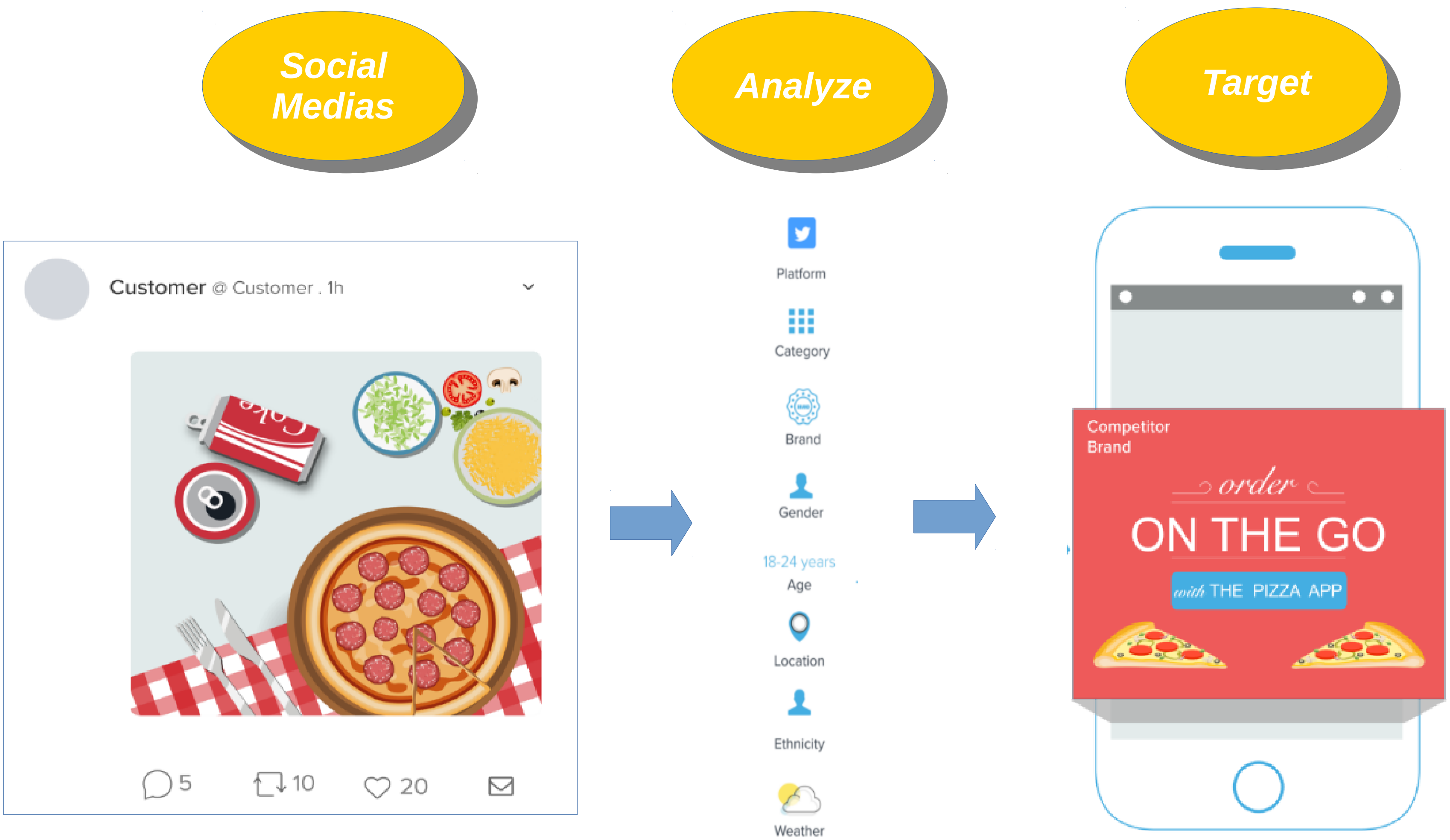
Logo Detection Using PyTorch

Nithiroj Tripatarasit (Lek)

Ad Tech

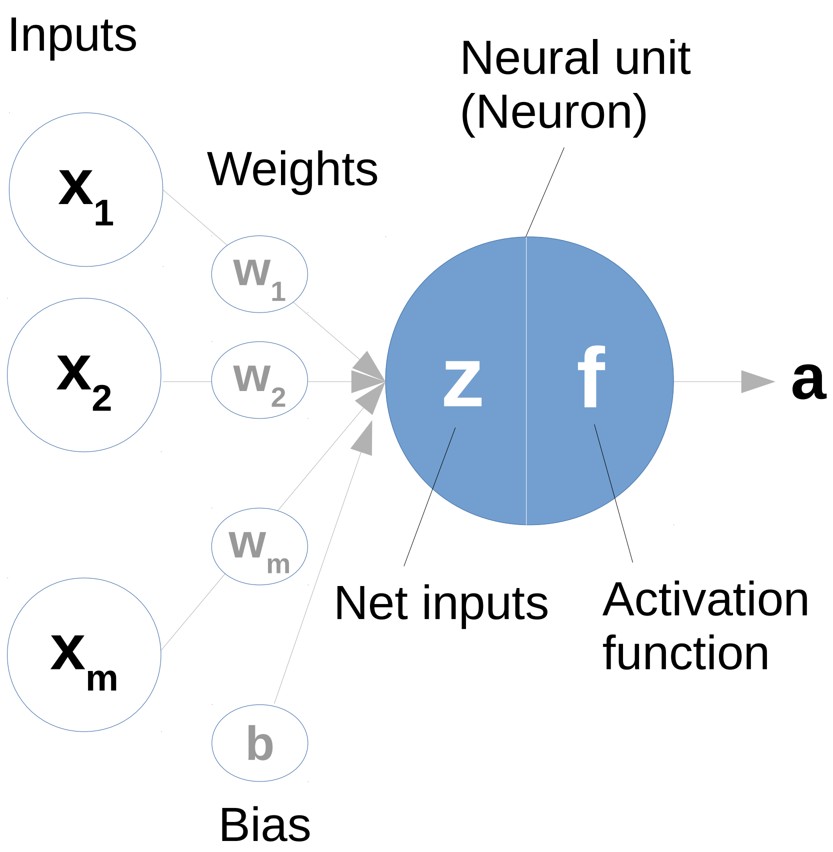


Source: https://cluep.com/

Deep Learning

1. **Create the Network**
2. **Train the Network**
3. **Deploy the Network**

Single-Layer Neural Network

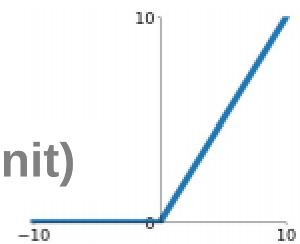


$$z = w_1x_1 + w_2x_2 + \ldots + w_mx_m + b$$

$$f = \max(0, z)$$

ReLU

(Rectified Linear Unit)



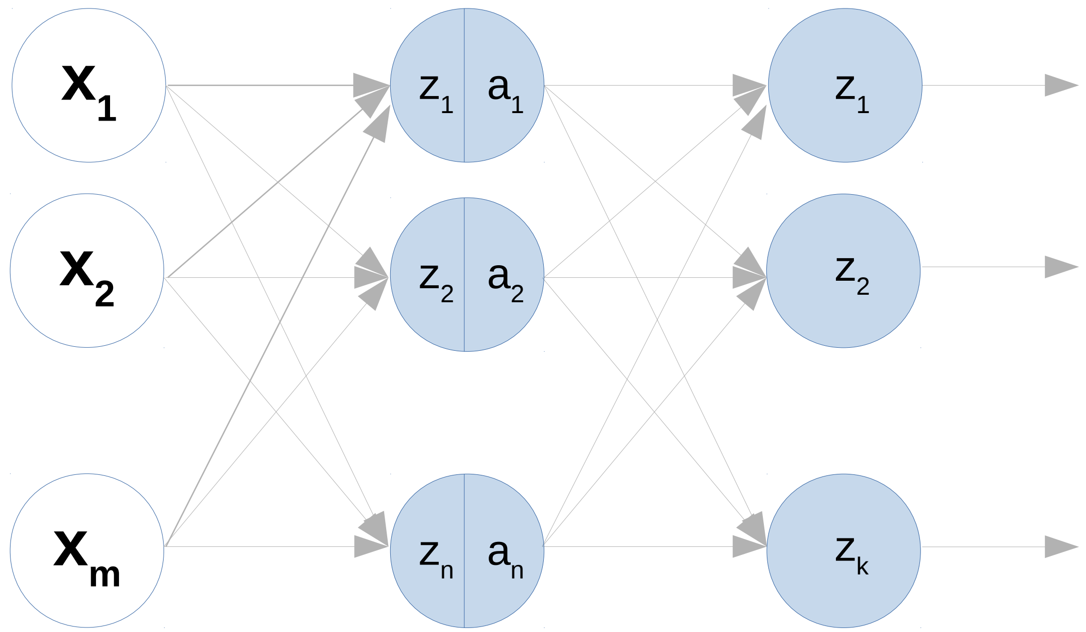
$$a = f(z)$$

Fully Connected Network

Input layer with
m input units

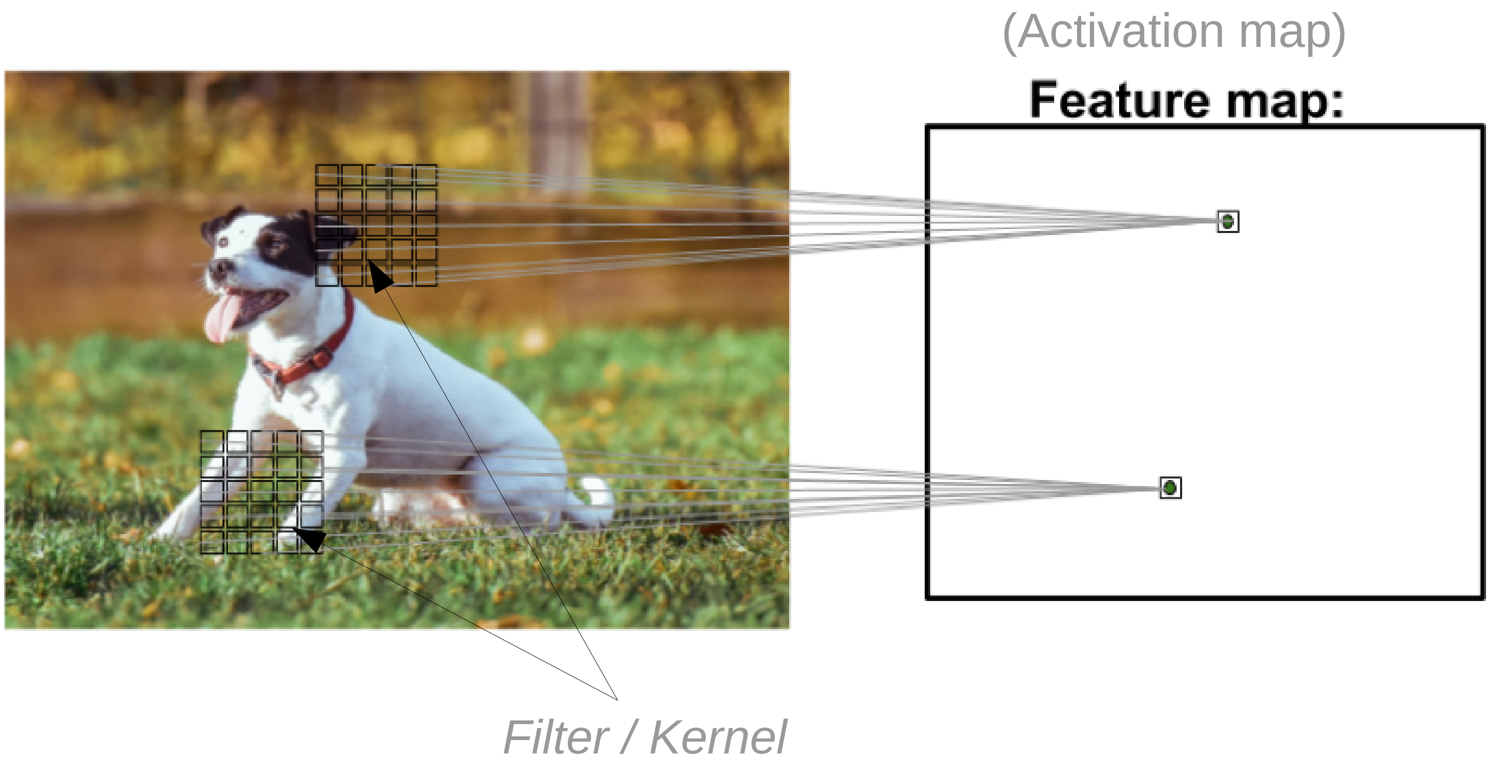
Hidden layer with
n neural units

Output layer with
k neural units

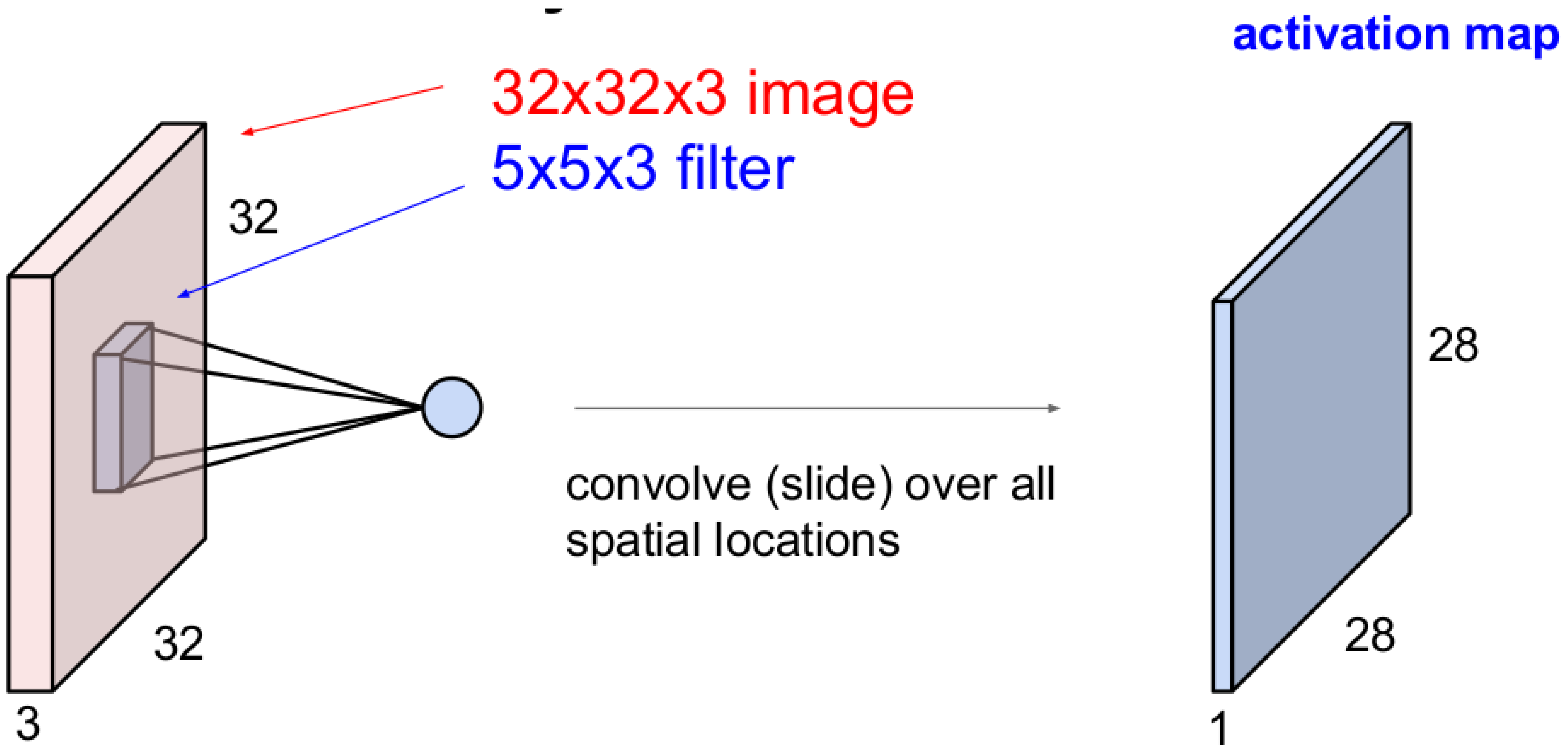


Convolutional Neural Networks
(CNN)

It is an important tool for object recognition.



Convolutional Layer



Source: http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture06.pdf

Dot Product

INPUT
5 x 5 x 1

1	2	0	1	1
1	1	1	1	1
0	0	2	1	1
2	2	0	0	1
2	1	1	2	0

Filter
3 x 3 x 1
Stride = 1, Padding = 0

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

weights

1

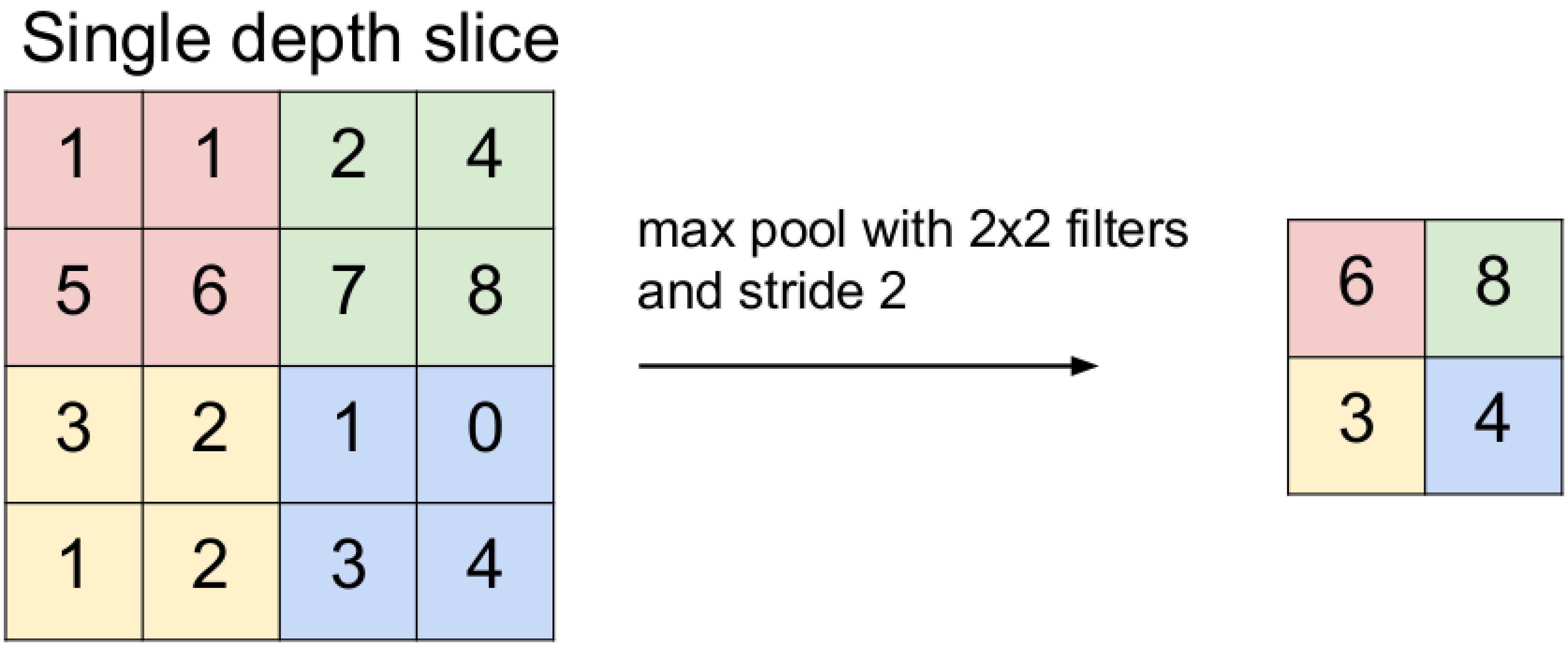
bias

OUTPUT
3 x 3 x 1
(Activation Map)

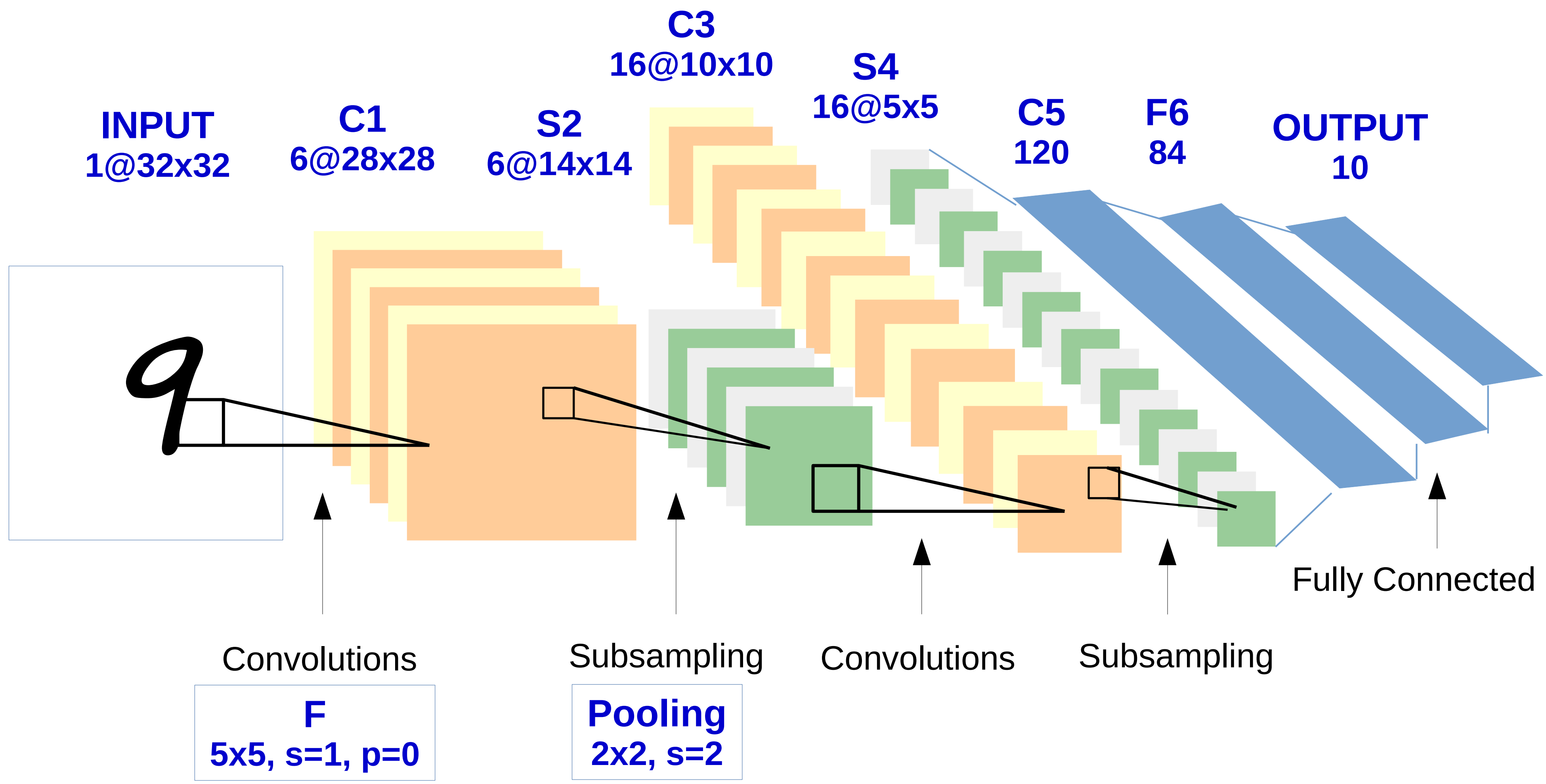
0	-3	2
3	1	-2
-1	2	1

(1 * -1) + (2 * 1) + (0 * 0) + (1 * -1) + (1 * -1) + (2 * 0) + (2 * 1) + (0 * 0) +((1 * 0) + 1 = 3

Max Pooling

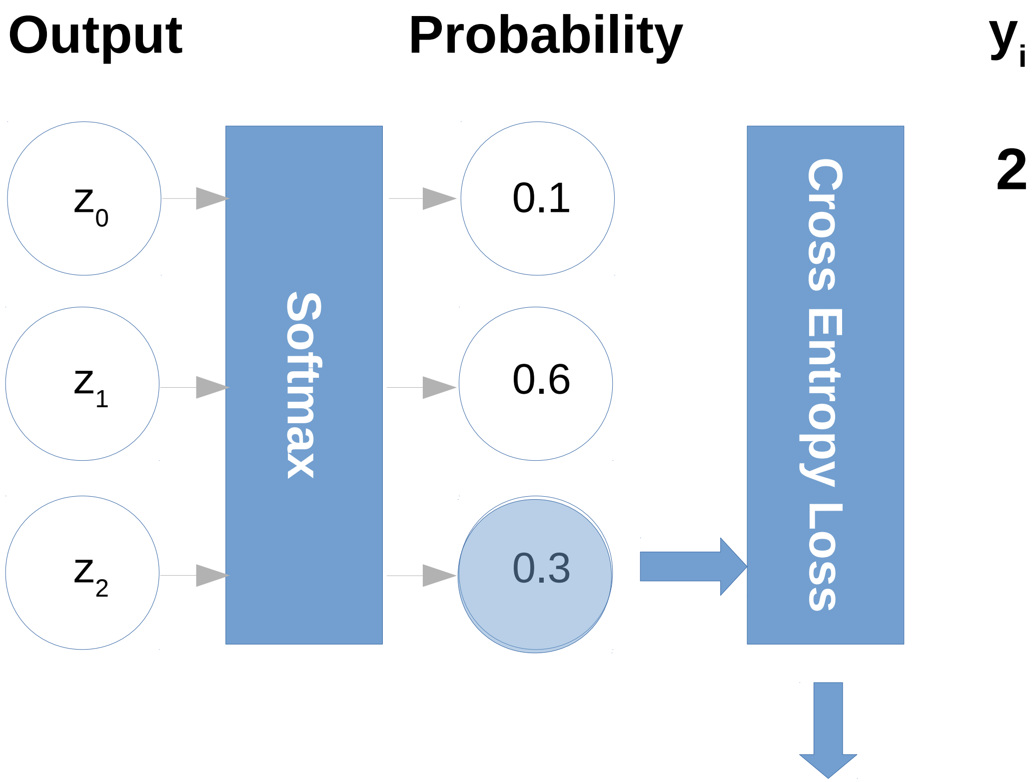

Single depth slice
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

LeNet-5



Source: LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

Loss Function



L_i = -log(0.3) = 0.523

Softmax

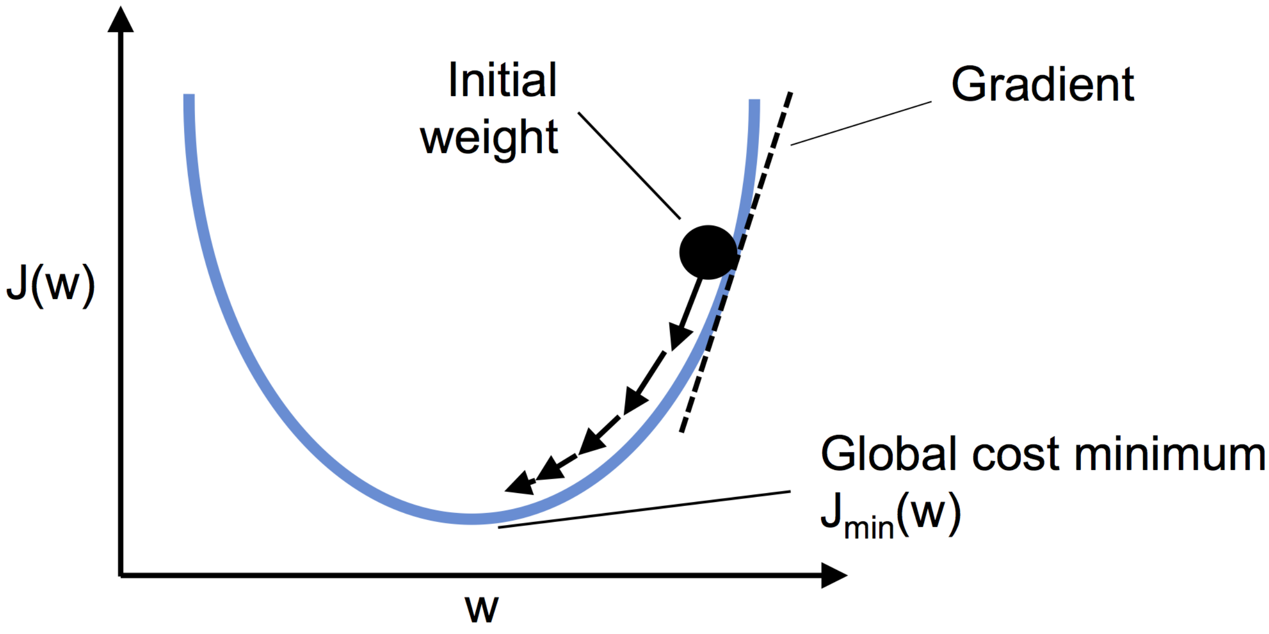
$$\boxed{P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

Cross Entropy Loss

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Gradient Descent

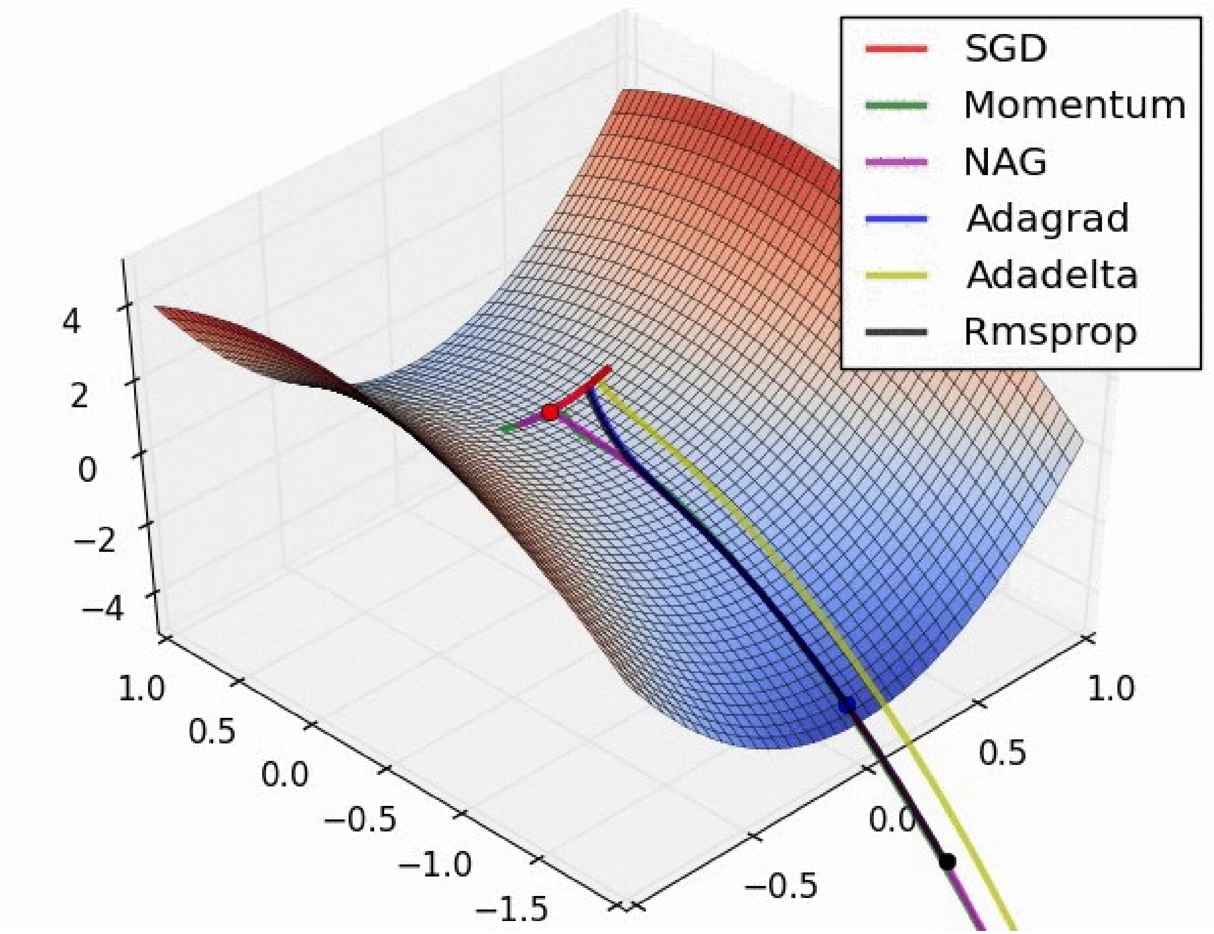
An algorithm used to optimize the network



Update weights : w += - learning_rate * gradient

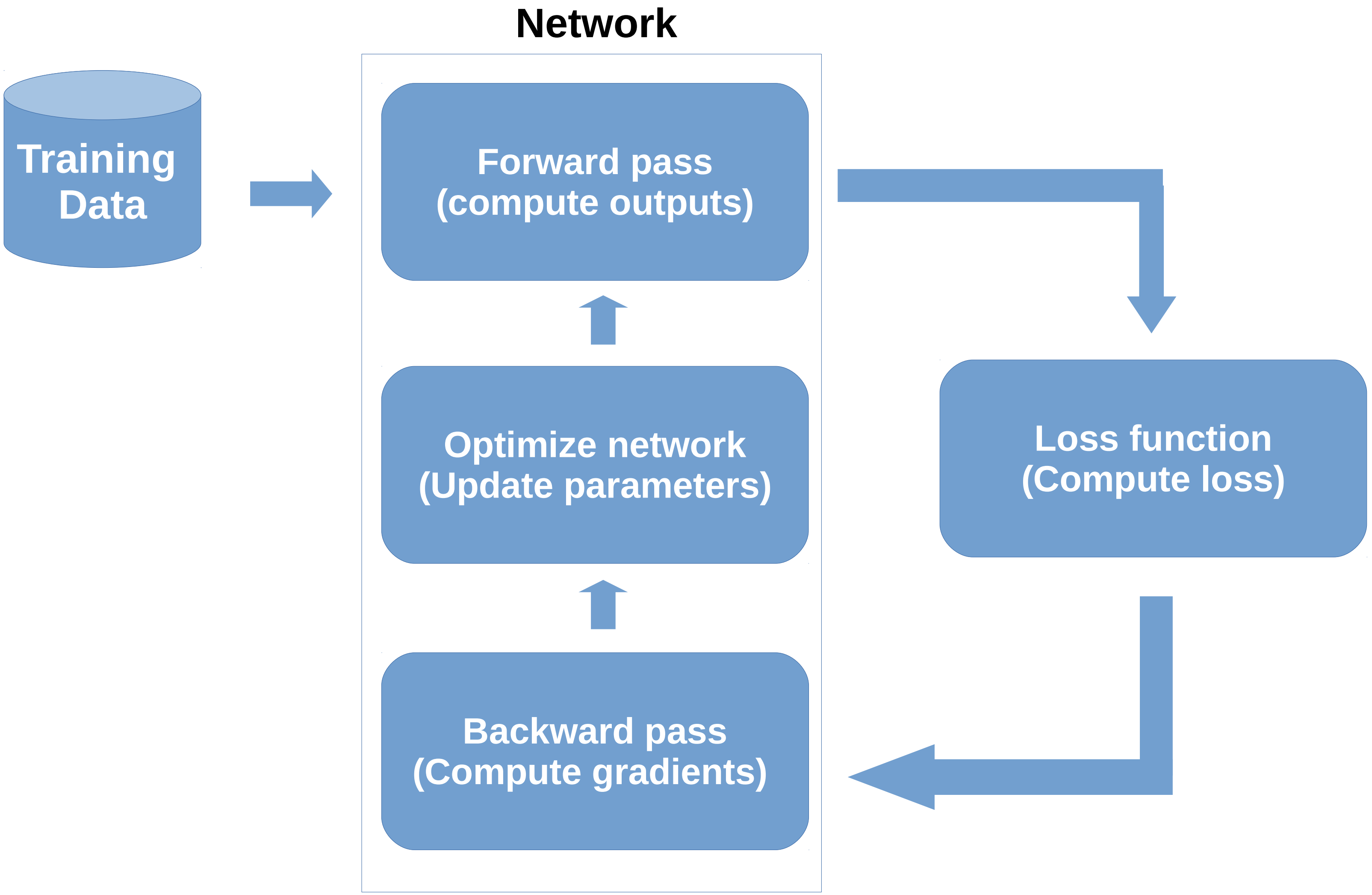
Optimizers

Algorithms used to optimize the network



Source: http://cs231n.github.io/neural-networks-3/

Network Training Loop



Deep Learning Framework

PyTorch

- Deep learning framework
- Autograd
- Tools to create and train deep learning easily and efficiently
- GPU support

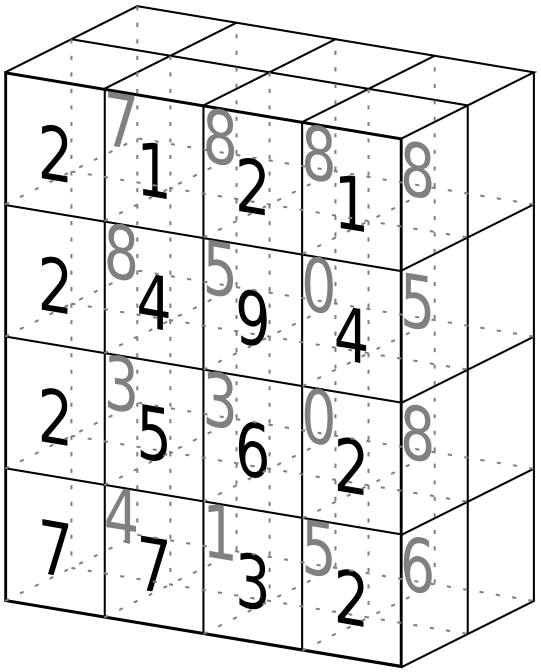
Tensors

't'
'e'
'n'
's'
'o'
'r'

tensor of dimensions [6]
(vector of dimension 6)

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3
2	3	8	4
6	2	6	4

tensor of dimensions [6,4]
(matrix 6 by 4)



tensor of dimensions [4,4,2]

Logo Detection Using PyTorch

Project Pipeline

1. **Get the data.**
2. **Prepare data for network.**
3. **Create network.**
4. **Train network.**
5. **Evaluate.**

1. Get the Data

- Flickrlogos-32
- 32 Different logo brands from Flickr

  Adidas, Aldi, Apple, Becks, BMW, Carlsberg, Chimay, Coca-Cola, Corona, DHL, Erdinger, Esso, Fedex, Ferrari, Ford, Foster's, Google, Guiness, Heineken, HP, Milka, Nvidia, Paulaner, Pepsi, Ritter Sport, Shell, Singha, Starbucks, Stella Artois, Texaco, Tsingtao and UPS.
- Training logo set 320 images
- Validation logo set 960 images
- Test set 3,960 images
- No-logo set 3,000 images

http://www.multimedia-computing.de/flickrlogos *(see dowload on page)*

1. Get the Data

- Load FlickrLogos-32_dataset_v2.zip and unzip

```python
import urllib.request
import os
import zipfile
import shutil
from pathlib import Path

FLICKLOGOS_URL = '.../FlickrLogos-32_dataset_v2.zip'
SOURCE_DIR = Path('FlickrLogos-v2')
DATA_DIR = Path('data')

def load_datasets(url, dst_dir):
    zip_file = url.split(sep='/')[-1]
    if not dst_dir.is_dir():
        if not zip_file.is_file():
            urllib.request.urlretrieve(url, zip_file)
        with zipfile.ZipFile(zip_file) as zip_ref:
            zip_ref.extractall()

load_datasets(FLICKLOGOS_URL, SOURCE_DIR)
```

2. Prepare Data for Network

- List image paths from text files and add half of no-logo paths to train and val paths

```
def list_image_paths(txt_relpath):
    with open(txt_relpath) as f:
        image_paths = f.read().splitlines()
    return image_paths

train_logo_relpaths = list_image_paths(SOURCE_DIR / 'trainset.relpaths.txt')
val_logo_relpaths = list_image_paths(SOURCE_DIR / 'valset-logosonly.relpaths.txt')
val_nologo_relpaths = list_image_paths(SOURCE_DIR / 'valset-nologos.relpaths.txt')
test_relpaths = list_image_paths(SOURCE_DIR / 'testset.relpaths.txt')


train_relpaths = train_logo_relpaths \
                    + val_nologo_relpaths[:int(len(val_nologo_relpaths) / 2)]
val_relpaths = val_logo_relpaths \
                    + val_nologo_relpaths[int(len(val_nologo_relpaths) / 2):]
```

2. Prepare Data for Network

../adidas/2325670.jpg → ../Dataset/Class/image.jpg
../train/adidas/2354545.jpg
../val/adidas/5553232.jpg
../test/adidas/7353256.jpg

```
SETS = ['train', 'val', 'test']
relpaths = [train_relpaths, val_relpaths, test_relpaths]
dataset_paths = dict(zip(SETS, relpaths))

def prepare_datasets(src_dir, dst_dir, keep_source=True):
    for dataset, paths in dataset_paths.items():
        num_files = 0
        for path in paths:
            num_files += 1
            src = src_dir / path
            dst = dst_dir / (path.replace('classes/jpg', dataset))
            dst.parent.mkdir(parents=True, exist_ok=True)
            shutil.copy2(src, dst)
        print(dataset, 'dataset:', str(num_files))
    if not keep_source: shutil.rmtree(src_dir)

prepare_datasets(SOURCE_DIR, DATA_DIR)
```

2. Prepare Data for Network

- Create data_transforms, datasets, and dataloaders.

```
import torch
import torchvision
from torchvision.transforms import transforms
from torch.utils.data import DataLoader

train_mean = np.array([0.44943, 0.4331 , 0.40244])
train_std = np.array([0.29053, 0.28417, 0.30194])

data_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(train_mean, train_std)
])

datasets = {i : torchvision.datasets.ImageFolder(DATA_DIR / i, data_transforms)
            for i in SETS}

bz = 32

dataloaders = {i : DataLoader(datasets[i], batch_size=bz,
                shuffle=(i == 'train'), num_workers=4) for i in SETS}
```

2. Prepare Data for Network

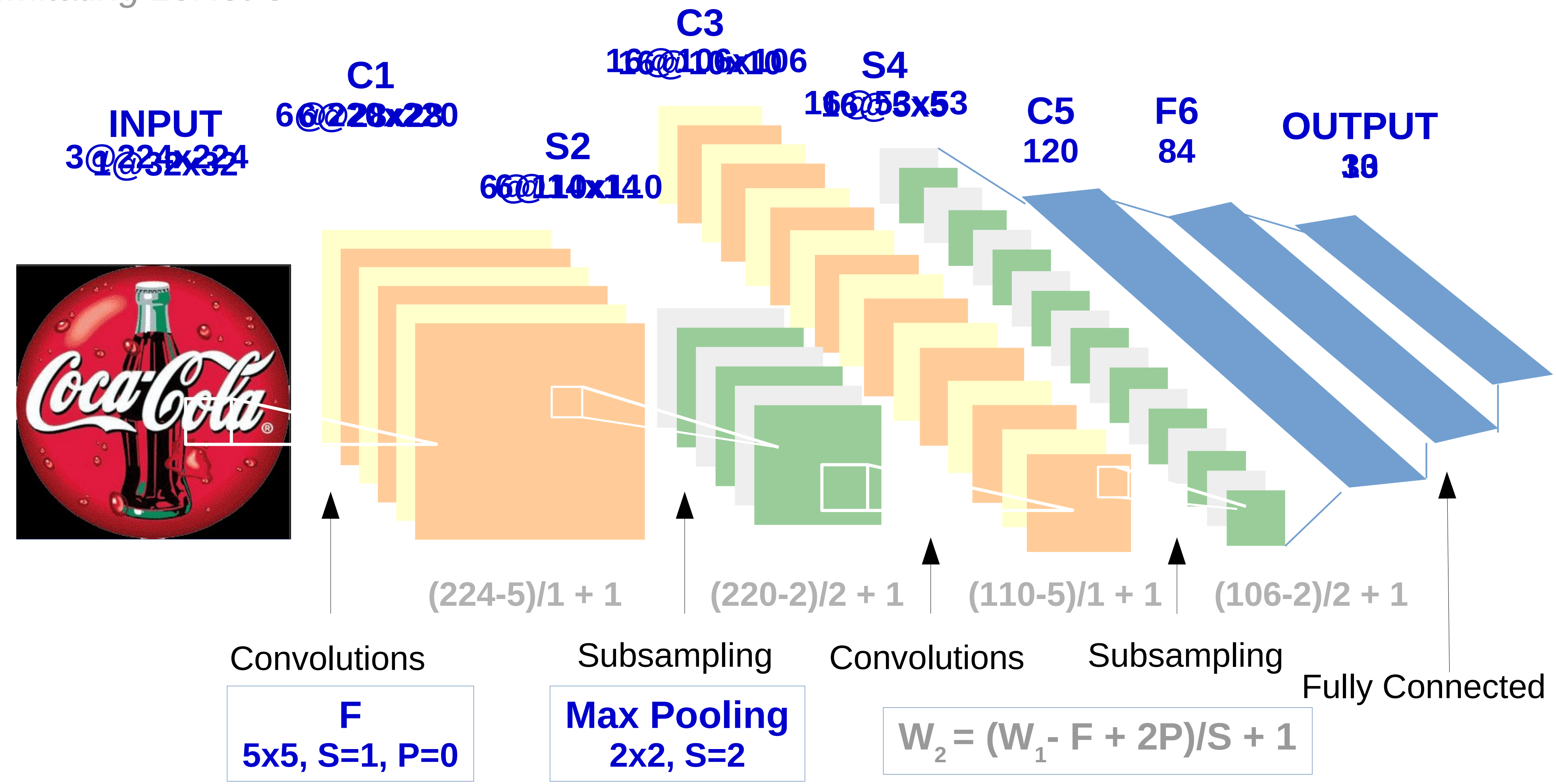
- Visualize datasets

```
import numpy as np
import matplotlib.pyplot as plt

def imshow(img):
    npimg = img.numpy().transpose((1, 2, 0))
    npimg = npimg * train_std + train_std    # denorm
    npimg = np.clip(npimg, 0, 1)
    plt.imshow(npimg)

imgs, labels = next(iter(dataloaders['train']))

img = torchvision.utils.make_grid(imgs[:4])
classes = datasets['train'].classes
print(', '.join(classes[i] for i in labels[:4]))
imshow(img)
```

```
guiness, no-logo, carlsberg, no-logo
```

3. Create Network

Imitating LeNet-5



Source: LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

3. Create Network

- Create network by subclass torch.nn.Module

```
import torch.nn as nn
import torch.nn.functional as F

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 53 * 53, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 33)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 53 * 53)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

3. Create Network

- cnn

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
cnn = CNN()
cnn = cnn.to(device)
```

```
Out[51]: CNN(
           (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
           (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
           (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
           (fc1): Linear(in_features=44944, out_features=120, bias=True)
           (fc2): Linear(in_features=120, out_features=84, bias=True)
           (fc3): Linear(in_features=84, out_features=33, bias=True)
         )
```

4. Train Network

- Define loss function and optimizer

```
import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(cnn.parameters(), lr=0.001, momentum=0.9)
```

4. Train Network

- Create train and validate function

```
def train_val(model, criterion, optimizer, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoc {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()
            running_loss = 0.0
            running_corrects = 0
            for inputs, labels in dataloaders[phase]:

                inputs, labels = inputs.to(device), labels.to(device)
```

4. Train Network

- Create train and validate function

```
def train_val(model, criterion, optimizer, num_epochs=25):
                                  ...
                for inputs, labels in dataloaders[phase]:
                    inputs, labels = inputs.to(device), labels.to(device)
                    optimizer.zero_grad()

                    with torch.set_grad_enabled(phase == 'train'):
                        # Forward pass
                        outputs = model(inputs)
                        _, preds = torch.max(outputs, 1)

                        # Compute loss
                        loss = criterion(outputs, labels)

                        # Compute gradients and update parameters if train
                        if phase == 'train':
                            loss.backward()
                            optimizer.step()
```

Source: *https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html*

4. Train Network

- Create train and validate function

```
def train_val(model, criterion, optimizer, num_epochs=25):
                                        ...
    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best Accuracy: {:.2f} %'.format(best_acc * 100))

    model.load_state_dict(best_model_wts)

    return model
```

Source: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

4. Train Network

- Train the network

```
model_cnn = train_val(cnn, criterion, optimizer)
```

```
Epoc 0/24
----------
Train Loss: 1.8010 Acc.: 75.71 %
Val Loss: 2.2365 Acc.: 60.98 %

Epoc 1/24
----------
Train Loss: 1.1586 Acc.: 82.42 %
Val Loss: 2.0667 Acc.: 60.98 %
 ...
Epoc 20/24
----------
Train Loss: 0.5419 Acc.: 86.98 %
Val Loss: 3.0452 Acc.: 61.87 %

 ...

Epoc 24/24
----------
Train Loss: 0.3455 Acc.: 92.14 %
Val Loss: 2.6079 Acc.: 57.72 %

Training complete in 11m 4s
Best Accuracy: 61.87 %
```

5. Evaluate

- Create test function to evaluate network on test set

```
def test(model):
    model.eval()
    running_corrects = 0
    with torch.no_grad():
        for inputs, labels in dataloaders['test']:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            running_corrects += torch.sum(preds == labels).item()

    test_acc = running_corrects / len(datasets['test'])
    print('Test Acc.: {:.2f} %'.format(test_acc * 100))

test(model_cnn)
```

```
Test Acc.: 75.91 %
```

Transfer Learning

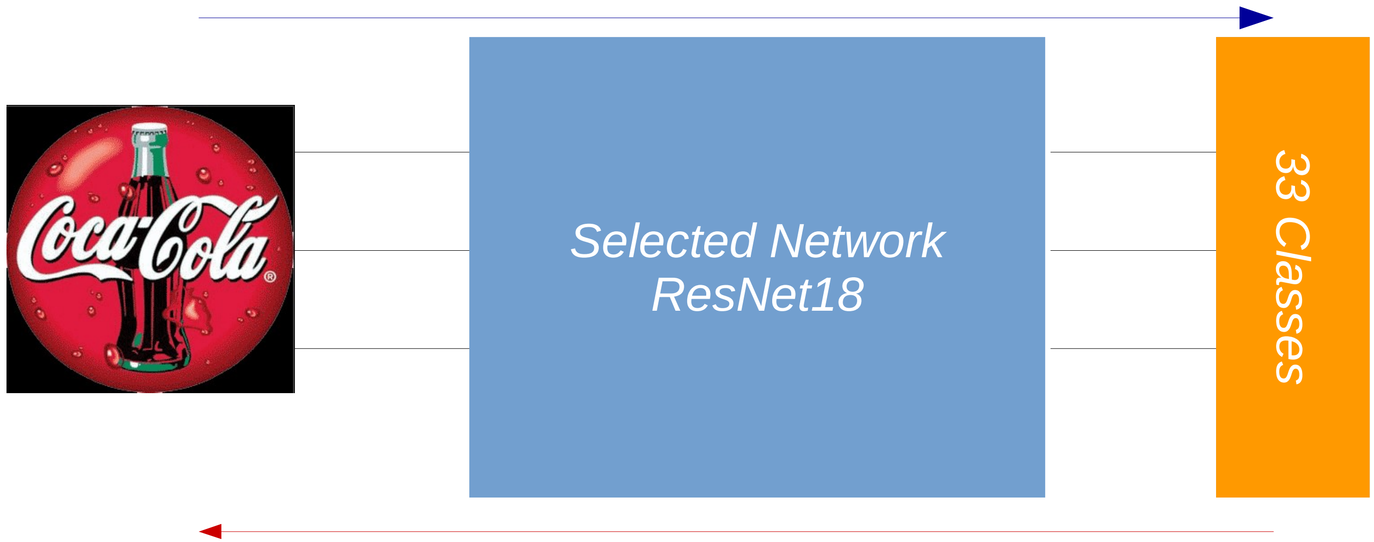
What's Transfer Learning?

"Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task."

Source: https://machinelearningmastery.com/transfer-learning-for-deep-learning/

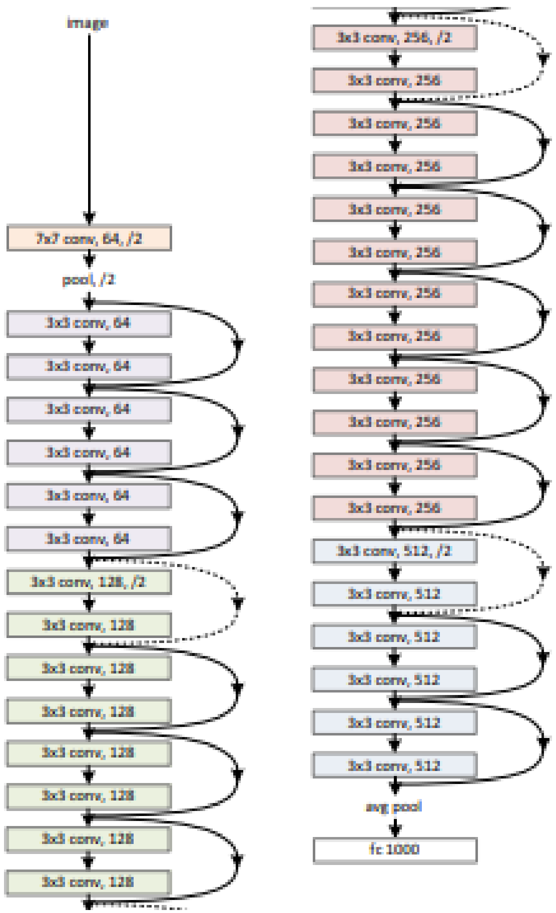
How?

1. Select network.
2. Match input format.
3. Replace output layer.
4. Retrain network



ResNet

34-layer residual



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Source: https://arxiv.org/pdf/1512.03385.pdf

Match Input Format

- Create matched transforms

```
train_mean = np.array([0.485, 0.456, 0.406])
train_std = np.array([0.229, 0.224, 0.225])
```

```
data_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(train_mean, train_std)])

datasets = {i : torchvision.datasets.ImageFolder(DATA_DIR / i, data_transforms)
            for i in SETS}

dataloaders = {i : DataLoader(datasets[i], batch_size=bz,
               shuffle=(i == 'train'), num_workers=4) for i in SETS}
```

Load Pretrained Network

- Load pretrained model. Define output layer (head).

```
model_ft = torchvision.models.resnet18(pretrained=True)
model_ft
```

```
Out[19]: ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(

 ...

    )
  )
  (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0)
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

Replace Output Layer

- Replace output layer.

```
num_ftrs = model_ft.fc.in_features                    # 512
model_ft.fc = nn.Linear(num_ftrs, 33)
model_ft = model_ft.to(device)
criterion = nn.CrossEntropyLoss()
lr = 0.001
optimizer_ft = optim.SGD(model_ft.parameters(), lr=lr, momentum=0.9)
```

Train Network

- Fine tune the network

```
model_ft = train_val(model_ft, criterion, optimizer_ft)
```

```
Epoc 0/24
----------
Train Loss: 1.0930 Acc.: 78.35 %
Val Loss: 1.6996 Acc.: 61.38 %

Epoc 1/24
----------
Train Loss: 0.6551 Acc.: 83.30 %
Val Loss: 1.6968 Acc.: 61.87 %
...

Epoc 22/24
----------
Train Loss: 0.0142 Acc.: 100.00 %
Val Loss: 1.1601 Acc.: 75.69 %

...

Epoc 24/24
----------
Train Loss: 0.0132 Acc.: 99.95 %
Val Loss: 1.1957 Acc.: 75.65 %

Training complete in 13m 9s
Best Accuracy: 75.69 %
```

Evaluate

- Evaluate network on test set

```
test(model_ft)
```

```
Test Acc.: 85.58 %
```

Thank you

All materials:

https://github.com/nithiroj/pycon-thailand-2018

Nithiroj Tripatarasit (Lek)

Email: nithiroj@gmail.com

Twitter: @nithiroj

URL: medium.com/@nithiroj

GitHub: https://github.com/nithiroj