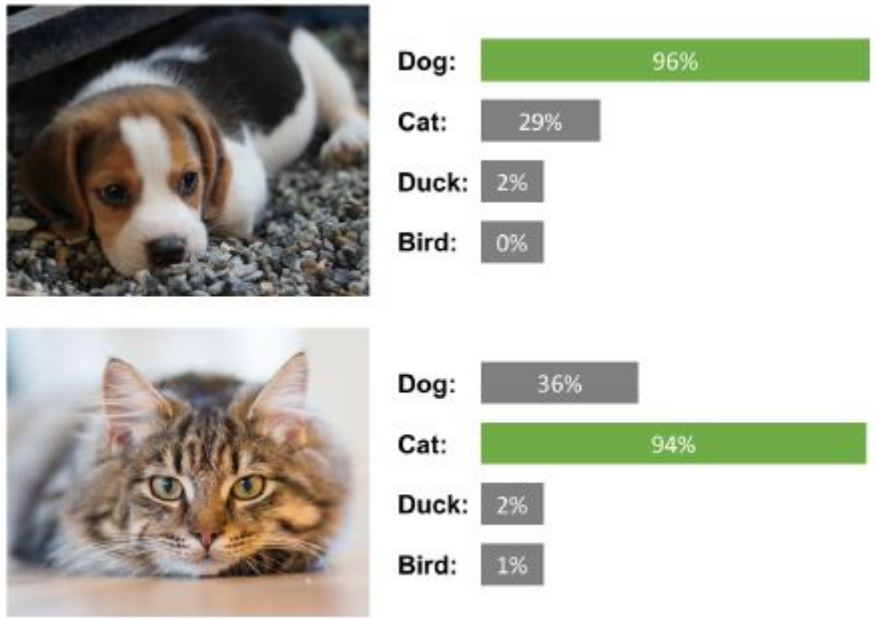
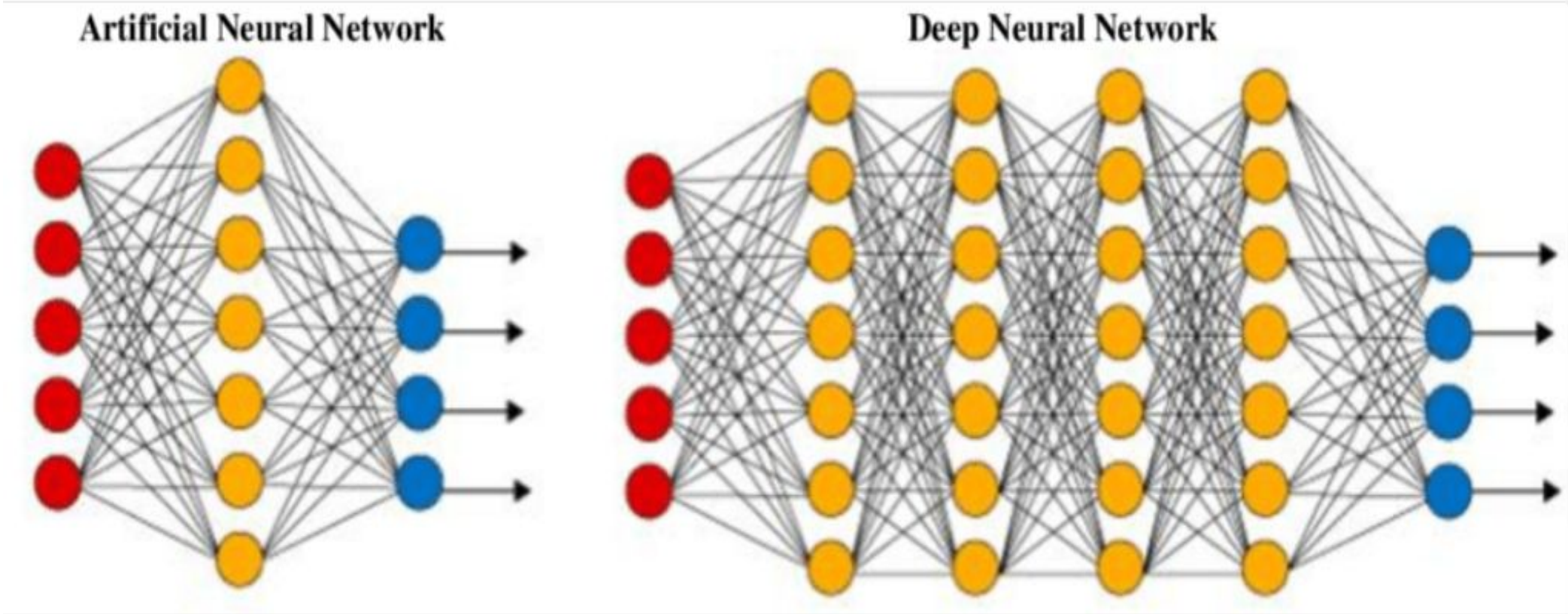
Deep Learning for Computer Vision
Classification

- Image classification is the task of labelling the whole image with an object or concept with confidence.
- The applications include
  - gender classification given an image of a person's face,
  - identifying the type of pet,
  - tagging photos,
  - and so on.



Deep Neural Network

- A Deep Neural Network (DNN) is a type of artificial neural network (ANN) with multiple layers between the input and output layers.
- These layers, often called hidden layers, allow the network to model complex and abstract features from the data, making DNNs particularly powerful for tasks like image and speech recognition, natural language processing, and more.
- Key Concepts of Deep Neural Networks Neurons (Nodes): The basic units of a neural network. Each neuron receives input, processes it (often applying a non-linear function), and passes the output to the next layer.



ANN vs DNN - Image Source

Layers

- Input Layer: The first layer of the network that receives the input data.
- Hidden Layers: Intermediate layers between the input and output layers. These layers perform feature extraction and transformation.
- Output Layer: The final layer that produces the output of the network.
- Weights and Biases: Parameters that the network learns during training. Weights determine the importance of each input to a neuron, while biases allow the model to fit the data better by providing a degree of freedom.

- Activation Functions: Functions applied to the output of each neuron to introduce non-linearity into the model, allowing it to learn complex patterns.
- Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh.
- Forward Propagation: The process of passing the input data through the network, layer by layer, to obtain the output.
- Loss Function: A function that measures the difference between the predicted output and the actual output (ground truth). Common loss functions include Mean Squared Error (MSE) for regression and Cross-Entropy Loss for classification.
- Backpropagation: The process of updating the weights and biases in the network by calculating the gradient of the loss function with respect to each parameter and adjusting them to minimize the loss.
- Optimization Algorithms: Methods used to update the network's parameters during training. Common optimization algorithms include Stochastic Gradient Descent (SGD), Adam, and RMSprop.

TensorFlow

- **TensorFlow 1.x**: You need to build a computational graph and then execute it within a session.
- **TensorFlow 2.x**: Eager execution is enabled by default, so operations are executed immediately as they are defined, making the code more intuitive and easier to work with.
- By moving to TensorFlow 2.x and using eager execution, it is able to write more straightforward and Pythonic code, which is easier to read, debug, and maintain

Computational Graph in TensorFlow

- A computational graph is a representation of the computations you want to perform. In the context of TensorFlow, it's a directed graph where:
- **Nodes** represent operations (like addition, multiplication, etc.) or variables (like constants or placeholders).
- **Edges** represent the data (tensors) flowing between these operations.

- In TensorFlow, when you define operations and variables, you're essentially building this computational graph. This graph doesn't perform the computations immediately. Instead, it defines the series of steps that TensorFlow will execute once you run the graph in a session.
- Let's break down a simple example to illustrate this concept:

1. **Define the Graph**: Create tensors and operations.
2. **Execute the Graph**: Run the defined graph in a session to perform the actual computations

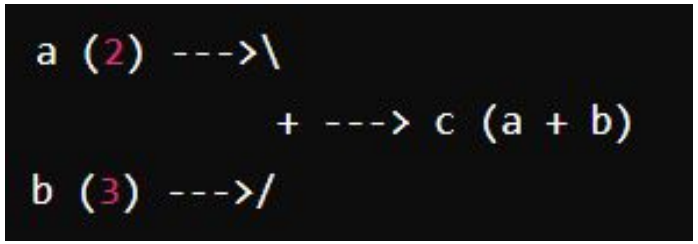
TensorFlow 1.x (Graph Execution):

```
import tensorflow as tf

# Build the computational graph
a = tf.constant(2)
b = tf.constant(3)
c = tf.add(a, b)

# Create a session to run the graph
with tf.compat.v1.Session() as sess:
    result = sess.run(c)
    print(result)  # Output: 5
```

- In this example:
- Nodes: a, b, and c are nodes in the graph. a and b are constant tensors, and c is an addition operation.
- Edges: The values of a and b flow into the addition operation c.



Execution Process

- Building the Graph: When you define a, b, and c, TensorFlow creates a computational graph with these nodes and edges.
- Running the Graph: When you create a session and run sess.run(c), TensorFlow executes the operations defined in the graph, computes the result, and returns it

TensorFlow 2.x (Eager Execution):

```
import tensorflow as tf

# Eager execution is enabled by default in TensorFlow 2.x

# Define and immediately execute the operations
a = tf.constant(2)
b = tf.constant(3)
c = a + b

print(c)  # Output: 5
```

Constants (tf.constant)

- Constants (tf.constant): These are immutable tensors whose values are fixed once they are set. They cannot be changed during execution.
- **Usage**: They are used when you have known values that should not change during the execution of the graph.

```
import tensorflow as tf

# Creating a constant tensor
a = tf.constant(5)
b = tf.constant(3)

# Performing operations
c = tf.add(a, b)

# Running in a session (TensorFlow 1.x)
with tf.compat.v1.Session() as sess:
    result = sess.run(c)
    print(result)  # Output: 8
```

Placeholders (tf.placeholder)

- Placeholders (tf.placeholder): These are tensors that act as nodes in the computational graph where data will be fed in at runtime. They are used to feed external data into the graph.
- **Usage**: They are used when you need to input external data into the graph, such as feeding data during training or inference.

```
import tensorflow as tf

# Defining placeholders
x = tf.compat.v1.placeholder(tf.float32, shape=[None, 3])
y = tf.compat.v1.placeholder(tf.float32, shape=[None, 3])

# Performing operations
z = tf.add(x, y)

# Running in a session (TensorFlow 1.x)
with tf.compat.v1.Session() as sess:
# Feeding data into the placeholders
    feed_dict = {x: [[1, 2, 3]], y: [[4, 5, 6]]}
    result = sess.run(z, feed_dict=feed_dict)
    print(result)  # Output: [[5. 7. 9.]]
```

Constants (tf.constant):

- Immutable tensors with fixed values.
- Values are set when the tensor is created.
- Useful for known, unchanging values.

Placeholders (tf.placeholder):

- Nodes that must be fed with data at runtime.
- Do not hold data themselves; used to specify the shape and type of input data.
- Useful for feeding external data into the graph during execution.

Keras

- Keras is a high-level neural networks API, written in Python and capable of running on top of several deep learning frameworks, including TensorFlow, Microsoft Cognitive Toolkit (CNTK), and Theano.
- It was developed to enable fast experimentation and to provide a simple, user-friendly way to build and train deep learning models.

Key Features of Keras:

- User-Friendly: Keras provides a simple, consistent interface optimized for common use cases, which allows for easy and fast prototyping.
- Modular: A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be connected together.
- Extensible: New modules can be easily added, and it allows for total customization in the building process.
- Multi-Backend and Multi-Platform: It can run seamlessly on CPU and GPU and can be deployed on various platforms, such as mobile devices using TensorFlow Lite.

KERAS VS TENSORFLOW

Keras	TensorFlow
A tool to *build models easily*	A big library for *machine learning and deep learning*
Originally independent, now part of TensorFlow	Created by Google
Runs *inside TensorFlow* as tf.keras	Keras is included in TensorFlow
Slower training in standalone Keras; improved in tf.keras	Faster due to Graph execution
Uses eager execution in tf.keras (from TF 2.x)	Supports eager and graph execution

Models:

- Sequential API: A linear stack of layers. Best for straightforward, layer-by-layer models. It's easy to use and sufficient for many common use cases
- .Functional API: Required for complex models with multiple inputs/outputs, shared layers, and non-linear data flows. It provides the flexibility needed for advanced architectures.

Introduction to OpenCV

- OpenCV, - Open Source Computer Vision Library,
- OpenCV is a huge open-source library for computer vision, machine learning, and image processing.
- Originally developed by Intel, it is now maintained by a community of developers under the OpenCV Foundation.
- OpenCV supports a wide variety of programming languages like Python, C++, Java, etc.
- It can process images and videos to identify objects, faces, or even the handwriting of a human.

Features of OpenCV:

- Feature Detection and Description: Functions for detecting and describing key points in images, such as SIFT(Scale-Invariant Feature Transform), SURF(Speeded-Up Robust Features), and ORB(Oriented FAST and Rotated BRIEF).
- Object Tracking: Algorithms for tracking objects in video streams, such as
  - Meanshift - shifts a window (or region of interest) toward the maximum density of features (e.g., color) in a given area
  - Camshift- extension of MeanShift that adapts to changes in object size and rotation
- 3D Reconstruction: Tools for reconstructing 3D structures from 2D images.
- Camera Calibration: Functions for calibrating camera parameters to remove distortion.
- Deep Learning: Integration with deep learning frameworks like TensorFlow, Caffe, and PyTorch for tasks like object detection and recognition.

Assignment

- perform basic image processing tasks using OpenCV such as reading an image, extracting pixel RGB values, defining a Region of Interest (ROI), resizing, rotating the image, drawing a rectangle, and displaying text on the image

SqueezeNet Model

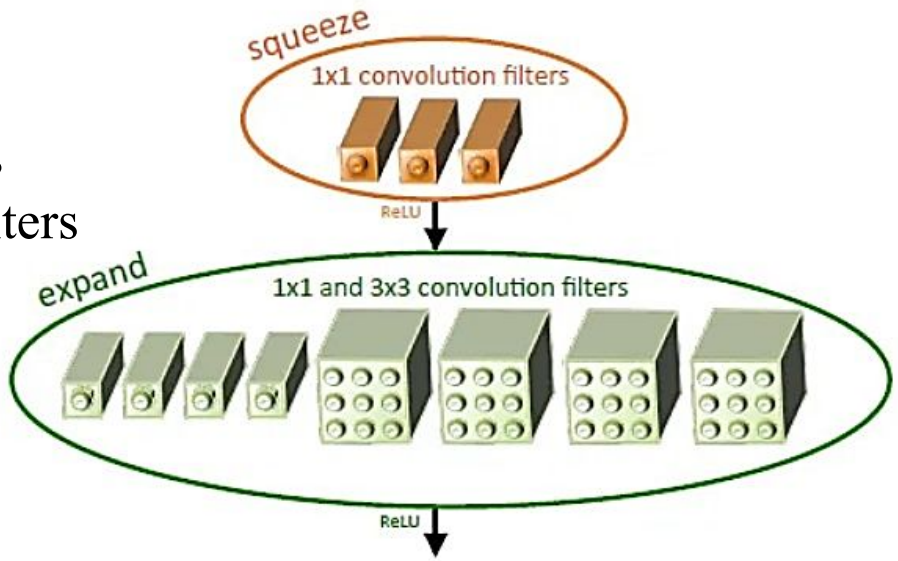
- Squeezenet is a CNN architecture which has 50 times less parameters than AlexNet but still maintains AlexNet level accuracy
- SqueezeNet model uses certain strategies to trim the majority of parameters
  - Replacing the 3x3 filters with 1x1 filters
    - Thus the model has 9 times fewer parameters than a traditional filter.
  - Decreasing the number of input channels to 3x3 filters
    - As the filter size is reduced to 1x1, the number of input channels also needs to be reduced. This is done using the squeeze layer
  - Downsampling later at the network
    - By downsampling later at the network, we are able to get larger activation maps for the convolution layers which leads to better accuracy.

SqueezeNet Model-Fire Module

- A Fire module is the basic building block of SqueezeNet, designed to reduce the number of parameters while keeping good accuracy.
- The Fire module has two main parts:

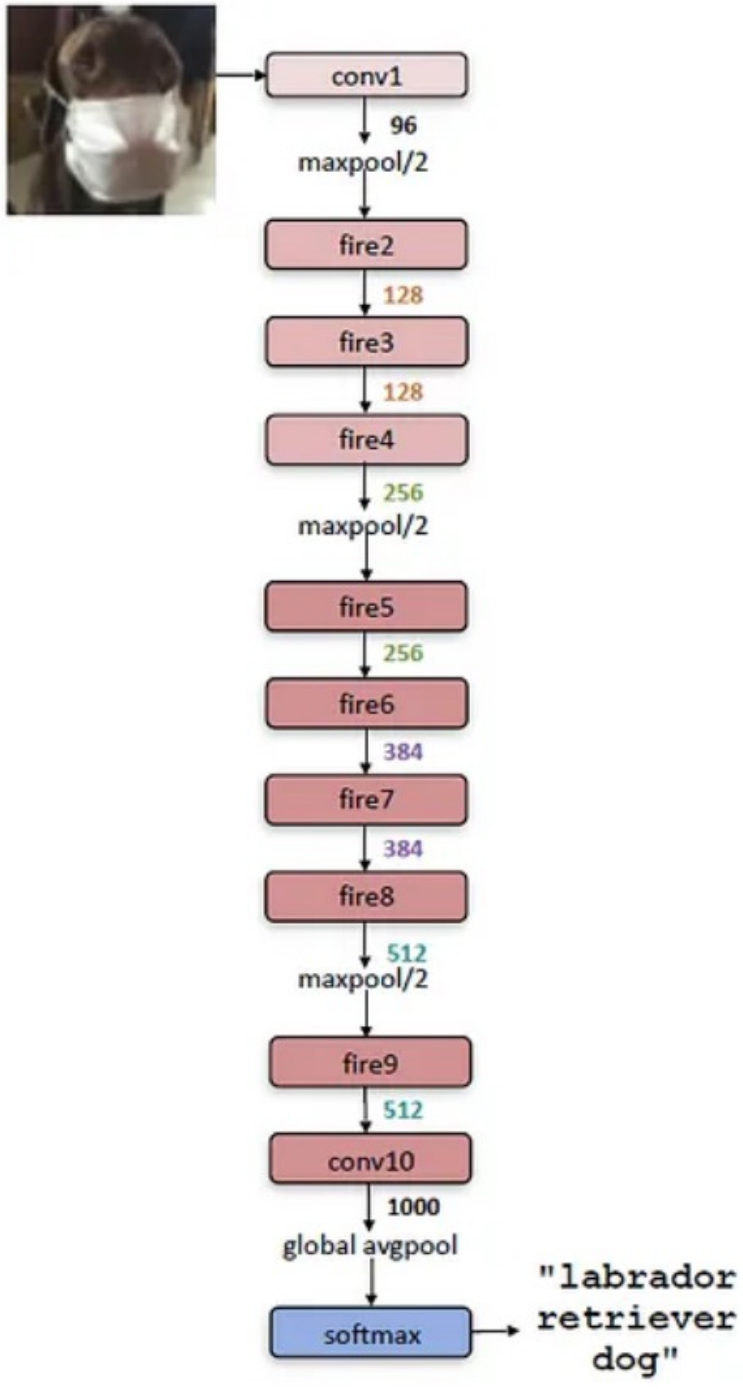
1. Squeeze Layer:
    - Uses only 1×1 convolution filters
    - Goal: Shrink (or "squeeze") the number of input channels
    - Controlled by hyperparameter s1x1 → Number of 1×1 filters

2. Expand Layer
    - Expands the data again using:
        - 1×1 convolution filters → Count: e1x1
        - 3×3 convolution filters → Count: e3x3
    - Output of both 1×1 and 3×3 filters is concatenated



Fire Module with hyperparameters: s1×1 = 3, e1×1 = 4, and e3×3 = 4

SqueezeNet Model



Input Image:

The network takes an input image, which in this case is of a dog.

conv1:

- This is the first convolutional layer.
- It uses 96 filters (as indicated by the number 96) to process the input image.
- The size of these filters is typically 7x7 (common in SqueezeNet), but the size is not specified in the diagram.

maxpool/2:

- This is a max-pooling layer with a stride of 2, which means it reduces the spatial dimensions of the feature maps by half.
- Pooling helps to reduce the spatial size and computational complexity, and it also helps to make the detection of features invariant to small translations.

fire2 to fire9:

- These are the "fire modules" which are the core components of SqueezeNet. Each fire module consists of:
  - A "squeeze" layer that has 1x1 convolutions.
  - An "expand" layer that has both 1x1 and 3x3 convolutions.
- The numbers next to each fire module (e.g., 128, 256, 384, 512) indicate the number of filters in the expand layers of those modules.

maxpool/2 (after fire4 and fire8):

- Additional max-pooling layers are placed after fire4 and fire8 modules.
- These layers again reduce the spatial dimensions of the feature maps by half, further downsampling the data while keeping the number of parameters low.

conv10:

- This is another convolutional layer with 1000 filters.
- This layer reduces the spatial dimensions to 1x1 for each of the 1000 classes, which prepares the data for the final classification step.

global avgpool:

- Global average pooling is used to reduce each feature map to a single value by taking the average of all the values in the feature map.
- This helps in converting the feature maps into a 1x1x1000 tensor, which matches the number of classes (1000 classes in ImageNet).

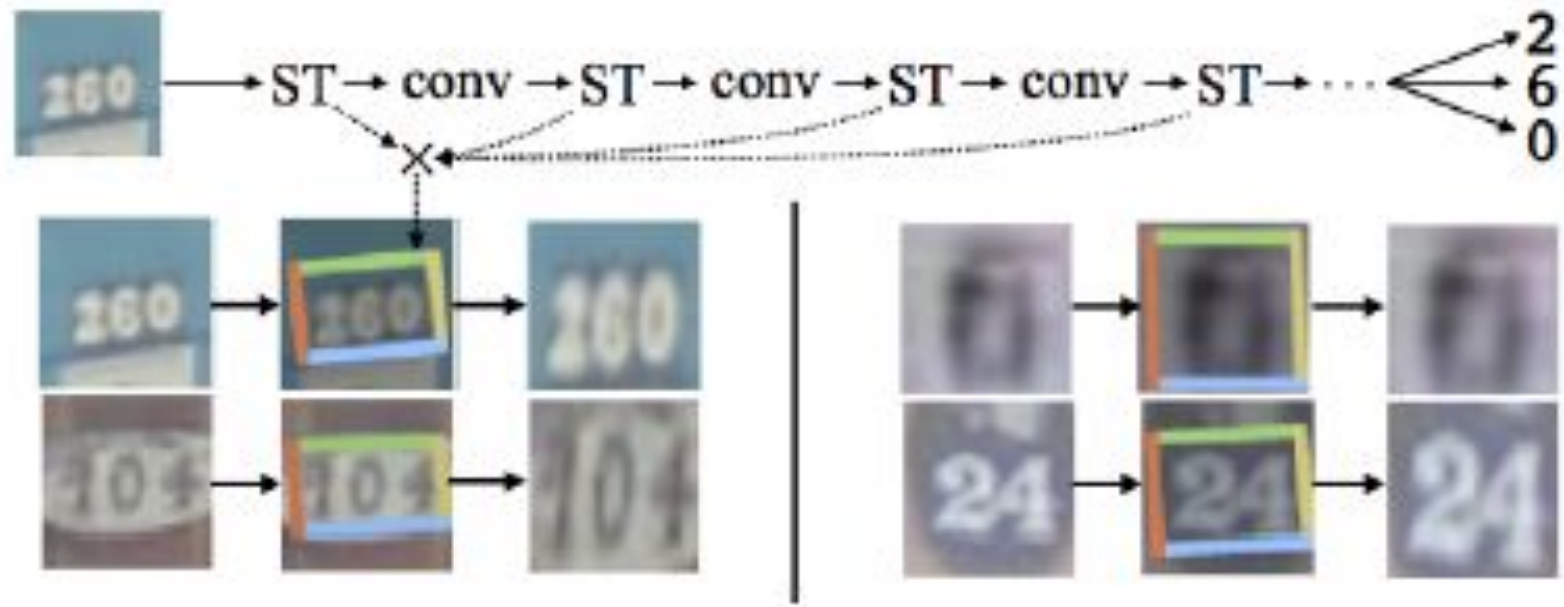
softmax:

- The final layer is a softmax layer.
- It converts the 1x1x1000 output into a probability distribution over the 1000 classes.
- The class with the highest probability is the network’s prediction, which in this case is identified as "labrador retriever dog."

Spatial Transformer Networks (STNs)

Spatial Transformer Networks (STNs)

- STNs take a different approach by learning a transformation of the input data before it is processed by the convolutional layers.
- This transformation is learned during training and can include affine transformations such as scaling, rotation, and translation.
- The parameters are learned for an **affine transformation.**
- By applying an affine transformation, spatial invariance is achieved **(model learns to recognize objects regardless of their position, size, or orientation).**


ST
conv
ST
conv
ST
conv
ST
2
6
0