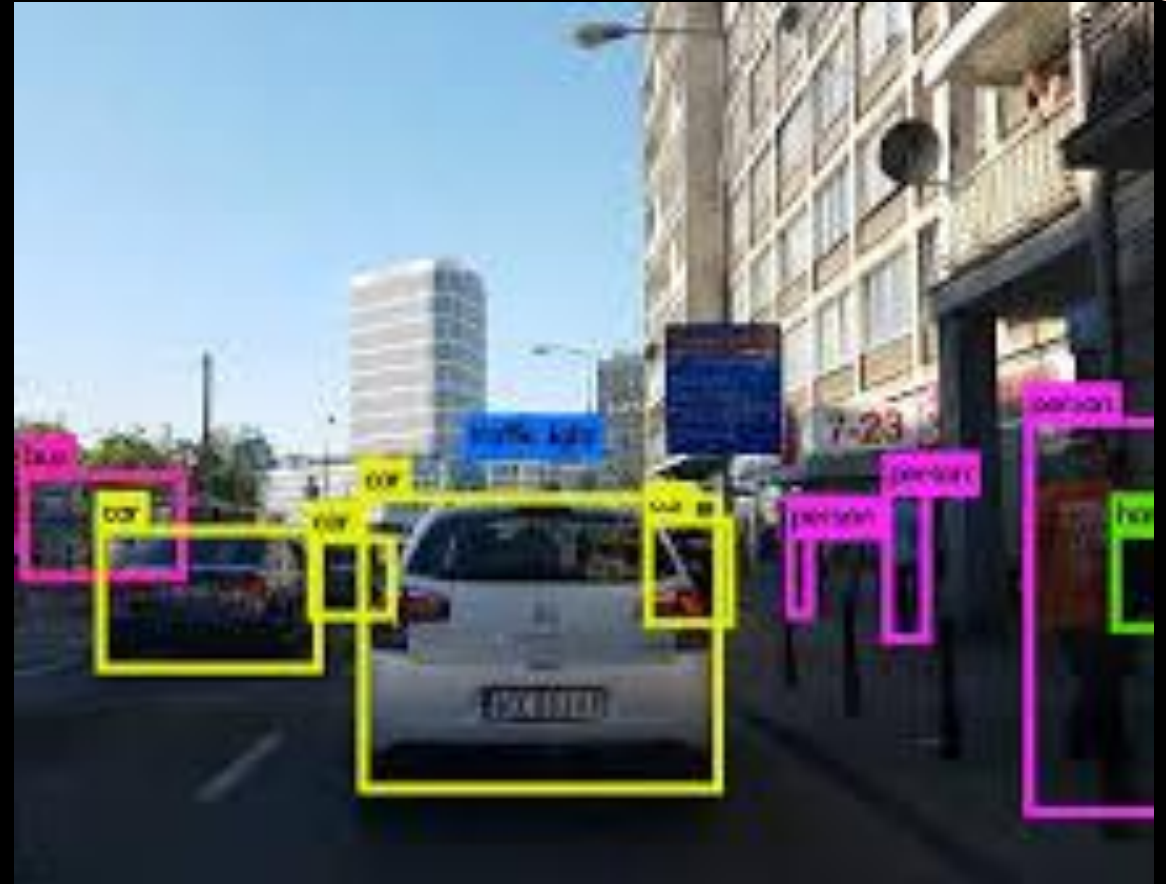


Lecture 21

CNNs, DNN, and OpenCV

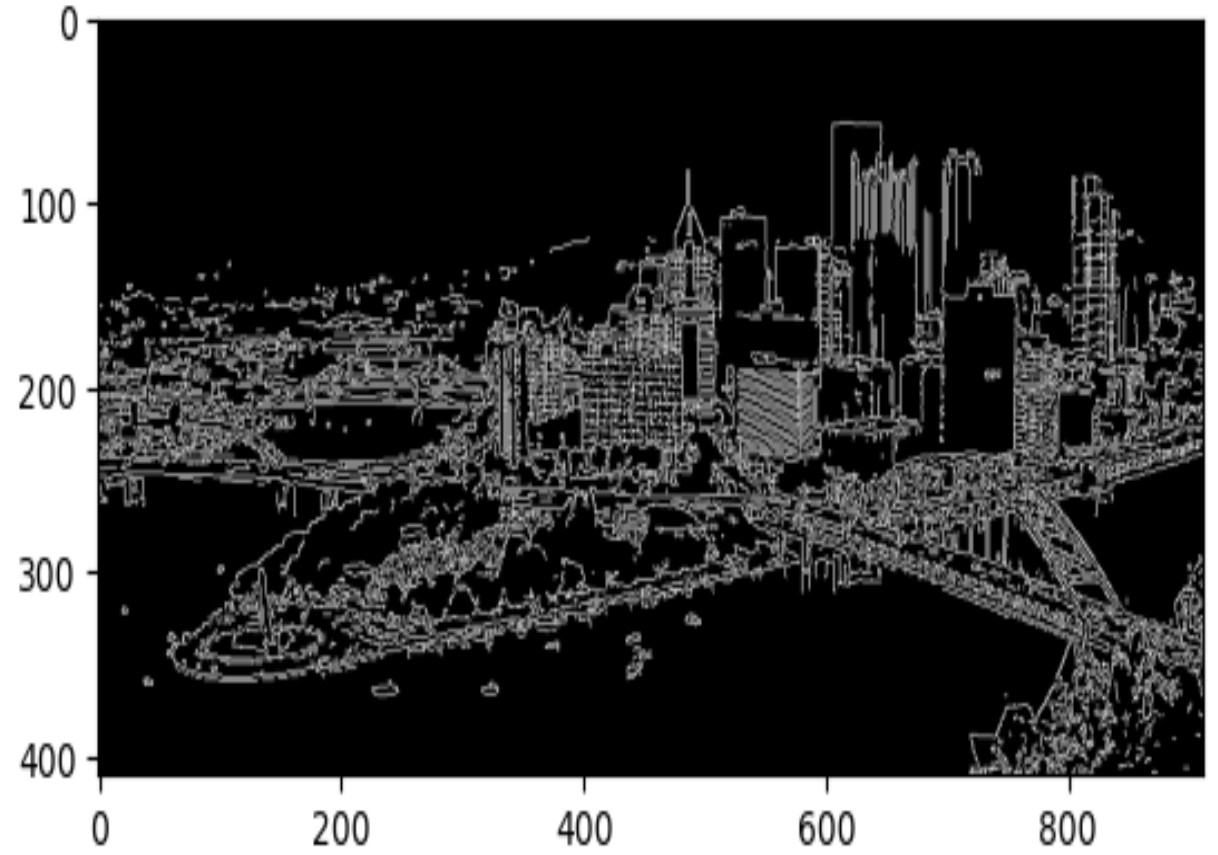
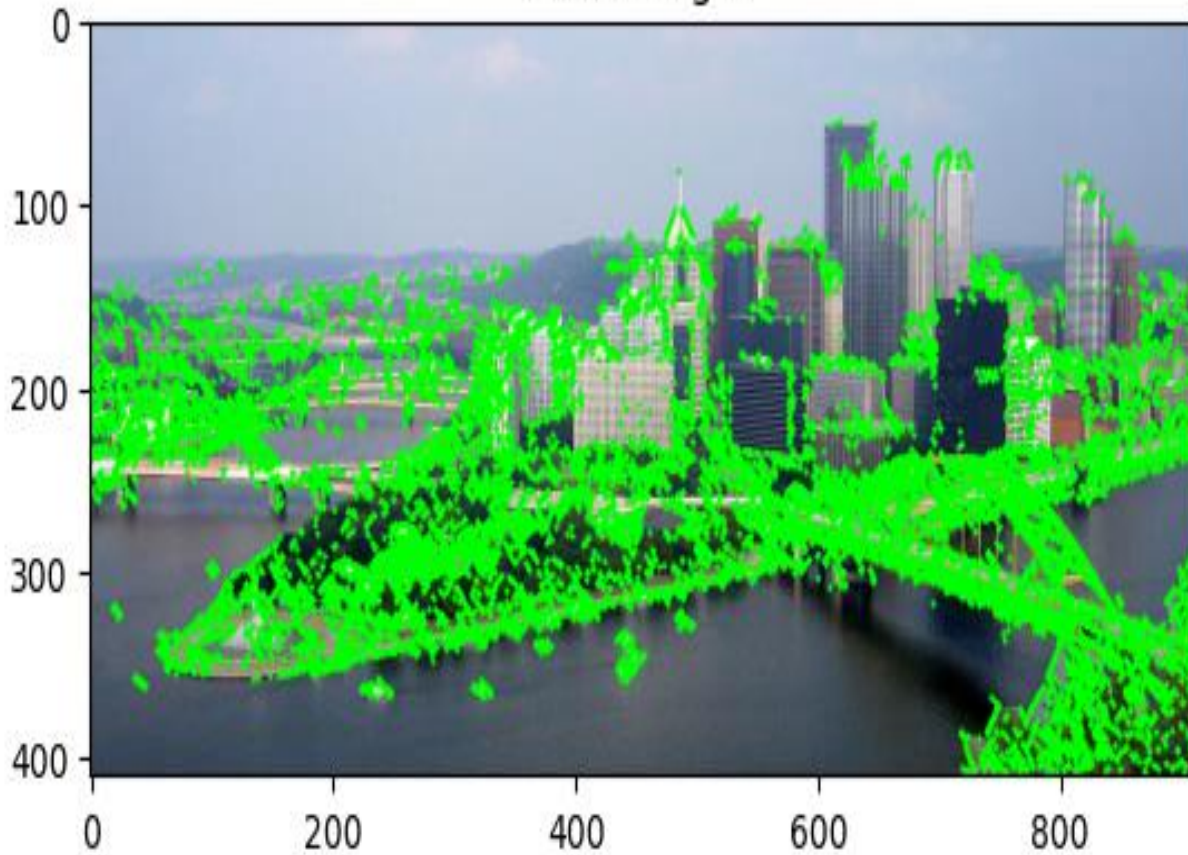
ECE 1390/2390



What is an image feature?

Edges

Sobel edges

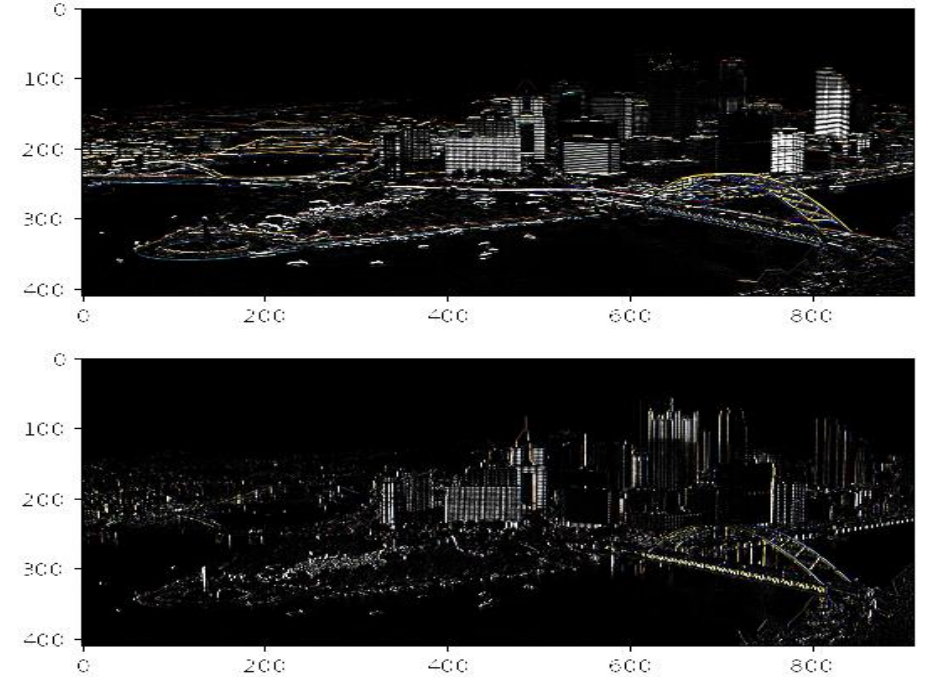


What is an image feature?

Harris Corners

$$\text{Kernel} = \begin{bmatrix} -1 & 1 \end{bmatrix}$$

$$\text{Kernel} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

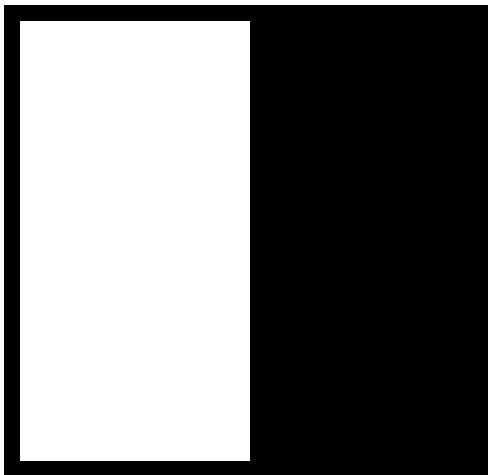


Structure matrix

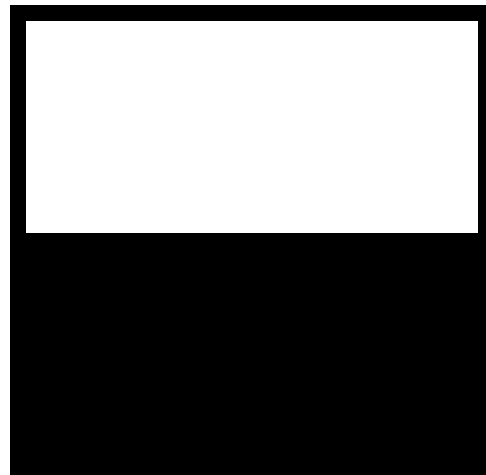
$$M[a, b] = \sum_{\{i, j\} \in W[a, b]} \begin{bmatrix} I_{x,x}(i, j) & I_{x,y}(i, j) \\ I_{y,x}(i, j) & I_{y,y}(i, j) \end{bmatrix}$$

- Viola and Jones, "[Rapid object detection using a boosted cascade of simple features](#)", [Computer Vision and Pattern Recognition](#), 2001
 - Haar-like feature extraction kernel (not actual Haar wavelets)
 - Series of convolution kernels to extract features
 - More features than pixels
 - E.g. 24 x 24 probe has 18,000 possible rectangle arrangements
 - ` Uses only small subset of the possibilities

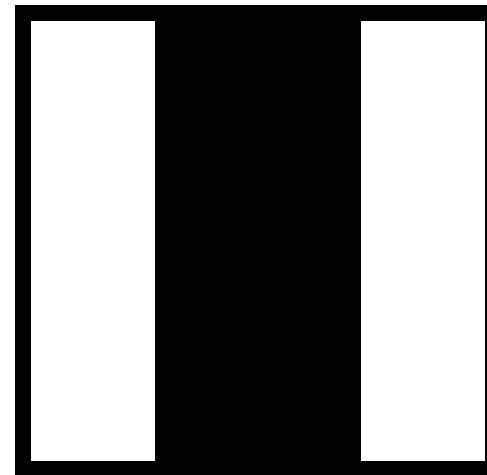
Horizontal Boundary



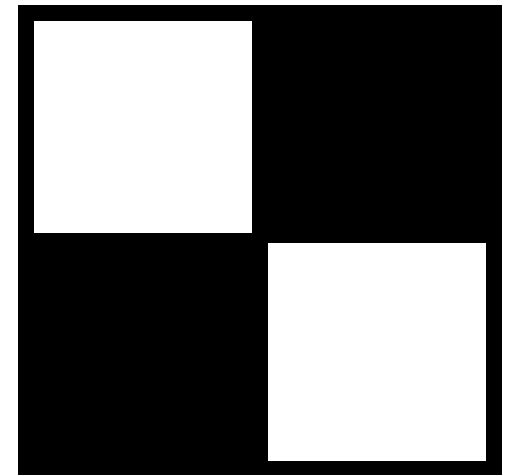
Vertical Boundary

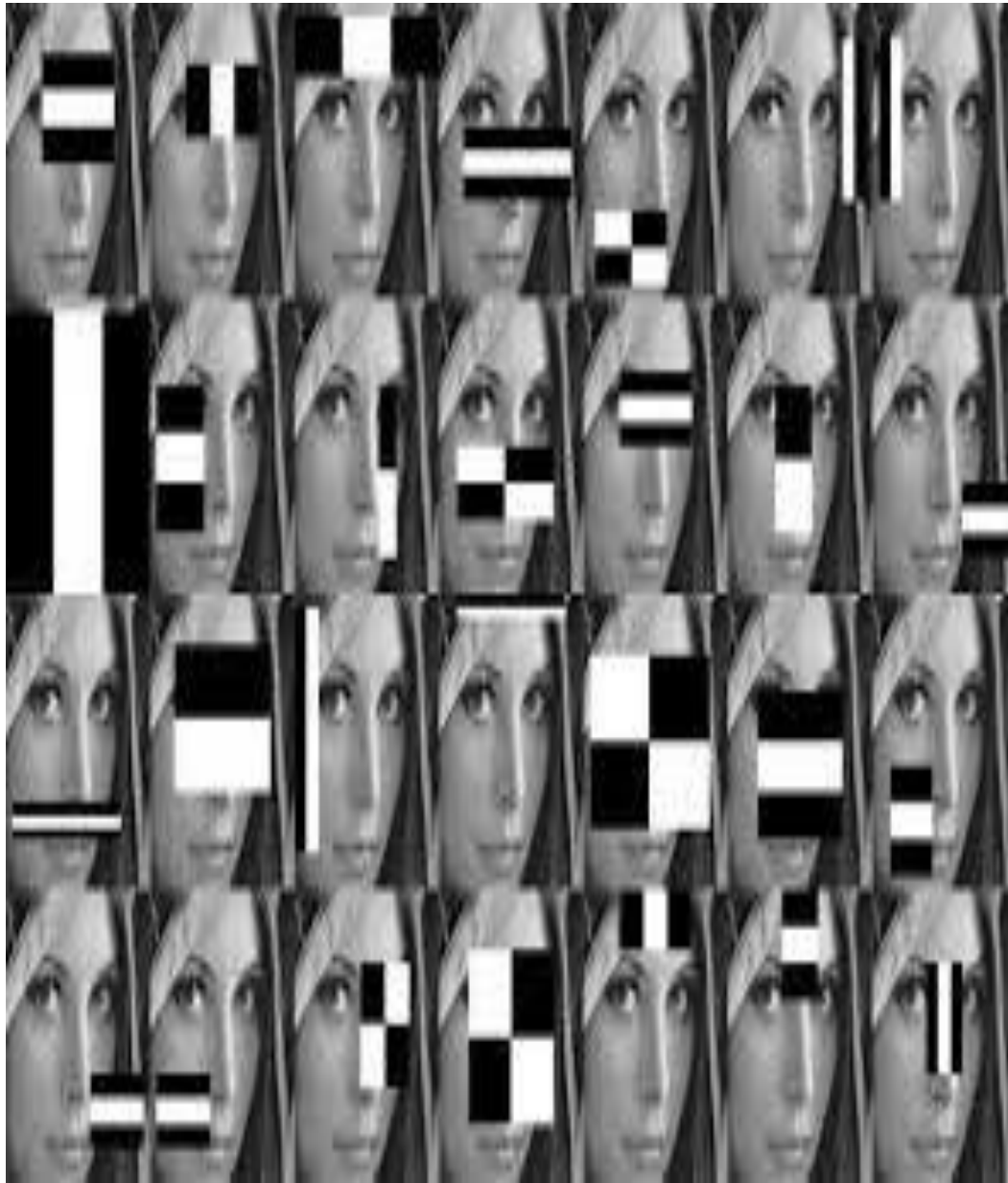


Vertical Line

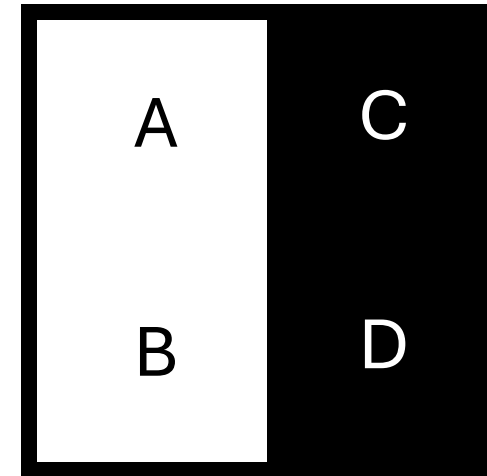


Crossing

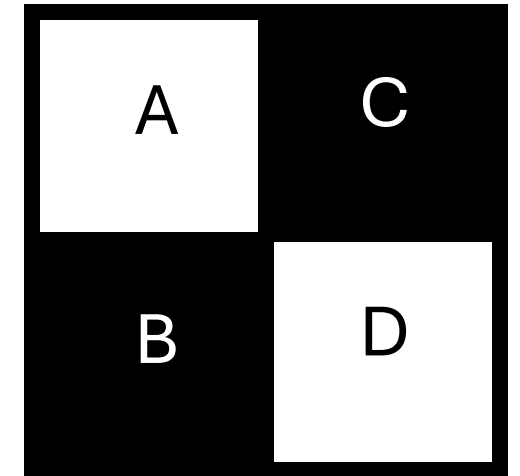




Feature Extraction:
Sum over rectangular windows



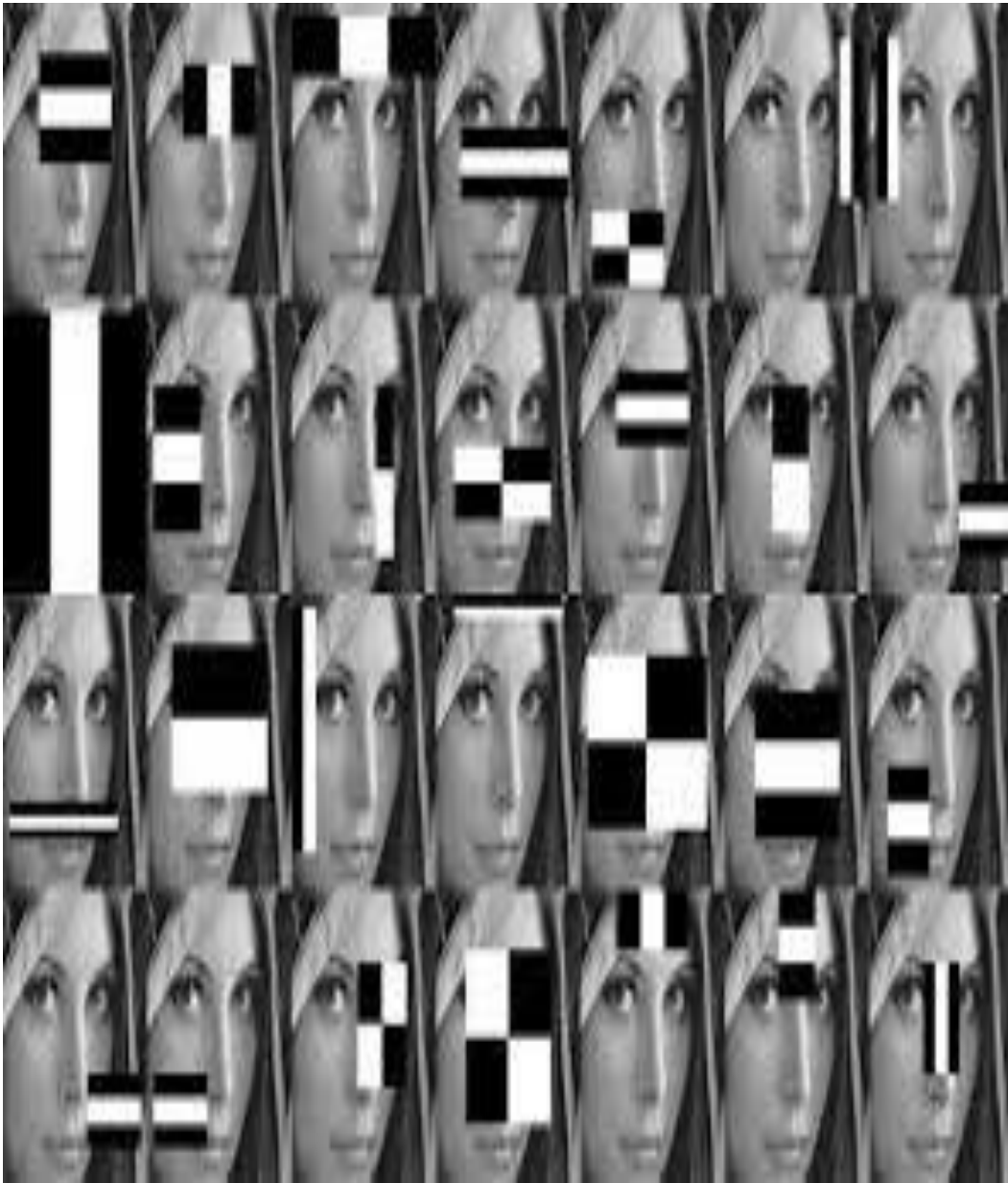
$$A+B - (C+D)$$



$$A+D - (B+C)$$

→ 28 features

Each feature is very
weakly predictive



Adaboost (Adaptive boosting)

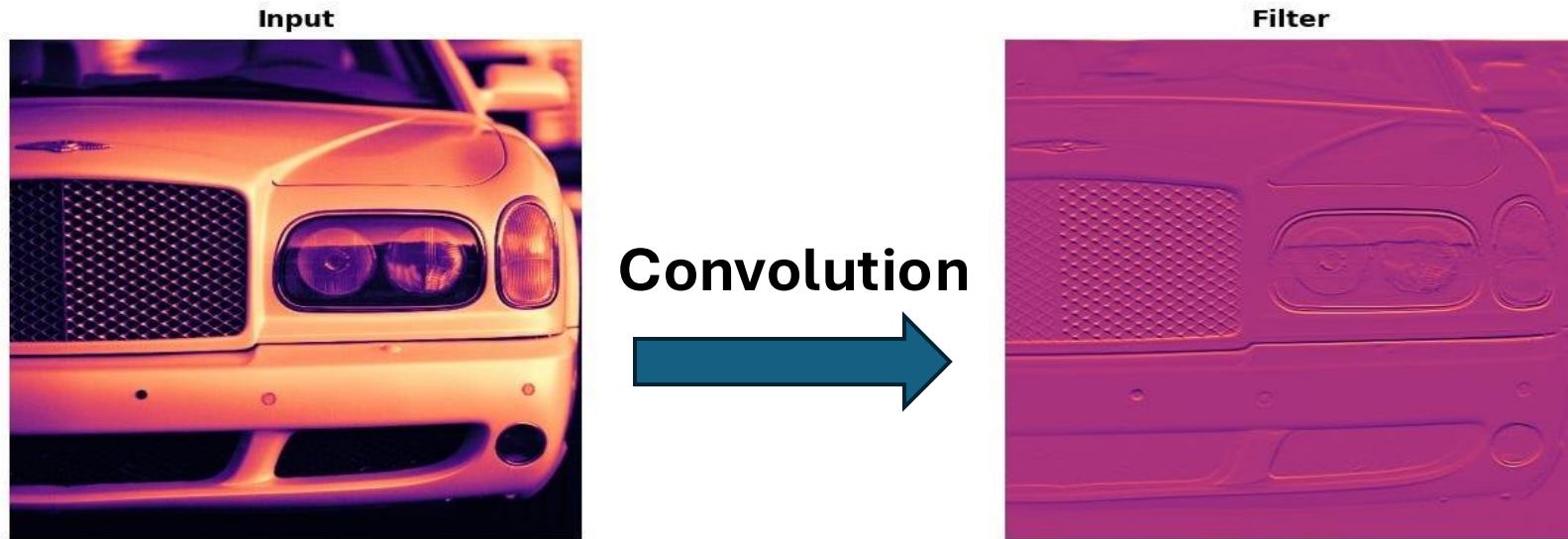
$$F = \sum_i f_i(x)$$

Boasted classifier

Weak binary (-1,1) classifiers

Yoav Freund and Robert Schapire (1995), [*A desicion-theoretic generalization of on-line learning and an application to boosting*](#), Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 23–37

Convolutional Neural Networks



Kernel size and count

- CNN's fit the convolution parameters as part of the model.
- This gives more flexibility than simply an edge or Harr feature

Padding and Stride:

- Padding can be added to the input to preserve spatial information and avoid the reduction in size during convolution.
- Stride determines how much the filter moves during each step

Convolutional Neural Networks

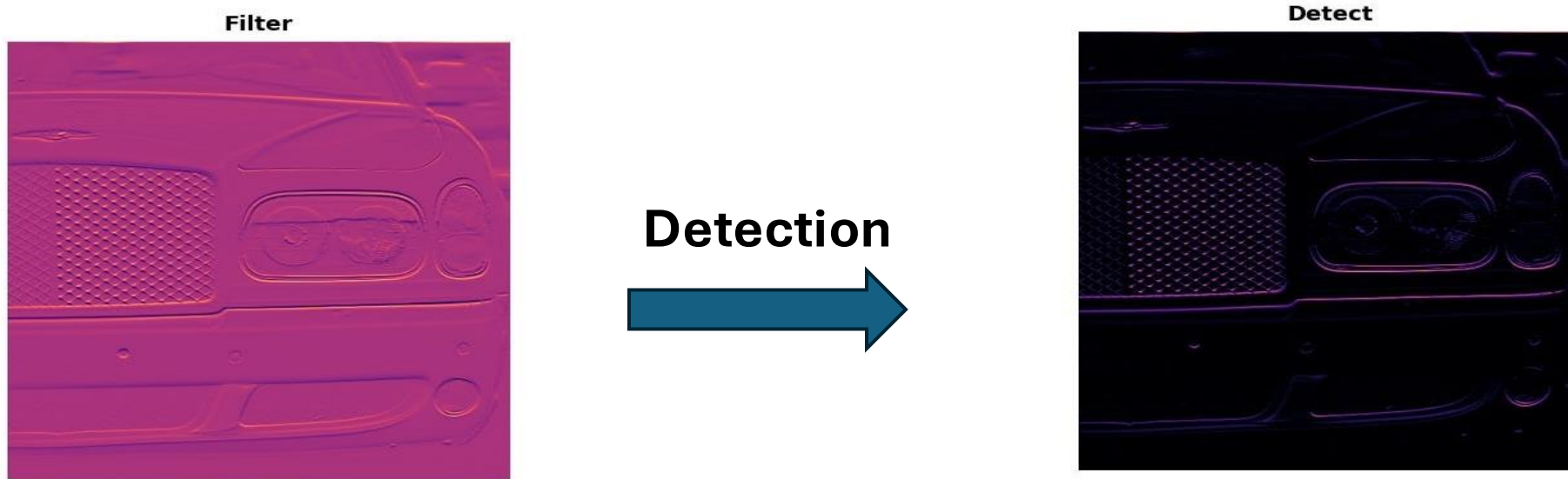
-1	-1
1	1

-1	-2	-1
0	0	0
1	2	1

-2	-1	0
-1	1	1
0	1	2



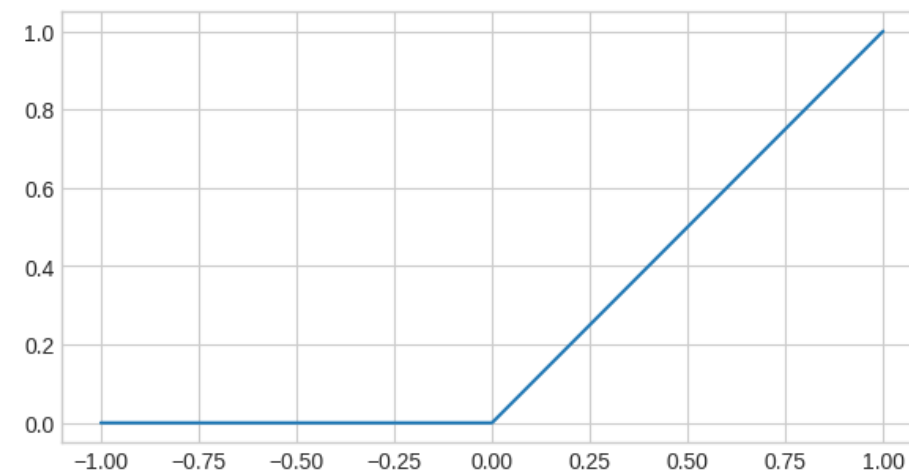
Convolutional Neural Networks



Detection/Activation layers

- An activation function (commonly ReLU — Rectified Linear Unit) is applied element-wise to introduce non-linearity to the network.
- The activation function helps the network learn complex, non-linear relationships in the data.

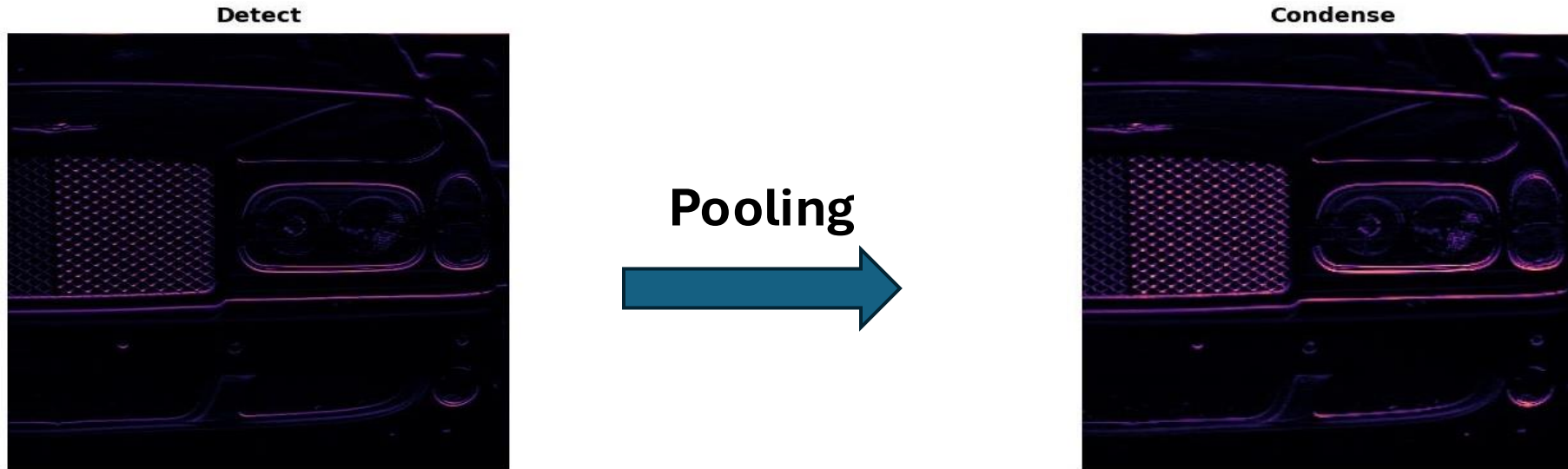
The Rectifier Function



Convolutional Neural Networks



Convolutional Neural Networks

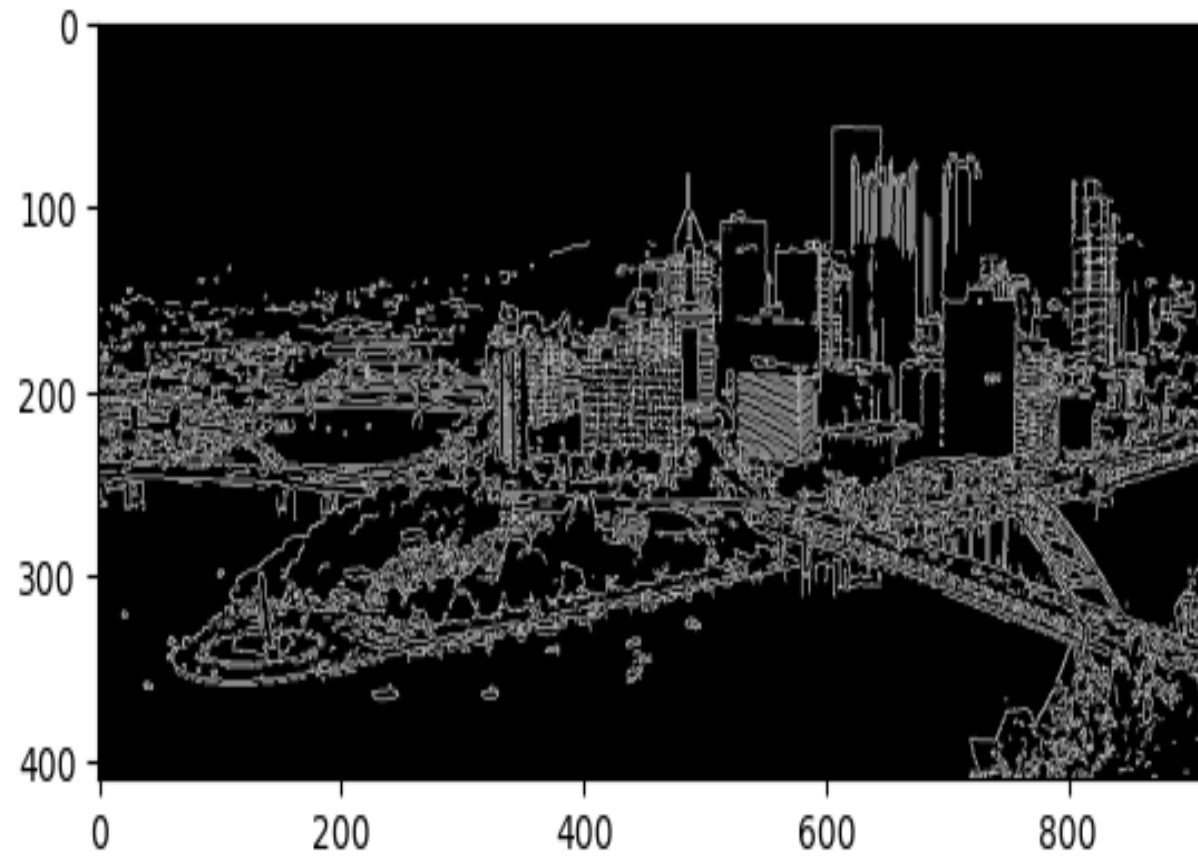
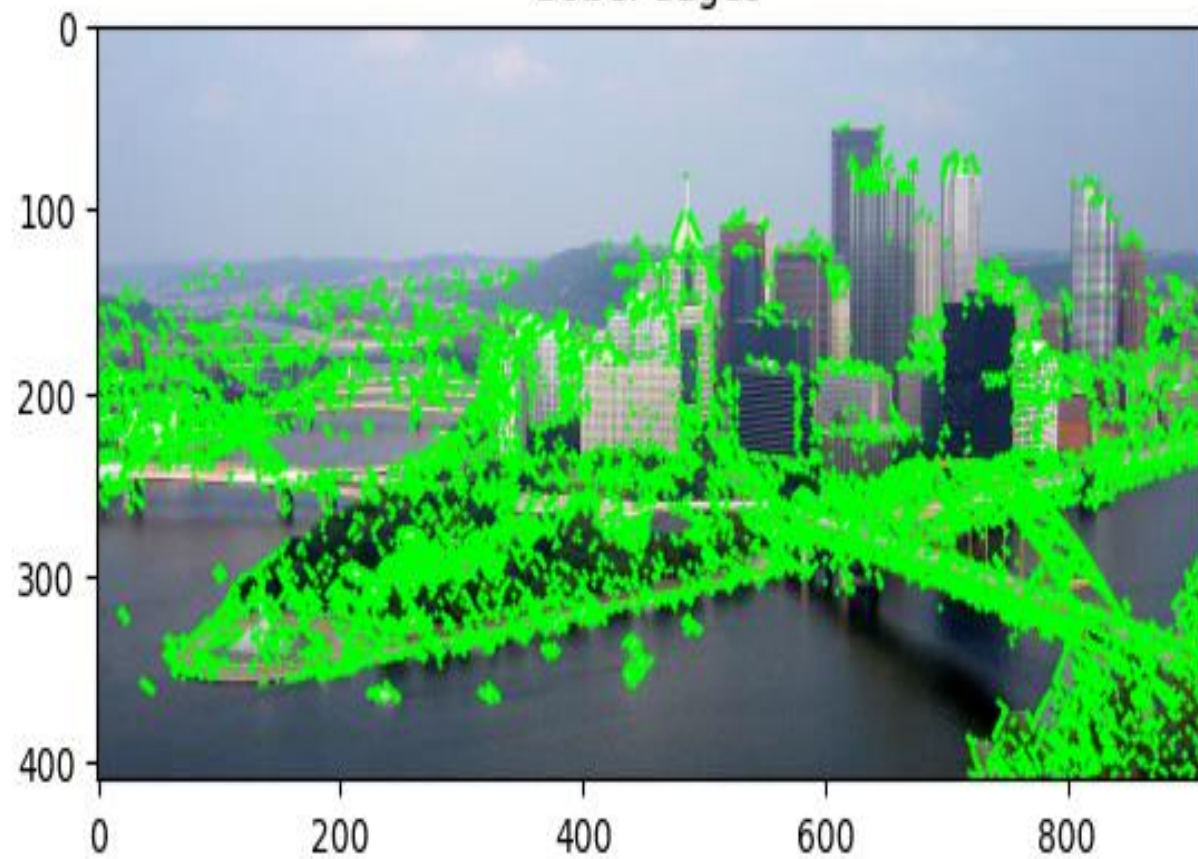


Pooling Layers:

- After convolution, pooling layers are often applied to reduce the spatial dimensions of the feature maps, decreasing the computational load and increasing the receptive field.
- Max pooling and average pooling are common pooling operations.

Edges

Sobel edges



Convolutional Neural Network

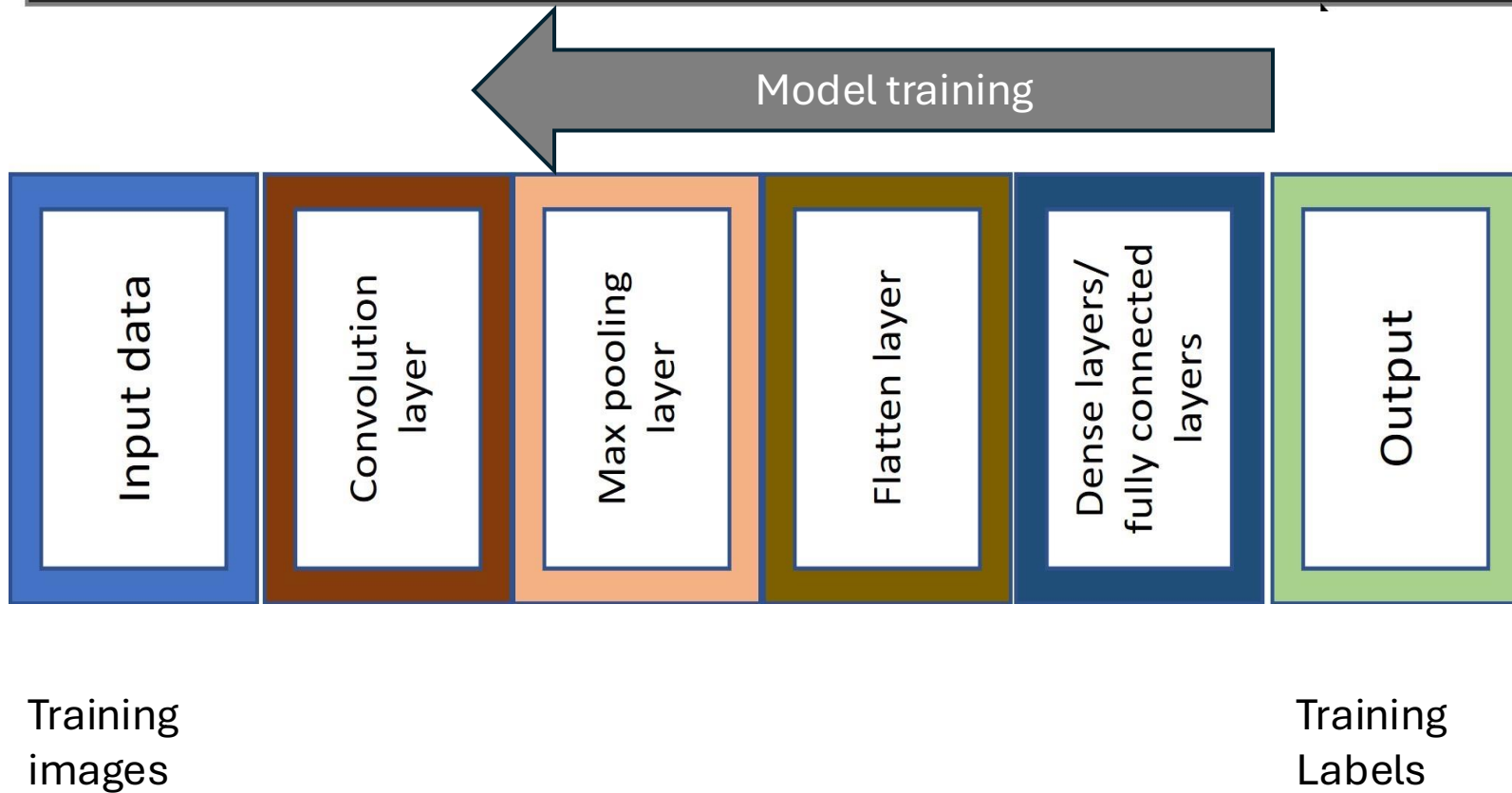
Feed Forward



**Feature
extraction**

Classification

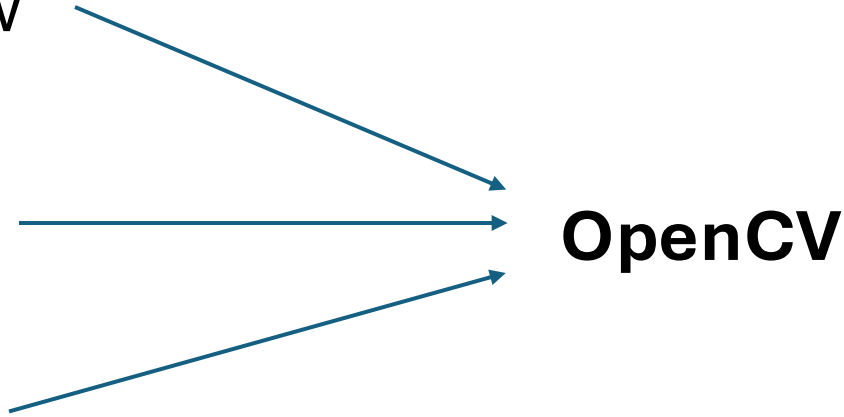
Convolutional Neural Network



Back-projection

- Some variation of steepest-decent
- Active area of development
- CNN designs to make back-projection more stable (e.g. ResNet)

- TensorFlow
- ONNX
- Caffe
- PyTorch
- DarkNet



OpenCV

OpenCV 3 introduced the DNN (deep neural network) module as an interface to other machine learning models

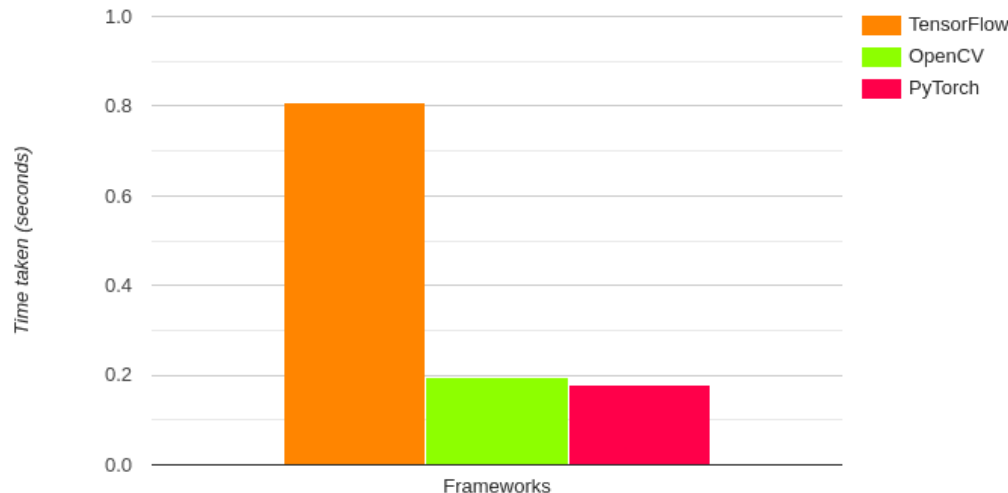


Image Classification Time Comparisons (Average of three images)

Benefits:

- OpenCV
 - Abstracts TensorFlow/PyTorch/etc backend
 - Access to NVIDIA or Intel GPU models
 - Better implementation on mobile or RaspPi frameworks
 - (Claimed) Comparable or better forward model performance over native packages (especially for CPU based computations)

CNN Model Zoos

- <https://modelzoo.co/>
- https://github.com/opencv/opencv_zoo

Limitations:

- OpenCV does NOT do model training or CNN construction
- Limited to image-based inputs
- Growing number of supported layer types (and can add custom ones)

Using cv2.dnn

cv2.dnn.readNet(.)

Generic CNN loading function. Works with many ML model structures

Inputs:


- **model:** This is the path to the pre-trained weights file.
- **config:** This is the path to the model configuration file,
- **framework:** provide the framework name that we are loading the models.
E.g. "Caffe"

Using cv2.dnn

1. `readNetFromCaffe()`: This is used to load pre-trained Caffe models and accepts *two arguments*. They are the *path to the prototxt file* and the *path to the Caffe model file*.
2. `readNetFromTensorflow()`: We can use this function to directly load the TensorFlow pre-trained models. This also accepts *two arguments*. One is the *path to the frozen model graph* and the other is the *path to the model architecture protobuf text file*.
3. `readNetFromTorch()`: We can use this to load Torch and PyTorch models which have been saved using the `torch.save()` function. We need to provide the *model path as the argument*.
4. `readNetFromDarknet()`: This is used to load the models trained using the DarkNet framework. We need to provide *two arguments* here as well. One of the *path to the model weights* and the other is the *path to the model configuration file*.
5. `readNetFromONNX()`: We can use this to load ONNX models and we only need to provide the path to the ONNX model file.

Preprocessing

Extend this dimension for multiple images



`Cv2.dnn.blobFromImage(s)(.)`

→ Image 4D array of size $\langle 1 \times 3 \times n \times m \rangle$

Inputs:

- image: This is the input image.
- scalefactor: This value scales the image by the provided value. It has a default value of 1 which means that no scaling is performed. E.g. many CNN models expect data in the range of 0-1, so this might be 1/255 to rescale uint8's
- size: This is the size that the image will be resized to. This is set by the model being used.
- mean: These are actually the mean values that are subtracted from the image's RGB color channels. This normalizes the input and makes the final input invariant to different illumination scales. This is dictated by the model being used.
- swapBR. Flag to swap BGR and RGB channels (used instead of cv2.cvtColor command)

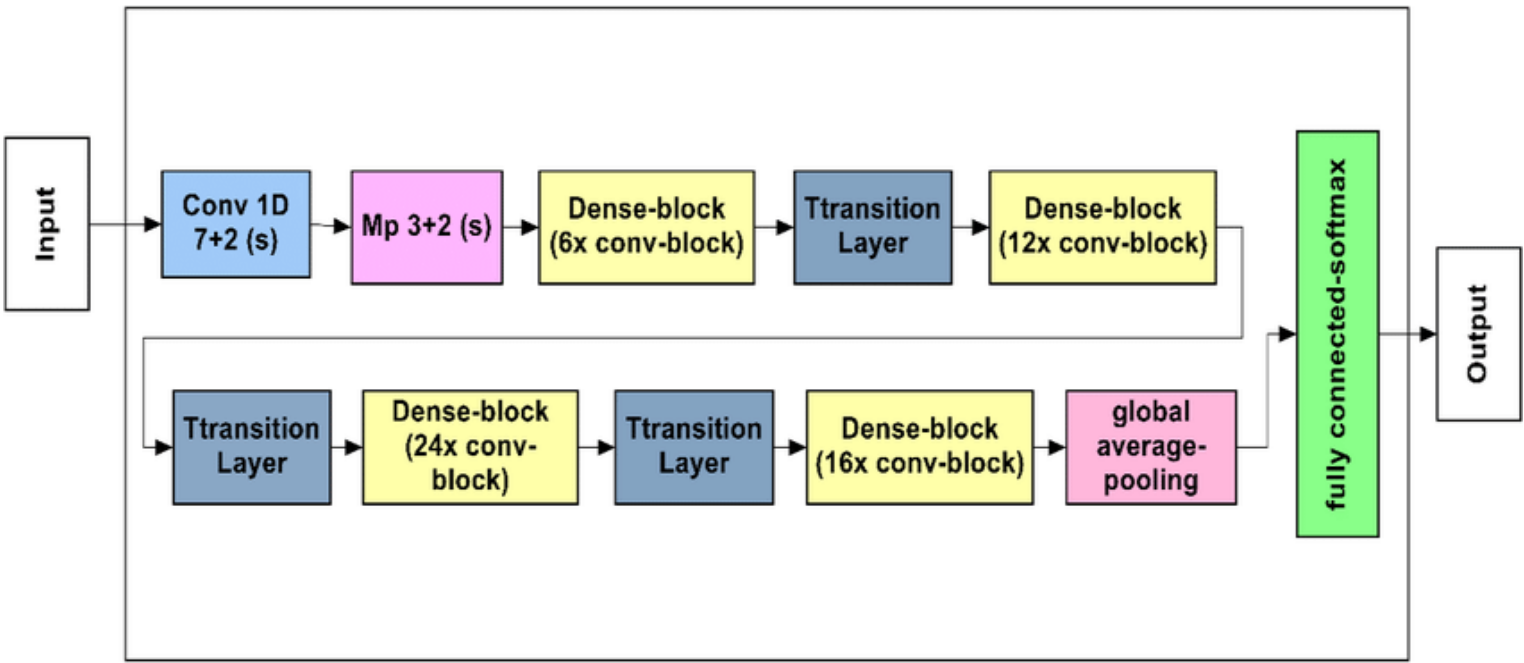
- Example: DenseNet121

- ImageNet classified
- 1000 classes

- 1) Cloned from <https://github.com/BVLC/caffe/wiki/Model-Zoo>

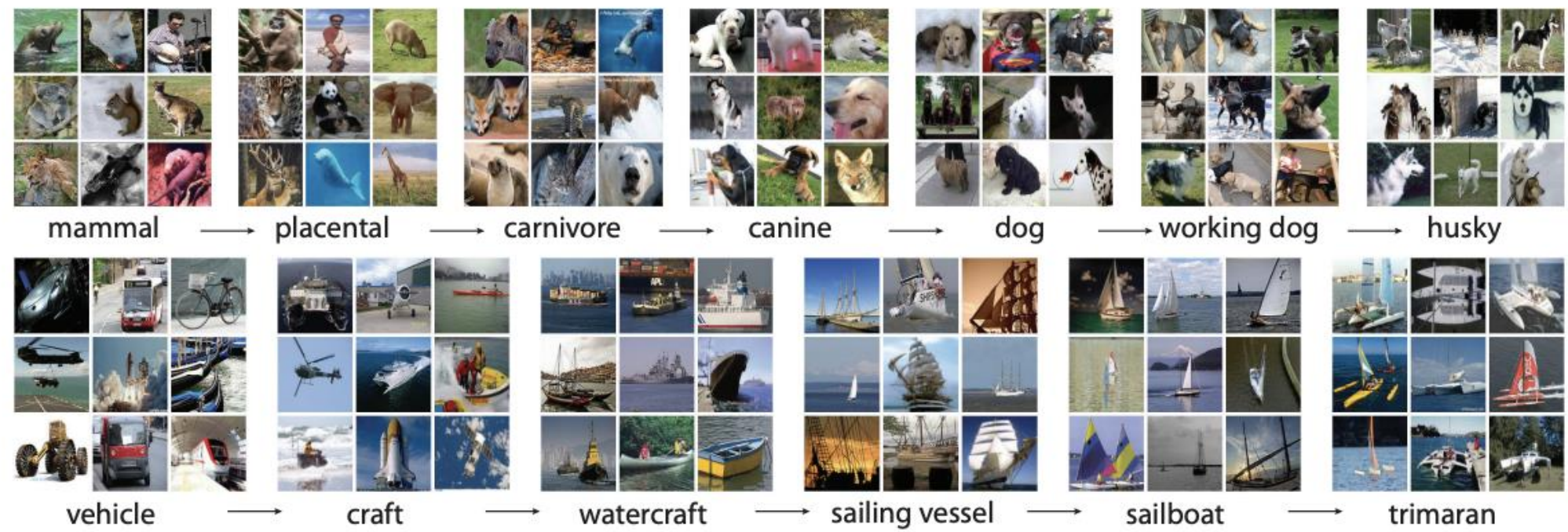
- Accept license from http://caffe.berkeleyvision.org/model_zoo.html#bvlc-model-license
- DenseNet_121.caffemodel
- DenseNet_121.prototxt
- Imagenet_classes.txt

(Placed these in the LectureNotebooks/Lecture21 folder)



ImageNet
<https://www.image-net.org/>

1000 labels
 1.2million training images
 100,000 testing images



Download a sample image

Download an example image from the pytorch website

```
import urllib
```

```
url, filename = ("https://github.com/pytorch/hub/raw/master/images/dog.jpg", "dog.jpg")
```

```
try: urllib.URLopener().retrieve(url, filename)
```

```
except: urllib.request.urlretrieve(url, filename)
```

Load Model

```
weights = os.path.relpath('./DenseNet_121.caffemodel')
```

```
arch = os.path.relpath('./DenseNet_121.prototxt')
```

```
net = cv2.dnn.readNetFromCaffe(arch,weights)
```

Preprocessing

```
blob = cv2.dnn.blobFromImage(img,1/255,(224,224),(103.94,116.78,123.68))
```

These numbers come from reading the ReadMe from DenseNet121 and are the image size and range that it expects for inputs

Set data for classifier model

```
net.setInput(blob)
```

Optional

```
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
```

```
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
```

The actual run command

```
outputs = model.forward()
```

Classifiers: The outputs will depend on the model being used. For classifier models (like DenseNet), this will be a vector of the classification scores for the 1000 possible classes

Softmax

$$\text{Prob}_i = \frac{e^{(\text{class_value}_i - \max(\text{class_value}))}}{\sum e^{(\text{class_value}_i - \max(\text{class_value}))}}$$



Predicted Samoyed
with probability 87.3%

Detection model

Load Model

```
weights = os.path.relpath('./yolov4.weights')
arch = os.path.relpath('./yolov4.cfg')
net = cv2.dnn.readNet(weights, arch)

model = cv2.dnn.DetectionModel(net)
model.setInputParams(size=(416, 416), scale=1/255, swapRB=False)

class_label, class_probability, bounding_location = model.detect(img)

array([16, 16], dtype=int32),
array([0.84245753, 0.81188846], dtype=float32)
array([[ 170, 44, 1238, 1096], [ 173, 39, 1231, 1107]], dtype=int32))
```

