max_len)
dataloader = DataLoader(dataset, batch_size=2)

# Initialize a simple BERT model with attention outputs enabled
config = BertConfig(
    hidden_size=128,
    num_attention_heads=2,
    num_hidden_layers=2,
    intermediate_size=256,
    output_attentions=True  # Enable output attentions
)
model = BertModel(config)

# Define optimizer
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)

# Training loop
epochs = 10
for epoch in range(epochs):
    for batch in dataloader:
        optimizer.zero_grad()
        input_ids = batch['input_ids']
        attention_mask = batch['attention_mask']
        outputs = model(input_ids, attention_mask=attention_mask)
        loss = torch.mean(outputs.last_hidden_state)
        loss.backward()
        optimizer.step()

    print(f"Epoch {epoch+1} completed with loss: {loss.item()}")
```

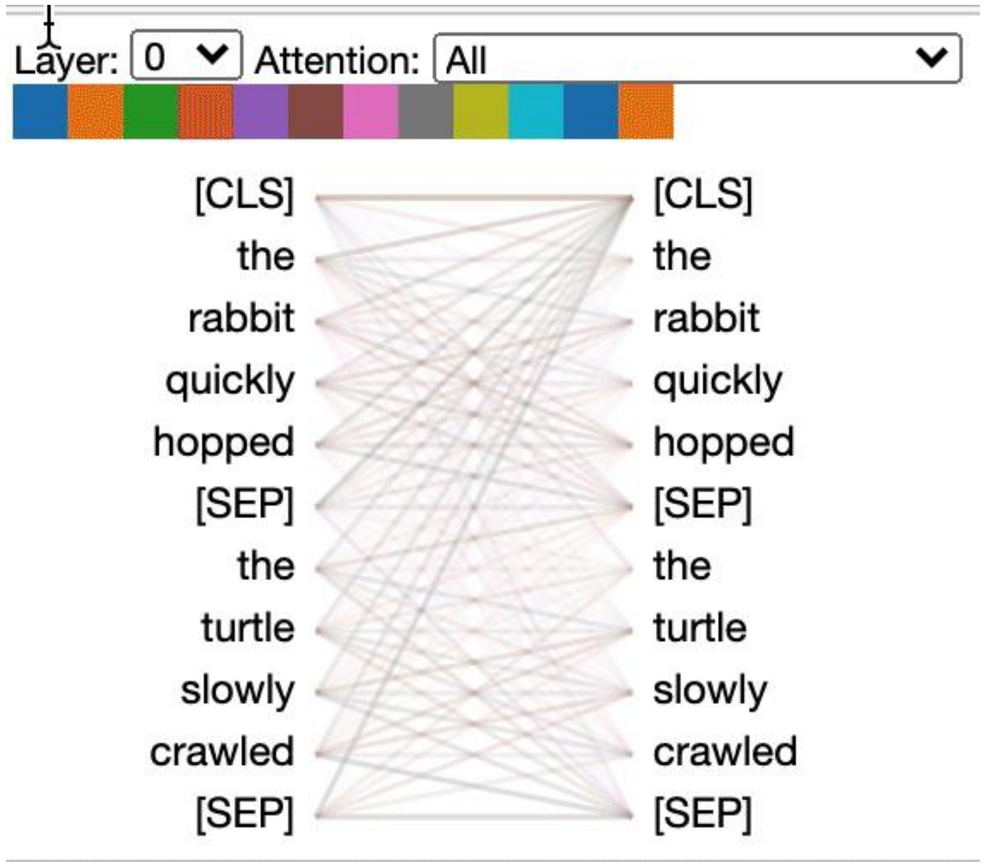
```python
def plot_attention_matrix(attention, tokens):
    # Calculate the mean of the attention weights across all attention heads
    attention = attention.mean(dim=1).squeeze(0).detach().numpy()

    # Create a plot
    fig, ax = plt.subplots(figsize=(10, 10))  # Increased figure size for better visibility

    # Generate a heatmap
    heatmap = sns.heatmap(attention, xticklabels=tokens, yticklabels=tokens, cmap='viridis', ax=ax, annot=True)

    # Set the title of the heatmap
    ax.set_title('Attention Matrix')

    # Rotate tick labels for better visibility
    plt.xticks(rotation=45, ha='right')
    plt.yticks(rotation=0)

    # Adjust layout to make room for token labels
    plt.tight_layout()

    # Display the plot
    plt.show()

# Get attention weights from the model
input_ids = tokenizer(sentences[0], return_tensors='pt')['input_ids']
with torch.no_grad():
    outputs = model(input_ids)
    attention = outputs.attentions[0]  # Get attention weights for the first layer

# Tokenize the first sentence and plot the attention matrix
tokens = tokenizer.convert_ids_to_tokens(input_ids.squeeze())
plot_attention_matrix(attention, tokens)
```

```
Epoch 1 completed with loss: -7.450580707946131e-10
Epoch 2 completed with loss: -2.9853730666218325e-05
Epoch 3 completed with loss: -5.976557804387994e-05
Epoch 4 completed with loss: -9.015276737045497e-05
Epoch 5 completed with loss: -0.00011957809329032898
Epoch 6 completed with loss: -0.0001500532089266926
Epoch 7 completed with loss: -0.00017997770919464529
Epoch 8 completed with loss: -0.00021101682796142995
Epoch 9 completed with loss: -0.0002409063308732584
Epoch 10 completed with loss: -0.0002727670653257519
```



BertViz: Visualize Attention in NLP Models

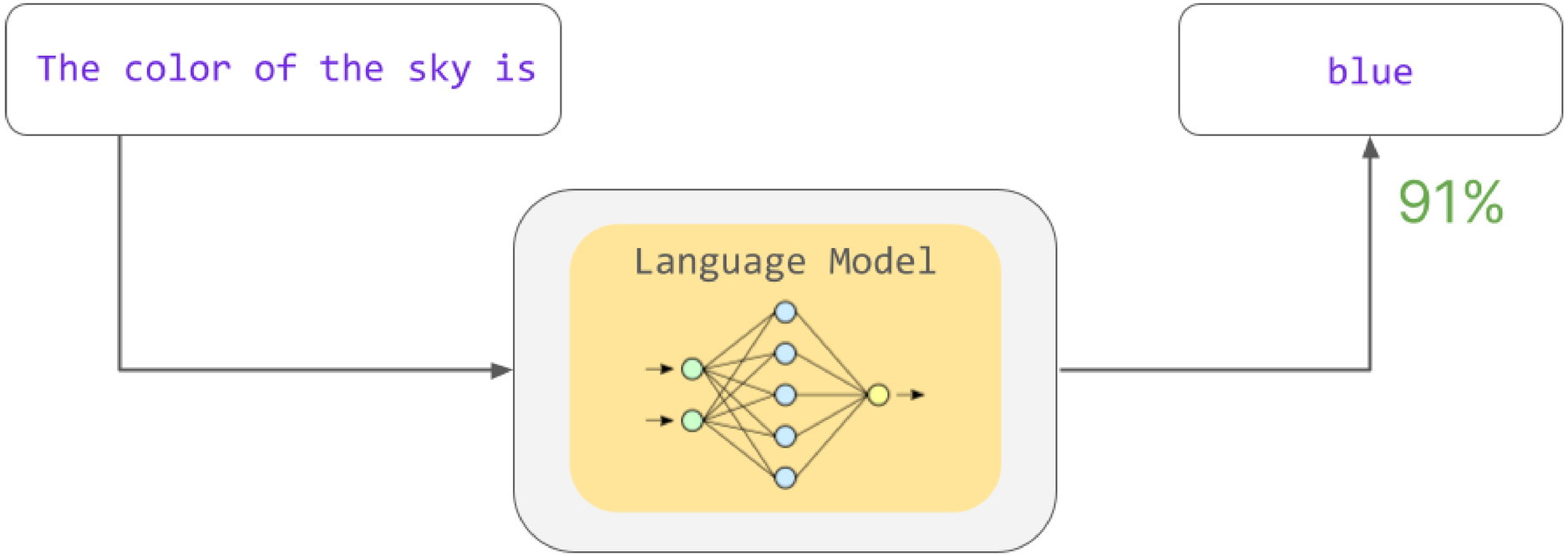
https://github.com/jessevig/bertviz?tab=readme-ov-file



LARGE LANGUAGE MODELS

LANGUAGE MODELS

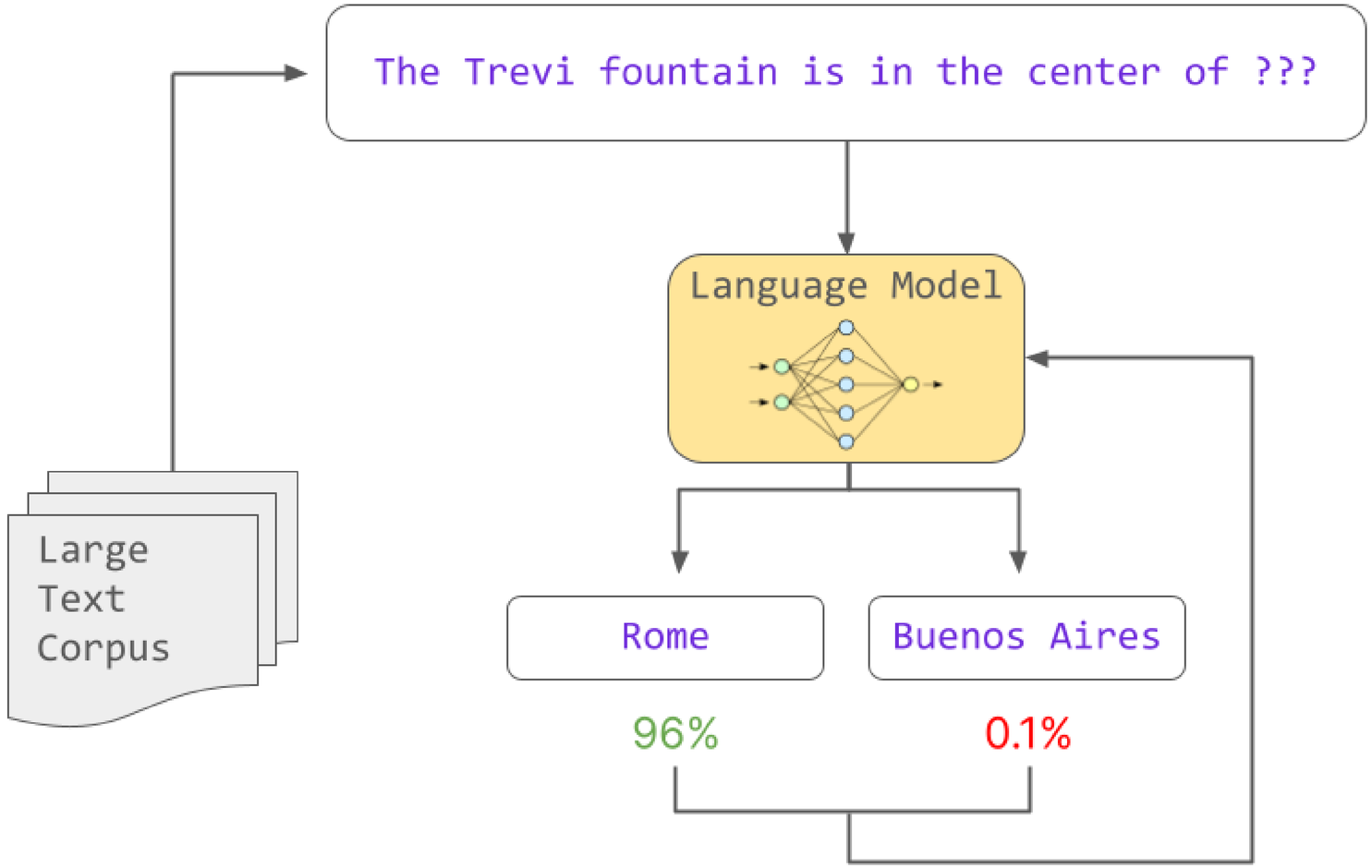
Language Models (LMs) are a class of probabilistic models explicitly tailored to identify and learn statistical patterns in natural language. The primary function of a language model is to calculate the **probability** that a word succeeds a given input sentence.



Source: https://www.assemblyai.com/blog/the-full-story-of-large-language-models-and-rlhf/

LANGUAGE MODELS

During the training process, text sequences are extracted from the corpus and truncated. The language model calculates probabilities of the missing words, which are then slightly adjusted and fed back to the model to match the ground truth, via a gradient descent based optimization mechanism. This process is repeated over the whole text corpus.



Source: https://www.assemblyai.com/blog/the-full-story-of-large-language-models-and-rlhf/

LARGE LANGUAGE MODELS

Large Language Models
from scratch

Steve Seitz

Source: https://www.youtube.com/watch?v=lnA9DMvHtfI

LARGE LANGUAGE MODELS

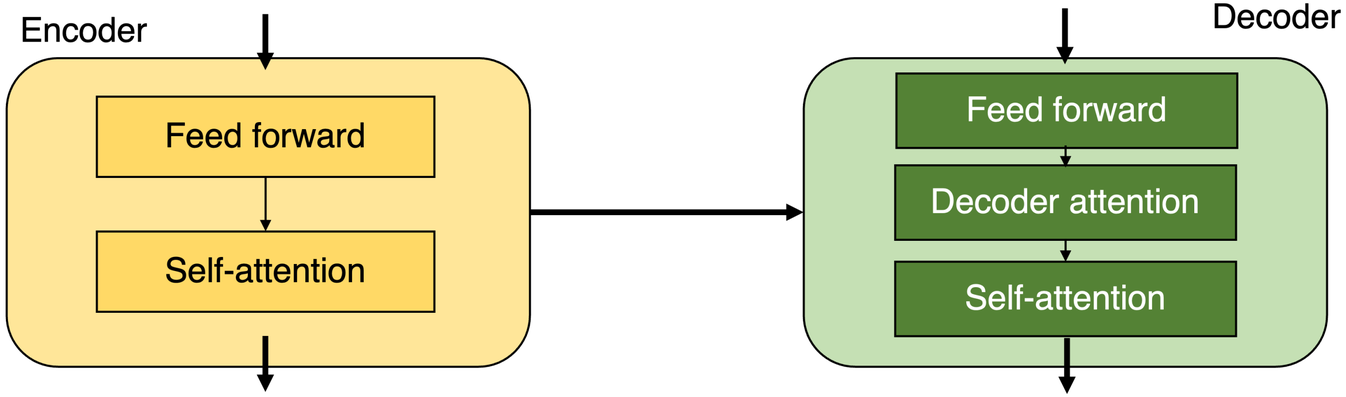
Large Language Models (LLMs) are a type of artificial deep neural networks designed to understand and generate human language. They are trained on vast amounts of text data and can perform a variety of language tasks, such as translation, summarization, and question answering.

LARGE LANGUAGE MODELS: KEY CONCEPTS

- **Transformer Architecture**: The foundation for most modern LLMs. Introduced in the paper "Attention is All You Need" by Vaswani et al., the Transformer uses self-attention mechanisms to process input data.
- **Pre-training and Fine-tuning**: LLMs are typically pre-trained on large datasets and then fine-tuned for specific tasks.
- **Tokens**: The basic units of text used by LLMs, which can be words, subwords, or characters.

TRANSFORMER MODEL

The original Transformer model is like a **LEGO** set. It has different bricks, such as encoders, decoders, embedding layers, positional encoding methods, multi-head attention layers, masked multi-head attention layers, post-layer normalization, feed-forward sub-layers, and linear output layers. These bricks work together to build the **Transformer model**.

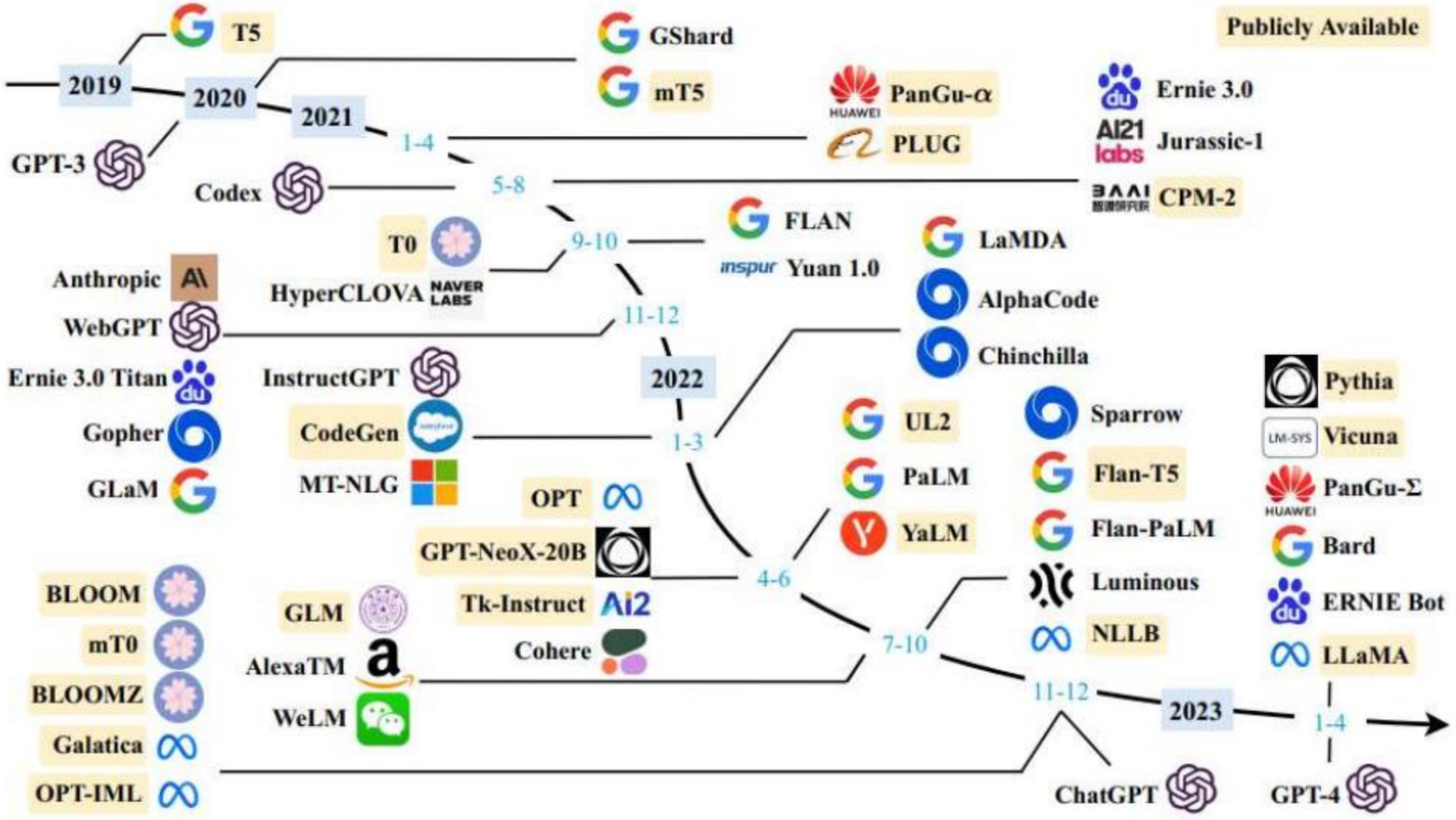


(Source: dominodatalab.com)

(Source: amazon.com)

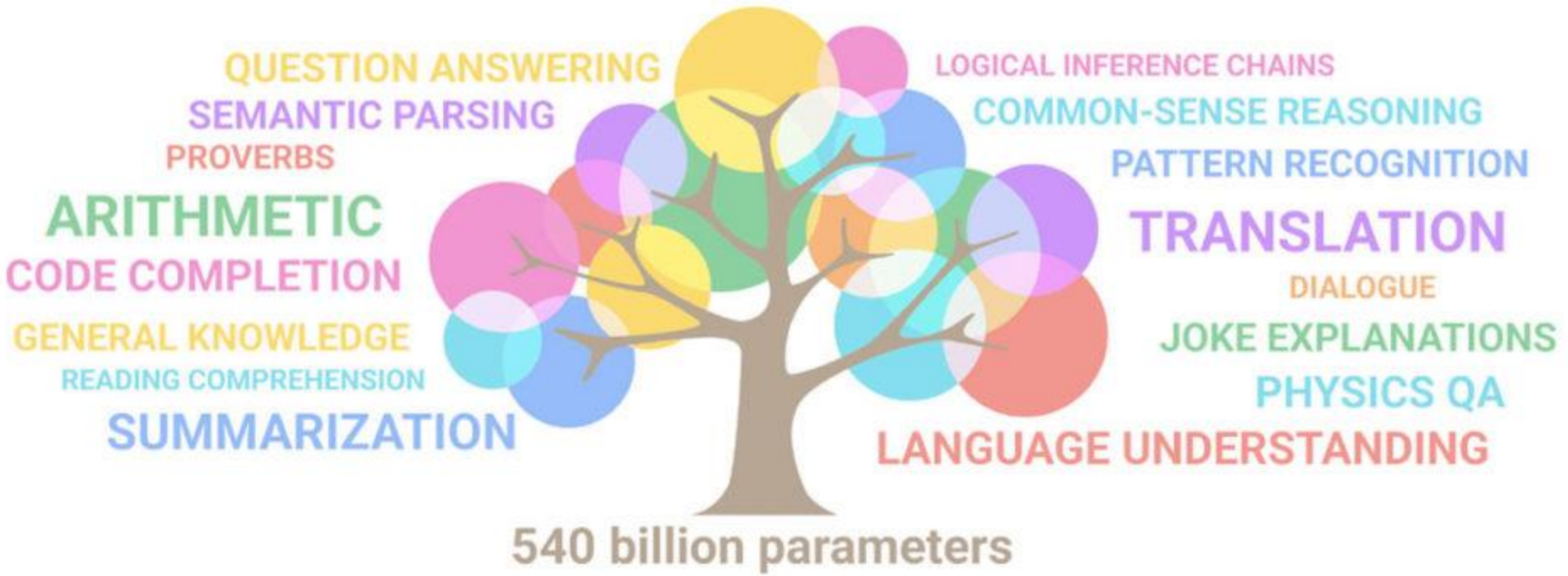
LARGE LANGUAGE MODELS (LLMs)

The bricks come in various sizes and forms. You can spend hours building all sorts of models using the same building kit! Some constructions will only require some of the bricks. Other constructions will add a new piece, just like when we obtain additional bricks for a model built using LEGO components.



(Source: Zhao et. al., 2023)

LARGE LANGUAGE MODELS (LLMs)



Source: https://www.topbots.com/leading-nlp-language-models-2020/

LARGE LANGUAGE MODELS (LLMs)

Encoder only

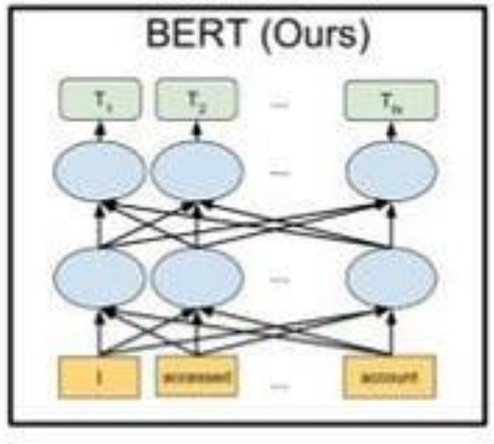
- BERT
- RoBerta
- Reformer
- FlauBERT
- CamemBERT
- Electra*
- MobileBERT
- Longformer

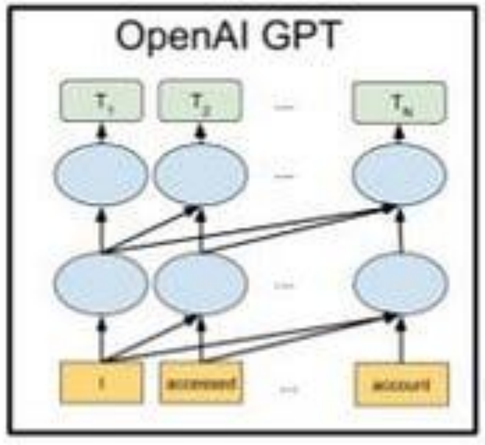
Decoder only

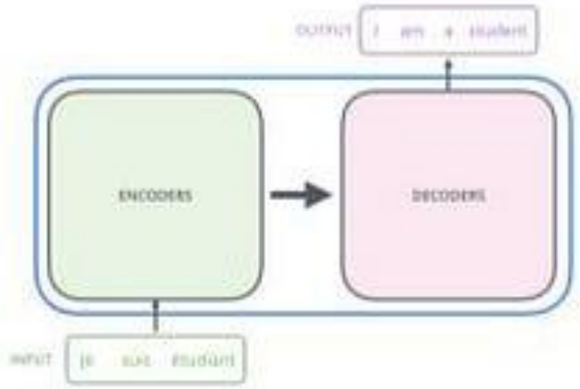
- Transformer-XL
- XLNet
- GPT series
- DialoGPT

Encoder + Decoder

- Transformer
- XLM
- T5
- BART
- XLM-RoBerta
- Pegasus
- mBART

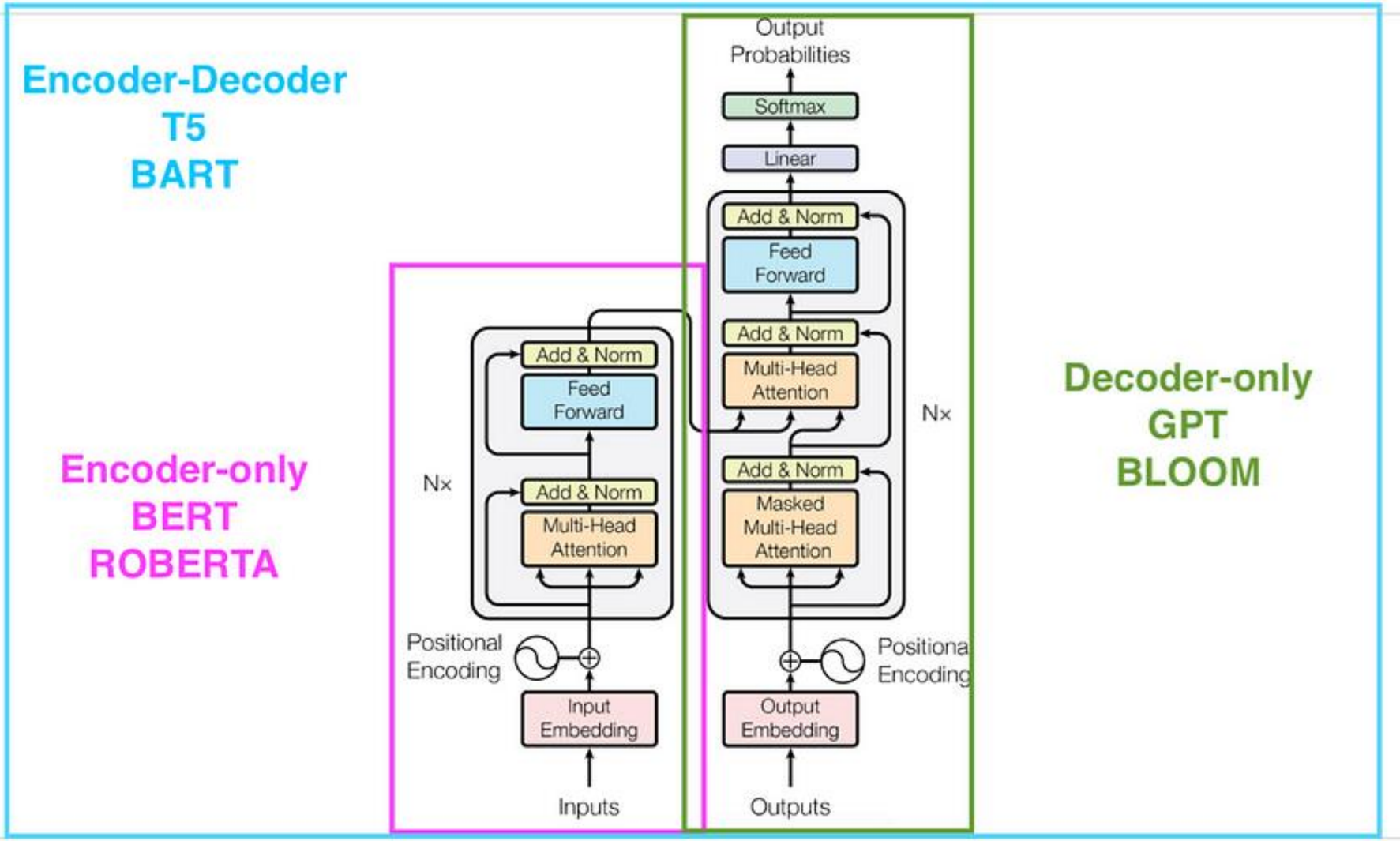






Source: https://towardsdatascience.com/gpt-3-transformers-and-the-wild-world-of-nlp-9993d8bb1314

LARGE LANGUAGE MODELS (LLMs)



Source: https://medium.com/@reyhaneh.esmailbeigi/bert-gpt-and-bart-a-short-comparison-5d6a57175fca

COMMONLY USED LLMs

	BERT	GPT	BART
Model Type	Encoder Only	Decoder Only	Encoder-Decoder
Direction	Bidirectional	Unidirectional (left-to-right)	Bidirectional
Pre-training Objective	Masked language modeling (MLM)	Autoregressive (casual) language modeling	Span Corruption (Masking entire spans of words)
Fine-tuning	Task-specific layer added on top of the pre-trained BERT model	Providing task-specific prompts using few-shot or one-shot adaptation and adapting the model's parameters	Versatile and can be used for various NLP tasks
Use Case	Sentiment Analysis Named entity Recognition Word Classification	Text generation Text completion creative writing	Translation Text Summarisation Question & Answer
Original Organisations	Google AI	OpenAI	Facebook AI

Comparison between BERT, GPT and BART

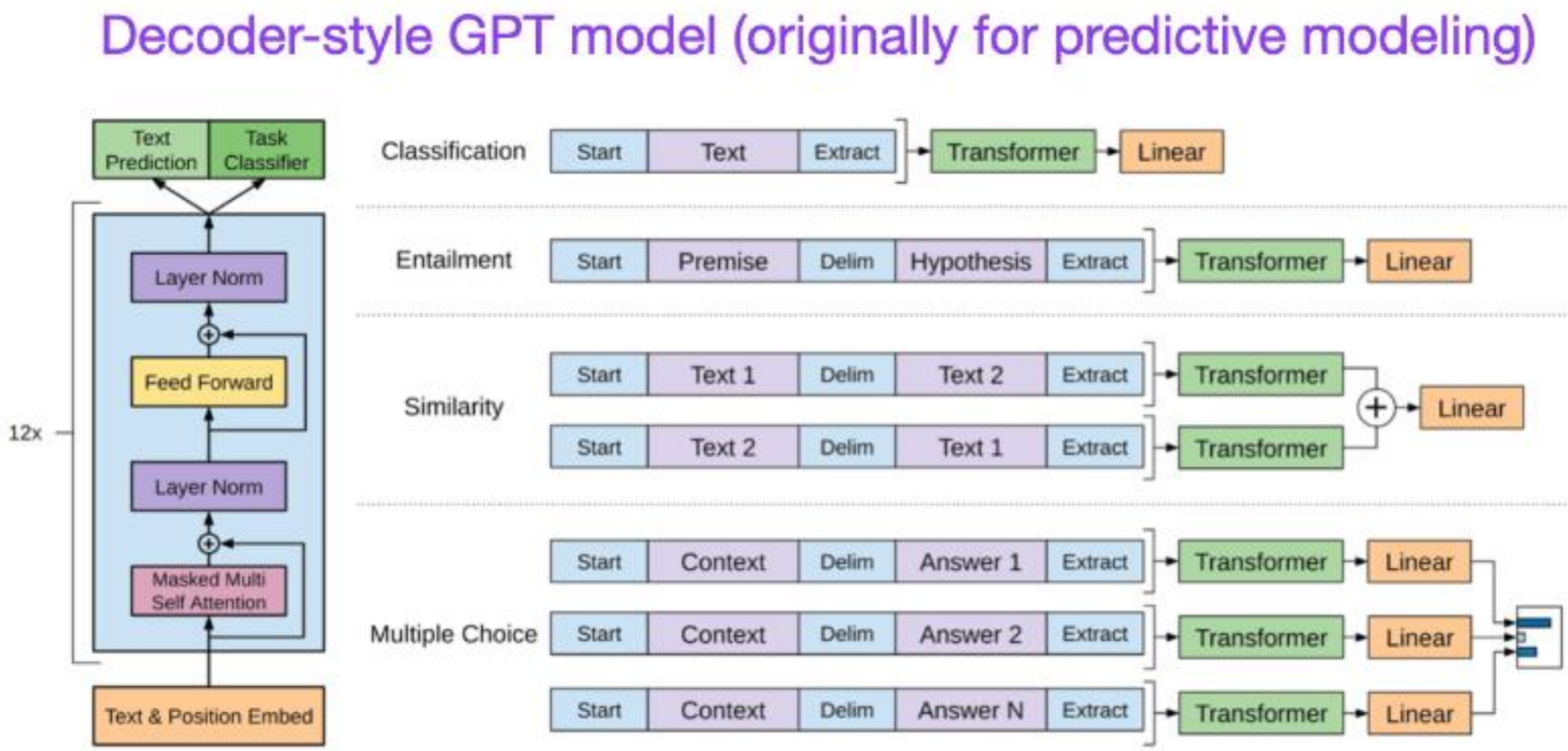
Source: https://medium.com/@reyhaneh.esmailbeigi/bert-gpt-and-bart-a-short-comparison-5d6a57175fca

	Masked Language Modeling (MLM)	Autoregressive(Causal) Language Modeling	Span Corruption (Masking)
Summery	Bidirectional contextual relationships between words by making words	Predict the next word in a sequence based on the preceding words. It operates in a left-to-right fashion and models the conditional probability of each word given the previous words.	Span corruption involves masking not only individual words but entire spans or sequences of text, consecutive phrases or sentences may be masked
Direction	Bidirectional	Unidirectional (left-to-right)	Bidirectional
Level	word	word	phrases or sentences
Used for	Contextual meaning and relationships between words	Predicting sequences where the order of words is critical	Longe range dependencies captured. document-level understanding or handling long sequences of text
Objective	Reconstruct Text	Predict Next token	Reconstruct Span
Model example	BERT	GPT	BART
Use case	Text classification Named entity recognition Question-answering	Text generation tasks Creative writing	Translation Text Summarisation Question & Answer

COMMONLY USED LLMs

Source:
https://medium.com/@reyhaneh.esmailbeigi/bert-gpt-and-bart-a-short-comparison-5d6a57175fca

OPENAI: GPT (GENERATIVE PRE-TRAINED TRANSFORMER)

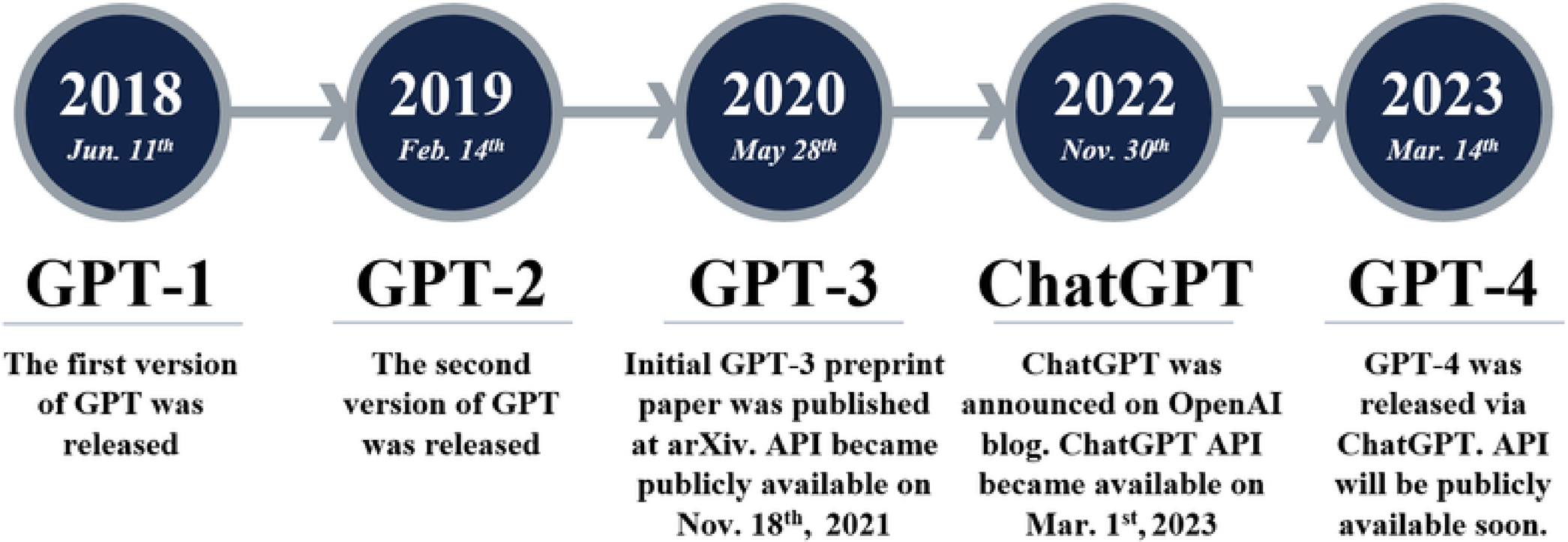


Improving Language Understanding by Generative Pre-Training (2018) by Radford and Narasimhan

Source: https://www.semanticscholar.org/paper/Improving-Language-Understanding-by-Generative-Radford-Narasimhan/cd18800a0fe0b668a1cc19f2ec95b5003d0a5035

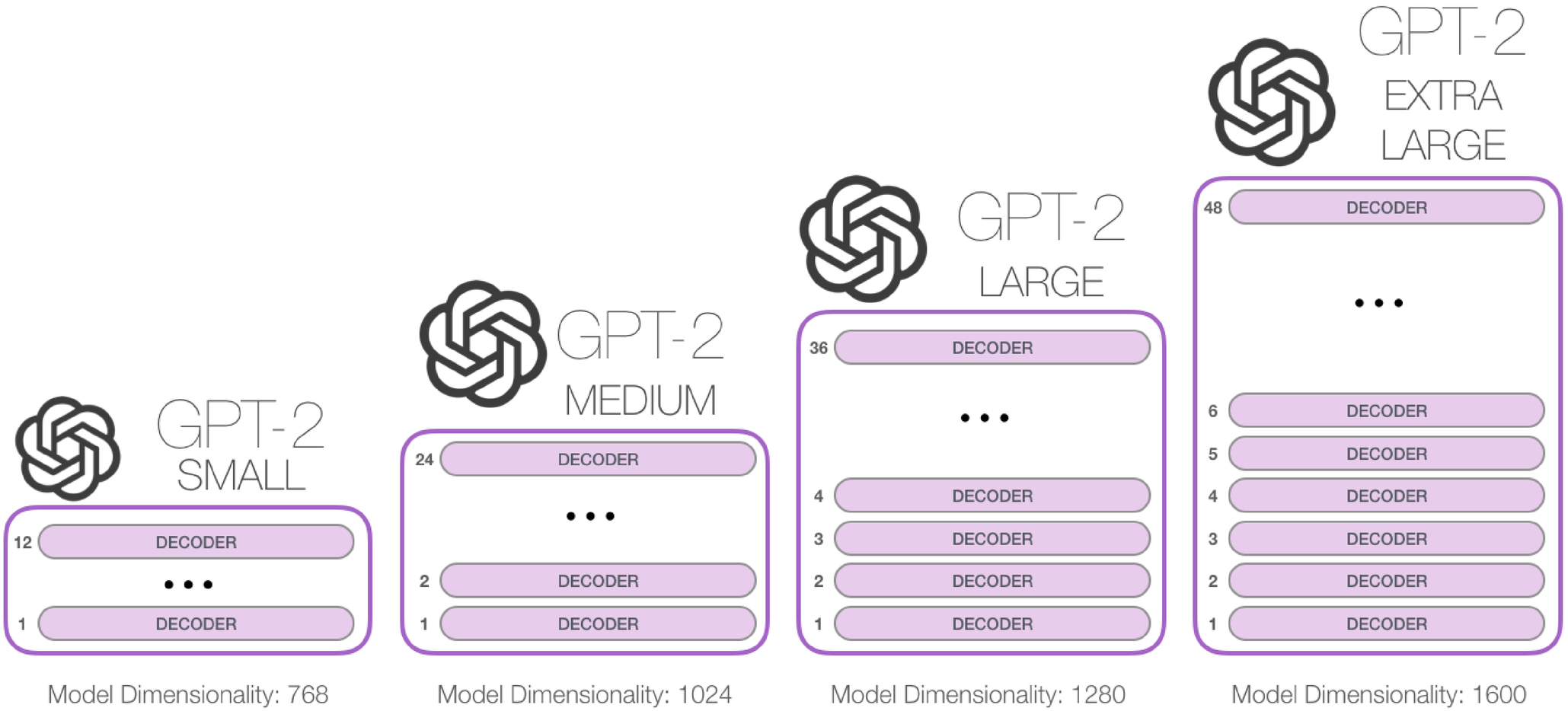
(Credit: sebastianraschka.com)

EVOLUTION OF GPT MODELS



Source: https://generativeai.pub/a-look-into-the-evolution-of-gpt-models-from-gpt-1-to-gpt-4-38b68c2f275b

GPT-2 (GENERATIVE PRE-TRAINED TRANSFORMER 2)



Source: https://jalammar.github.io/illustrated-gpt2/

GPT-2 EXAMPLE

```python
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load pre-trained model and tokenizer
model_name = 'gpt2'
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

# Encode input text
input_text = "Once upon a time"
input_ids = tokenizer.encode(input_text, return_tensors='pt')

# Generate text
output = model.generate(input_ids, max_length=50, num_return_sequences=1)

# Decode and print the output
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
print(generated_text)

```

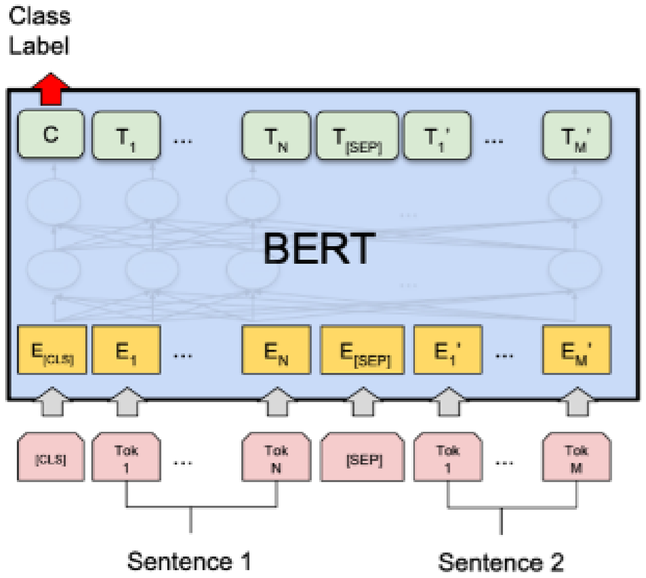
GPT-2 EXAMPLE

```python
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load pre-trained model and tokenizer
model_name = 'gpt2'
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

# Encode input text
input_text = "Once upon a time, the world was a place of great beauty and great"
input_ids = tokenizer.encode(input_text, return_tensors='pt')

# Generate text
output = model.generate(input_ids, max_length=50, num_return_sequences=1)

# Decode and print the output
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
print(generated_text)
```

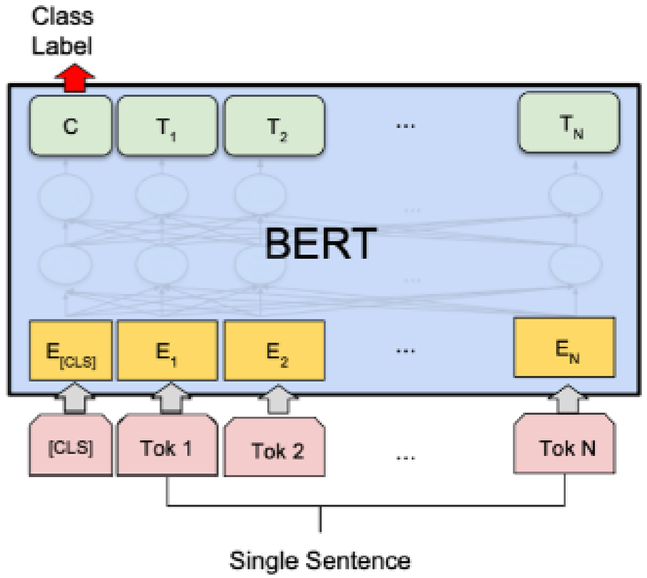
```
Once upon a time, the world was a place of great beauty and great danger. The world was a place of great danger, and the world was a place of great danger. The world was a place of great danger, and the world was a
```

GOOGLE: BERT (BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS)

Encoder-style BERT model for predictive modeling tasks



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018) by Devlin, Chang, Lee, and Toutanova

Source: https://arxiv.org/abs/1810.04805

(Credit: sebastianraschka.com)

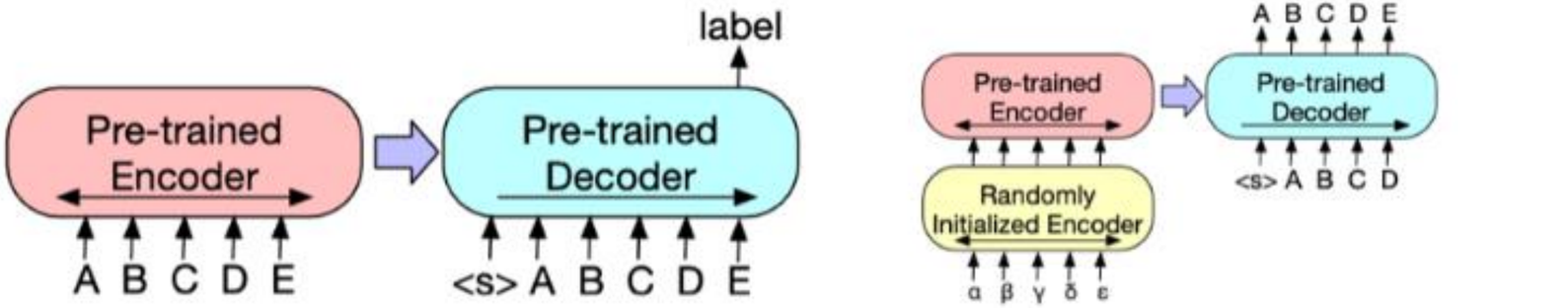
```python
from transformers import BertForMaskedLM, BertTokenizer
import torch

# Load pre-trained model and tokenizer
model_name = 'bert-base-uncased'
model = BertForMaskedLM.from_pretrained(model_name)
tokenizer = BertTokenizer.from_pretrained(model_name)
model.eval()  # Set the model to evaluation mode

# Encode input text with a masked token
input_text = "Once upon a [MASK] there was a kingdom."
input_ids = tokenizer(input_text, return_tensors='pt')['input_ids']

# Predict the masked token
with torch.no_grad():  # Disable gradient calculation for inference
    outputs = model(input_ids)
    predictions = outputs.logits

# Get the index of the masked token
mask_token_index = torch.where(input_ids == tokenizer.mask_token_id)[1]

# Get the predicted token at the masked position
predicted_index = torch.argmax(predictions[0, mask_token_index.item()], axis=-1)
predicted_token = tokenizer.convert_ids_to_tokens([predicted_index])[0]

# Replace the mask with the predicted token and print the output
output_text = input_text.replace("[MASK]", predicted_token)
print(output_text)

```

GOOGLE: BERT

```
Once upon a time there was a kingdom.
```

FACEBOOK META AI: BART (BIDIRECTIONAL AUTO-REGRESSIVE TRANSFORMERS)

BART combines encoder and decoder parts



(a) To use BART for classification problems, the same input is fed into the encoder and decoder, and the representation from the final output is used.

(b) For machine translation, we learn a small additional encoder that replaces the word embeddings in BART. The new encoder can use a disjoint vocabulary.

BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension (2019), by Lewis, Liu, Goyal, Ghazvininejad, Mohamed, Levy, Stoyanov, and Zettlemoyer

Source: https://arxiv.org/abs/1910.13461

(Credit: sebastianraschka.com)

```python
from transformers import BartForConditionalGeneration, BartTokenizer
import torch

# Load pre-trained model and tokenizer
model_name = 'facebook/bart-large-cnn'  # This checkpoint is optimized for summarization.
model = BartForConditionalGeneration.from_pretrained(model_name)
tokenizer = BartTokenizer.from_pretrained(model_name)
model.eval()  # Set the model to evaluation mode

# Input text for summarization
input_text = """
The Amazon rainforest, also known in English as Amazonia or the Amazon Jungle,
is a moist broadleaf tropical rainforest in the Amazon biome that covers most of
the Amazon basin of South America. This basin encompasses 7,000,000 square kilometers
(2,700,000 sq mi), of which 5,500,000 square kilometers (2,100,000 sq mi) are covered by
the rainforest. This region includes territory belonging to nine nations and is known
for being the largest tropical rainforest in the world. It is often described as the
"Lungs of the Earth" because it produces more than 20% of the world's oxygen.
"""

# Encode the input text
input_ids = tokenizer(input_text, return_tensors='pt')['input_ids']

```

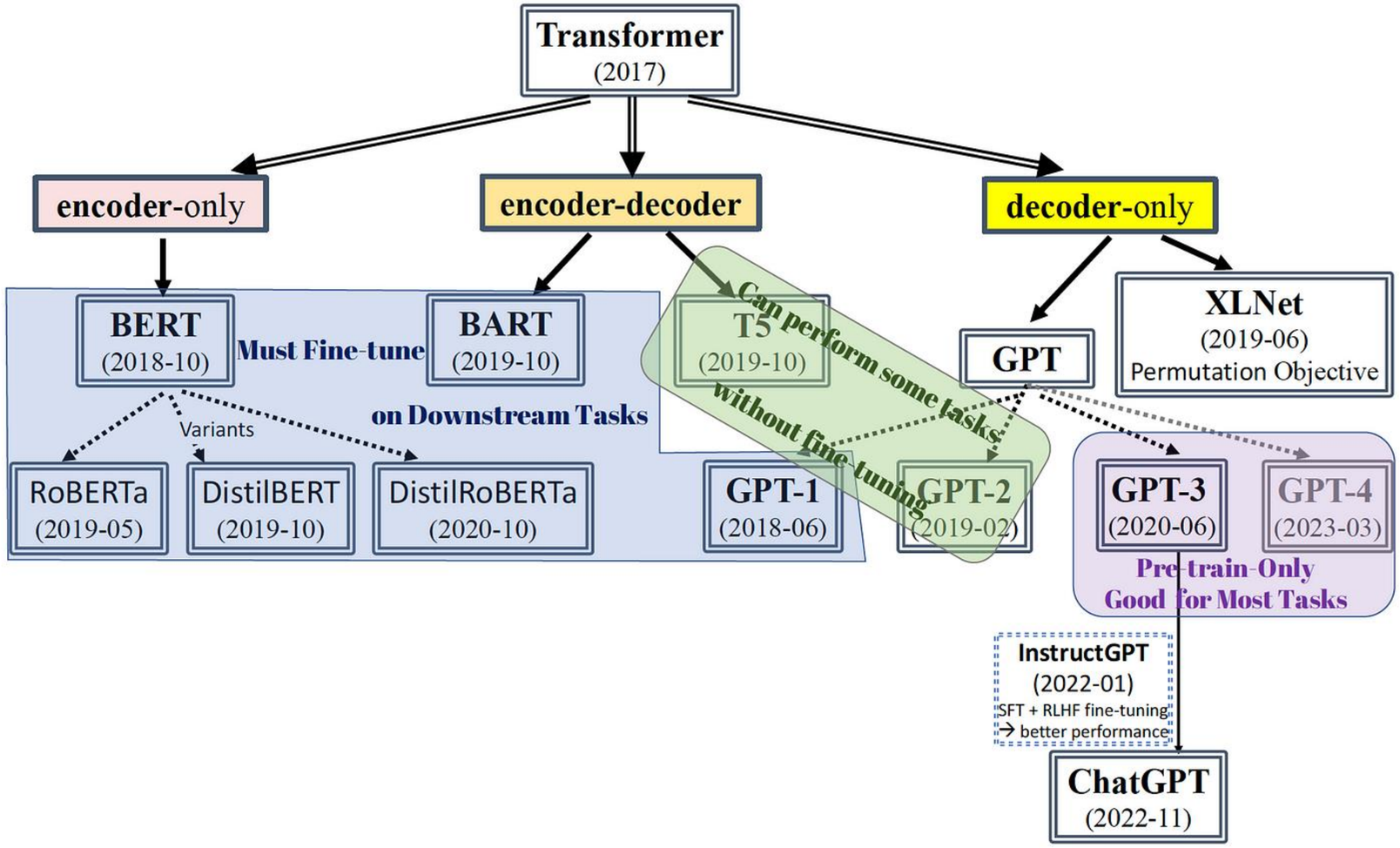
META AI: BART

```
23
24 # Generate a summary of the input text
25 output_ids = model.generate(
26     input_ids,
27     max_length=100,  # Reasonable maximum length for a summary
28     num_beams=4,     # Use beam search with 4 beams for better quality
29     length_penalty=2.0,  # Encourage shorter summaries
30     no_repeat_ngram_size=3,  # Prevent repeating 3-grams to ensure diversity
31     early_stopping=True  # Stop when the summary is complete
32 )
33
34 # Decode and print the generated summary
35 generated_summary = tokenizer.decode(output_ids[0], skip_special_tokens=True)
36 print(generated_summary)
37
```

META AI: BART

```
The Amazon rainforest is the largest tropical rainforest in the world. It is often described as the "Lungs of the Earth" becaus
e it produces more than 20% of the world's oxygen. It covers 7,000,000 square kilometers (2,700,000 sq mi) of South America.
```

TRANSFORMER-BASED LARGE LANGUAGE MODELS



(Credit: Dr. Yule Wang)

DATA SOURCES OF EXISTING LLMs

14

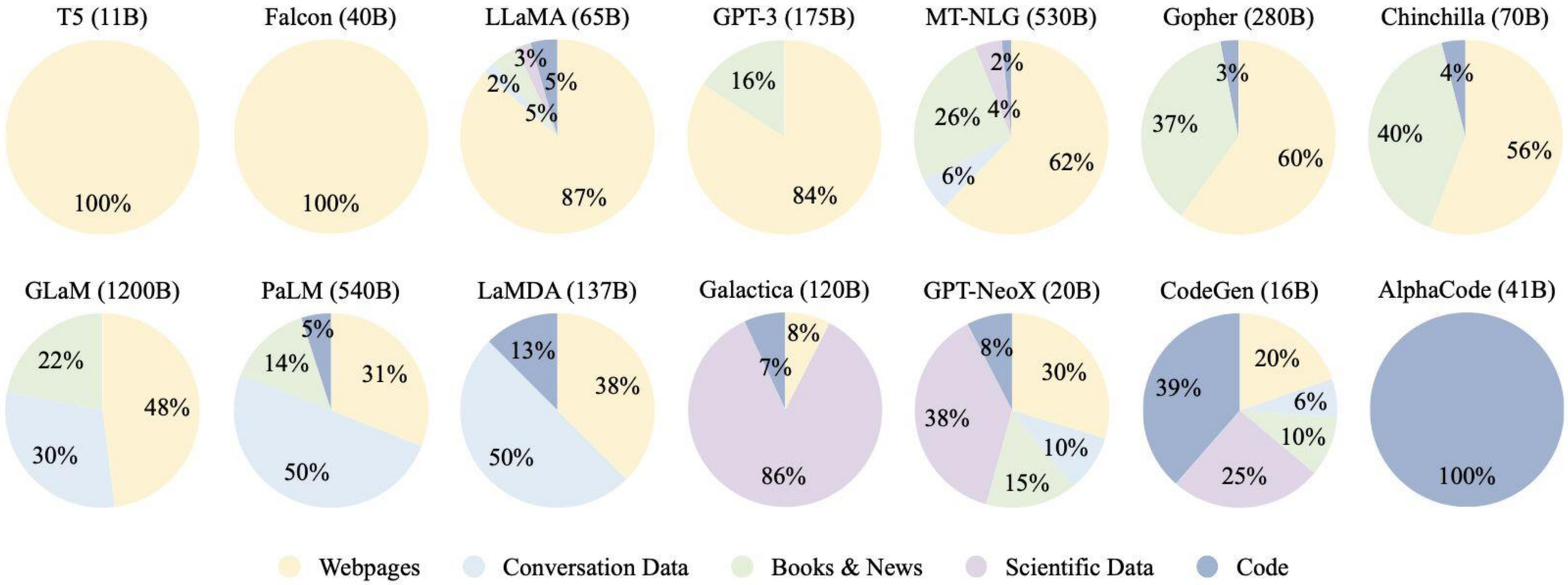


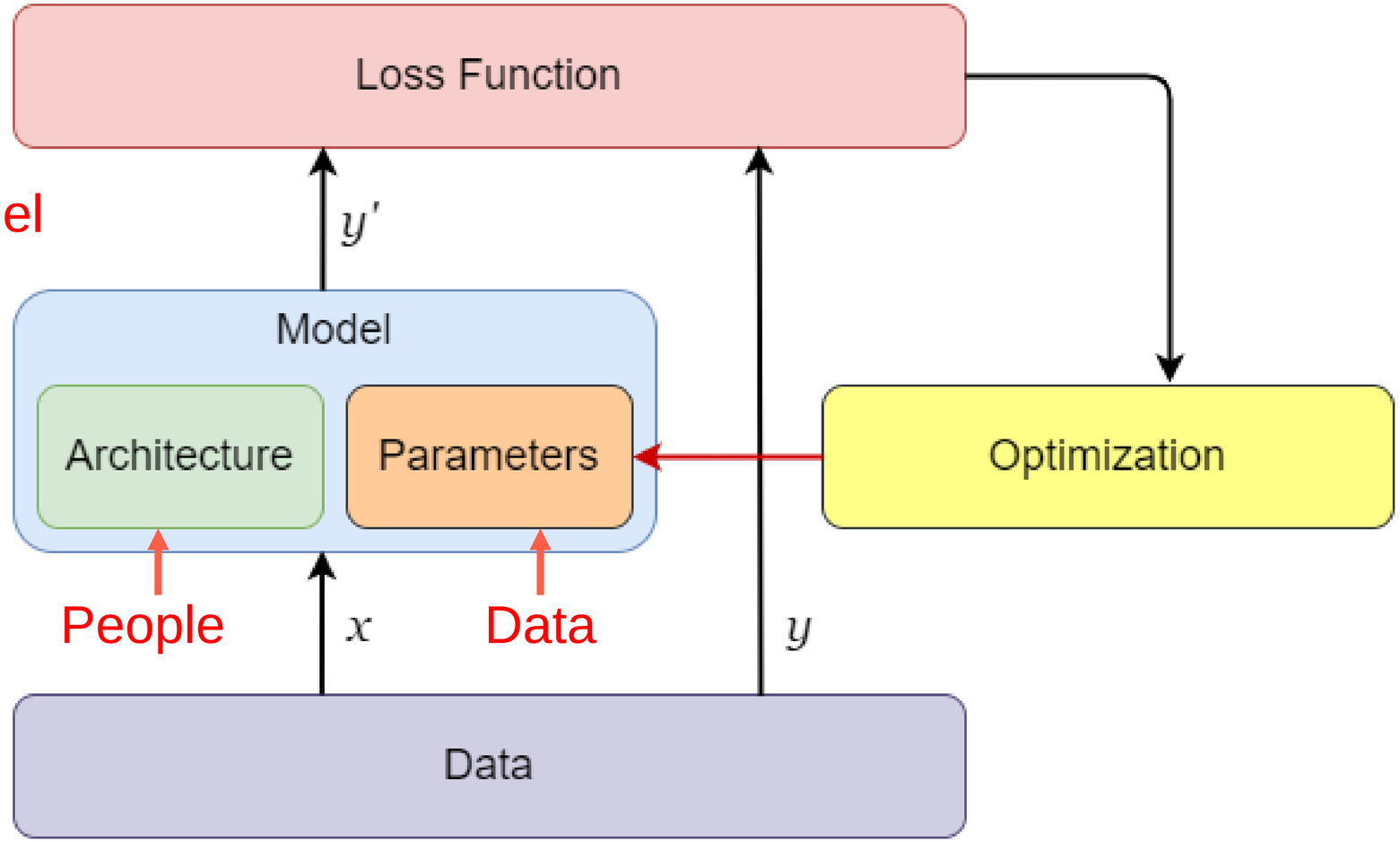
Fig. 5: Ratios of various data sources in the pre-training data for existing LLMs.

(Source: Zhao et. al., 2023)

ADDITIONAL TOPICS FOR YOUR GENERAL KNOWLEDGE OF NEURAL NETWORKS

RECAP: NEURAL NETWORK OVERVIEW

Parametric model

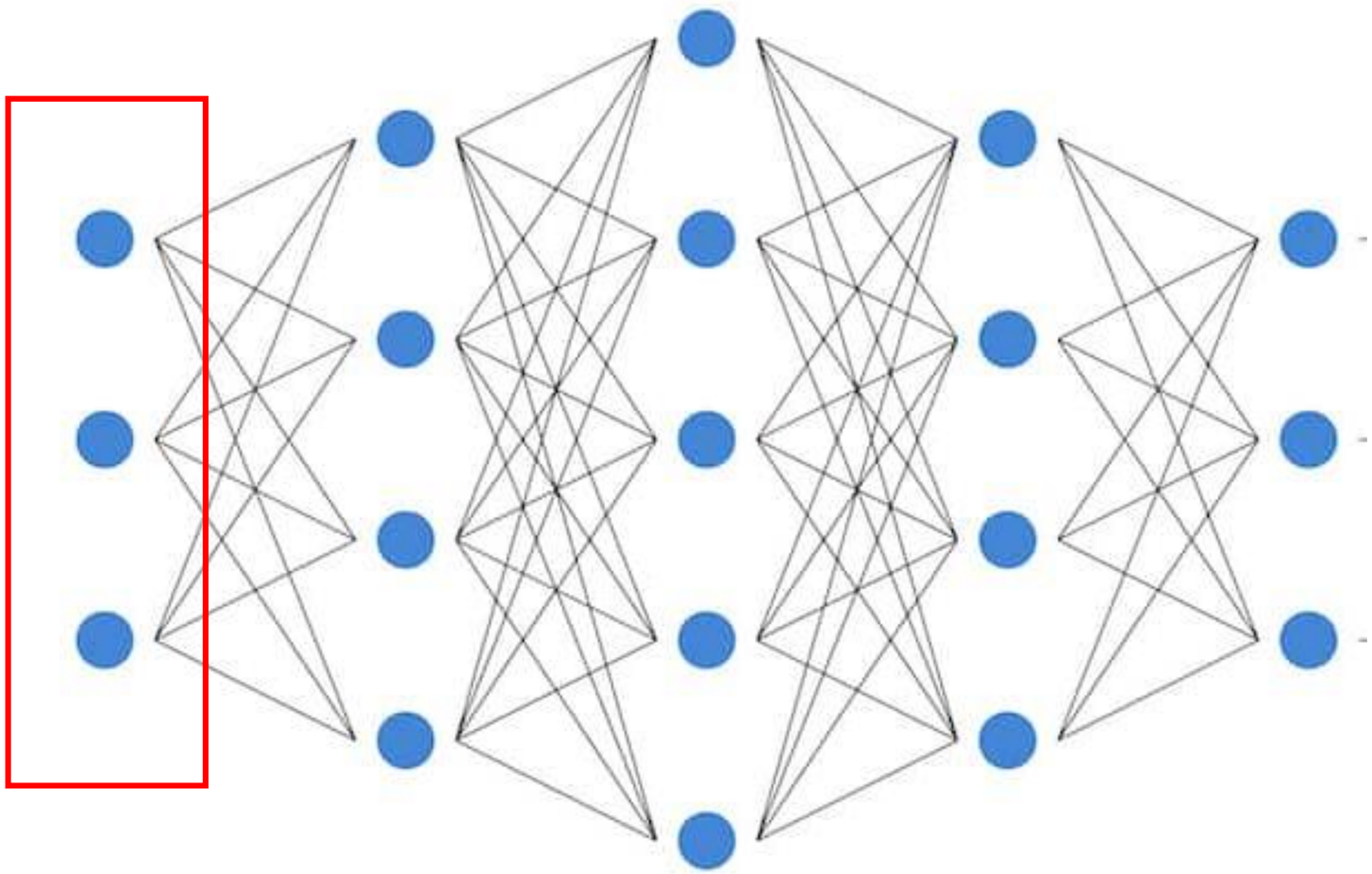


LAYER TYPES

[Image generated by DALLE-3]

INPUT LAYER

1. Independent variables
2. Taken as batches
3. Binned data
4. Categorized data

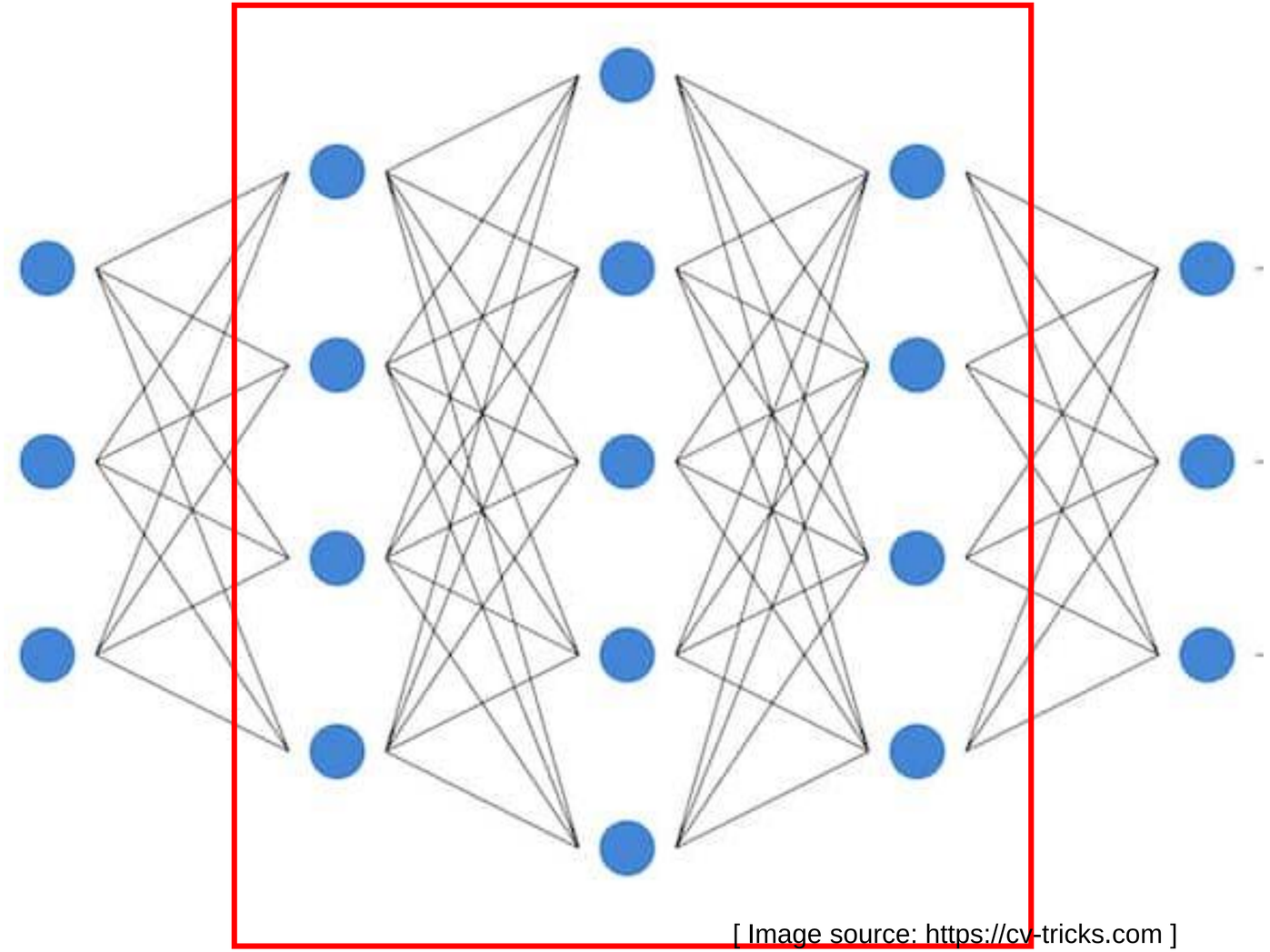


[Image source: https://cv-tricks.com]

HIDDEN LAYERS

1. Dense Layer (FC)
2. Convolution layer
3. RNN layer
4. LSTM layer
5. ResBlock layer
6. Attention layer

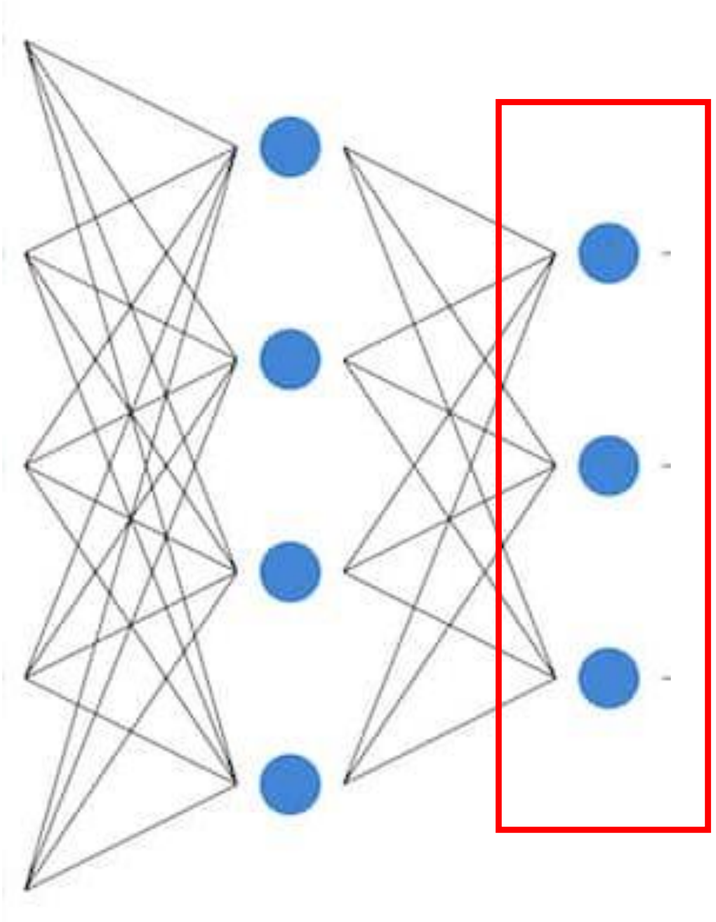
…



[Image source: https://cv-tricks.com]

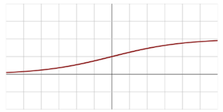
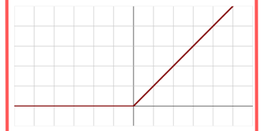
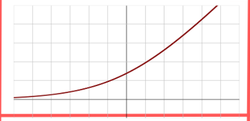
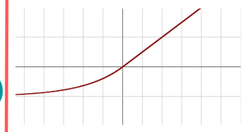
OUTPUT LAYER

Problem type	Nodes	Output Layer Activation
Regression	1	Linear
Multi-Target Regression	Number of targets	Linear
Binary Classification	1	Sigmoid
Multi-Label Classification	Number of labels	Softmax



ACTIVATION FUNCTIONS

ACTIVATION LAYER / ACTIVATION FUNCTIONS

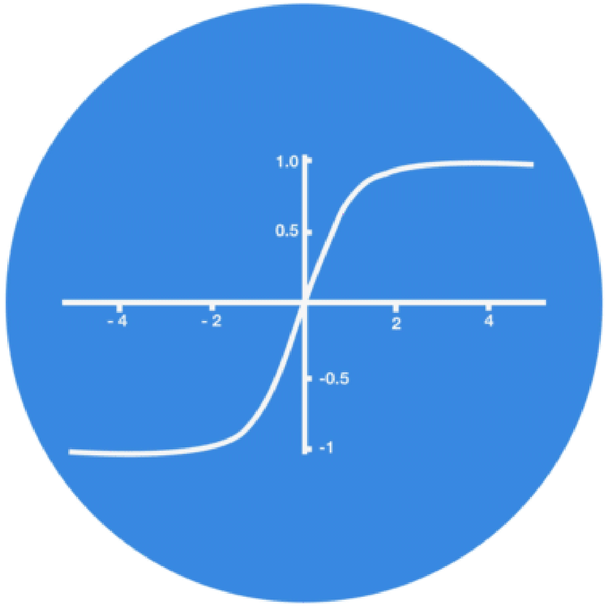
ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary Step		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$	$(0, 1)$
Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Softplus		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-1, 1)$
Exponential Linear Unit(ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$[0, \infty)$

[Image source: https://iq.opengenus.org]

ACTIVATION FUNCTIONS

Activation functions introduce non-linearity to the output of a neuron in an artificial neural network. This allows the network to learn complex, non-linear patterns in the data.

5
0.1
-0.5

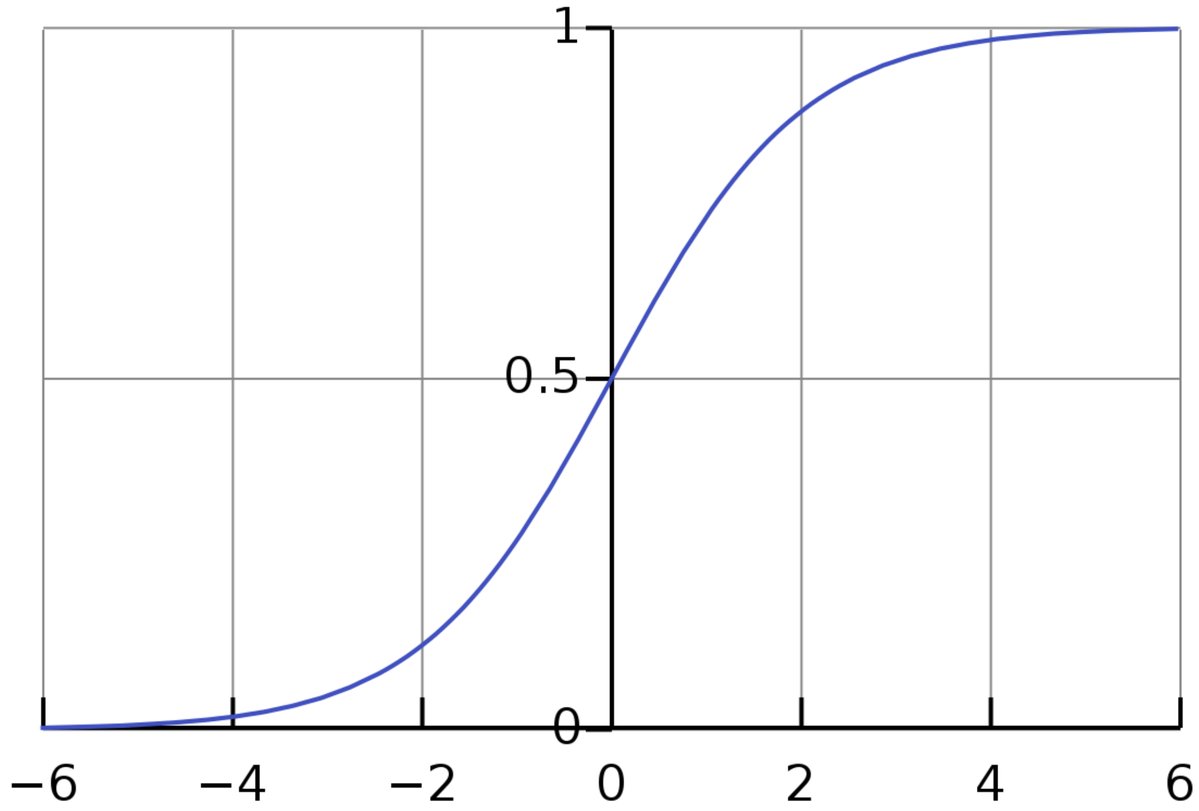


(Source: medium.com)

ACTIVATION FUNCTIONS: Sigmoid

The sigmoid function maps input values to a range between 0 and 1.
It has an "S" shape curve and is defined as:

$$f(x) = \frac{1}{1 + e^{-x}}$$



(Source: medium.com)

ACTIVATION FUNCTIONS: Sigmoid

Usage: Used in binary classification tasks where the output represents the probability of an input belonging to a specific class.

Pros:

With its range bound between 0 and 1, is useful for normalizing output values, predicting probabilities, having a smooth gradient, being differentiable, and yielding clear predictions close to 1 or 0.

Cons:

Suffers from vanishing gradients due to extreme outputs, lacks centered output causing weight update inefficiency, and has slower computation due to exponential operations.

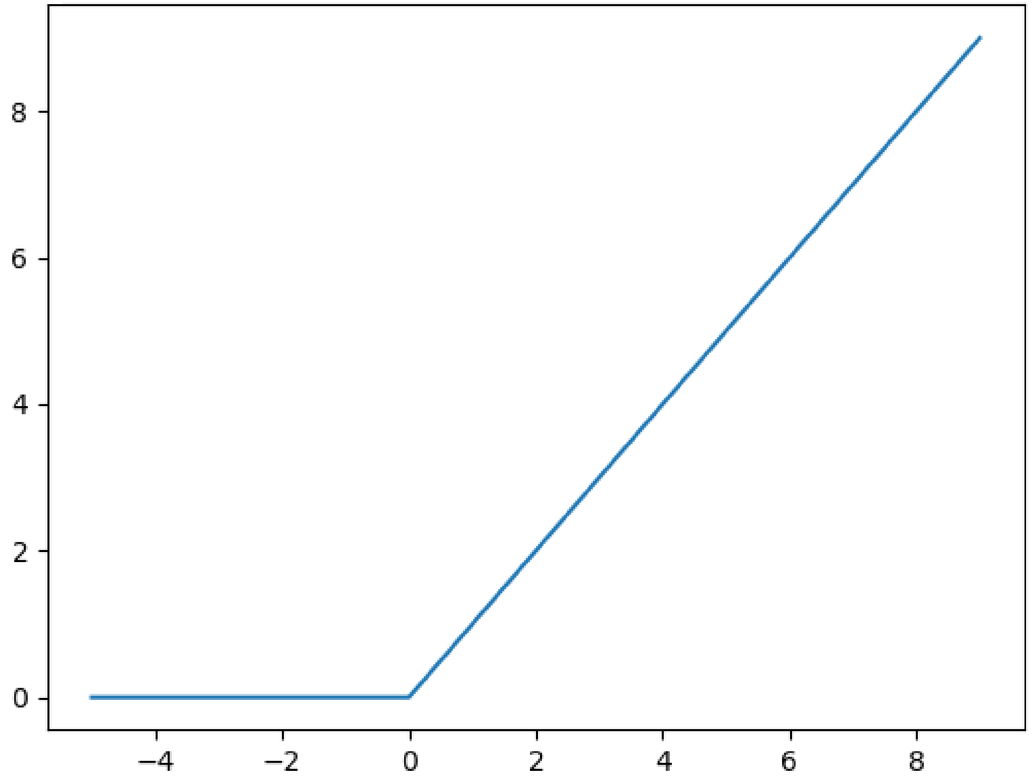
```python
import torch
import torch.nn as nn

x = torch.tensor([1.0, 2.0, 3.0])
sigmoid = nn.Sigmoid()
output = sigmoid(x)
print(output)
```

```
tensor([0.7311, 0.8808, 0.9526])
```

ACTIVATION FUNCTIONS: ReLU (Rectified Linear Unit)

ReLU is defined as the positive part of the input value. It is a piecewise linear function:

$$f'(x) = \begin{cases} max(0, x) & , x >= 0 \\ 0 & , x < 0 \end{cases}$$



(Source: medium.com)

ACTIVATION FUNCTIONS: ReLU (Rectified Linear Unit)

Usage: ReLU is widely used in deep learning architectures like CNNs (Convolutional Neural Networks) and MLPs (Multilayer Perceptrons) due to its simplicity and efficiency.

Pros:

Helps mitigate vanishing gradient problem.

Computationally efficient compared to other activation functions.

Cons:

Non-differentiable at zero.

Dying ReLU problem: Neurons can become inactive and stop learning if their weights are updated such that they always produce negative output.

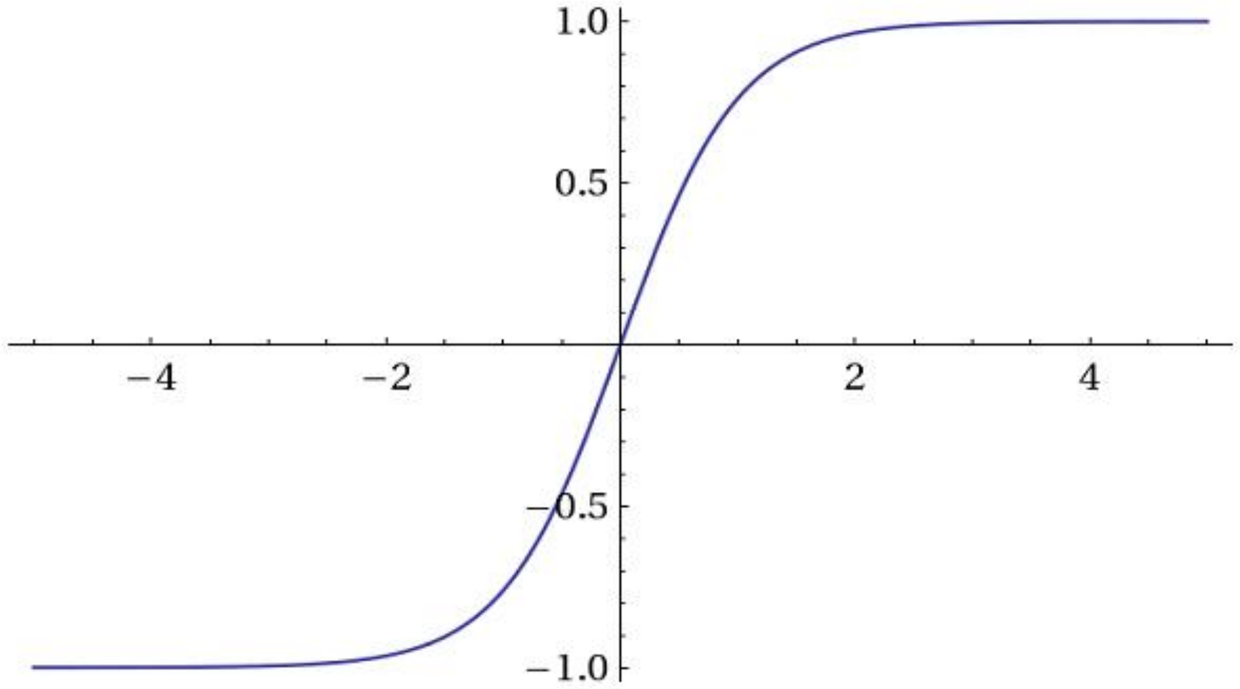
```
1 x = torch.tensor([-1.0, 0.0, 1.0])
2 relu = nn.ReLU()
3 output = relu(x)
4 print(output)
```

```
tensor([0., 0., 1.])
```

ACTIVATION FUNCTIONS: Tanh (Hyperbolic Tangent)

The tanh function maps input values to a range between -1 and 1. It is defined as:

$$f(x) = \frac{\left(e^{x} - e^{-x}\right)}{\left(e^{x} + e^{-x}\right)}$$



(Source: medium.com)

ACTIVATION FUNCTIONS: Tanh (Hyperbolic Tangent)

Usage: Used in tasks where the output is expected to be centered around zero, such as in recurrent neural networks (RNNs).

Pros:

Smooth function, which is differentiable.

Output range (-1, 1) is zero-centered.

Cons:

Suffers from vanishing gradients for large input values, which may slow down training.

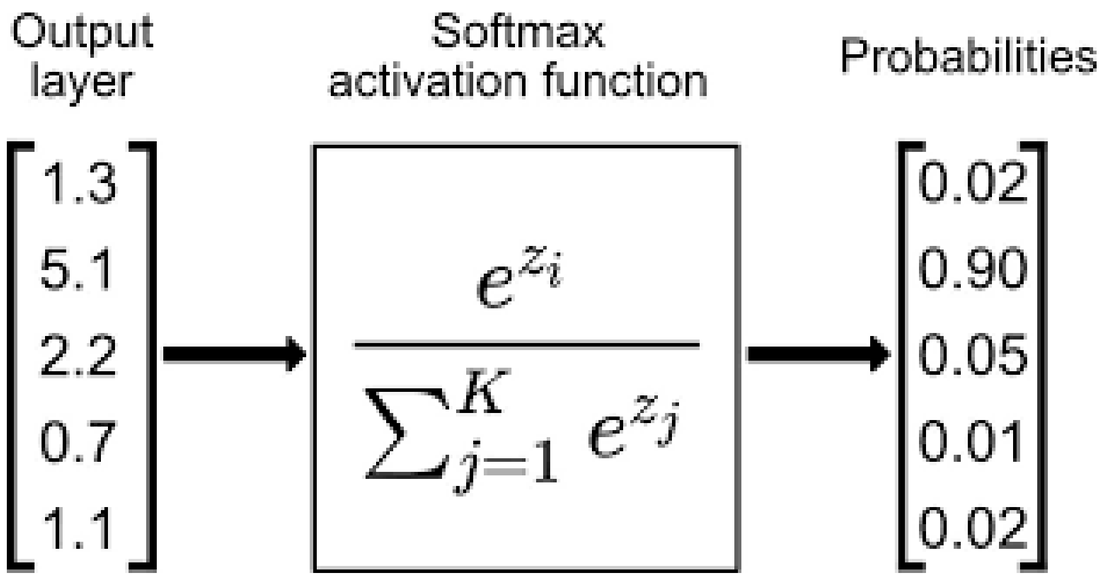
```
1  x = torch.tensor([-1.0, 0.0, 1.0])
2  tanh = nn.Tanh()
3  output = tanh(x)
4  print(output)
```

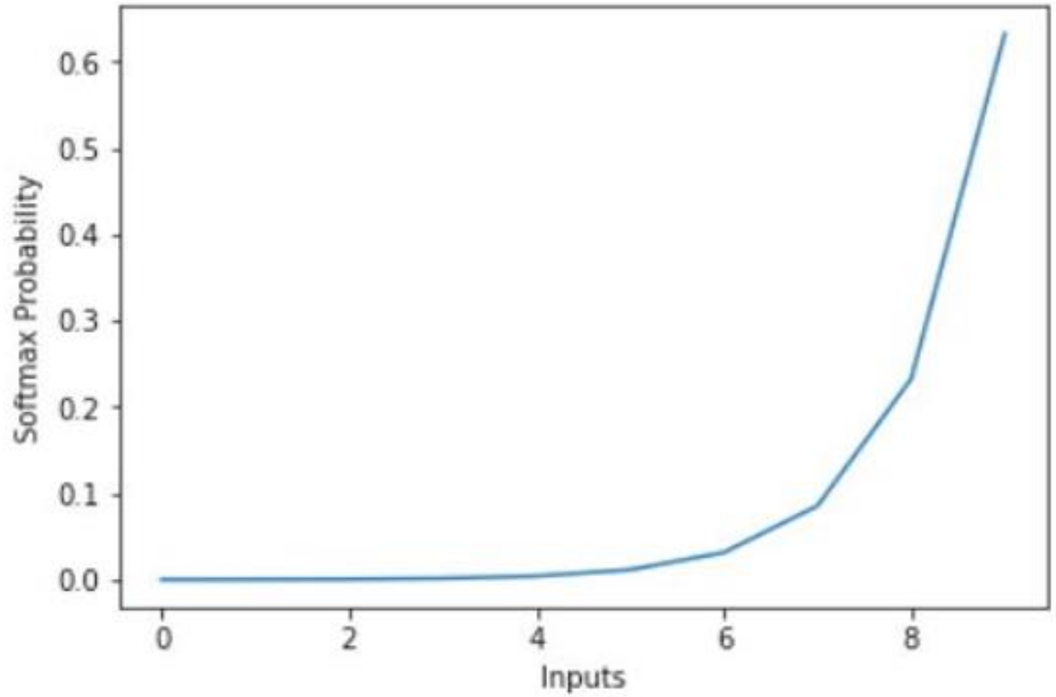
```
tensor([-0.7616,  0.0000,  0.7616])
```

ACTIVATION FUNCTIONS: Softmax

The softmax function maps input values to a probability distribution over multiple classes. It is defined as:

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$





(Source: medium.com)

ACTIVATION FUNCTIONS: Softmax

Usage: Used in multi-class classification tasks where the output represents the probability of an input belonging to a specific class.

Pros:

Outputs a probability distribution over classes, making it suitable for multi-class classification problems. The resulting probabilities sum up to 1, which is a desirable property for representing probabilities.

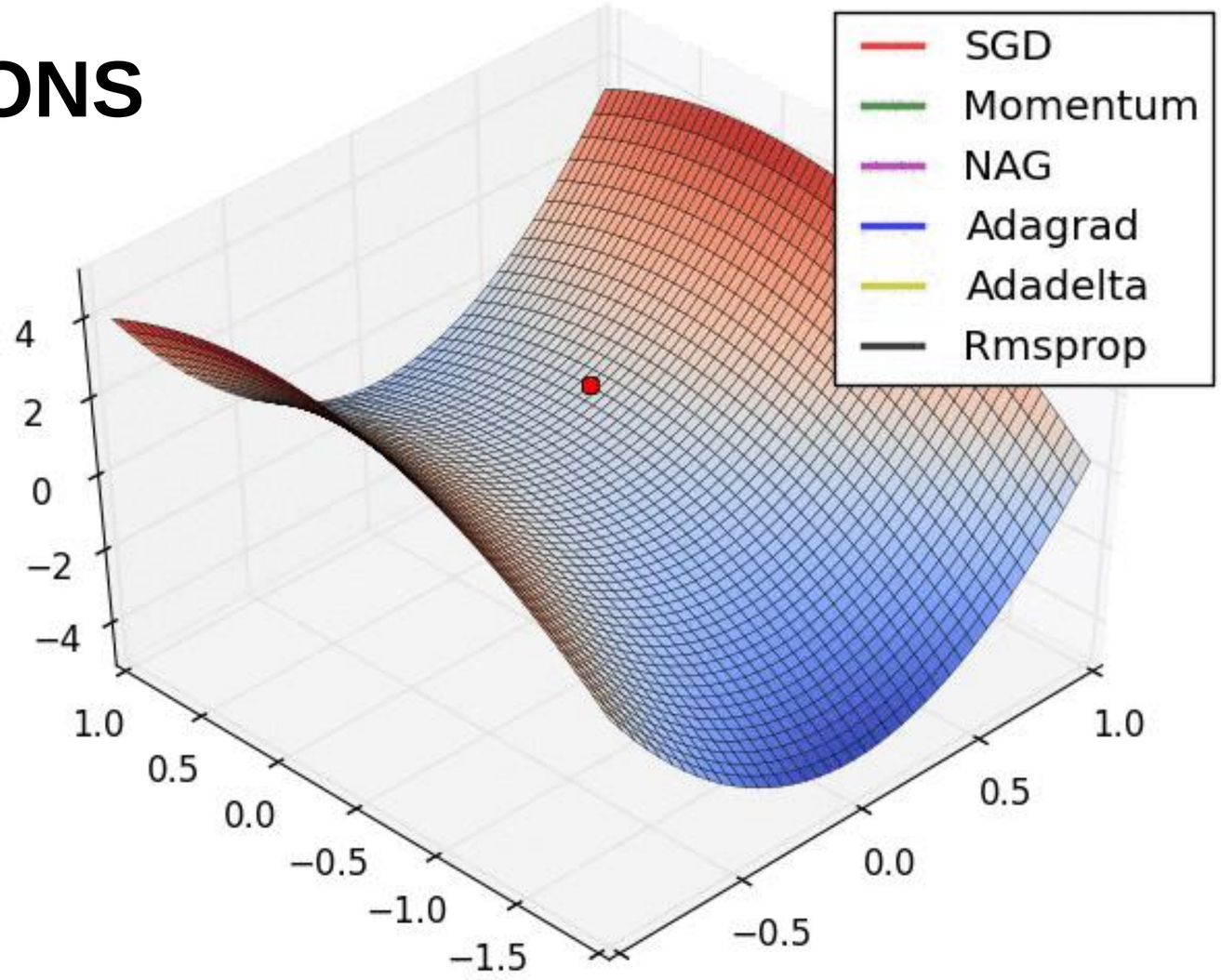
Cons:

Softmax is sensitive to large input values, which can lead to numerical instability. The exponential function used in softmax can be computationally expensive.

```
1  x = torch.tensor([1.0, 2.0, 3.0])
2  softmax = nn.Softmax(dim=0)
3  output = softmax(x)
4  print(output)
```

```
tensor([0.0900, 0.2447, 0.6652])
```

OPTIMIZATION FUNCTIONS



[Image source: https://www.ruder.io]

Optimization functions

```
import torch
import torch.optim as optim
```

- Optimization functions are algorithms used in neural network training to minimize the objective function.
- They iteratively update model parameters, guiding the network towards an optimal solution.
- These functions compute or estimate gradients to determine the direction and magnitude of parameter updates.
- Without proper optimization, neural networks may struggle to learn effectively or get stuck in suboptimal local minima.
- Different optimization functions offer various advantages and considerations, allowing customization to specific network architectures and data characteristics.
- Selecting an appropriate optimization function and tuning its hyperparameters can significantly impact model performance and training efficiency.

Gradient Descent

Gradient Descent is a first-order optimization algorithm that aims to find the minimum of a function by iteratively updating the parameters in the direction of the negative gradient.

Pros:

- Simplicity and ease of implementation
- Converges to a global minimum given certain conditions

Cons:

- Can be slow for large datasets or high-dimensional problems
- Sensitive to learning rate selection

Gradient Descent

Gradie e minimum of a fu e negative gradier

Pros:

- Sir
- Co

Cons:

- Ca
- Se

```
# Gradient Descent
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Training loop
for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is an optimization algorithm that computes the gradient and updates the parameters using a small randomly selected subset of the training data at each iteration.

Pros:

- Faster convergence compared to Gradient Descent for large datasets
- Can escape local minima more effectively

Cons:

- Noisy updates due to random sampling, which may lead to suboptimal solutions
- Learning rate tuning is still necessary

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is an optimization algorithm that computes the gradient and upd[illegible] the training data at e[illegible]

```
optimizer = optim.SGD(model.parameters(), lr=0.01)

for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

Pros:

- Faste[illegible]
- Can e[illegible]

Cons:

- Noisy [illegible] [illegible]utions
- Learni[illegible]

Momentum

Momentum is an optimization algorithm that introduces a momentum term to accelerate the convergence. It accumulates the past gradients' directions to determine the parameter update.

Pros:

- Faster convergence, especially in the presence of sparse gradients
- Helps navigate flat regions and escape local minima

Cons:

- Can overshoot the optimal solution in some cases
- Requires tuning of additional hyperparameters

Momentum

Momentum is an optimization algorithm that introduces a momentum term to ac… …mine the…

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

Pr…

- …
- …

Co…

- …
- Requires tuning of additional hyperparameters

AdaGrad (Adaptive Gradient)

AdaGrad is an adaptive learning rate optimization algorithm that individually adapts the learning rate for each parameter based on their historical gradients.

Pros:

- Automatic adjustment of learning rate for each parameter
- Effective for sparse data or problems with high dimensionality

Cons:

- Learning rate can become too small over time, hindering convergence
- Accumulation of squared gradients may dominate and suppress learning for important parameters

AdaGrad (Adaptive Gradient)

AdaGrad is an adaptive learning rate optimization algorithm that individually adapts the learning ra

```
optimizer = optim.Adagrad(model.parameters(), lr=0.01)

for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

Pros:

- Automa
- Effectiv

Cons:

- Learnin
- Accumu g for important parameters

RMSprop (Root Mean Square Propagation)

RMSprop is an adaptive learning rate optimization algorithm that addresses the diminishing learning rate issue in AdaGrad by using a moving average of squared gradients.

Pros:

- Better convergence compared to AdaGrad
- Automatically adjusts the learning rate

Cons:

- Requires manual tuning of additional hyperparameters
- May still suffer from the small learning rate problem in certain scenarios

RMSprop (Root Mean Square Propagation)

RMSpro es the diminish quared gradient

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

for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

Pros:

- Bette
- Autor

Cons:

- Requires manual tuning of additional hyperparameters
- May still suffer from the small learning rate problem in certain scenarios

Adam (Adaptive Moment Estimation)

Adam combines the advantages of both Momentum and RMSprop by using adaptive learning rates and momentum. It maintains an exponential moving average of past gradients and squared gradients.

Pros:

- Fast convergence and robustness to different types of data and architectures
- Applicable in various scenarios without extensive hyperparameter tuning

Cons:

- Relatively more memory intensive compared to other optimization algorithms
- Can converge to suboptimal solutions in certain cases

Adam (Adaptive Moment Estimation)

Ada adaptive
lear of past
gra

Pro

- F
- A

Cor

- F
- C

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

for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

Convolution model

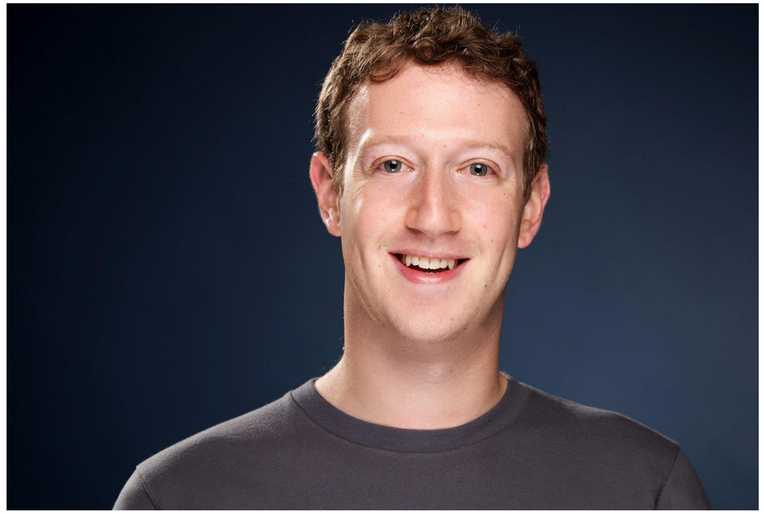
[Image source: google.com]

Before diving in …

[Image source: google.com]

DO YOU KNOW WHO THEY ARE?



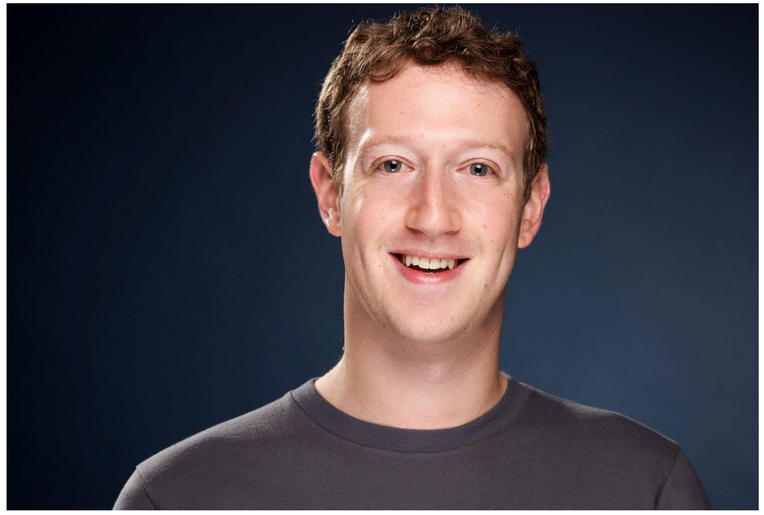
[Image source: google.com]

HOW DID YOU RECOGNIZE THEM?

Elon Musk

Barack Obama

Jack Ma



Mark Zuckerberg

[Image source: google.com]

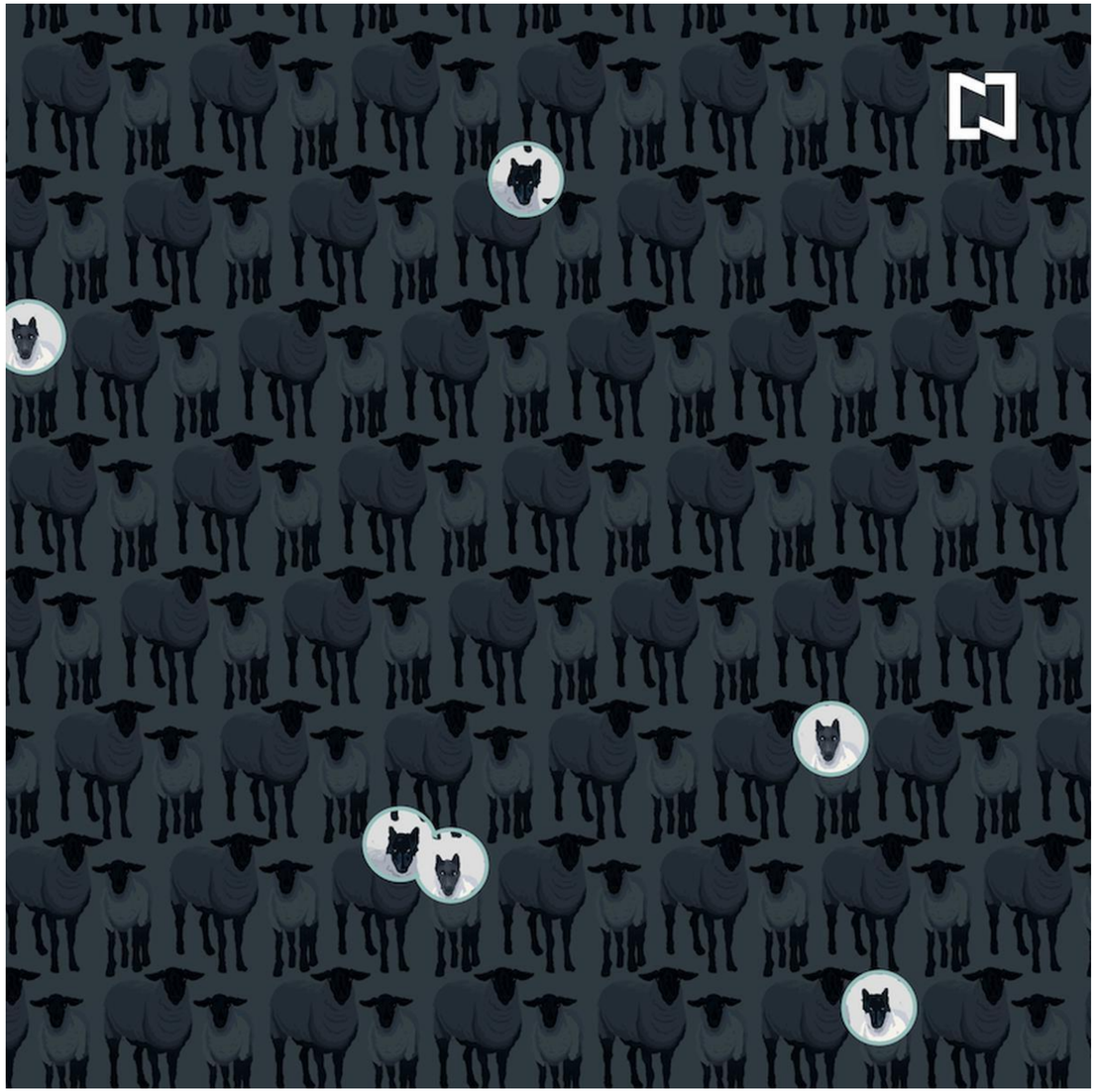
CAN YOU RECOGNIZE WHO THEY ARE?

[Image source: google.com]

HOW MANY WOLVES DO YOU SEE?

[Image source: google.com]

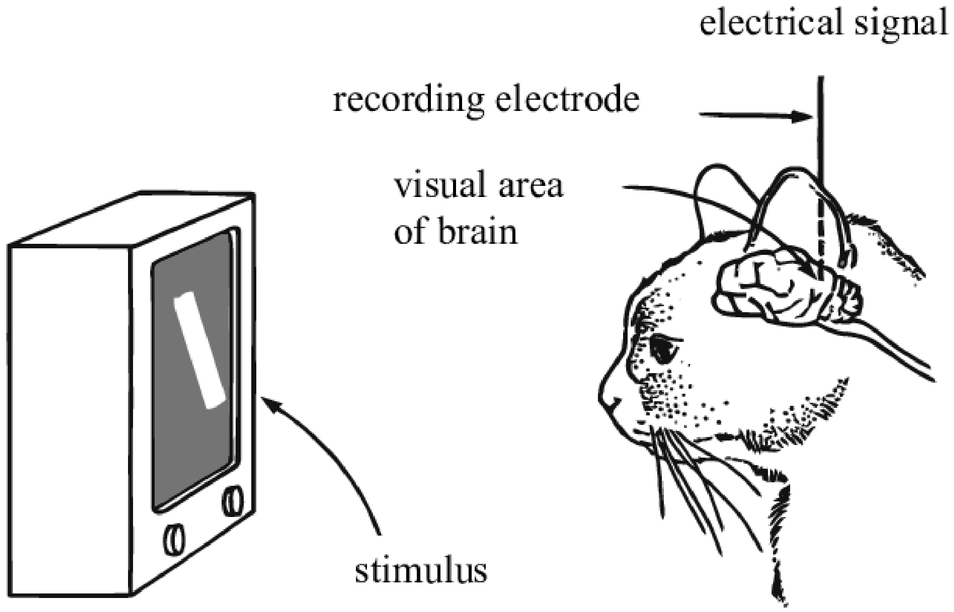
HOW DID YOU FIND THEM ?



[Image source: google.com]

HOW THE BRAIN RECOGNIZES WHAT THE EYE SEES?

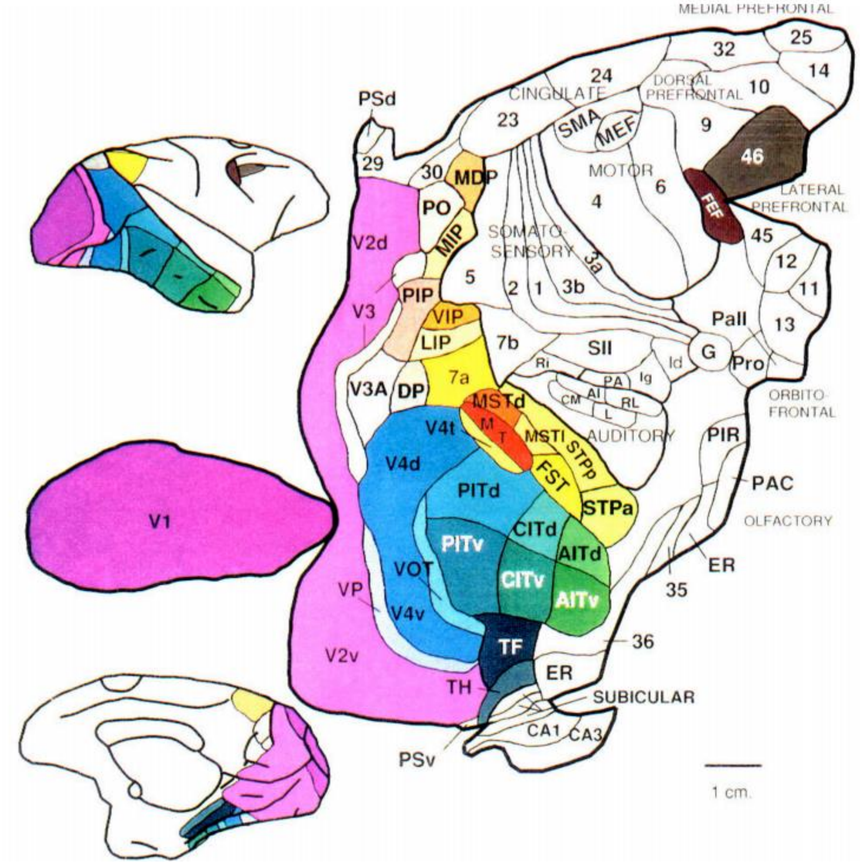
Simple and complex cells in visual cortex



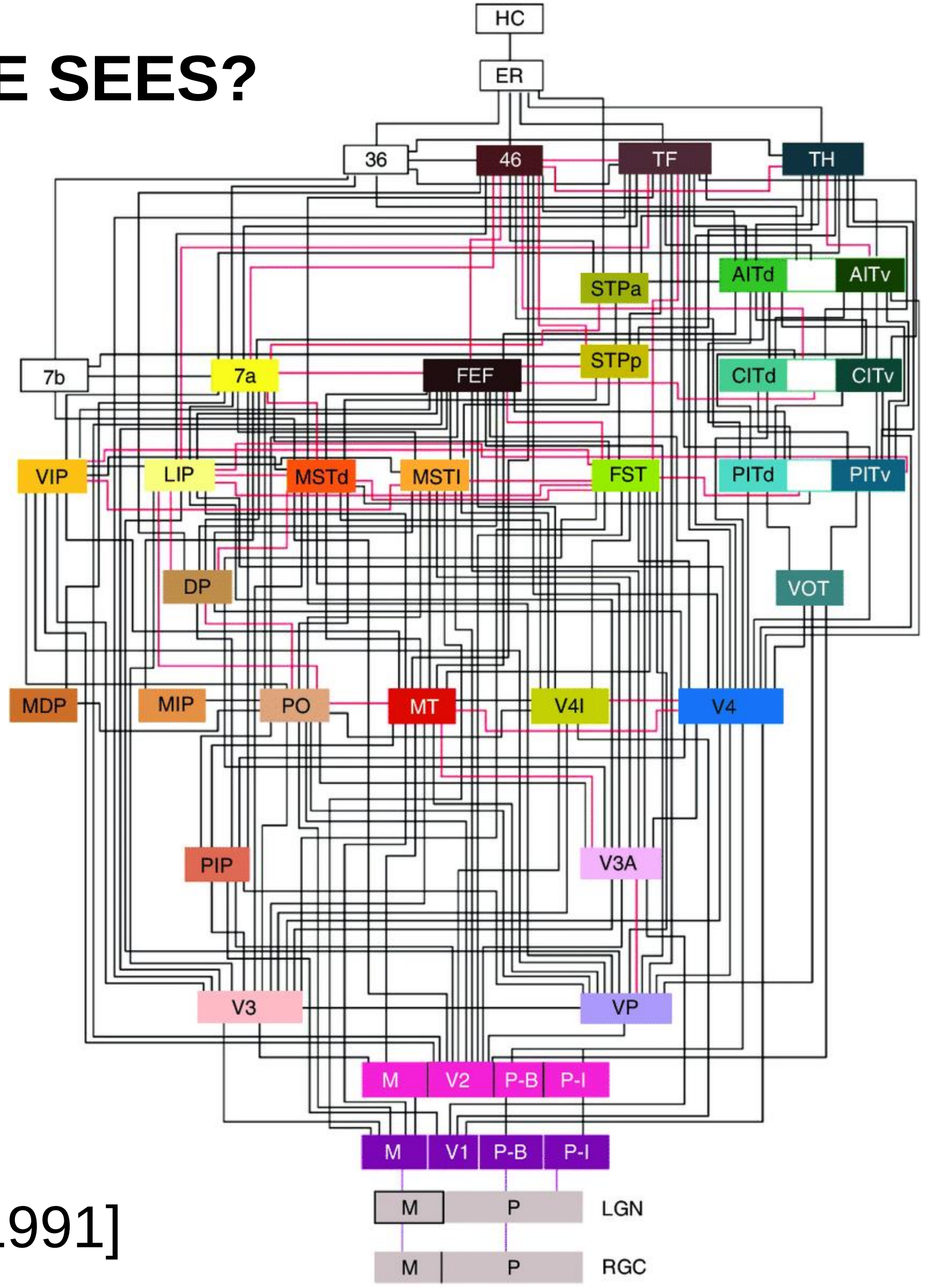
[Hubel & Wiesel, 1959]

HOW THE BRAIN RECOGNIZES WHAT THE EYE SEES?

Flat map of the Macaque brain



Vision Pipeline

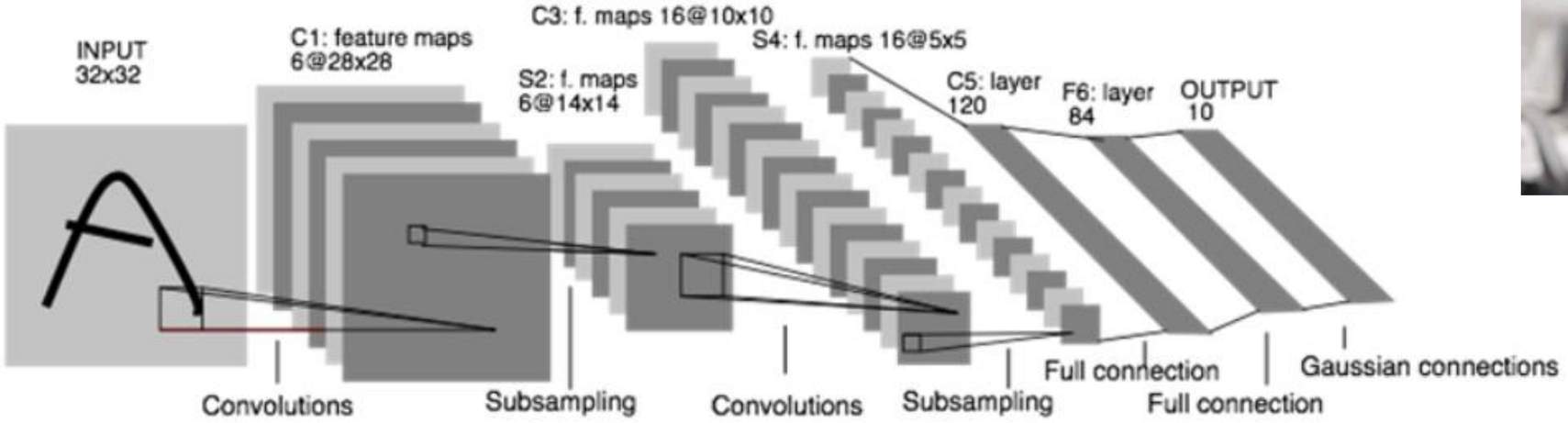


[Felleman & Van Essen, 1991]

Convolutional Neural Network

CONVOLUTIONAL NEURAL NETWORKS (CNNs)

LeNet-5 (1998)



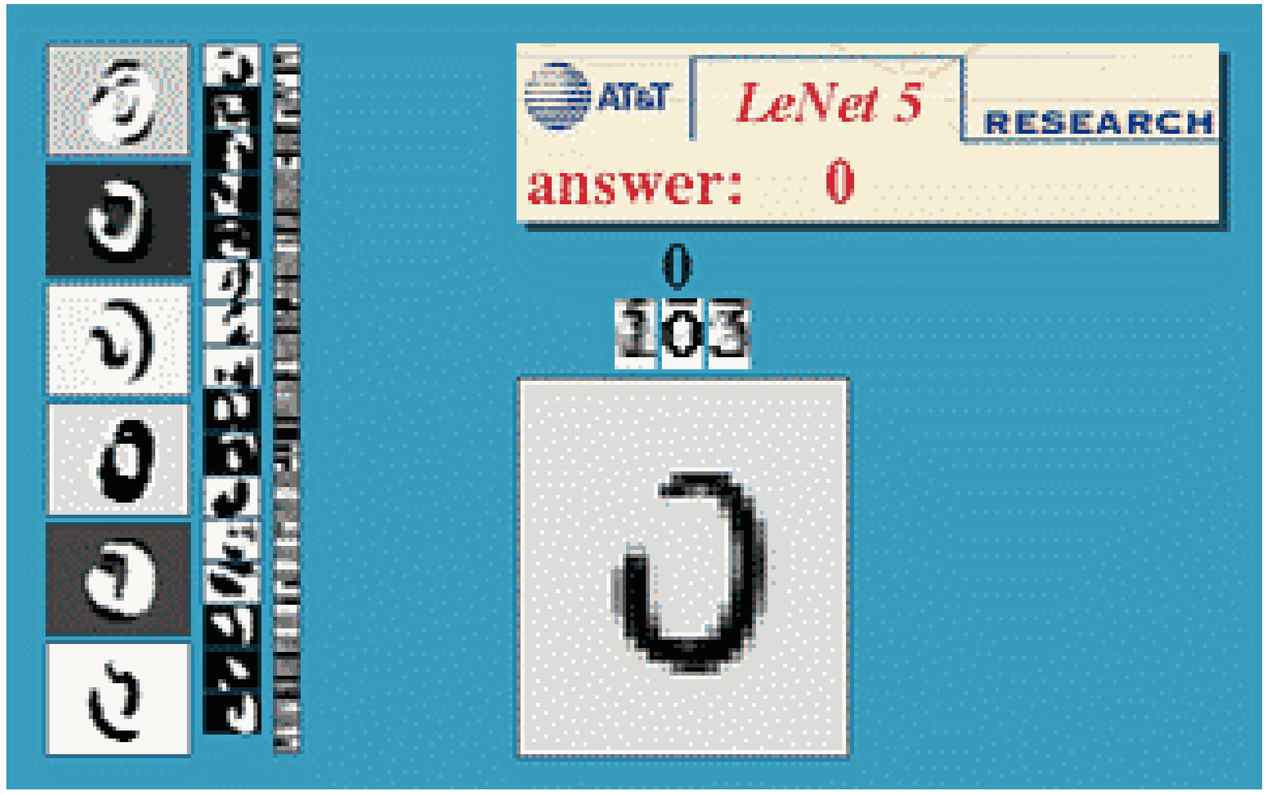
Papers: Backpropagation Applied to Handwritten Zip Code Recognition (LeCun-1989) Gradient-Based Learning Applied to Document Recognition(LeCun-1998)

Yann LeCun, VP and Chief AI Scientist, Facebook; Silver Professor of Computer Science, Data Science, Neural Science, and Electrical and Computer Engineering, New York University; ACM Turing Award Laureate

[Source: yann.lecun.com]

CONVOLUTIONAL NEURAL NETWORKS (CNNs)

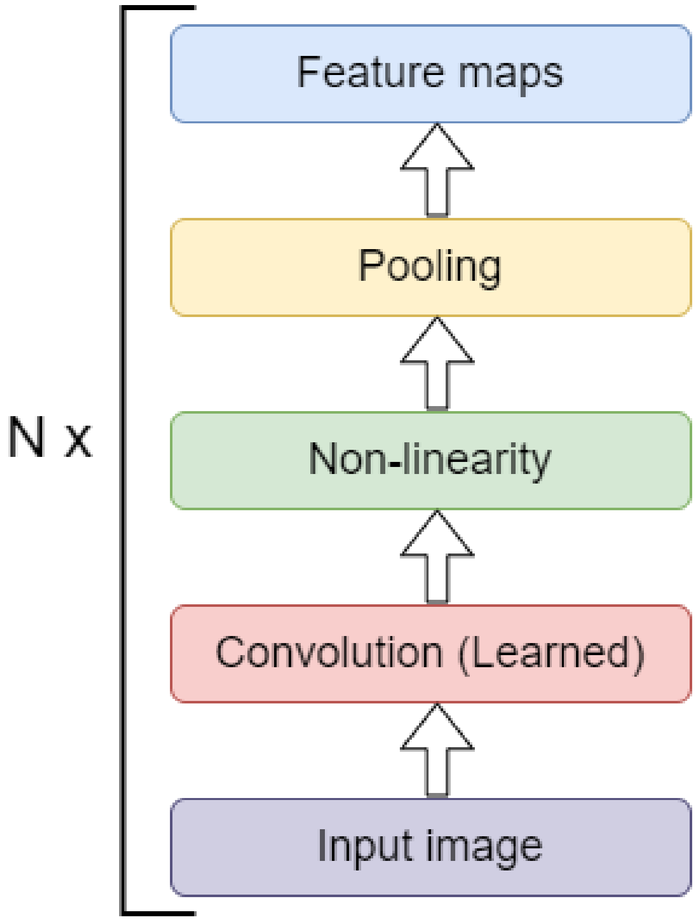
[LeCun et al. 1998]



LeNet-5 was used by the American Post office to automatically identify handwritten zip code numbers and was used on large scale to automatically classify hand-written digits on bank cheques in the United States

GENERAL STRUCTURE OF CNN

- Feedforward model
- 1 layer contains:
  - Convolution filter
  - Non-linearity
  - Pooling
- Deep = Many layers



2012: A BREAKTHROUGH YEAR FOR DEEP LEARNING

Google Hires Brains that Helped Supercharge Machine Learning

Google has hired the man who showed how to make computers learn much like the human brain.

Geoffrey Hinton (right), one of the machine learning scientists hard at work on The Google Brain. Photo: University of Toronto

AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

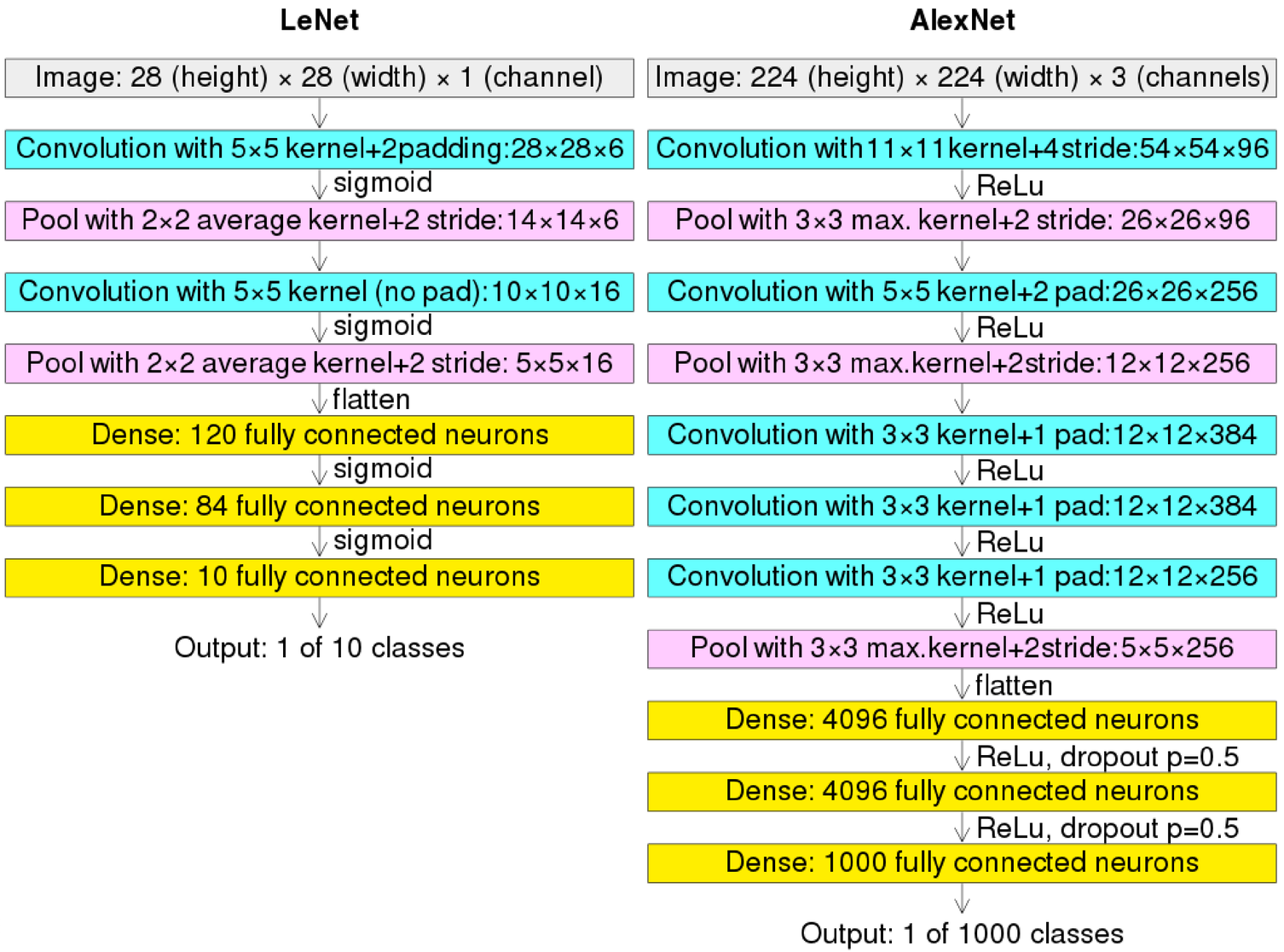
Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 dif-

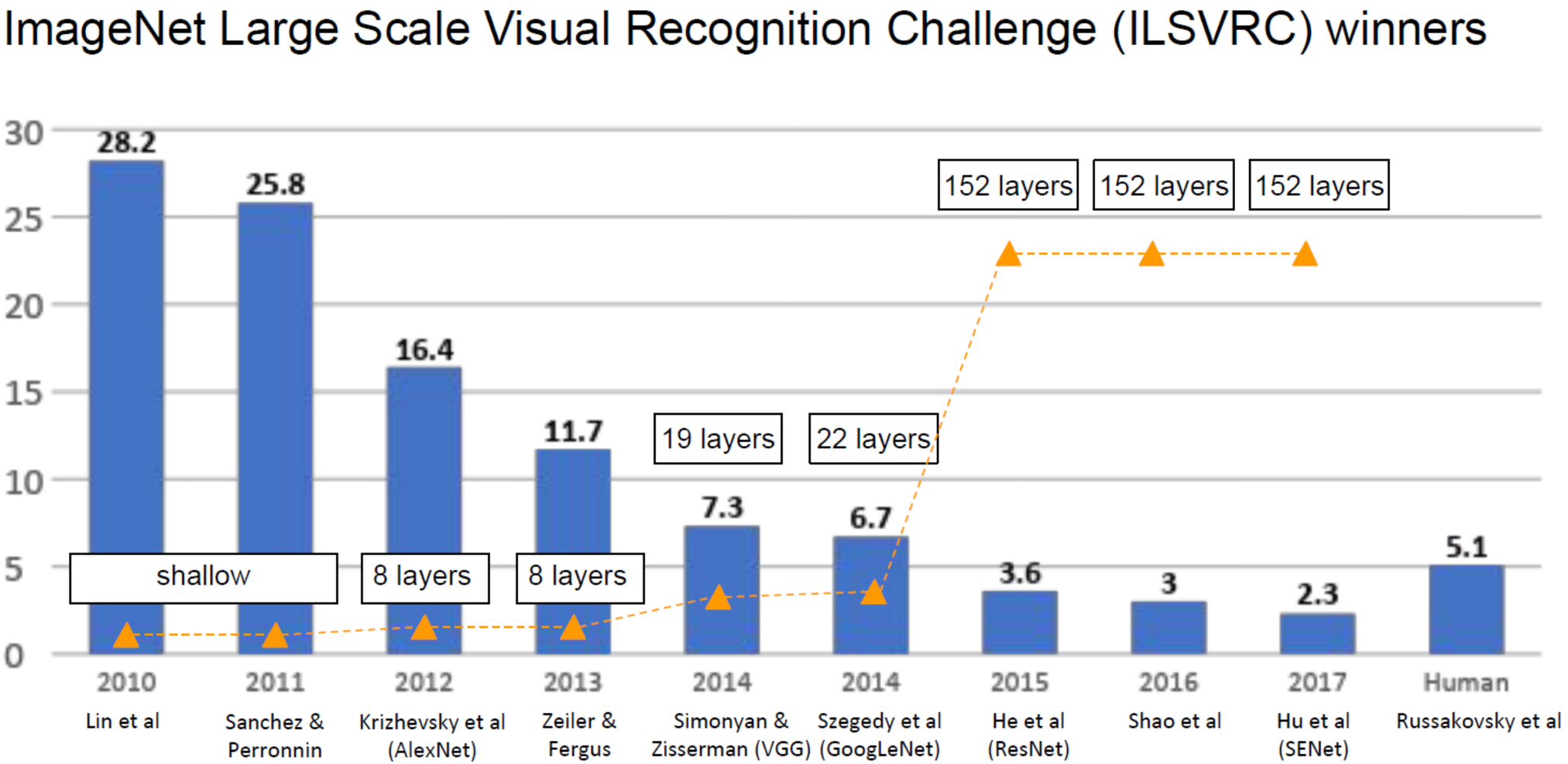
"The price climbed so high, Hinton shortened the bidding window from an hour to 30 minutes. The bids quickly climbed to $40 million, $41 million, $42 million, $43 million. "It feels like we're in a movie," he said. One evening, close to midnight, as the price hit $44 million, he suspended the bidding again. He needed some sleep."

Source: www.wired.com/story/secret-auction-race-ai-supremacy-google-microsoft-baidu/

COMPARISON OF THE LENET AND ALEXNET

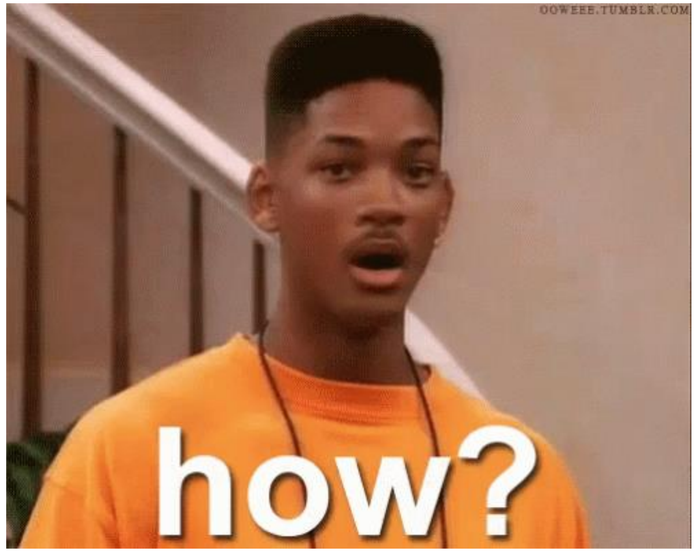


IMAGENET CLASSIFICATION



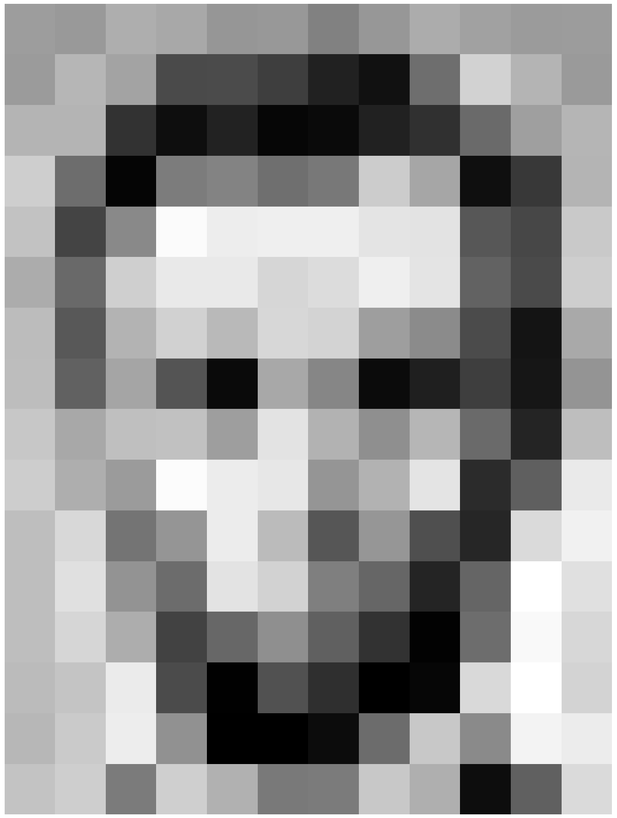
[Image source: google.com]

How does CNN actually work?



[Image source: google.com]

WHAT DO COMPUTERS SEE?



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
196	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
196	206	123	207	177	121	123	200	175	13	96	218

Source: https://openframeworks.cc/ofBook/chapters/image_processing_computer_vision.html

IMAGE DATA: 2 KEY PROPERTIES
LOCALITY AND TRANSLATION INVARIANCE

[Image source: google.com]

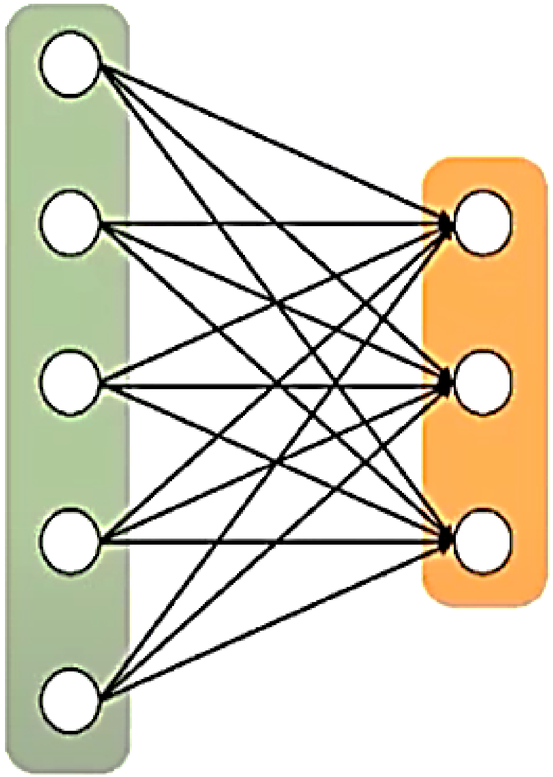
Locality: nearby pixels are more strongly correlated

Translation invariance: meaningful patterns can occur anywhere in the image

IMAGE DATA: 2 KEY PROPERTIES
LOCALITY AND TRANSLATION INVARIANCE

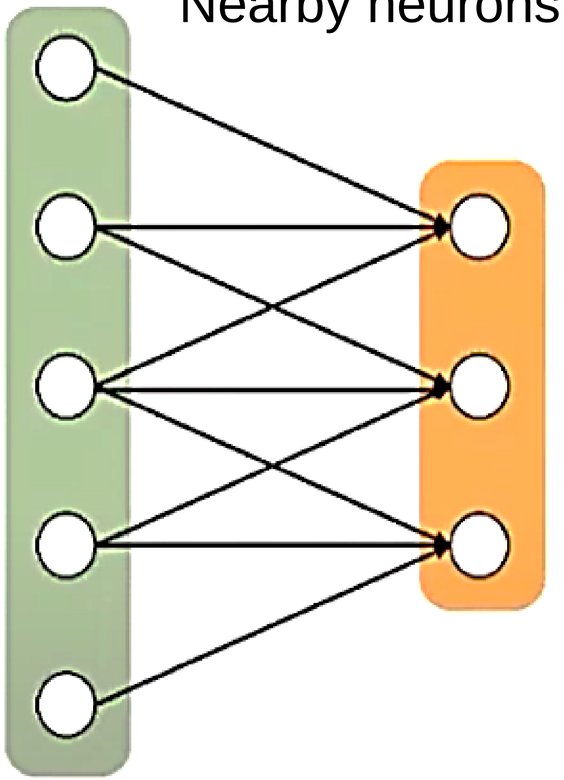
[Image source: google.com]

PROPERTY 1: LOCALITY



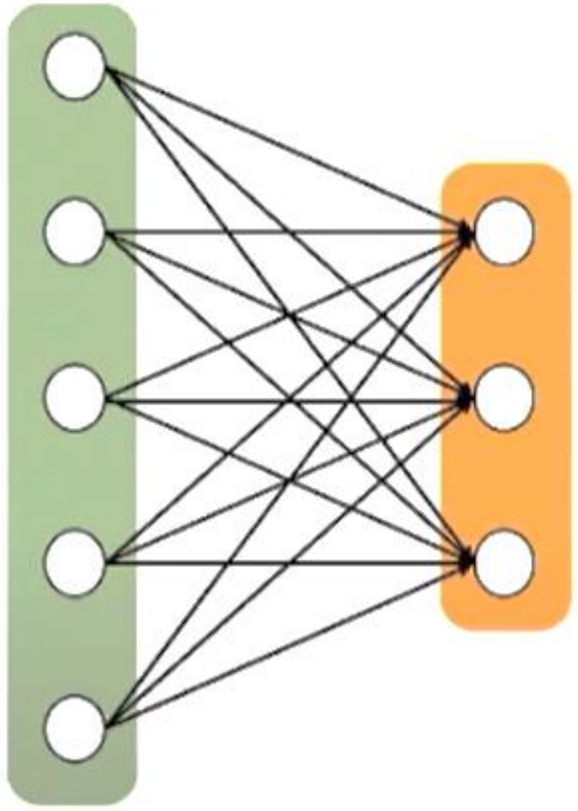
Fully-connected
(3 x 5 = 15 parameters)

Nearby neurons

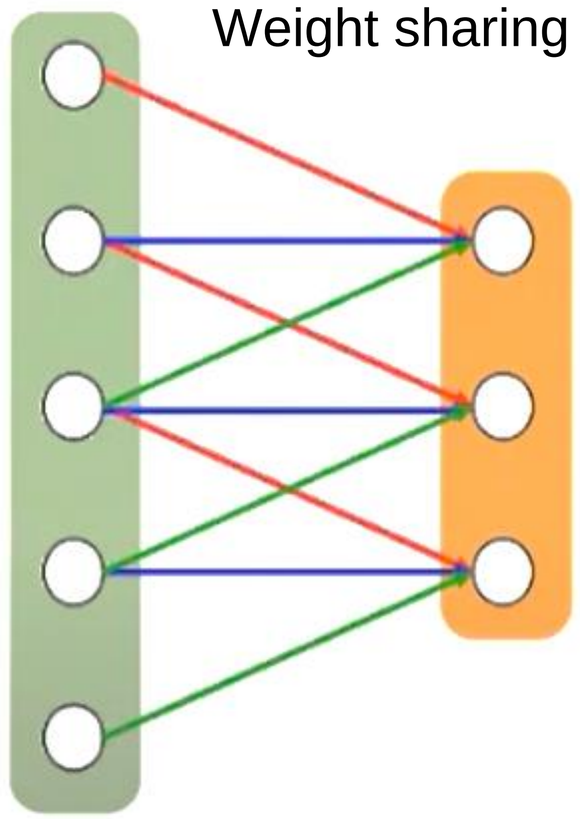


Locally-connected
(3 x 3 = 9 parameters)

PROPERTY 2: TRANSLATION INVARIANCE

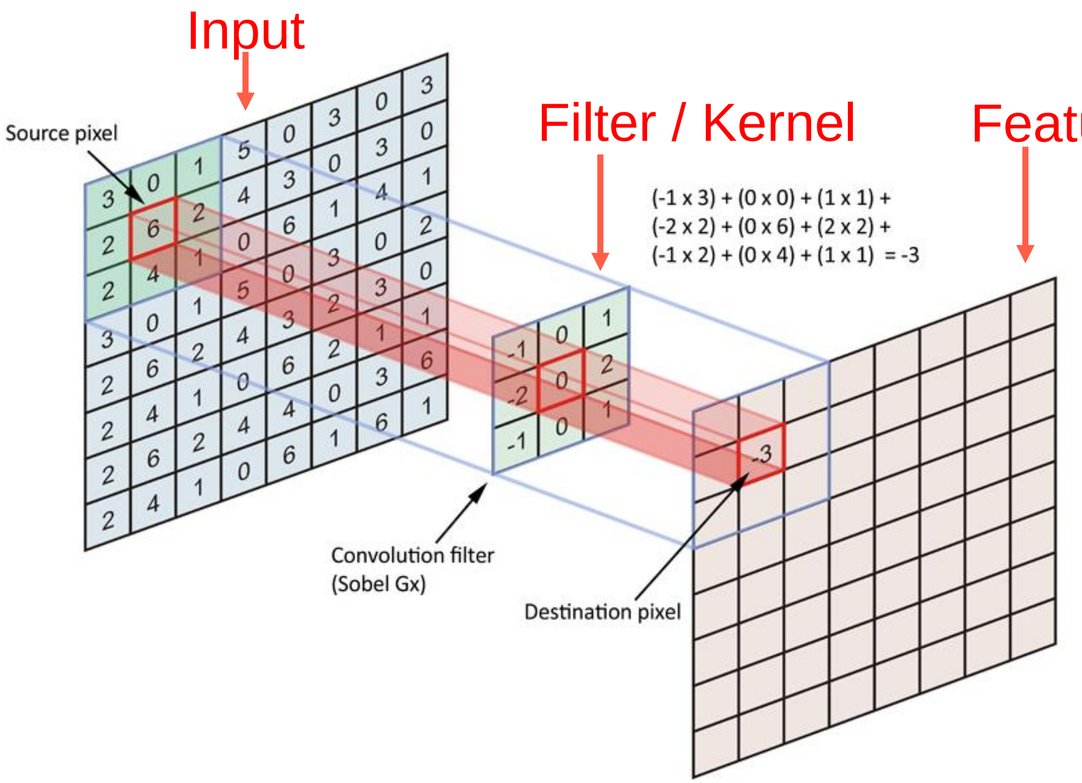


Fully-connected
(3 x 5 = 15 parameters)



Locally-connected
(3 parameters)

CONVOLUTION FILTER



7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

*

1	0	-1
1	0	-1
1	0	-1

=

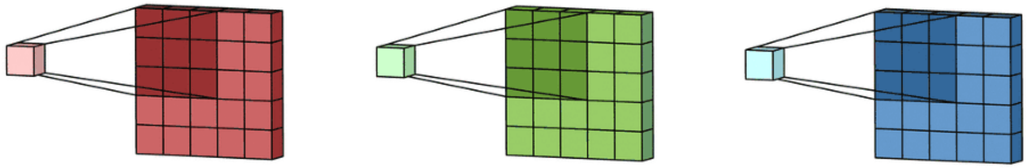
6		

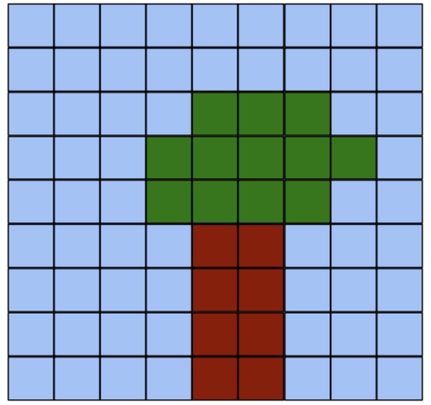
7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

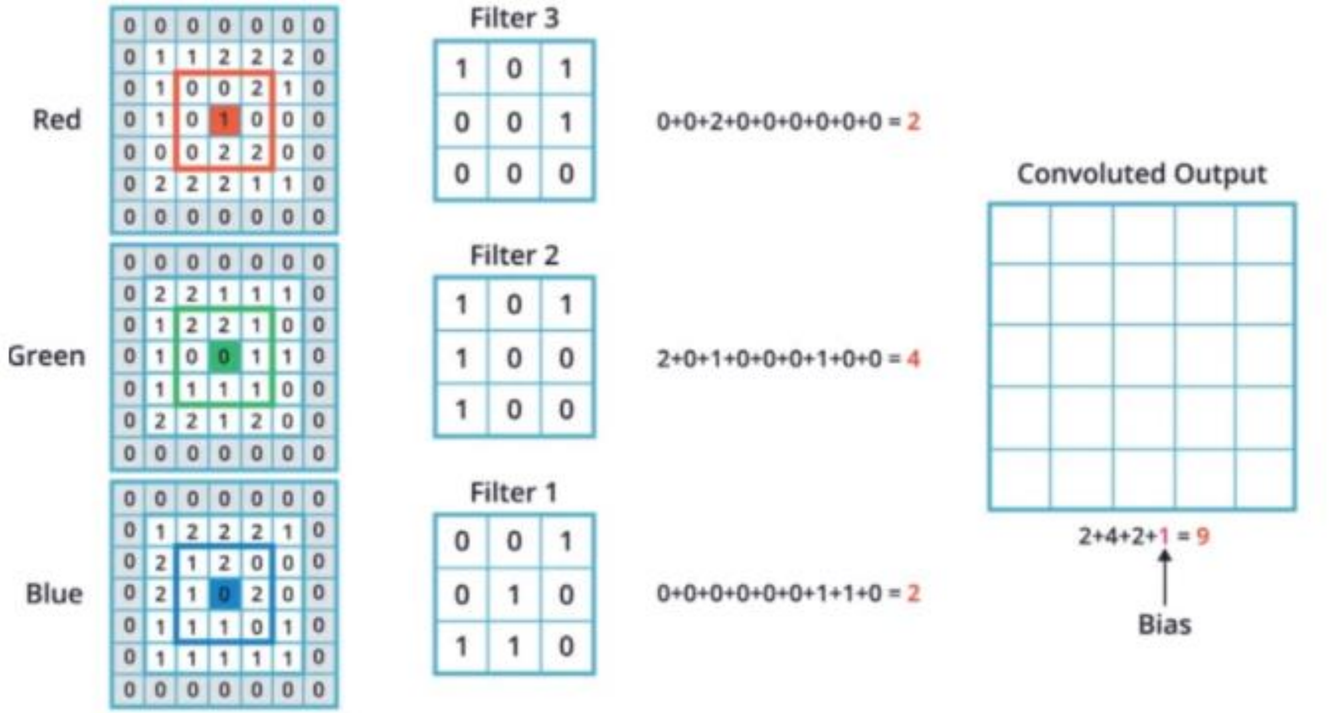
Slide over the image spatially, computing element wise dot products, and sum over all

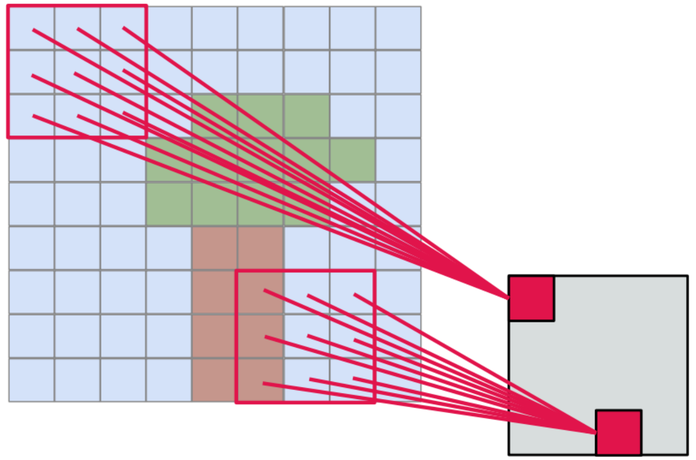
Convolution operation captures the local dependencies in the original image.

CONVOLUTION FILTER





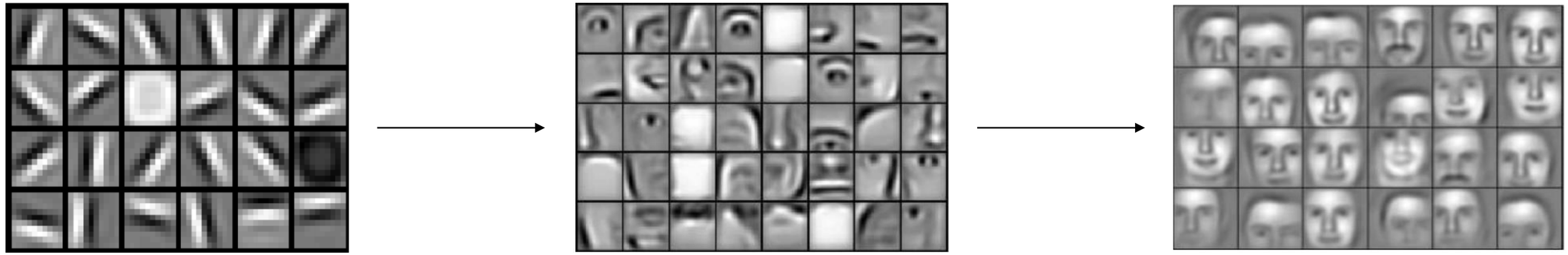


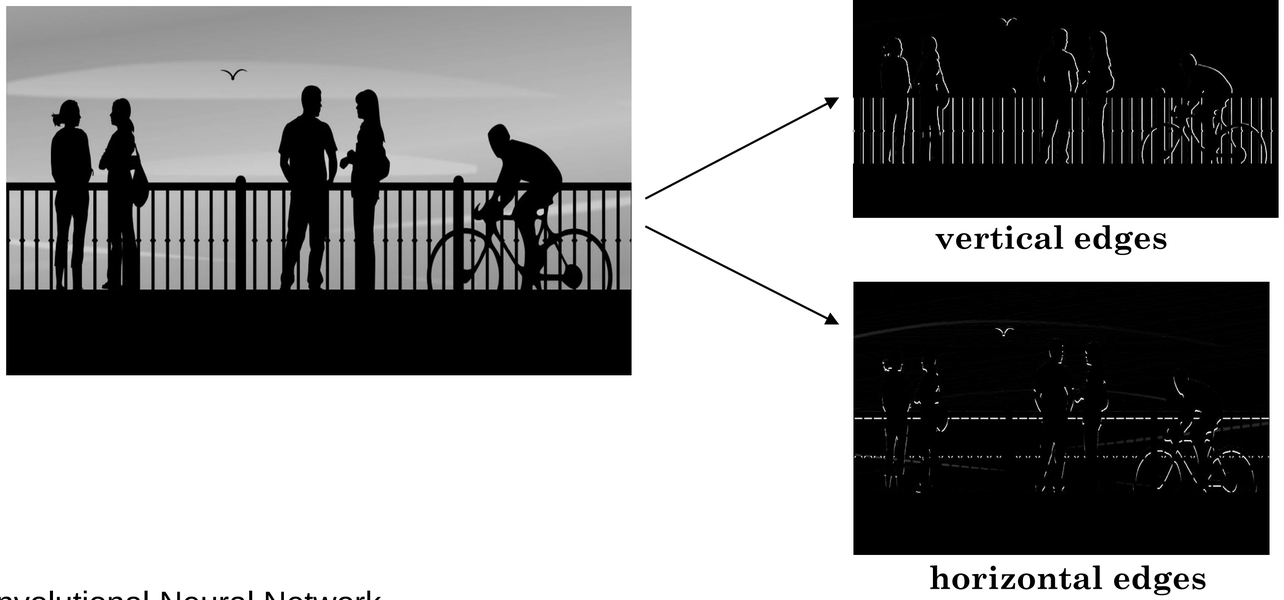


$$y = \mathbf{w} * \mathbf{x} + b$$

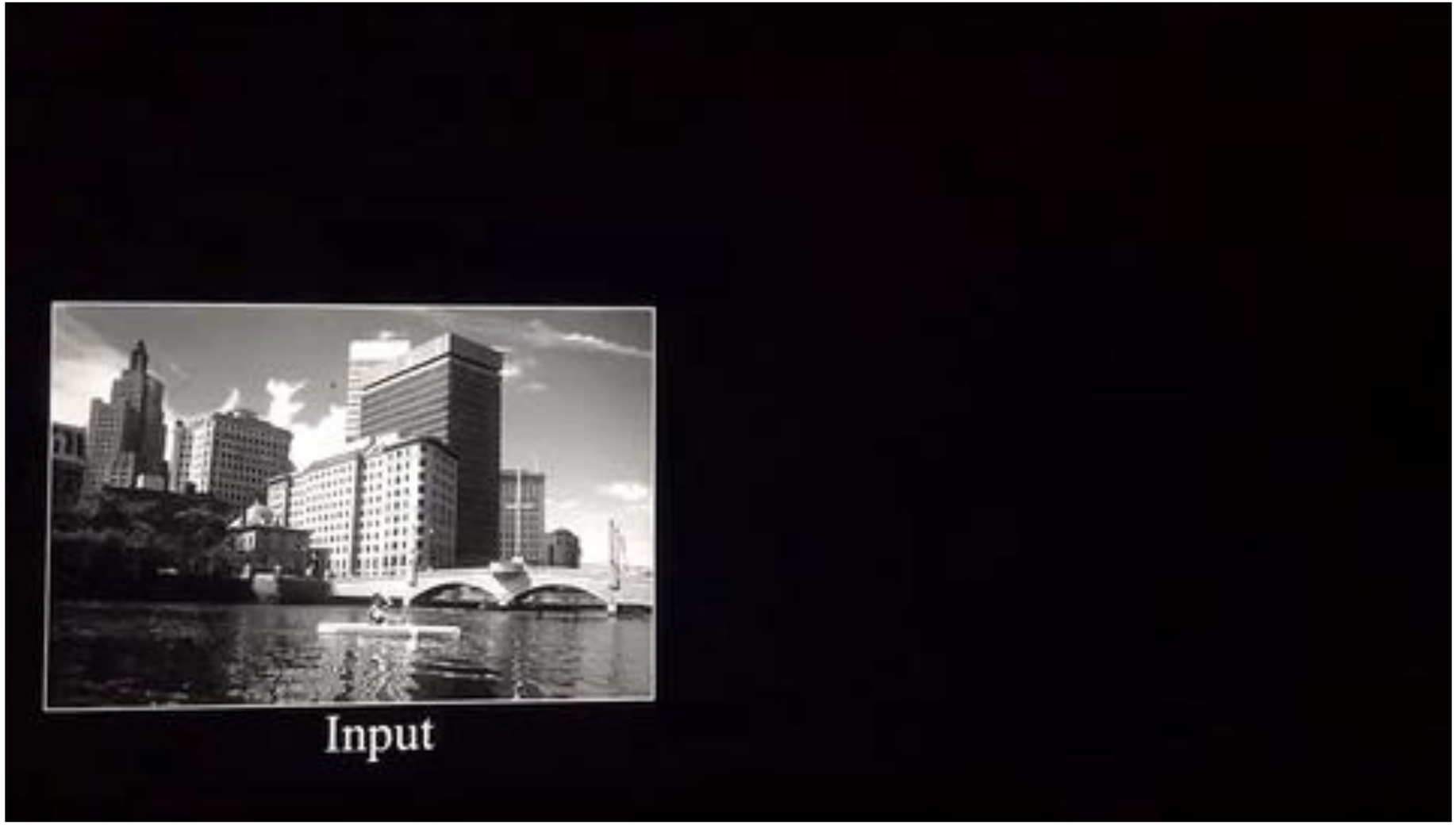
convolutional units
3×3 receptive field

CONVOLUTION FILTER



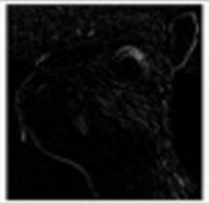


CONVOLUTION FILTER



[Image source: google.com]

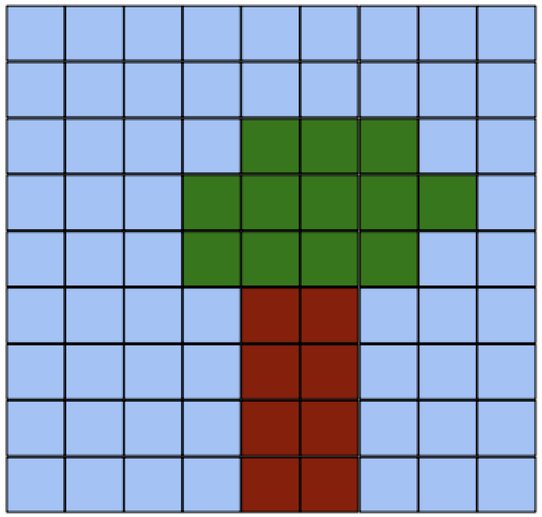
CONVOLUTION FILTER

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

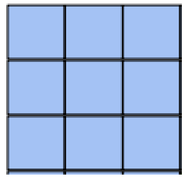
Operation	Filter	Convolved Image
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Different "filter" can produce different feature maps

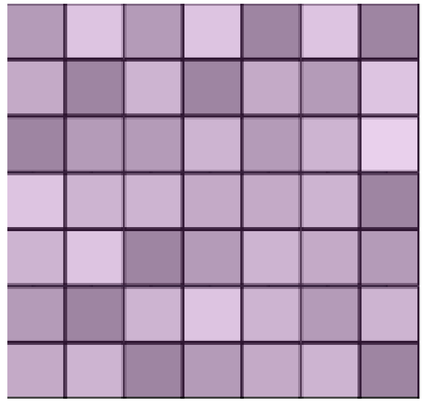
FEATURE MAP SIZE



n x n
(9 x 9)



k x k
(3 x 3)



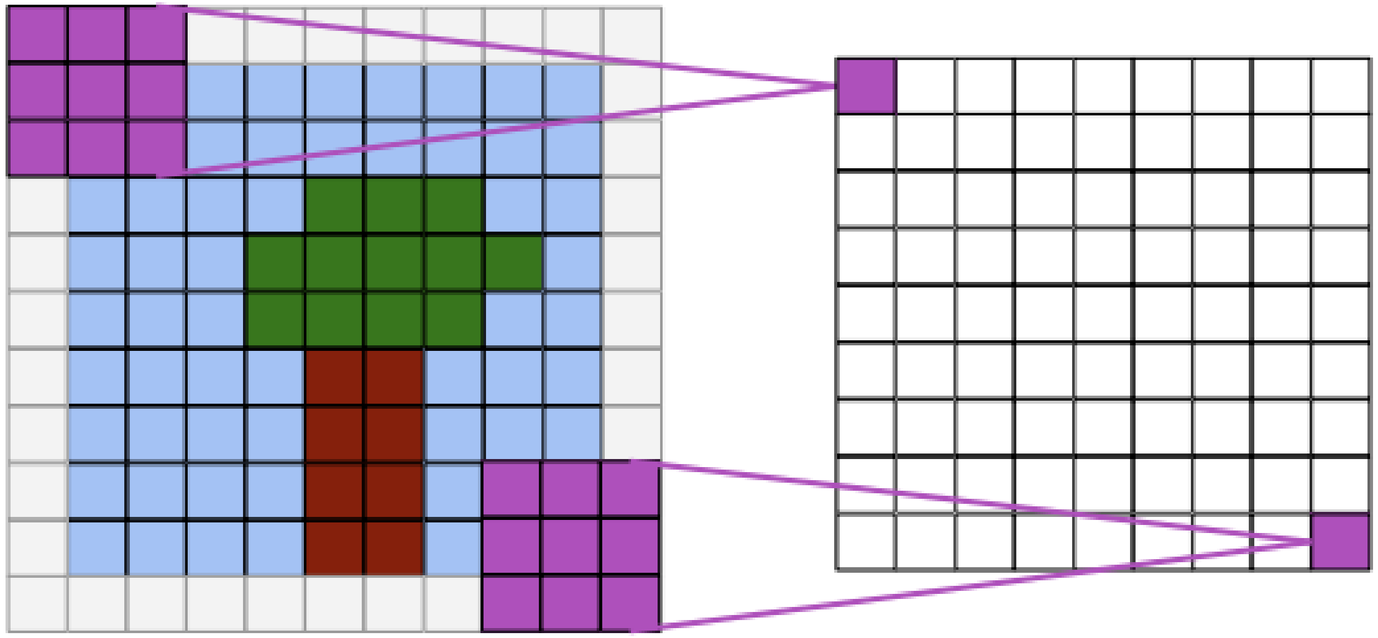
m x m
(7 x 7)

2 downsides:

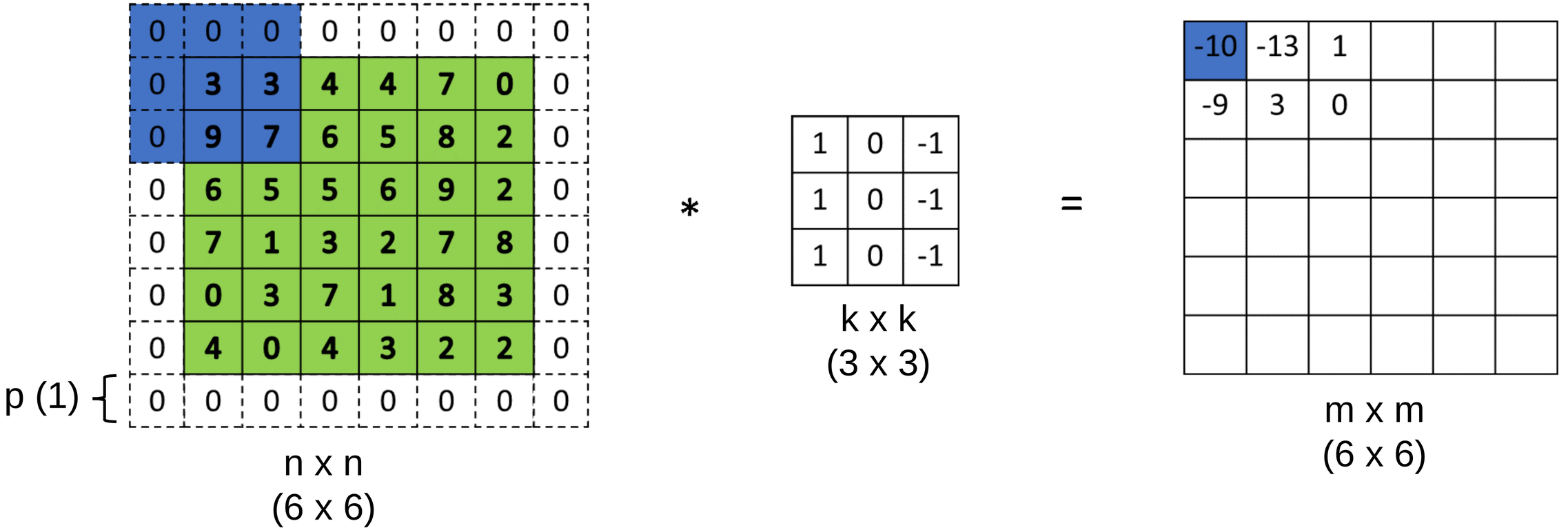
- Shrinking output
- Information loss near the edge of the image.

$$m = n - k + 1$$

PADDING



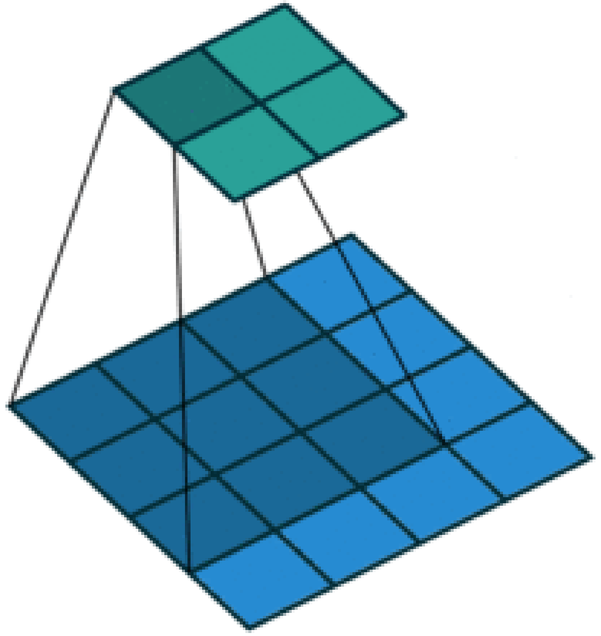
PADDING



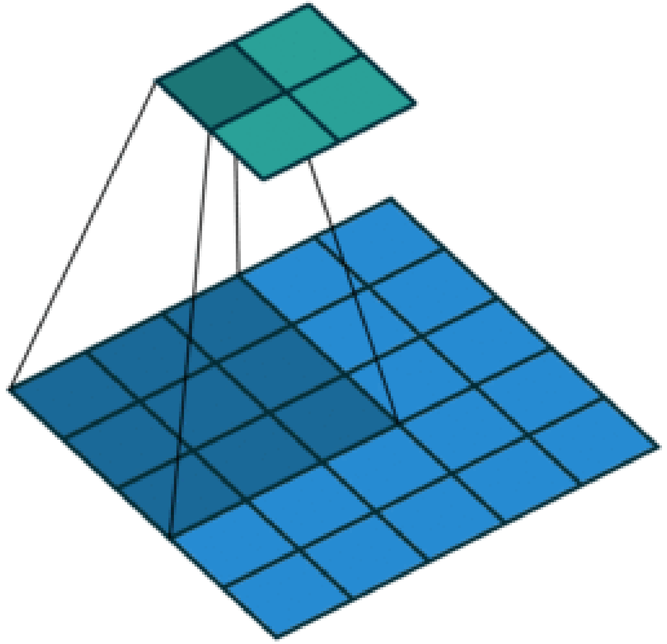
$$m = n + 2p - k + 1$$

6 +2*1 – 3 +1 = 6

STRIDE



Stride-1 convolution (“non-strided”)



Stride-2 convolution (“strided”)

Stride parameter controls the length of kernel step. By default it’s usually 1.
You can increase the stride (step) length in order to reduce calculation time.

FEATURE MAP SIZE

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features

n_{out}: number of output features

k: convolution kernel size

p: convolution padding size

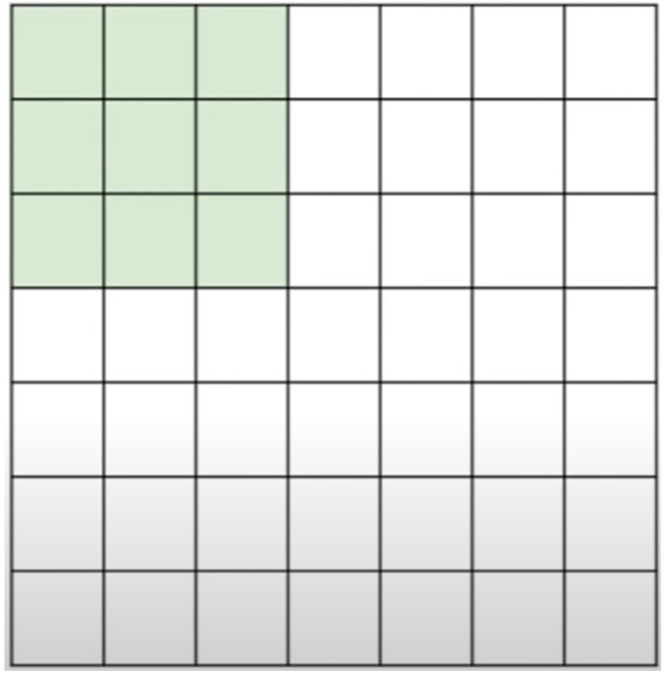
s: convolution stride size

VARIANTS OF THE CONVOLUTION OPERATION

1. Valid convolution — output size = input size - kernel size + 1
2. Same convolution — output size = input size
3. Full convolution — output size = input size + kernel size - 1
4. Strided convolution — kernel slides along the image with a step > 1

QUESTIONS?

WHAT'S THE OUTPUT SIZE OF THIS?

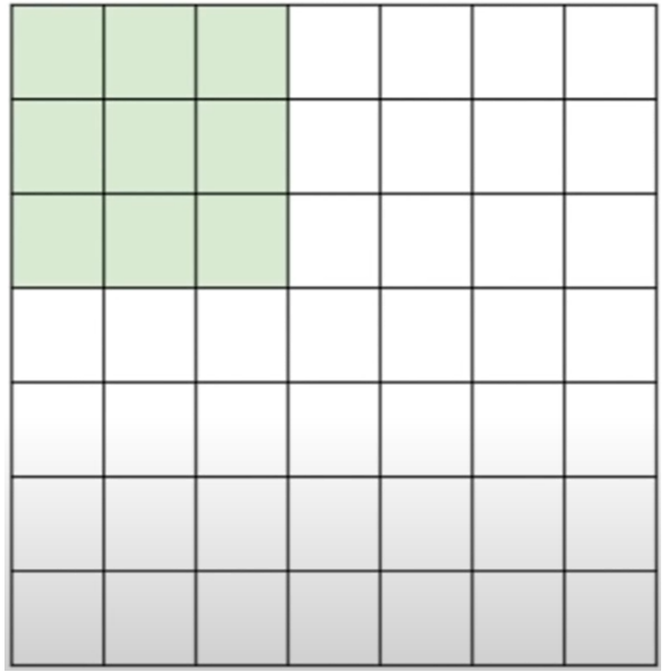


7

7

7 x 7 input, assume 3 x 3 filter applied with stride 3?

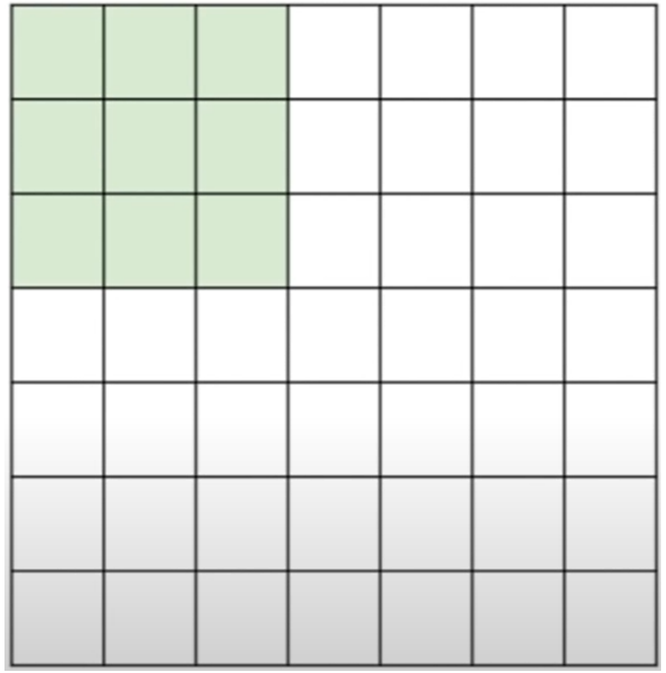
WHAT'S THE OUTPUT SIZE OF THIS?



7 x 7 input, assume 3 x 3 filter applied with stride 3?

Doesn't fit! (7 – 3) / 3 + 1 = 2.33
cannot apply 3 x 3 filter on
7 x 7 input with stride 3

WHAT'S THE OUTPUT SIZE OF THIS?

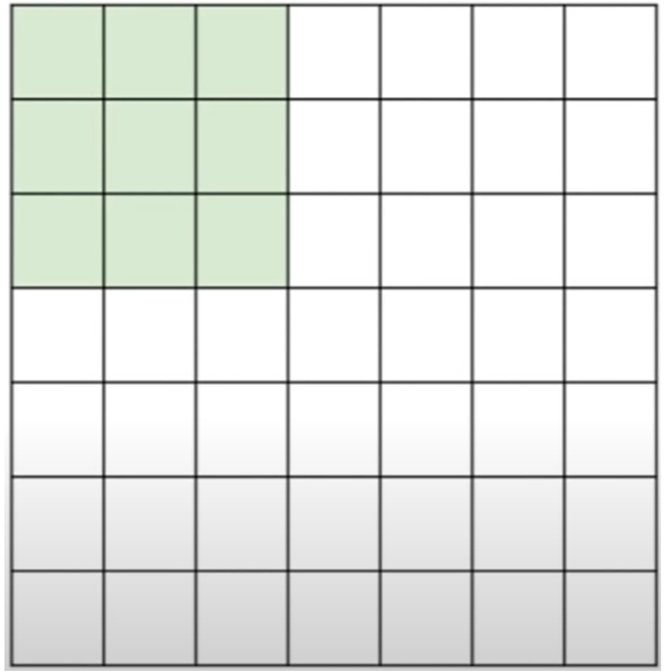


7

7

7 x 7 input, 3 x 3 filter applied with stride 2?

WHAT'S THE OUTPUT SIZE OF THIS?

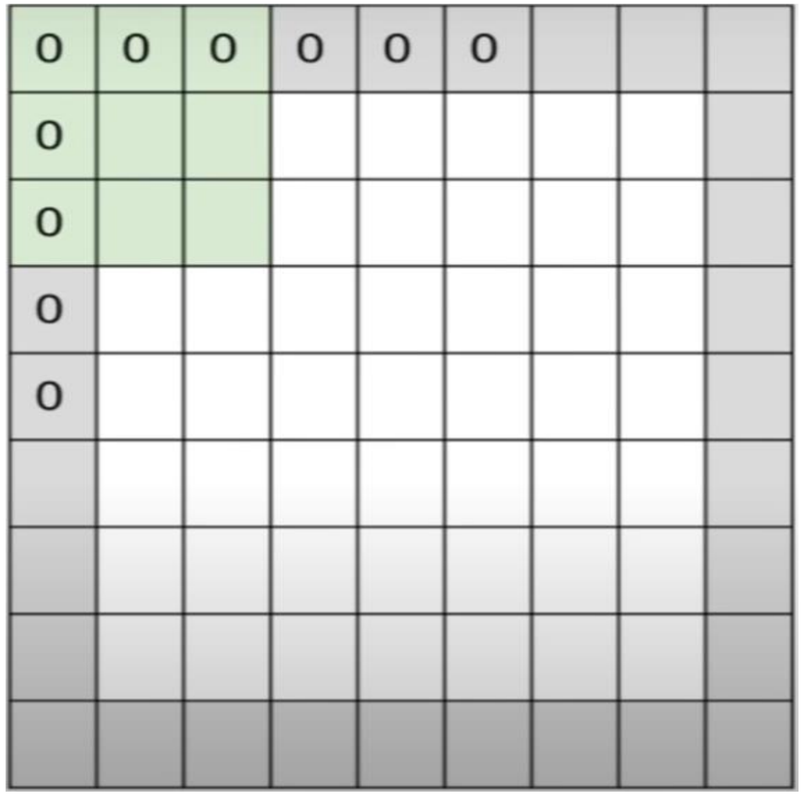


7 x 7 input, 3 x 3 filter applied with stride 2?

(7 – 3)/2 + 1 = 3

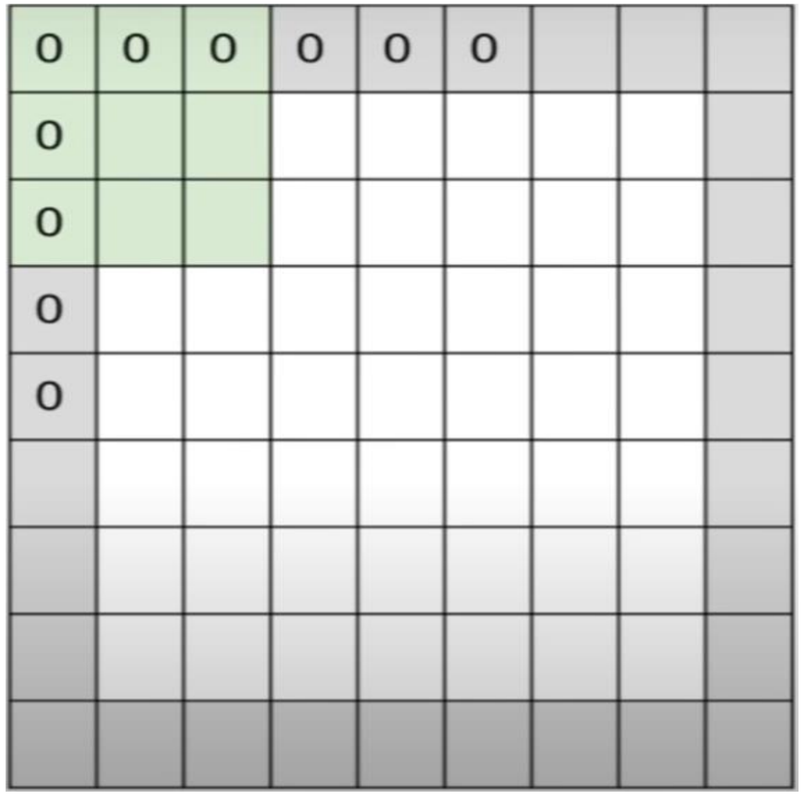
3 x 3

WHAT'S THE OUTPUT SIZE OF THIS?



Input 7 x 7:
3 x 3 filter, applied with stride 2,
pad with 1 pixel border?

WHAT'S THE OUTPUT SIZE OF THIS?



Input 7 x 7:
3 x 3 filter, applied with stride 2,
pad with 1 pixel border?

(7 + 2*1 – 3)/2 + 1 = 4

4 x 4

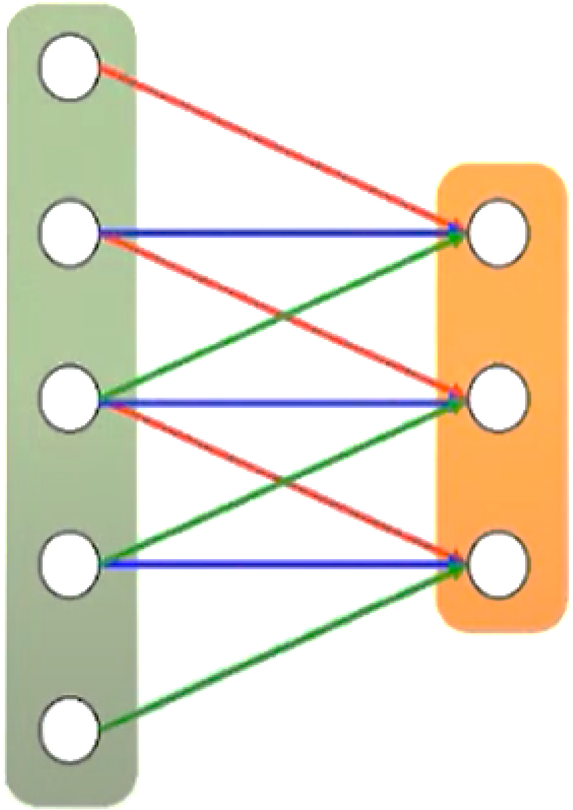
WHAT'S THE OUTPUT SIZE OF THIS?

1. 16 x 16 input, 4 x 4 filter applied with stride 4?
2. 24 x 24 input, 3 x 3 filter applied with stride 3?
3. 32 x 32 x 3 input, 20 5 x 5 filters applied with stride 1, pad 2?

WHAT'S THE OUTPUT SIZE OF THIS?

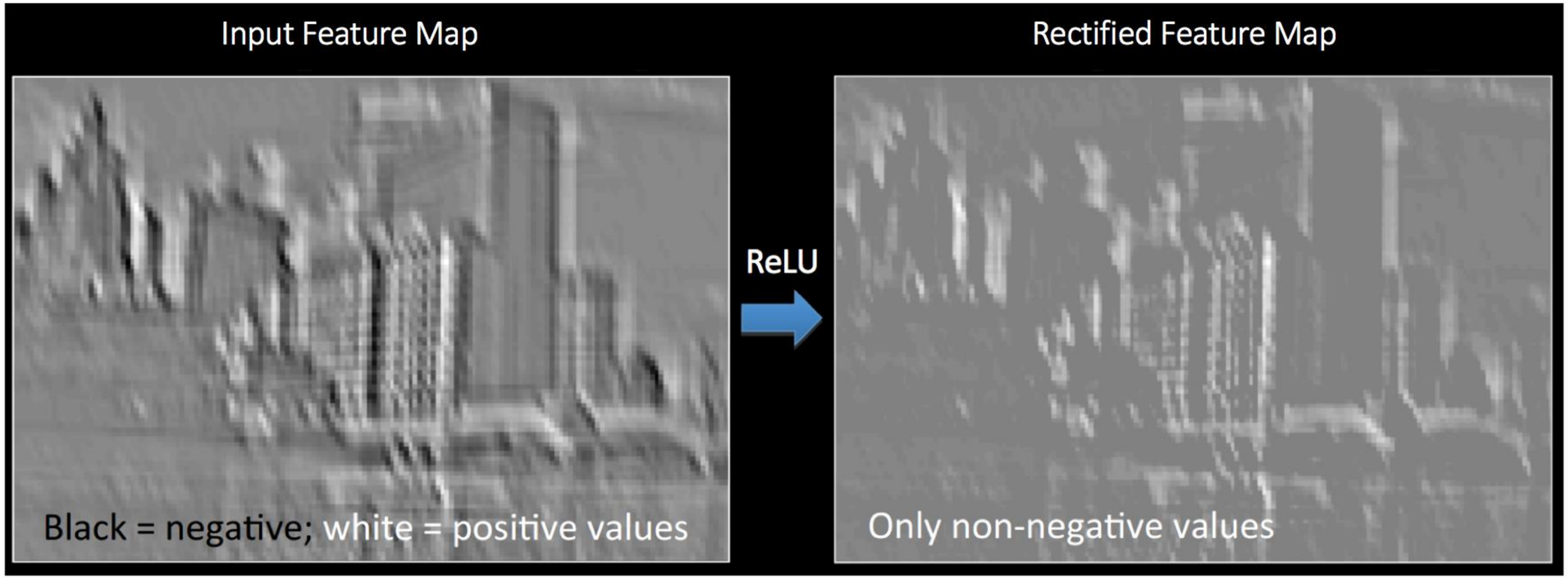
1. 16 x 16 input, 4 x 4 filter applied with stride 4?
   (16-4)/4+1 = 4 (4 x 4)
2. 24 x 24 input, 3 x 3 filter applied with stride 3?
   (24-3)/3+1 = 8 (8 x 8)
3. 32 x 32 x 3 input, twenty 5 x 5 filters applied with stride 1, pad 2?
   (32+2*2-5)/1+1 = 32 (32 x 32 x 20)

CONVOLUTION FILTER



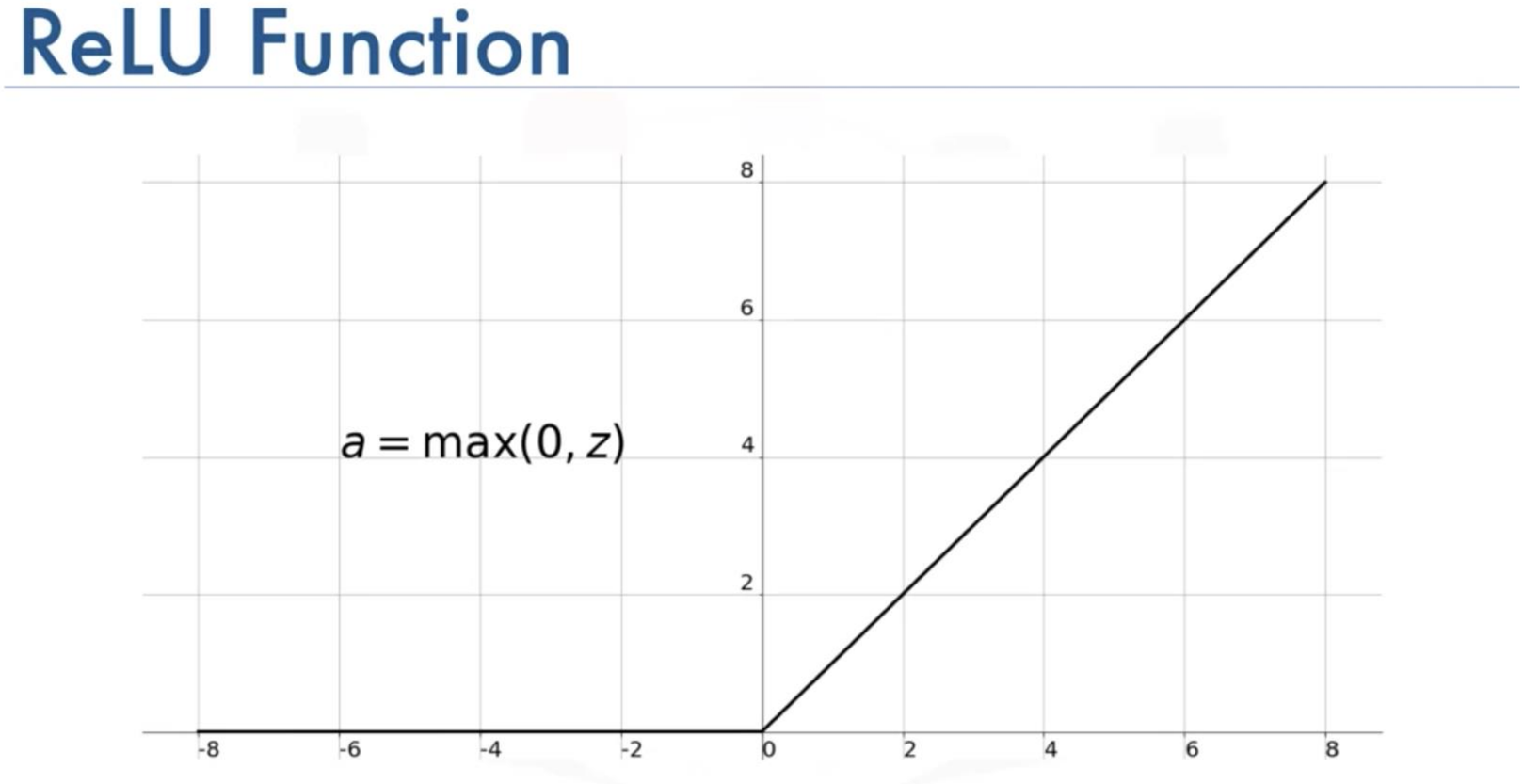
Convolution is linear operator. A special case of Fully Connected Network (the filter size is equal to the size of the input)

NON LINEARITY



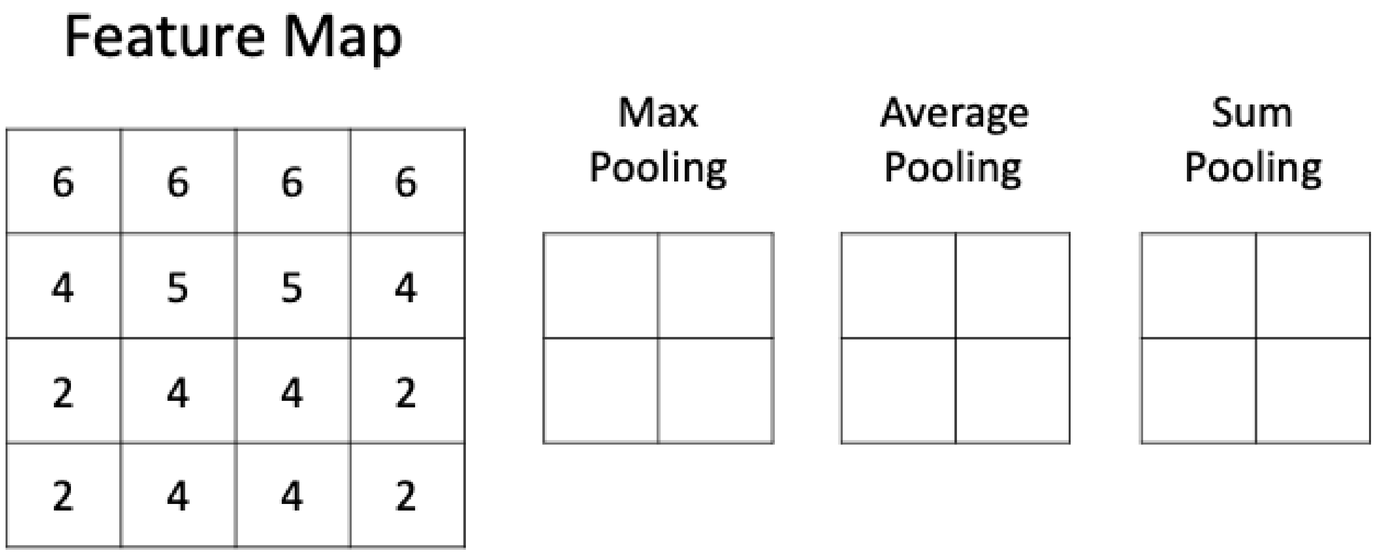
[Source: Rob Fergus, 2013]

NON LINEARITY



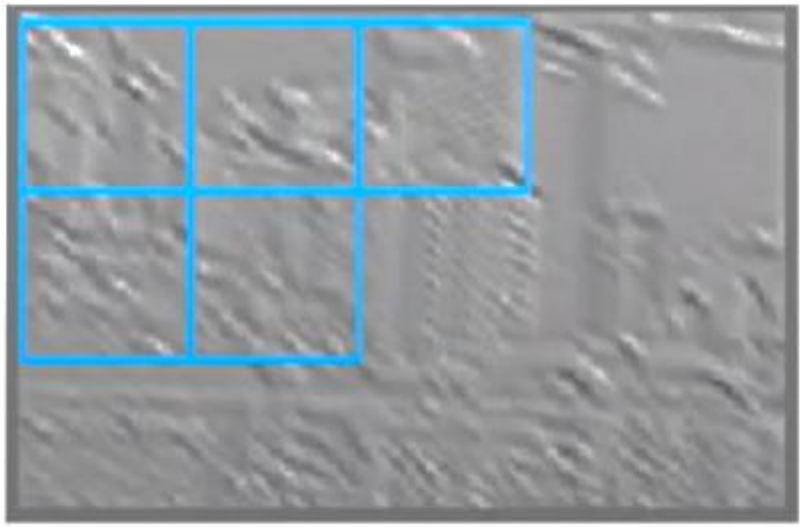
Other non linear functions such as **tanh** or **sigmoid** can also be used instead of ReLU, but ReLU has been found to perform better in most situations.

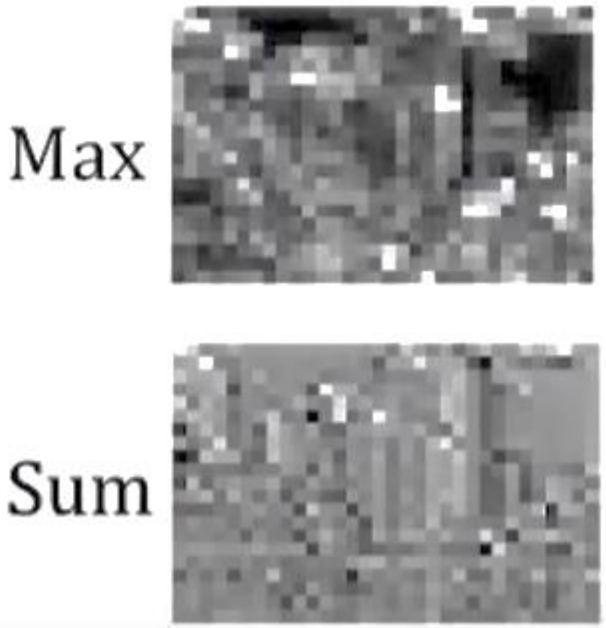
POOLING



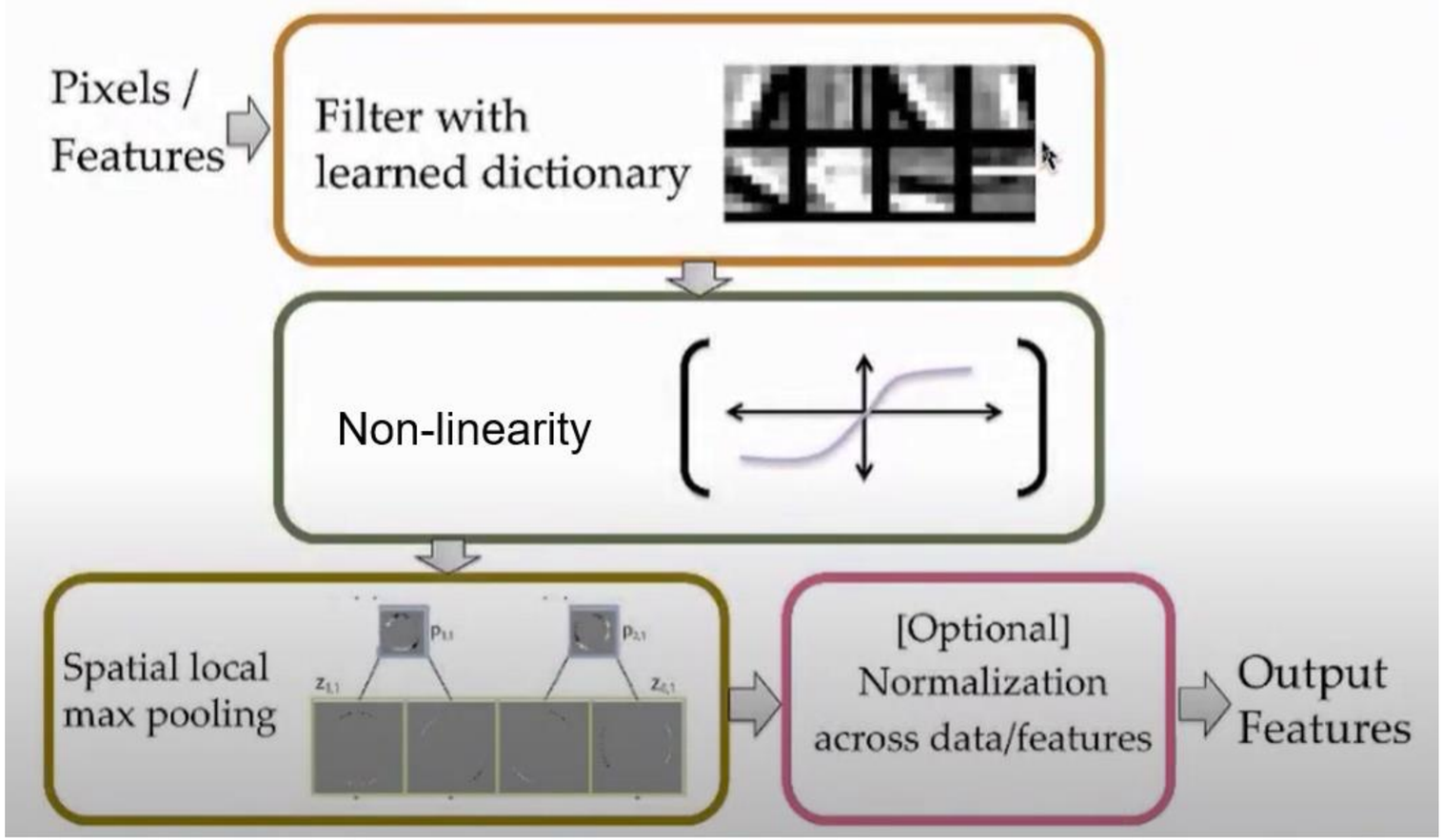
- Spatial Pooling
  - Non-overlapping / overlapping regions
  - Sum / Max / Avg
  - Boureau et al. ICML'10 for theoretical analysis

POOLING



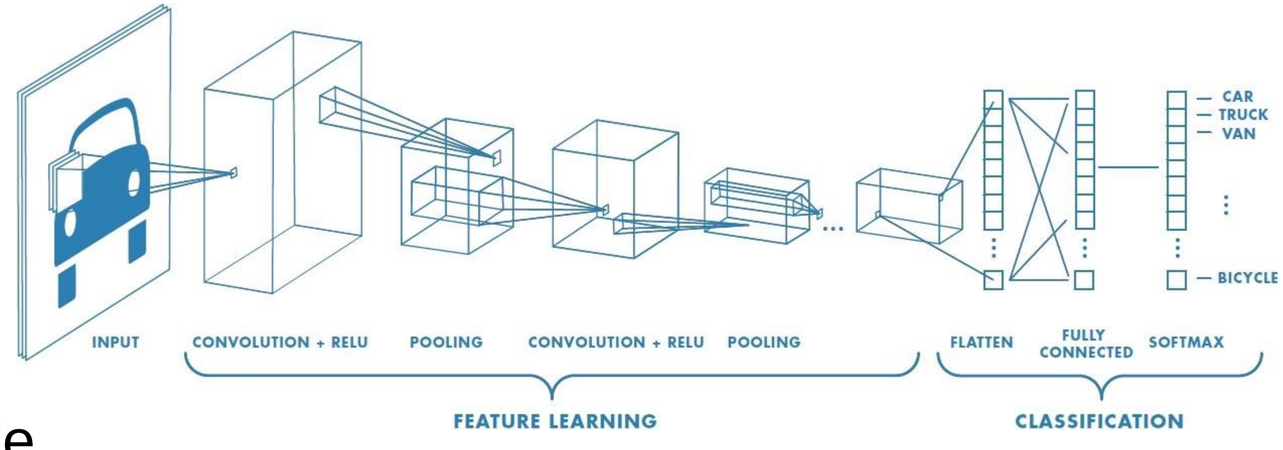


CONVOLUTION LAYER



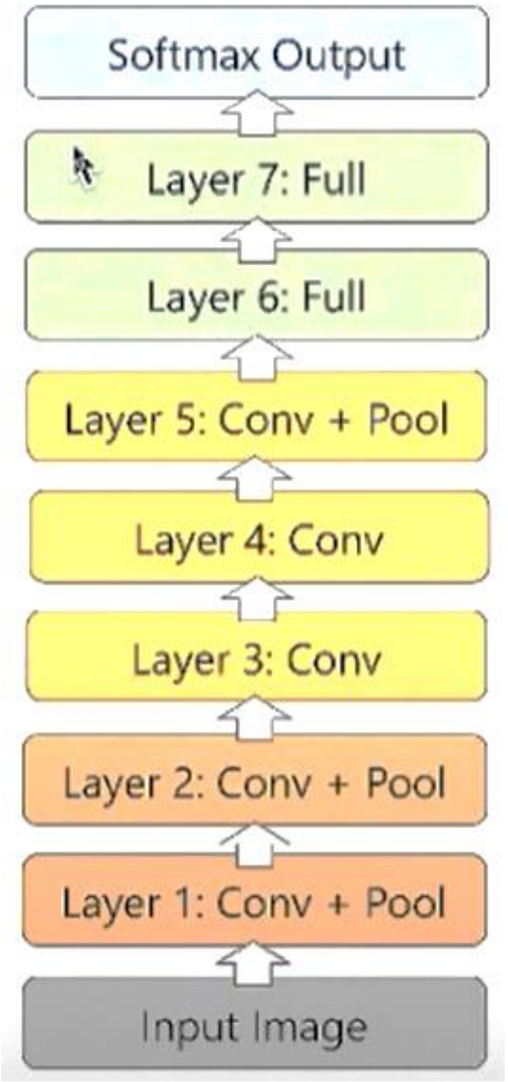
CONVOLUTION ARCHITECTURE (HYPERPARAMETERS)

- Convolution filter
  - Kernel size, Stride, Filters
- Non-linearity
  - Sigmoid, ReLU, Tanh
- Pooling
  - Max or Avg / Mean, Size, Stride
- Number of layers
- Validation

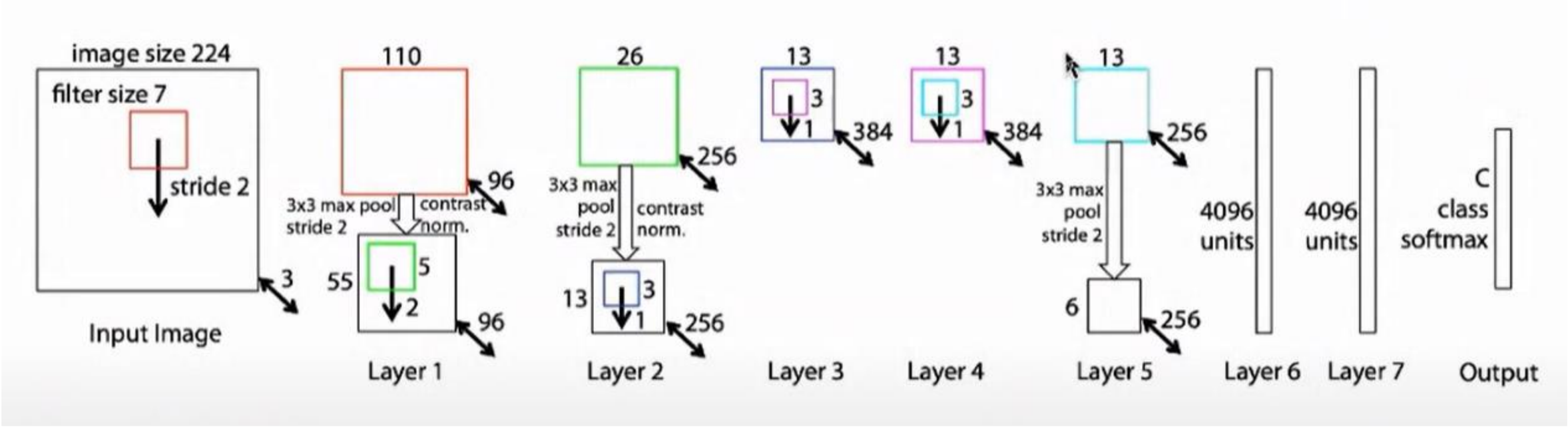


Architecture of Krizhevsky et al.

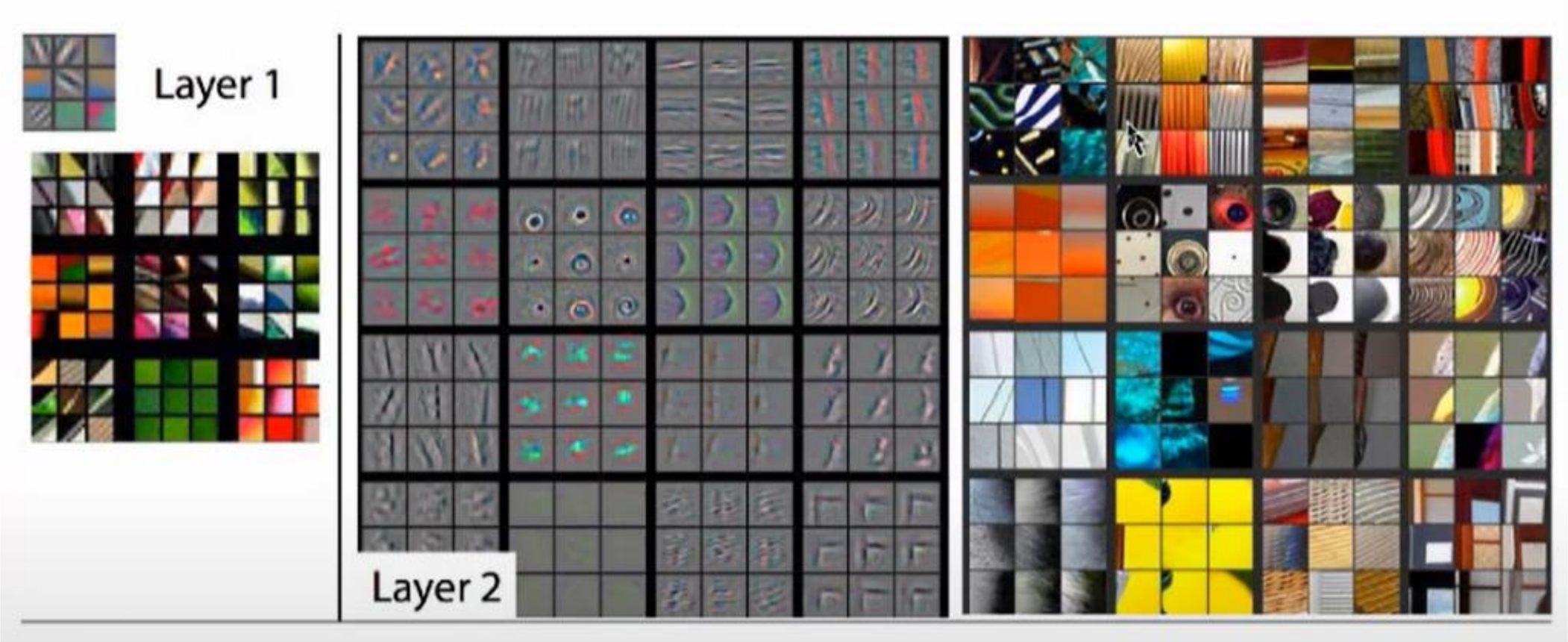
- 8 layers
- ImageNet data
- Top-5 error rate: 16.4%



Architecture of Krizhevsky et al.

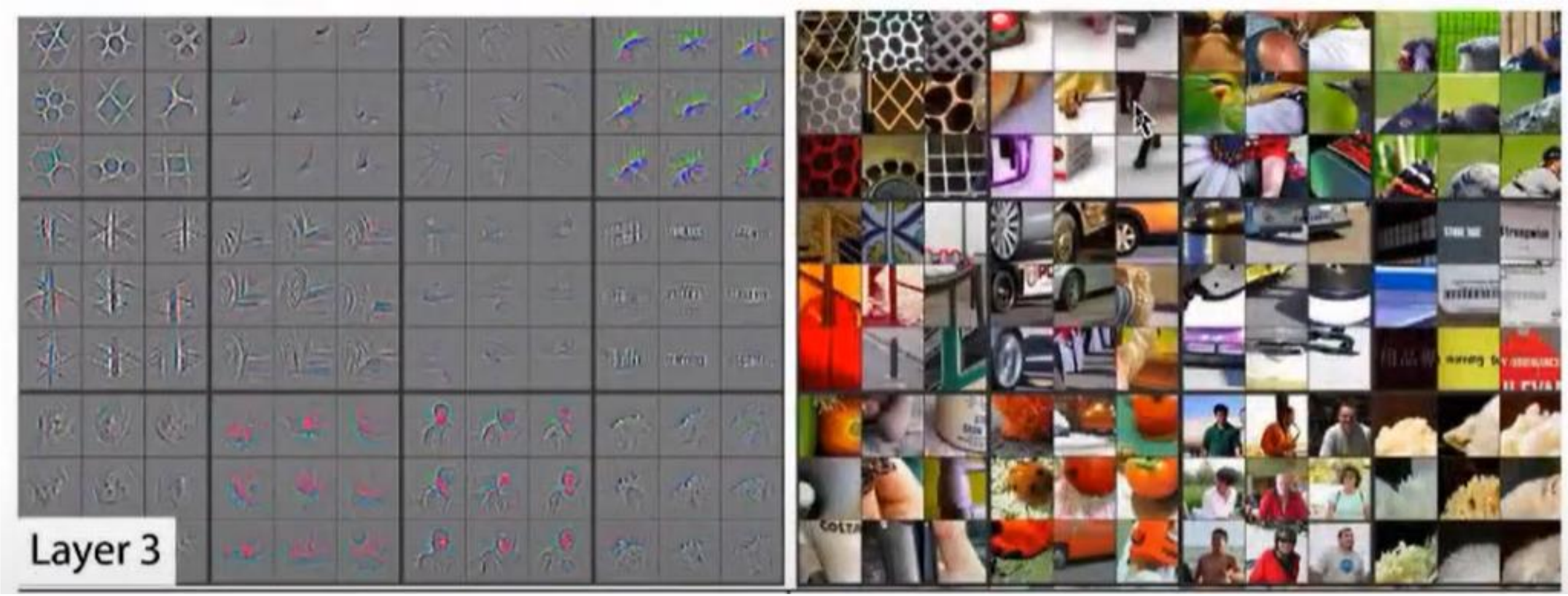


WHAT DO LAYERS LEARN?



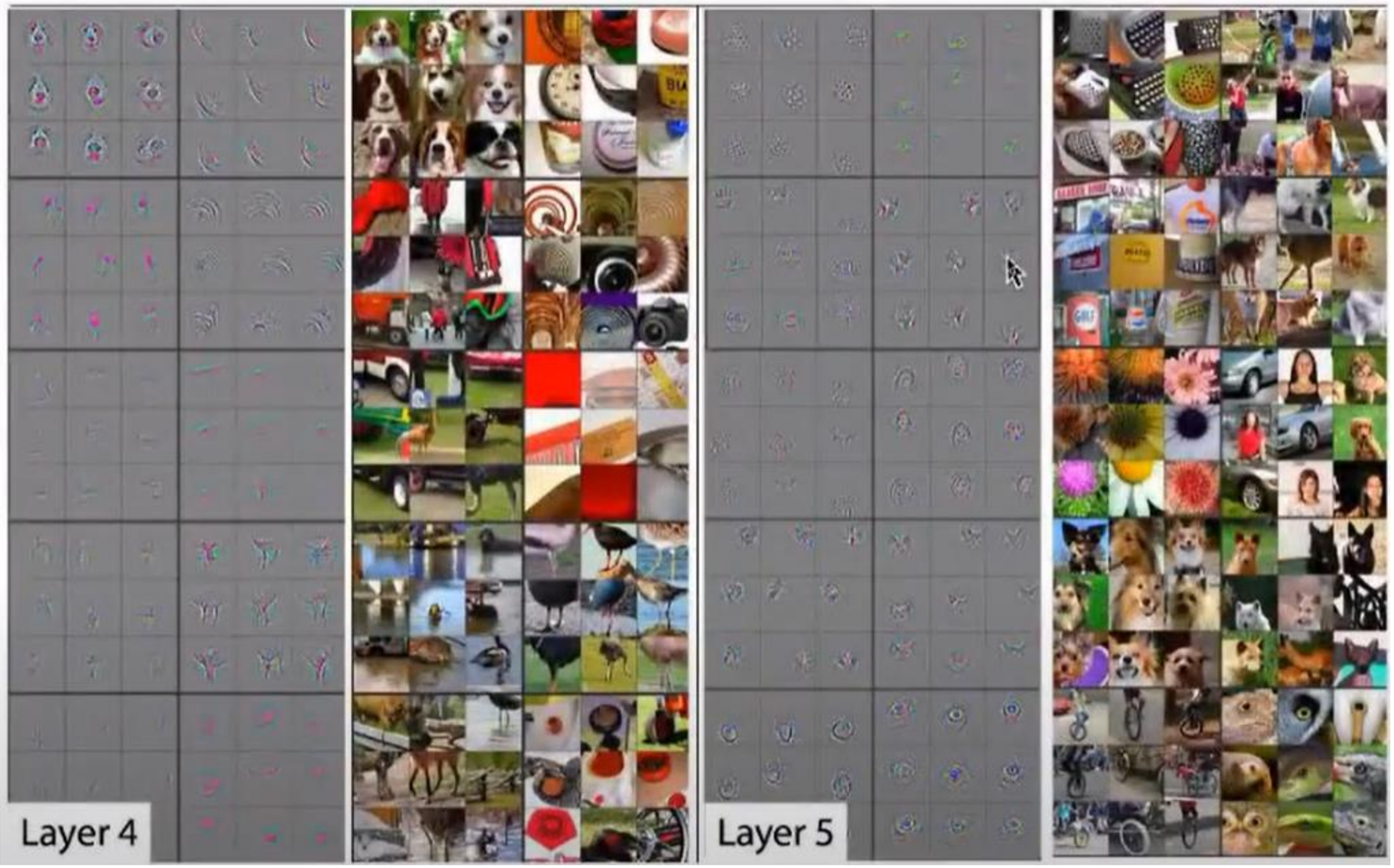
[Source: Zeiler & Fergus, 2013]

WHAT DO LAYERS LEARN?



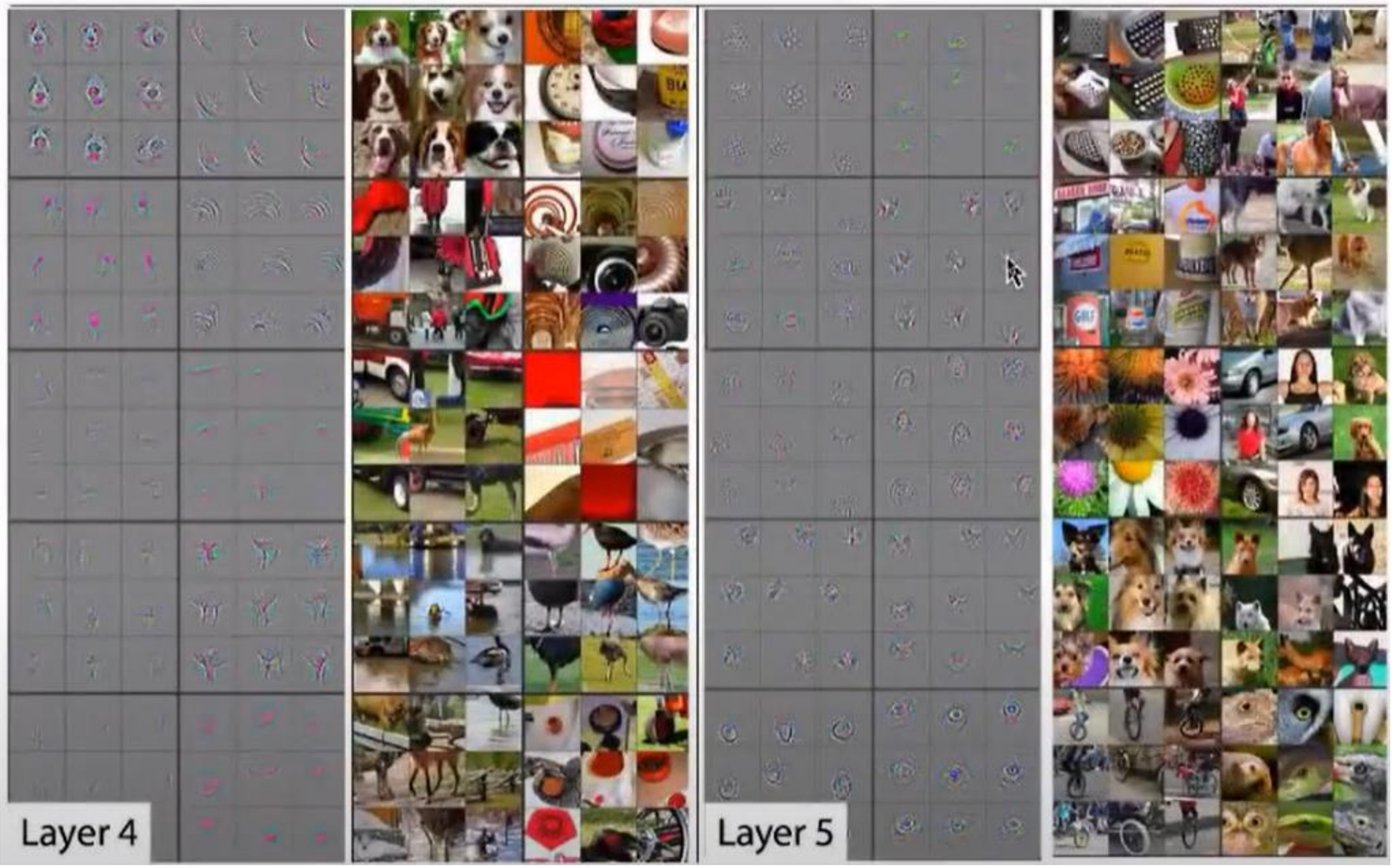
[Source: Zeiler & Fergus, 2013]

WHAT DO LAYERS LEARN?



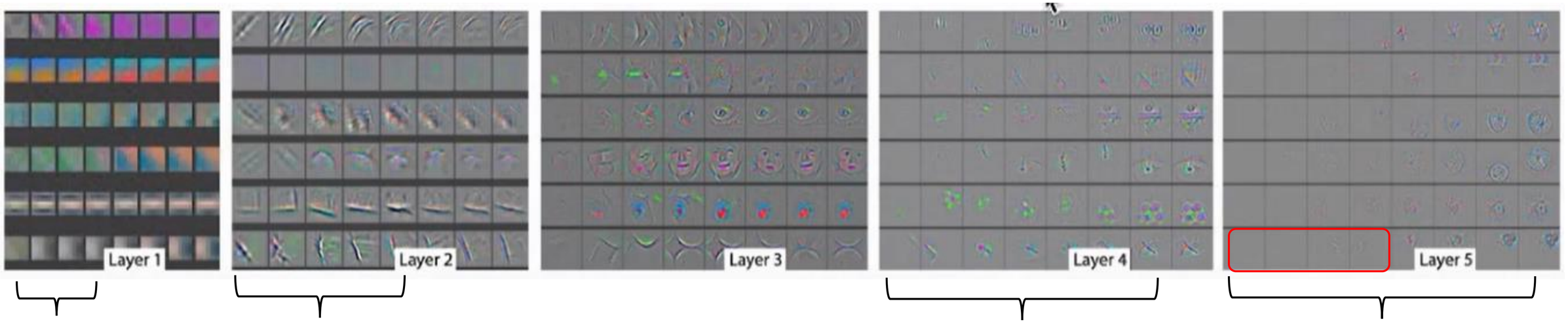
[Source: Zeiler & Fergus, 2013]

WHAT DO LAYERS LEARN?



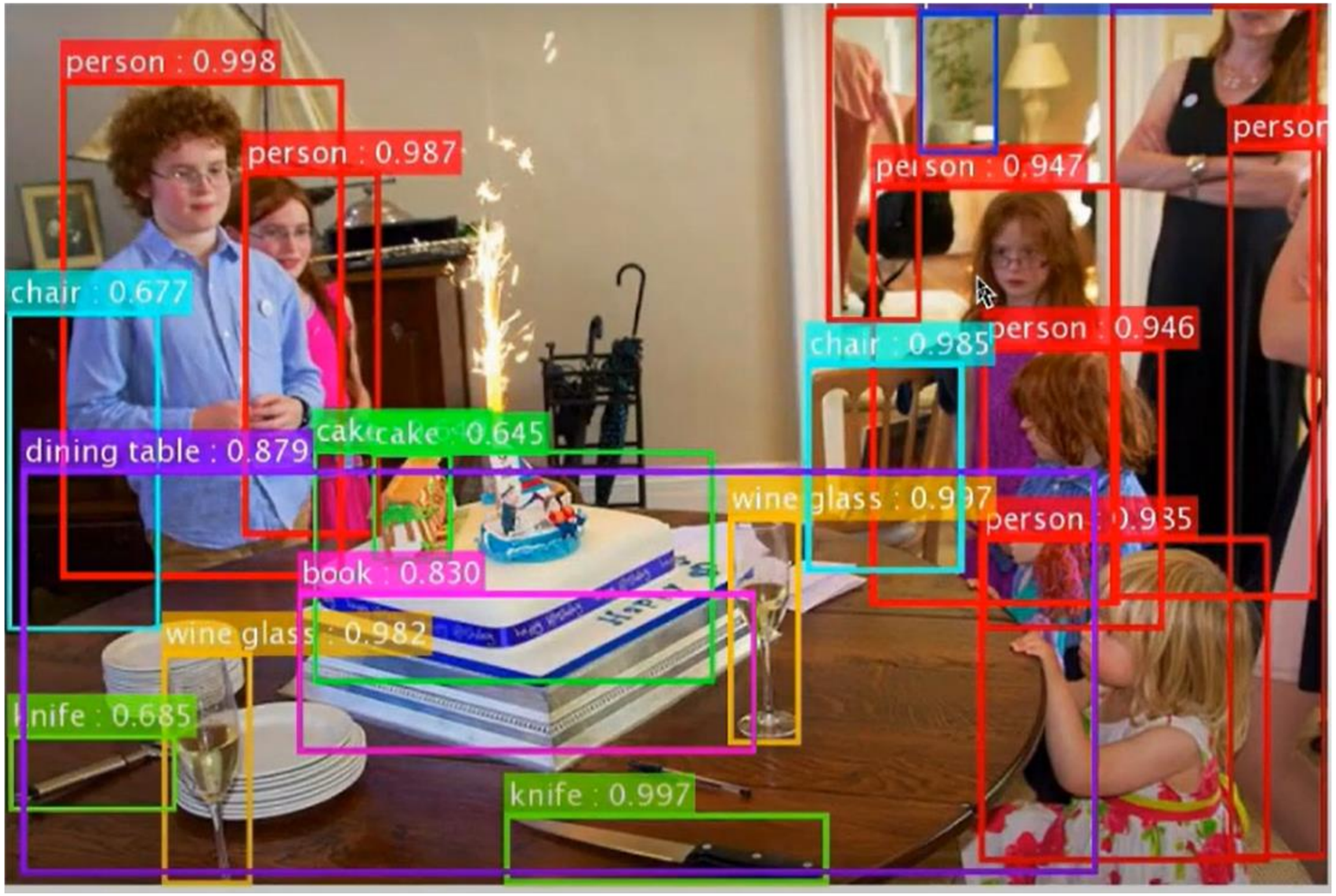
[Source: Zeiler & Fergus, 2014]

HOW FAST DO LAYERS LEARN?



[Source: Zeiler & Fergus, 2013]

IMAGE RECOGNITION



[Source: He, Zhang, Ren, & Sun, "Deep Residual Learning for Image Recognition". ICCV 2015]

IMAGE SEGMENTATION

[Source: He et al, “Mask R-CNN”. 2017]

GENERATIVE ADVERSARIAL NETWORKS

[Source: Karras et al, “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. 2017]