Lecture 13

Introduction to Convolutional Neural Networks Part 3

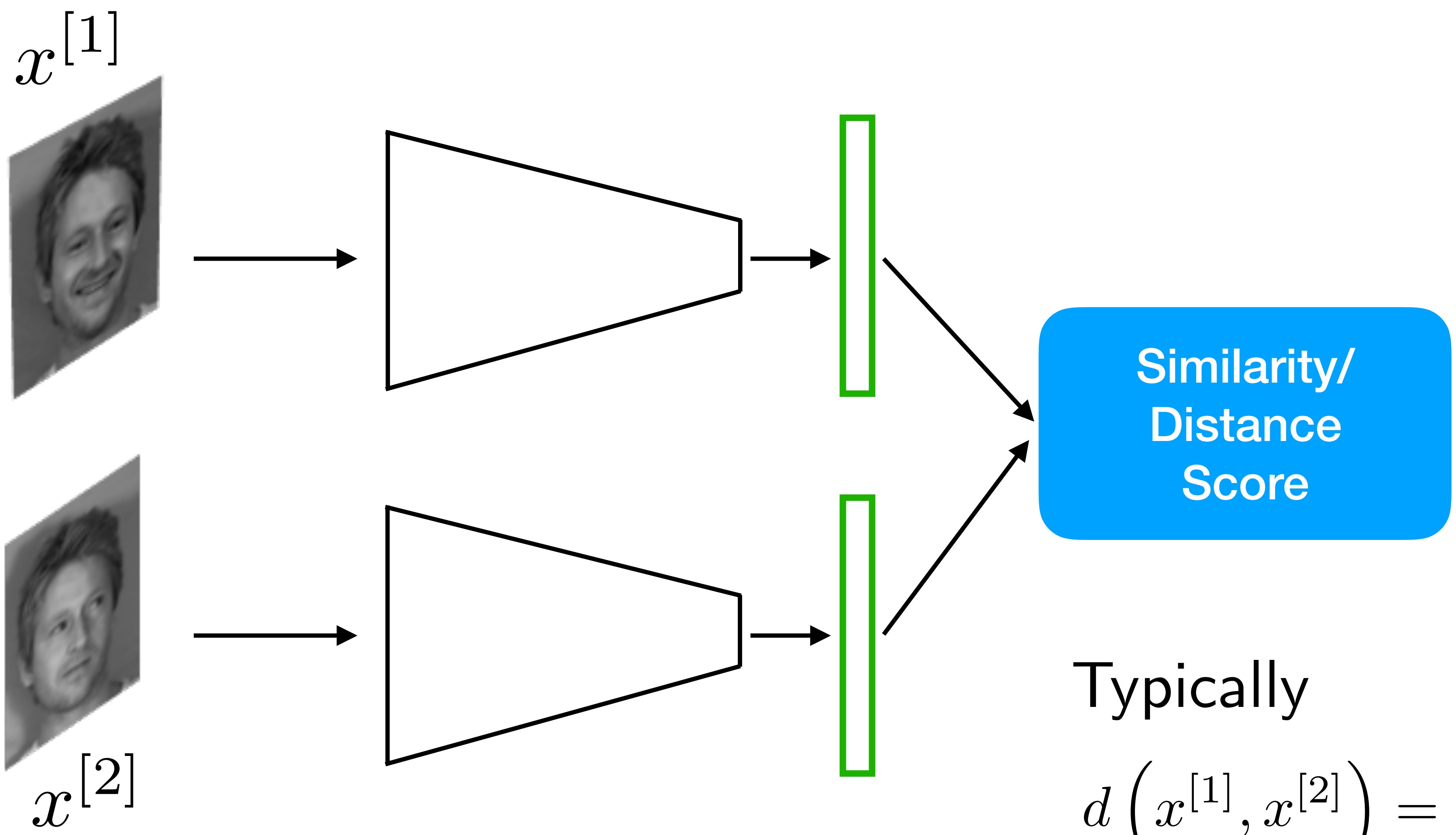
STAT 479: Deep Learning, Spring 2019

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat479-ss2019/

Face Recognition and Metric Learning

Siamese Networks



Typically

$$d\left(x^{[1]}, x^{[2]}\right) = \left\| f\left(x^{[1]}\right) - f\left(x^{[1]}\right) \right\|_2^2$$

or

$$d\left(x^{[1]}, x^{[2]}\right) = \left| f\left(x^{[1]}\right) - f\left(x^{[1]}\right) \right|_1$$

Siamese Networks

Often used for "One-shot learning"

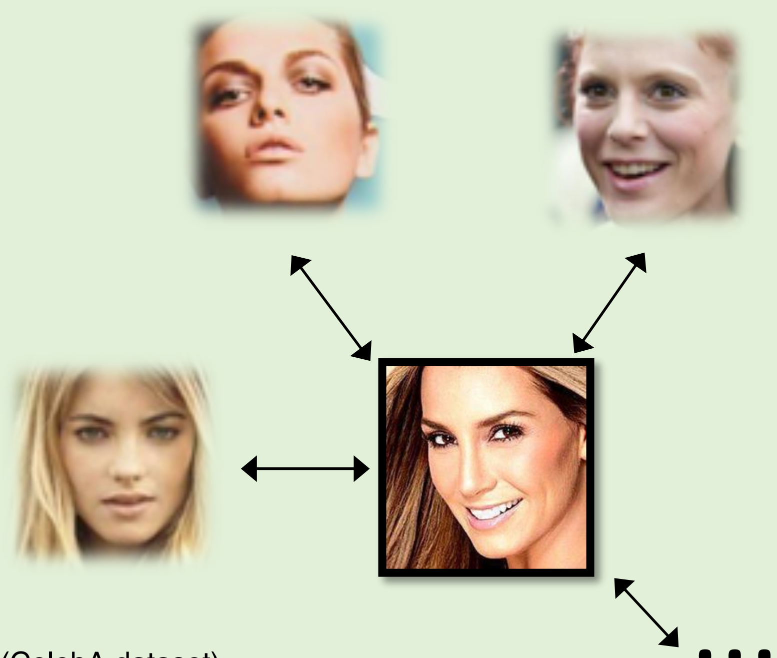
- Suppose you trained a Siamese network for verification tasks
- Now, suppose you have only ~1 object per class
- You can compare any new object to any object based on maximum similarity to your given images (somewhat related to K-nearest neighbors)

Face Recognition:
Face Identification vs Face Verification

A. Identification

Determine identity of an unknown person

1-to-*n* matching

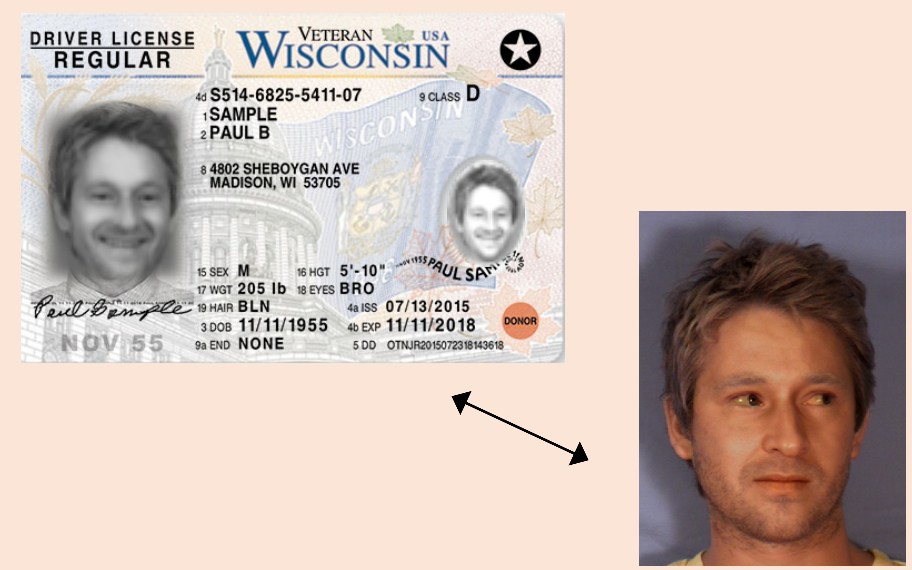


(CelebA dataset)

B. Verification

Verify claimed identity of a person

1-to-1 matching



(MUCT dataset)

dataset link: http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

dataset link: http://www.milbo.org/muct/

DeepFace

Taigman, Yaniv, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. "Deepface: Closing the gap to human-level performance in face verification." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701-1708. 2014.

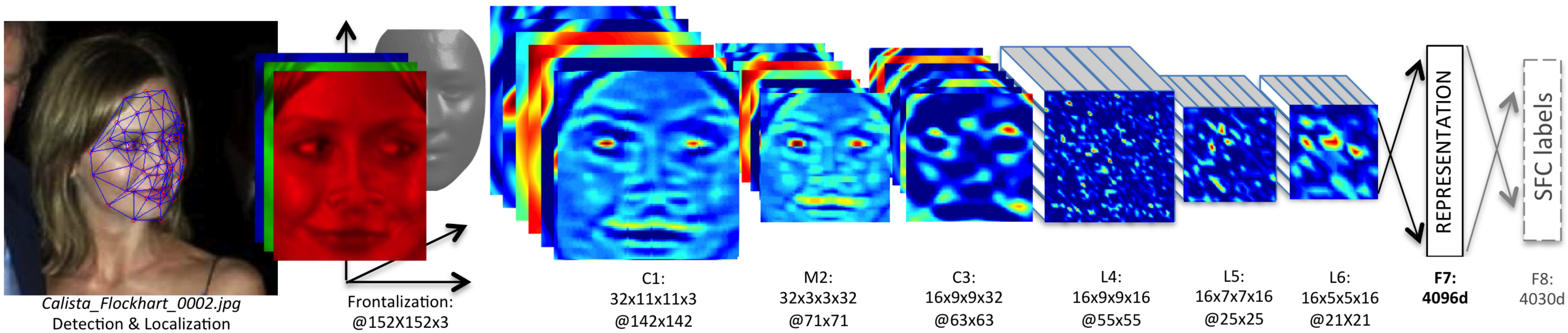
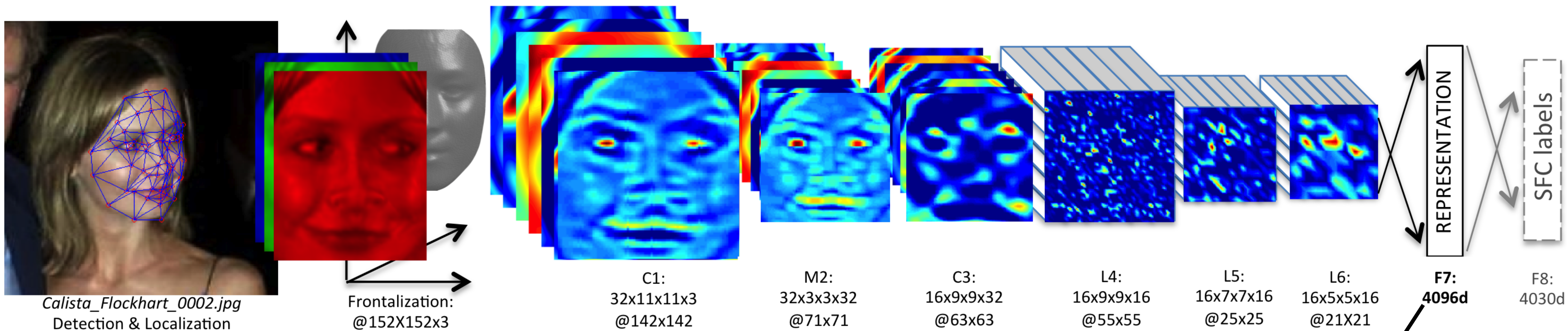


Figure 2. **Outline of the *DeepFace* architecture**. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate feature maps produced at each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

Hybrid between traditional methods and deep learning

DeepFace

Taigman, Yaniv, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. "Deepface: Closing the gap to human-level performance in face verification." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701-1708. 2014.



Given an image I, the representation $G(I)$ is then computed using the described feed-forward network. Any feed-forward neural network with L layers, can be seen as a composition of functions g_ϕ^l. In our case, the representation is: $G(I) = g_\phi^{F_7}(g_\phi^{L_6}(...g_\phi^{C_1}(T(I, \theta_T))...))$ with the net's parameters $\phi = \{C_1, ..., F_7\}$ and $\theta_T = \{x_{2d}, \vec{P}, \vec{r}\}$ as described in Section 2.

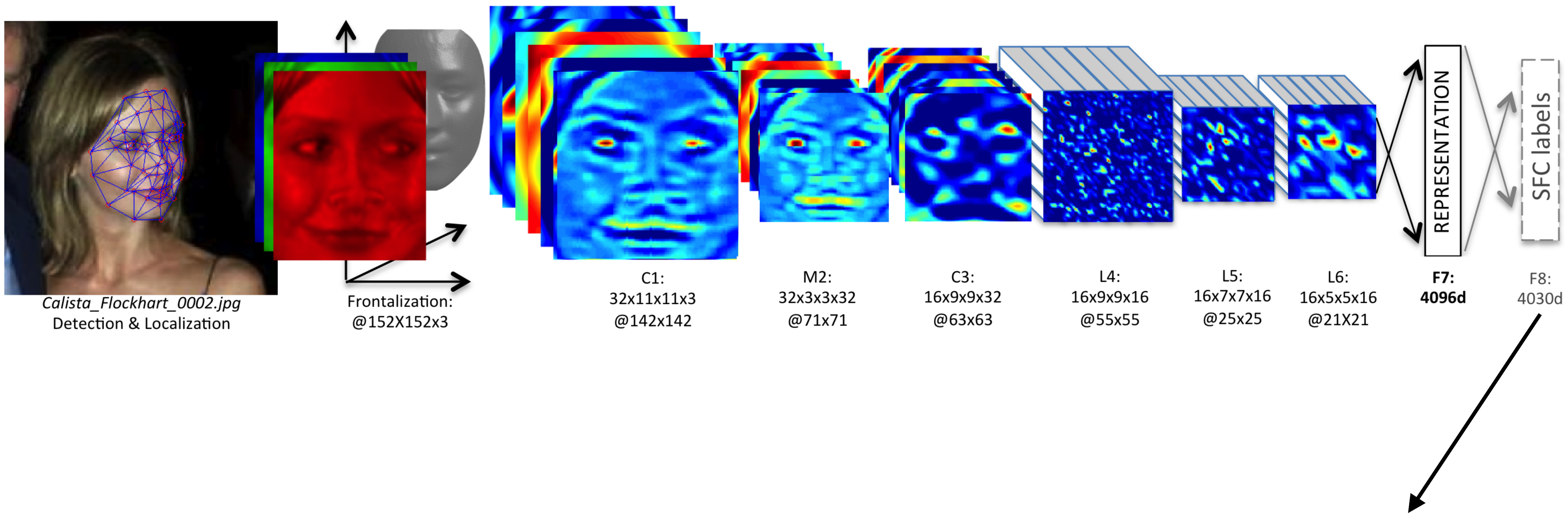
Normaliaztion As a final stage we normalize the features to be between zero and one in order to reduce the sensitivity to illumination changes: Each component of the feature vector is divided by its largest value across the training set. This is then followed by L_2-normalization: $f(I) := \bar{G}(I)/||\bar{G}(I)||_2$ where $\bar{G}(I)_i = G(I)_i / \max(G_i, \epsilon)$ [3]. Since we employ ReLU activations, our system is not invariant to re-scaling of the image intensities. Without bi-

[2] See the **supplementary** material for more details.
[3] $\epsilon = 0.05$ in order to avoid division by a small number.

normalized feature vectors

DeepFace - Face Recognition

Taigman, Yaniv, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. "Deepface: Closing the gap to human-level performance in face verification." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701-1708. 2014.

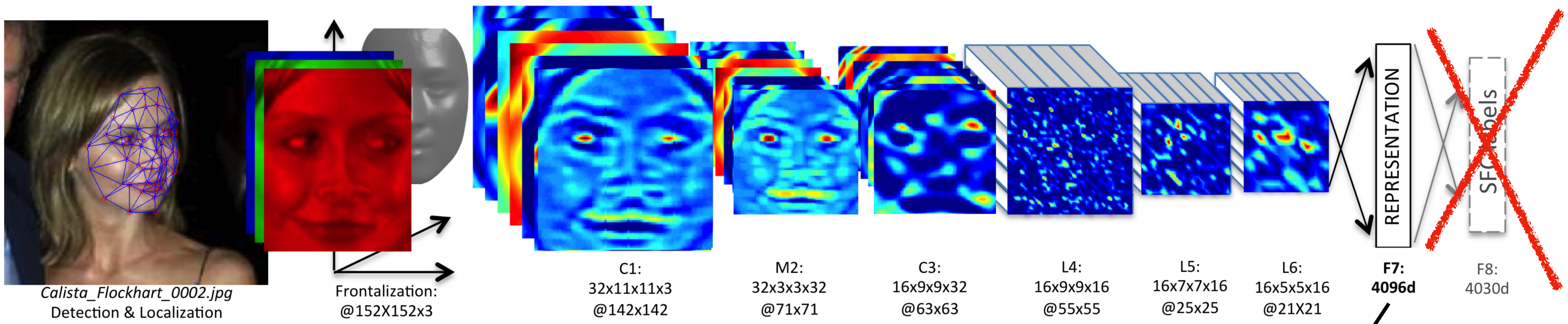


Regular softmax ouput layer for classifying faces (face IDs) optimized via cross-entropy loss.

Note they have 1-4k classes (they achieved a classification accuracy of ~93%).

DeepFace - Face Verification

Taigman, Yaniv, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. "Deepface: Closing the gap to human-level performance in face verification." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701-1708. 2014.



Weighted chi-square distance + SVM classifier for binary classification (predict whether two images depict the same person)

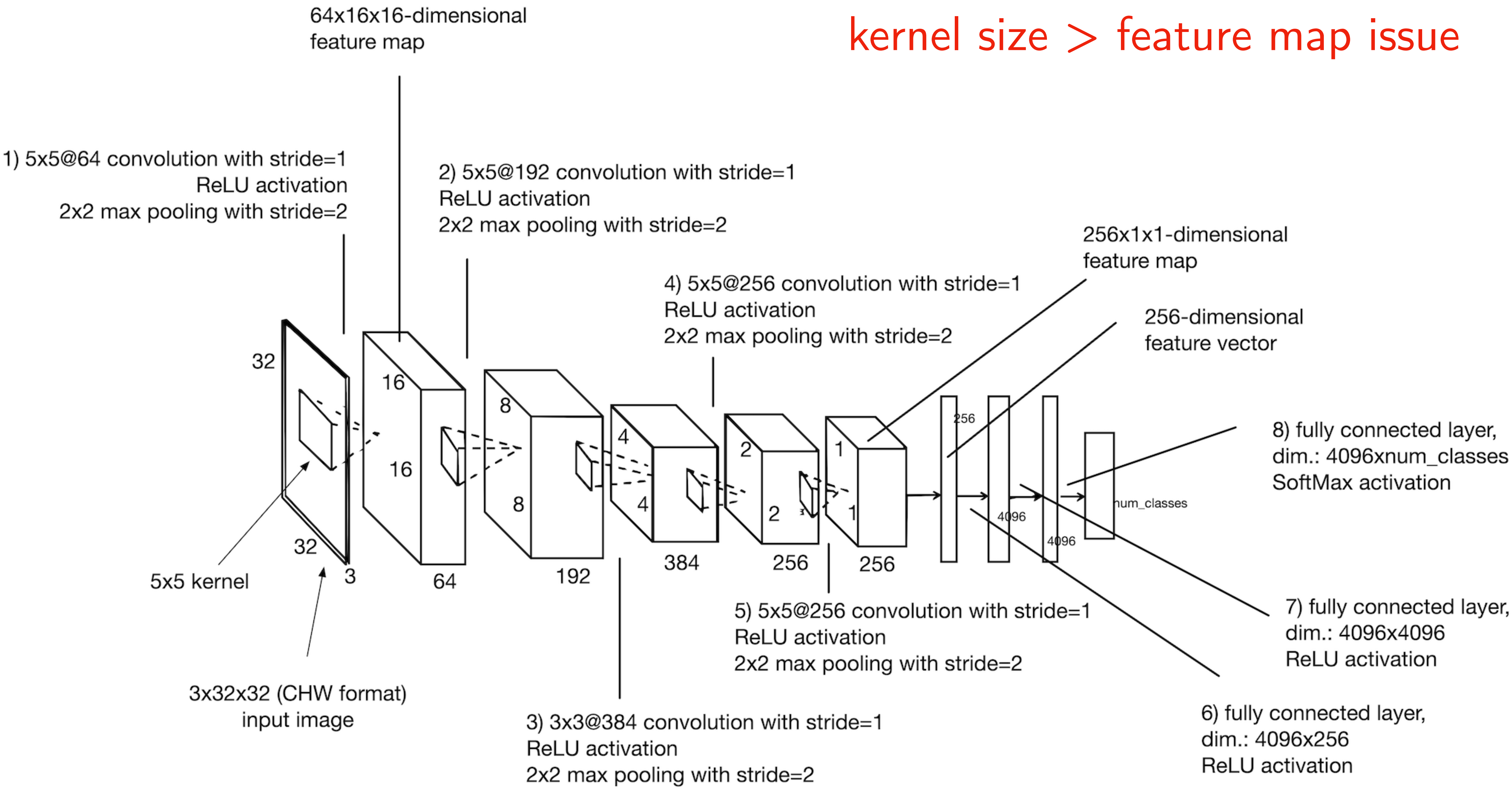
$$\chi^2(f_1, f_2) = \sum_i w_i \left(f_1|i] - f_2[i]\right)^2 / \left(f_1[i] + f_2[i]\right)$$

You may know this from other stats classes for comparing discrete probability distributions (histograms)

The weight is learned by the SVM.
(They achieved a classification accuracy is ~97%.)

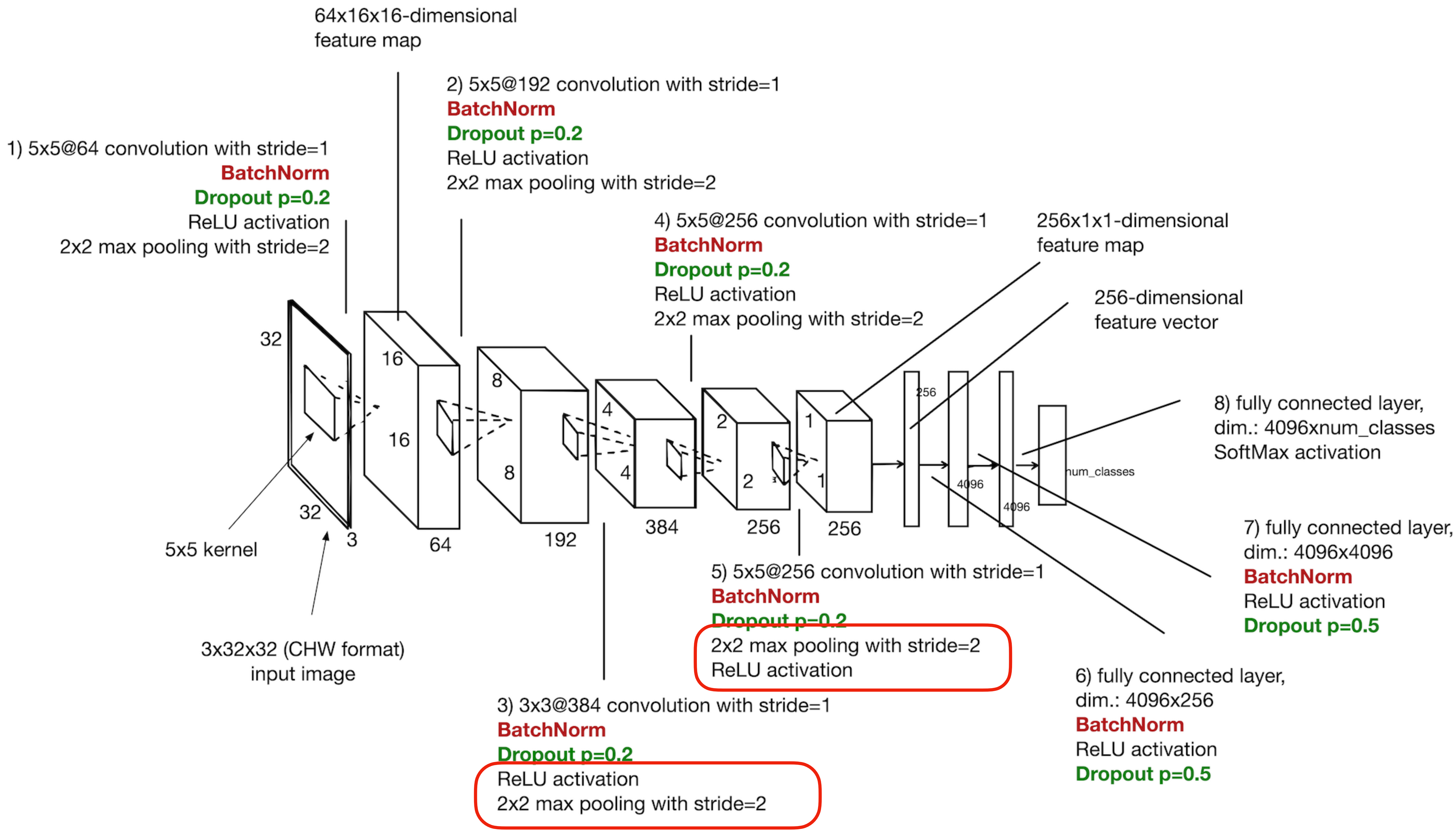
HW Clarification 1

kernel size > feature map issue



HW Clarification 2

ReLU -- MaxPool Order



HW Clarification 3

```python
    # 14x14x4 => 14x14x8
    self.conv_2 = torch.nn.Conv2d(in_channels=4,
                                  out_channels=8,
                                  kernel_size=(3, 3),
                                  stride=(1, 1),
                                  padding=1) # (1(14-1) - 14 + 3) / 2 = 1
    # 14x14x8 => 7x7x8
    self.pool_2 = torch.nn.MaxPool2d(kernel_size=(2, 2),
                                     stride=(2, 2),
                                     padding=0) # (2(7-1) - 14 + 2) = 0

    self.linear_1 = torch.nn.Linear(7*7*8, num_classes)


def forward(self, x):
    out = self.conv_1(x)
    out = F.relu(out)
    out = self.pool_1(out)

    out = self.conv_2(out)
    out = F.relu(out)
    out = self.pool_2(out)

    logits = self.linear_1(out.view(-1, 7*7*8))
    probas = F.softmax(logits, dim=1)
    return logits, probas
```

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/cnn-with-diff-init/default.ipynb

HW Clarification 3

From PyTorch Lecture (Lecture 6):

1)

```
class MultilayerPerceptron(torch.nn.Module):

    def __init__(self, num_features, num_classes):
        super(MultilayerPerceptron, self).__init__()

        self.my_network = torch.nn.Sequential(
            torch.nn.Linear(num_features, num_hidden
            torch.nn.ReLU(),
            torch.nn.Linear(num_hidden_1, num_hidden
            torch.nn.ReLU(),
            torch.nn.Linear(num_hidden_2, num_classe
        )

    def forward(self, x):
        logits = self.my_network(x)
        probas = F.softmax(logits, dim=1)
        return logits, probas
```

Much more compact and clear, but "forward" may be harder to debug if there are errors (we cannot simply add breakpoints or insert "print" statements

2) However, if you use Sequential, you can define "hooks" to get intermediate outputs.
For example:

```
[7]: model.net
```

```
[7]: Sequential(
       (0): Linear(in_features=784, out_features=128, bias=True)
       (1): ReLU(inplace)
       (2): Linear(in_features=128, out_features=256, bias=True)
       (3): ReLU(inplace)
       (4): Linear(in_features=256, out_features=10, bias=True)
     )
```

[ ]: If we want to get the output from the 2nd layer during the forward pass, we can register a hook as follows:

```
[8]: outputs = []
     def hook(module, input, output):
         outputs.append(output)

     model.net[2].register_forward_hook(hook)
```

```
[8]: <torch.utils.hooks.RemovableHandle at 0x7f659c6685c0>
```

Now, if we call the model on some inputs, it will save the intermediate results in the "outputs" list:

```
[9]: _ = model(features)

     print(outputs)
```

```
[tensor([[0.5341, 1.0513, 2.3542,  ..., 0.0000, 0.0000, 0.0000],
         [0.0000, 0.6676, 0.6620,  ..., 0.0000, 0.0000, 2.4056],
         [1.1520, 0.0000, 0.0000,  ..., 2.5860, 0.8992, 0.9642],
         ...,
         [0.0000, 0.1076, 0.0000,  ..., 1.8367, 0.0000, 2.5203],
         [0.5415, 0.0000, 0.0000,  ..., 2.7968, 0.8244, 1.6335],
         [1.0710, 0.9805, 3.0103,  ..., 0.0000, 0.0000, 0.0000]],
       device='cuda:3', grad_fn=<ThresholdBackward1>)]
```

HW Clarification 4

Model Settings

In [3]:
```
##############################
### NO NEED TO CHANGE THIS CELL
##############################



#-------------------------
### SETTINGS
#-------------------------

# Hyperparameters
RANDOM_SEED = 1
LEARNING_RATE = 0.001
BATCH_SIZE = 256
NUM_EPOCHS = 20

# Architecture
NUM_FEATURES = 32*32
NUM_CLASSES = 10

# Other
DEVICE = "cuda:0"
```

HW Clarification 5

23 minutes ago

It seems that the model simply takes too much memory.

I tried various numbers but the memory seems to always run out at some point in training

FaceNet - Face Verification

Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "[Facenet: A unified embedding for face recognition and clustering]." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815-823. 2015.

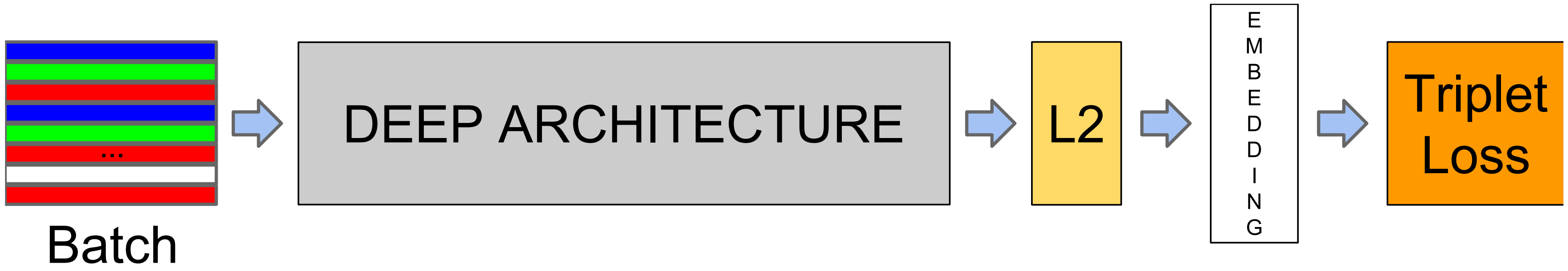


Figure 2. **Model structure.** Our network consists of a batch input layer and a deep CNN followed by L_2 normalization, which results in the face embedding. This is followed by the triplet loss during training.

FaceNet - Face Verification

Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "[Facenet: A unified embedding for face recognition and clustering]." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815-823. 2015.

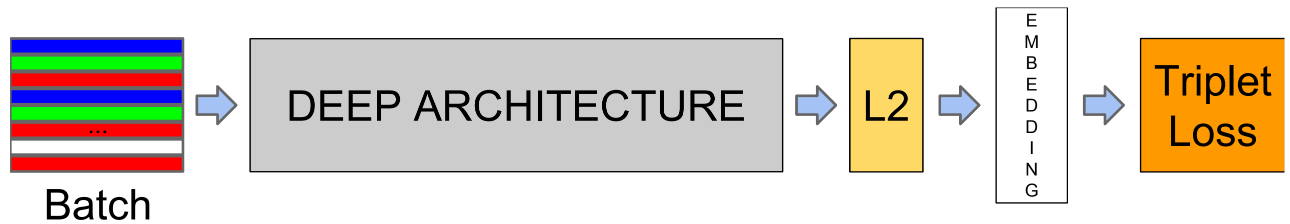


Figure 2. **Model structure.** Our network consists of a batch input layer and a deep CNN followed by L_2 normalization, which results in the face embedding. This is followed by the triplet loss during training.

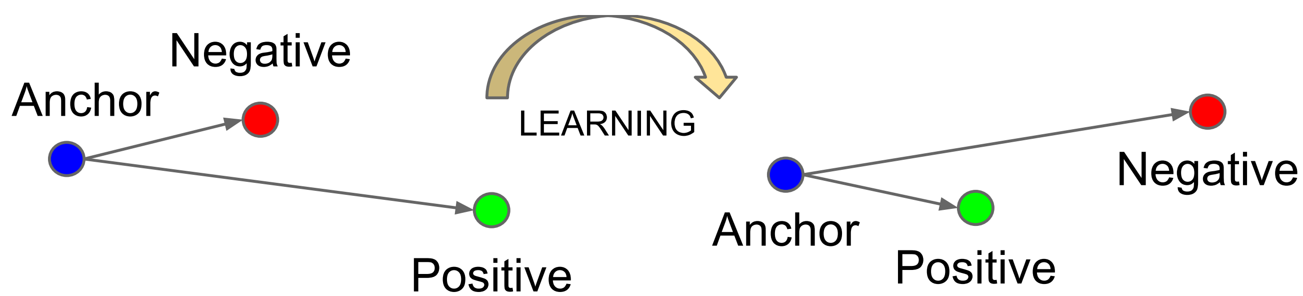


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

Triplet Loss


Anchor Positive

Want encodings to be very similar
(small distance)

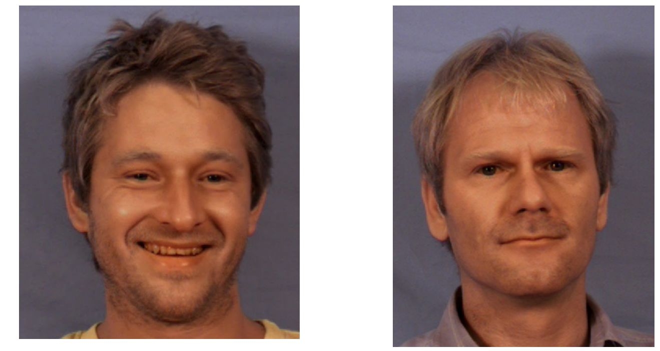

Anchor Negative

Want encodings to be very different
(large distance)

Triplet Loss

Anchor Positive

Want encodings to be very similar
(small distance)



Anchor Negative

Want encodings to be very different
(large distance)

$$d(A, P) \leq d(A, N)$$

$$\|f(A) - f(P)\|_2^2 \leq \|f(A) - f(N)\|_2^2$$

Triplet Loss

Anchor Positive

Want encodings to be very similar
(small distance)

Anchor Negative

Want encodings to be very different
(large distance)

$$d(A, P) + \alpha \leq d(A, N)$$

$$\|f(A) - f(P)\|_2^2 + \boxed{\alpha} \leq \|f(A) - f(N)\|_2^2$$

To make it a little harder

Triplet Loss


Anchor Positive

Want encodings to be very similar
(small distance)


Anchor Negative

Want encodings to be very different
(large distance)

$$d(A, P) + \alpha \leq d(A, N)$$

$$\|f(A) - f(P)\|_2^2 + \alpha \leq \|f(A) - f(N)\|_2^2$$

Rearrange

$$\boxed{\|f(A) - f(P)\|_2^2 + \alpha - \|f(A) - f(N)\|_2^2 \leq 0}$$

Triplet Loss

Anchor Positive

Want encodings to be very similar
(small distance)

Anchor Negative

Want encodings to be very different
(large distance)

Bounded loss function for training:

$$\mathcal{L}(A, P, N) = \max\left(\|f(A) - f(P)\|_2^2 + \alpha - \|f(A) - f(N)\|_2^2, 0\right)$$

Triplet Loss

Anchor Positive

Want encodings to be very similar
(small distance)

Anchor Negative

Want encodings to be very different
(large distance)

<u>In practice:</u> Selecting good pairs (those that are "hard")
is crucial during training

$$\mathcal{L}(A, P, N) = \max\left(\|f(A) - f(P)\|_2^2 + \alpha - \|f(A) - f(N)\|_2^2, 0\right)$$

Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815-823. 2015.

Architecture used

layer	**size-in**	**size-out**	**kernel**	**param**	**FLPS**
conv1	$220\times220\times3$	$110\times110\times64$	$7\times7\times3, 2$	9K	115M
pool1	$110\times110\times64$	$55\times55\times64$	$3\times3\times64, 2$	0	
rnorm1	$55\times55\times64$	$55\times55\times64$		0	
conv2a	$55\times55\times64$	$55\times55\times64$	$1\times1\times64, 1$	4K	13M
conv2	$55\times55\times64$	$55\times55\times192$	$3\times3\times64, 1$	111K	335M
rnorm2	$55\times55\times192$	$55\times55\times192$		0	
pool2	$55\times55\times192$	$28\times28\times192$	$3\times3\times192, 2$	0	
conv3a	$28\times28\times192$	$28\times28\times192$	$1\times1\times192, 1$	37K	29M
conv3	$28\times28\times192$	$28\times28\times384$	$3\times3\times192, 1$	664K	521M
pool3	$28\times28\times384$	$14\times14\times384$	$3\times3\times384, 2$	0	
conv4a	$14\times14\times384$	$14\times14\times384$	$1\times1\times384, 1$	148K	29M
conv4	$14\times14\times384$	$14\times14\times256$	$3\times3\times384, 1$	885K	173M
conv5a	$14\times14\times256$	$14\times14\times256$	$1\times1\times256, 1$	66K	13M
conv5	$14\times14\times256$	$14\times14\times256$	$3\times3\times256, 1$	590K	116M
conv6a	$14\times14\times256$	$14\times14\times256$	$1\times1\times256, 1$	66K	13M
conv6	$14\times14\times256$	$14\times14\times256$	$3\times3\times256, 1$	590K	116M
pool4	$14\times14\times256$	$7\times7\times256$	$3\times3\times256, 2$	0	
concat	$7\times7\times256$	$7\times7\times256$		0	
fc1	$7\times7\times256$	$1\times32\times128$	maxout p=2	103M	103M
fc2	$1\times32\times128$	$1\times32\times128$	maxout p=2	34M	34M
fc7128	$1\times32\times128$	$1\times1\times128$		524K	0.5M
L2	$1\times1\times128$	$1\times1\times128$		0	
total				140M	1.6B

Table 1. **NN1.** This table show the structure of our Zeiler&Fergus [22] based model with 1×1 convolutions inspired by [9]. The input and output sizes are described in $rows \times cols \times \#filters$. The kernel is specified as $rows \times cols, stride$ and the maxout [6] pooling size as $p = 2$.

FaceNet - Results

Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815-823. 2015.

jpeg q	val-rate
10	67.3%
20	81.4%
30	83.9%
50	85.5%
70	86.1%
90	86.5%

#pixels	val-rate
1,600	37.8%
6,400	79.5%
14,400	84.5%
25,600	85.7%
65,536	86.4%

Table 4. **Image Quality.** The table on the left shows the effect on the validation rate at 10E-3 precision with varying JPEG quality. The one on the right shows how the image size in pixels effects the validation rate at 10E-3 precision. This experiment was done with NN1 on the first split of our test hold-out dataset.

FaceNet - Results

Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815-823. 2015.

#dims	VAL
64	$86.8\% \pm 1.7$
128	$87.9\% \pm 1.9$
256	$87.7\% \pm 1.9$
512	$85.6\% \pm 2.0$

Table 5. **Embedding Dimensionality.** This Table compares the effect of the embedding dimensionality of our model NN1 on our hold-out set from section 4.1. In addition to the VAL at 10E-3 we also show the standard error of the mean computed across five splits.

FaceNet - Results

Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815-823. 2015.

#training images	VAL
2,600,000	76.3%
26,000,000	85.1%
52,000,000	85.1%
260,000,000	86.2%

Table 6. **Training Data Size.** This table compares the performance after 700h of training for a smaller model with 96x96 pixel inputs. The model architecture is similar to NN2, but without the 5x5 convolutions in the Inception modules.

FaceNet - Results

Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815-823. 2015.

#dims	VAL
64	$86.8\% \pm 1.7$
128	$87.9\% \pm 1.9$
256	$87.7\% \pm 1.9$
512	$85.6\% \pm 2.0$

Table 5. **Embedding Dimensionality.** This Table compares the effect of the embedding dimensionality of our model NN1 on our hold-out set from section 4.1. In addition to the VAL at 10E-3 we also show the standard error of the mean computed across five splits.

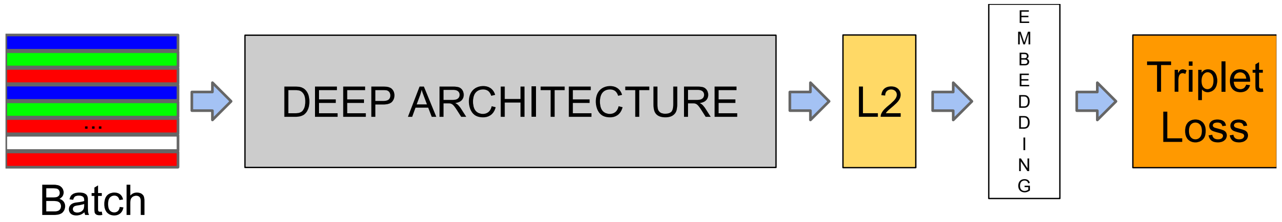


Figure 2. **Model structure.** Our network consists of a batch input layer and a deep CNN followed by L_2 normalization, which results in the face embedding. This is followed by the triplet loss during training.

Suppose we have 2 L2-normalized vectors:

$$\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2 = 1$$

The squared L2 distance is then proportional to the cosine similarity

$$\begin{aligned}
\|\mathbf{x} - \mathbf{y}\|_2^2 &= (\mathbf{x} - \mathbf{y})^\top (\mathbf{x} - \mathbf{y}) \\
&= \mathbf{x}^\top \mathbf{x} - 2\mathbf{x}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y} \\
&= 2 - 2\mathbf{x}^\top \mathbf{y} \\
&= 2 - 2\cos(\mathbf{x}, \mathbf{y}) \qquad \text{where } \cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} \in [-1, 1]
\end{aligned}$$

Optional: Recent Triplet Loss Variants

(not required), only for those who are interested

- Cosine Similarity-based triplet loss:

Li, Chao, Xiaokong Ma, Bing Jiang, Xiangang Li, Xuewei Zhang, Xiao Liu, Ying Cao, Ajay Kannan, and Zhenyao Zhu. "Deep speaker: an end-to-end neural speaker embedding system." *arXiv preprint arXiv:1705.02304* (2017).

- Angular Loss:

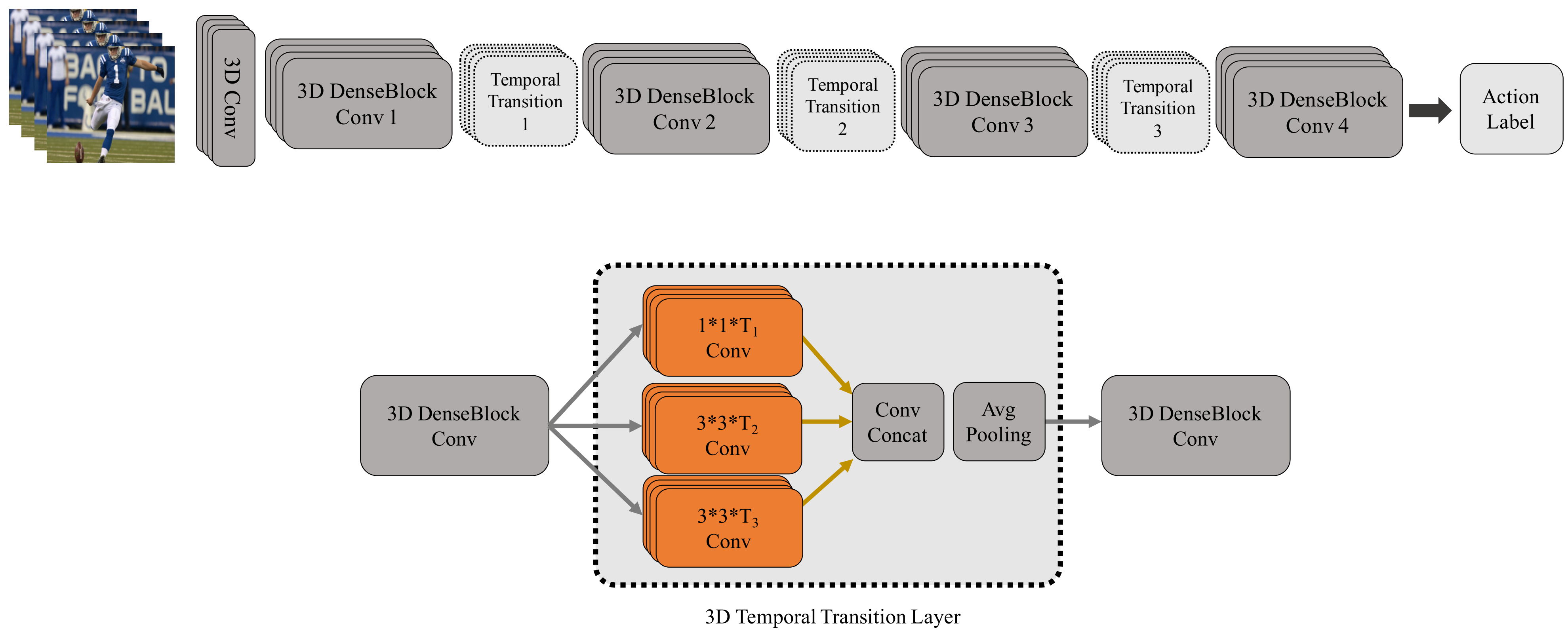
Wang, Jian, Feng Zhou, Shilei Wen, Xiao Liu, and Yuanqing Lin. "Deep metric learning with angular loss." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2593-2601. 2017.

- Large margin cosine loss:

Wang, Hao, Yitong Wang, Zheng` Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. "Cosface: Large margin cosine loss for deep face recognition." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5265-5274. 2018.

Additional Concepts to Wrap Up the Intro to Convolutional Neural Networks

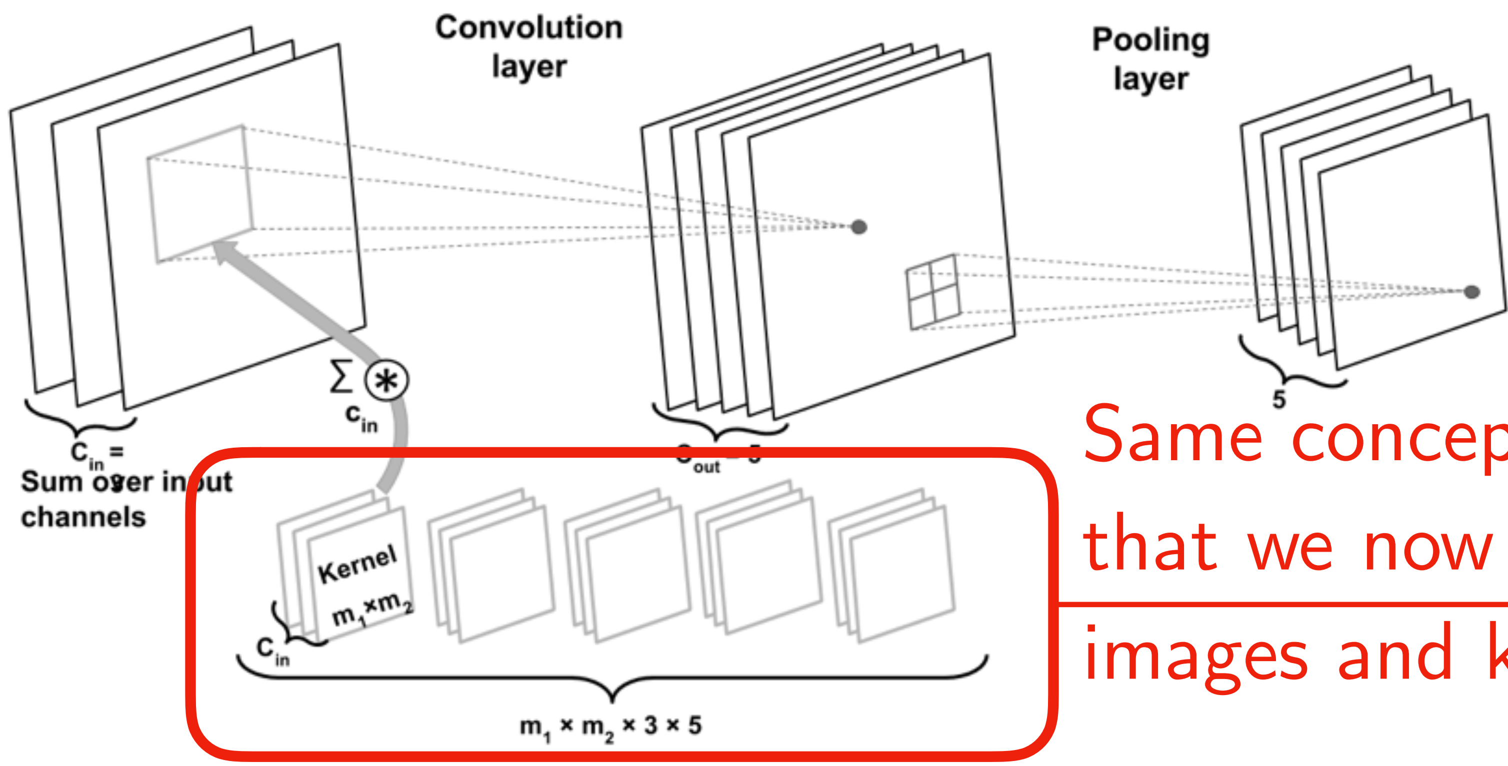
ConvNets and 3D Inputs



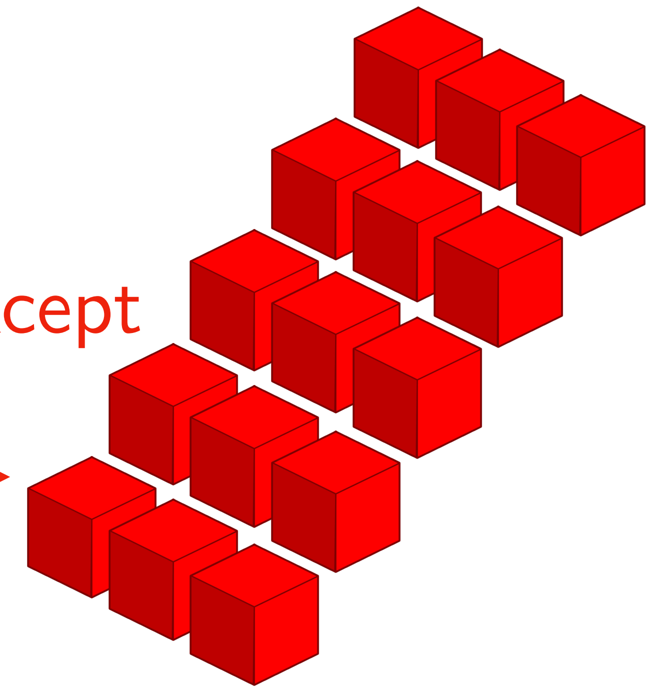
Diba, Ali, Mohsen Fayyaz, Vivek Sharma, Amir Hossein Karami, Mohammad Mahdi Arzani, Rahman Yousefzadeh, and Luc Van Gool. "Temporal 3d convnets: New architecture and transfer learning for video classification." *arXiv preprint arXiv: 1711.08200* (2017).

Also very popular for Medical Imaging (MRI, CT scans ...)

ConvNets and 3D Inputs



Same concept as before except that we now have 3D images and kernels



$$\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times c_{in}}$$

$$\mathbf{W} \in \mathbb{R}^{m_1 \times m_2 \times c_{in} \times c_{out}} \quad \mathbf{b} \in \mathbb{R}^{c_{out}}$$

ConvNets and 3D Inputs

Usage is similar to Conv2d, except that we now have 3 dimensional kernels

Conv3d

CLASS `torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)` [SOURCE]

Applies a 3D convolution over an input signal composed of several input planes.

https://pytorch.org/docs/stable/nn.html?highlight=conv3d#torch.nn.functional.conv3d

[1]:
```
import torch
import torch.nn as nn
```

[2]:
```
m = nn.Conv3d(16, 33, 3, stride=2)
m = nn.Conv3d(16, 33, (3, 5, 2), stride=(2, 1, 1), padding=(4, 2, 0))
input = torch.randn(20, 16, 10, 50, 100)
output = m(input)
```

[3]:
```
input.size()
```

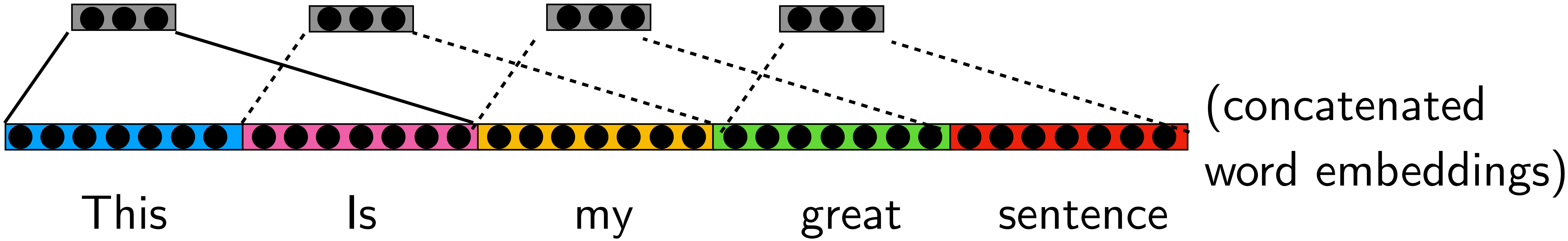
[3]:
```
torch.Size([20, 16, 10, 50, 100])
```

[4]:
```
output.size()
```

[4]:
```
torch.Size([20, 33, 8, 50, 99])
```

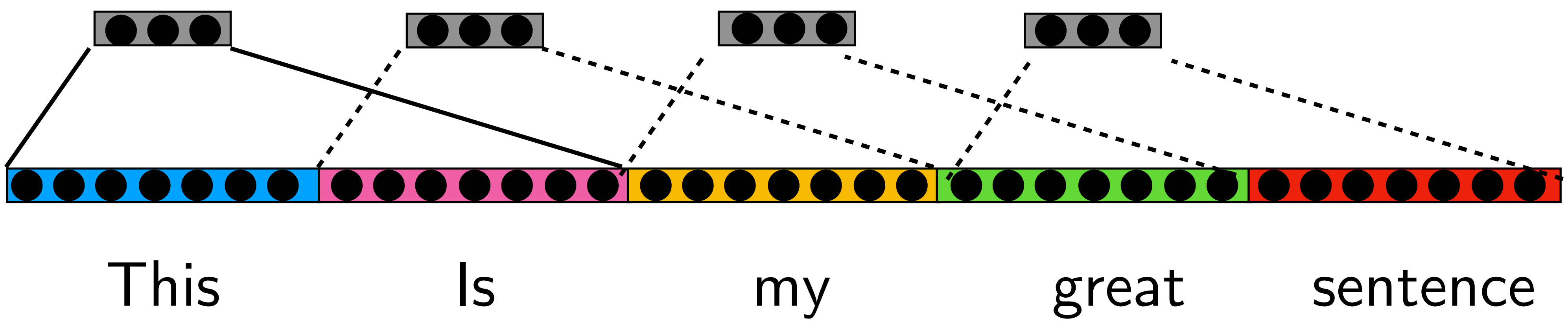
ConvNets for Text with 1D Convolutions

We can think of text as image with width 1



ConvNets for Text with 1D Convolutions

We can think of text as image with width 1



(concatenated word embeddings)

https://pytorch.org/docs/stable/nn.html#conv1d

Conv1d

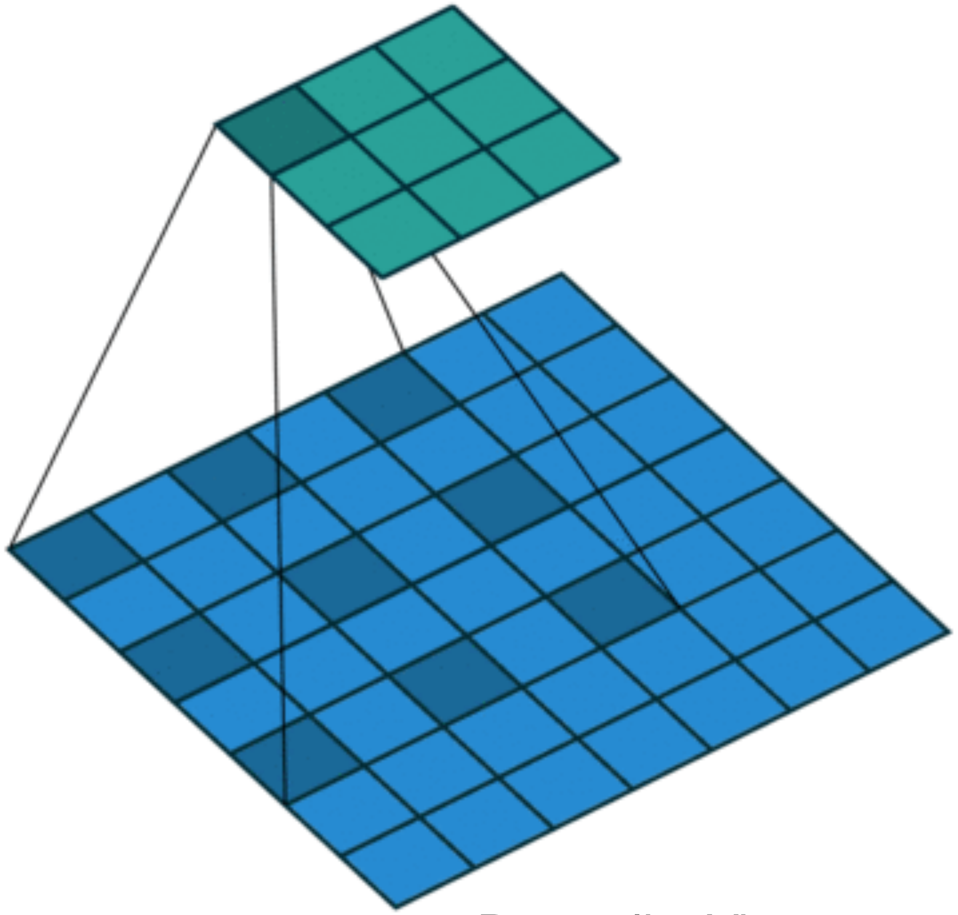
CLASS `torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)` [SOURCE]

Applies a 1D convolution over an input signal composed of several input planes.

Dilated Convolutions

$$o = \left\lfloor \frac{i + 2p - k - (k-1)(d-1)}{s} \right\rfloor + 1$$

A 2-dilated 2D convolution



Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

CNNs for Text (with 2D Convolutions)

Good results have also been achieved by representing a sentence as a matrix of word vectors and applying 2D convolutions (where each filter uses a different kernel size)

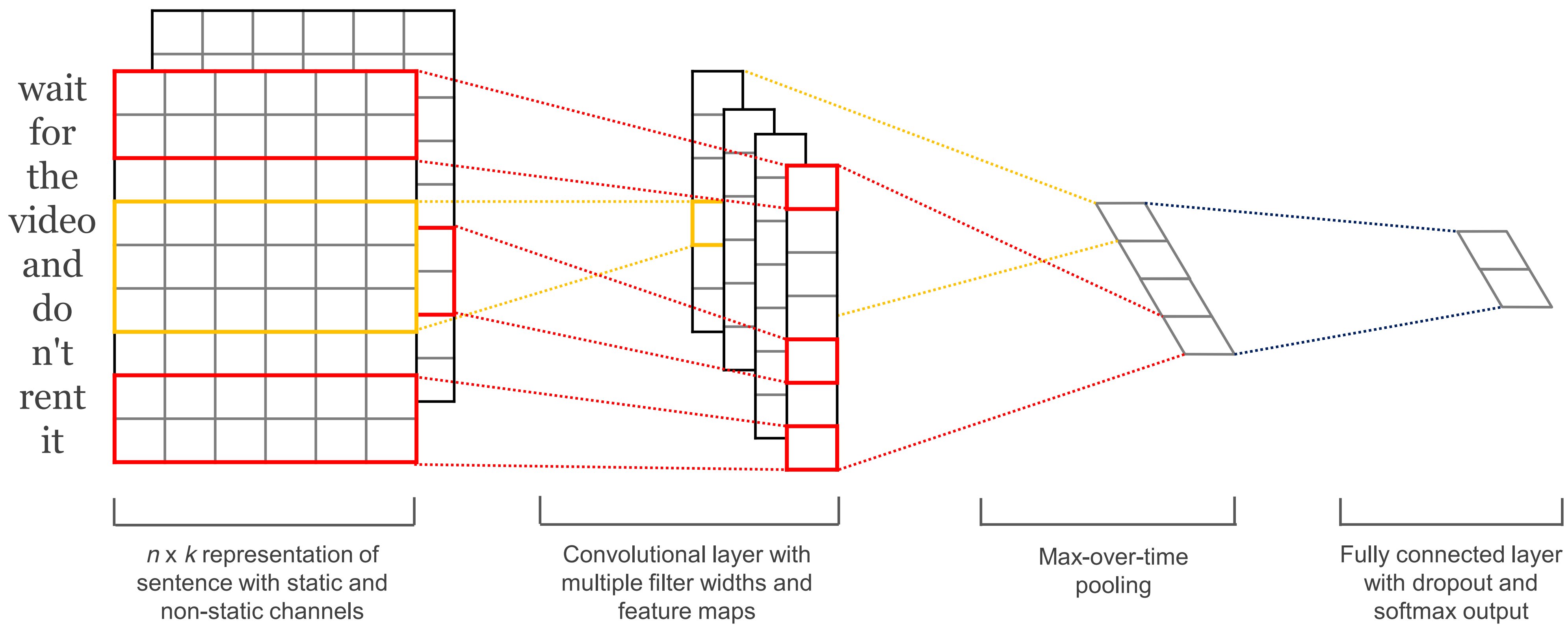


Figure 1: Model architecture with two channels for an example sentence.

Kim, Y. (2014). [Convolutional neural networks for sentence classification](). *arXiv preprint arXiv:1408.5882*.

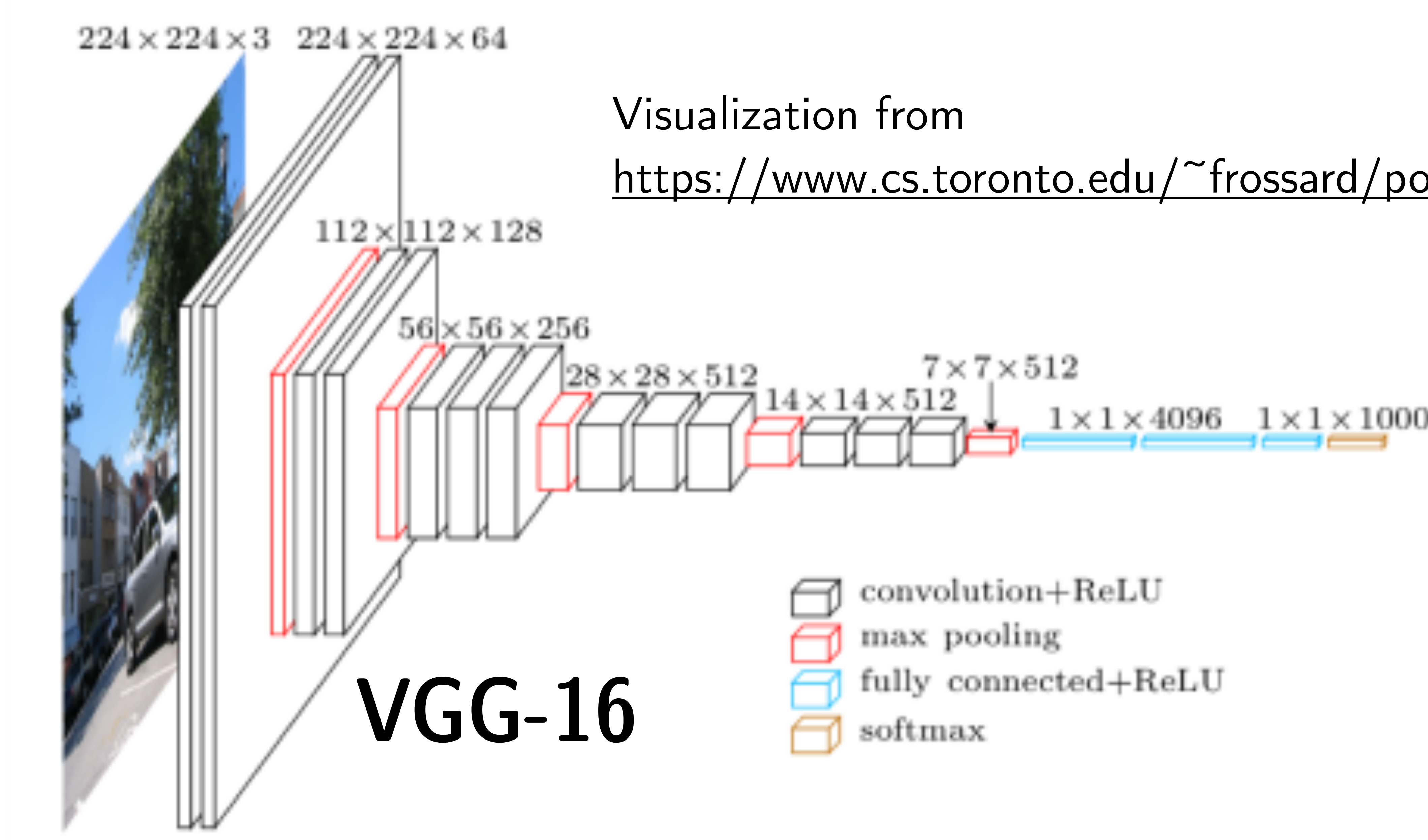
Transfer Learning

- A technique that may be useful for your class projects
- Key idea:
  - ✦ Feature extraction layers may be generally useful
  - ✦ Use a pre-trained model (e.g., pretrained on ImageNet)
  - ✦ Freeze the weights: Only train last layer (or last few layers)
- Related approach: Finetuning, train a pre-trained network on your smaller dataset

Transfer Learning

PyTorch implementation: https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/vgg16.ipynb

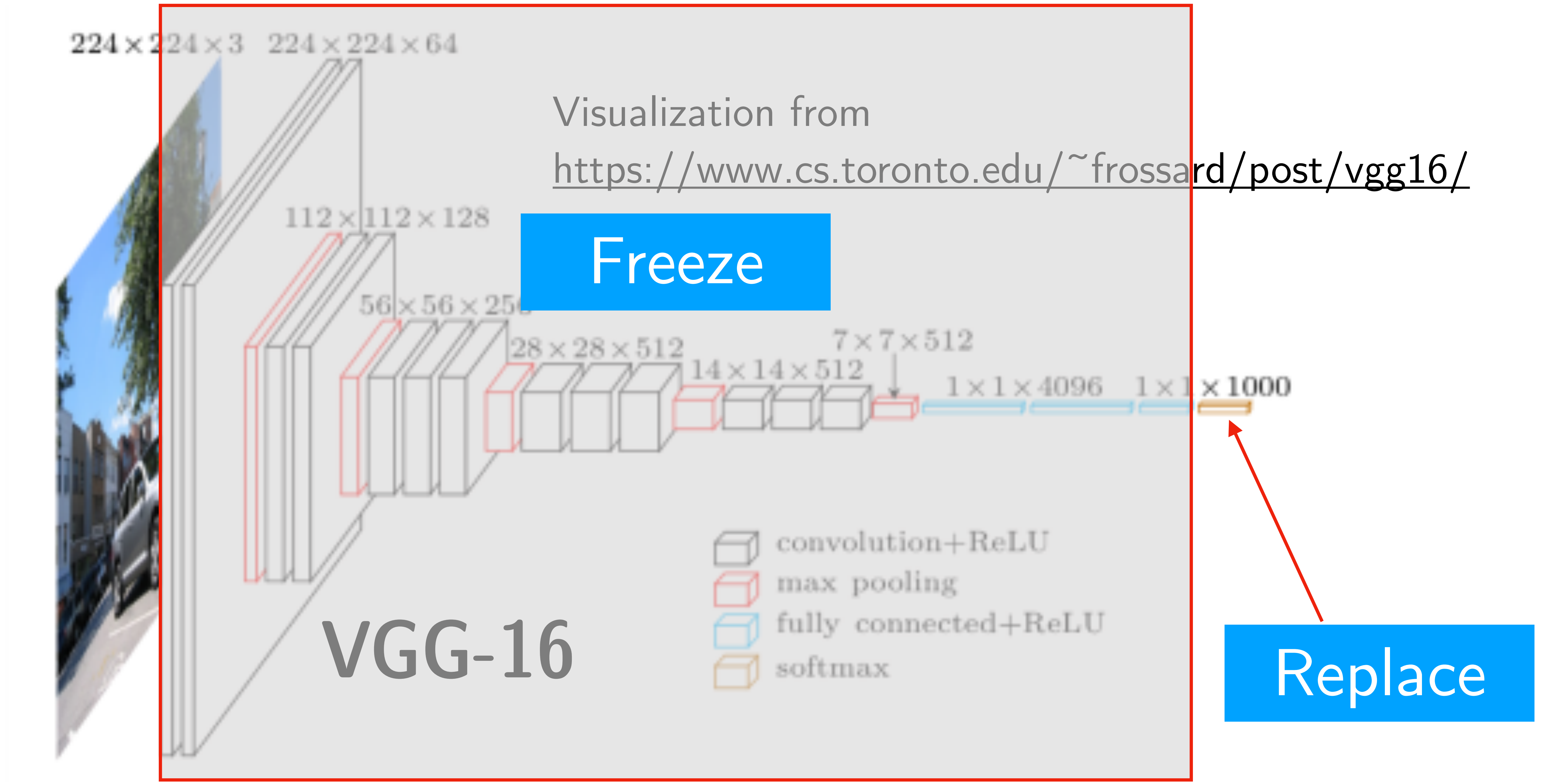
Visualization from
https://www.cs.toronto.edu/~frossard/post/vgg16/



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

Transfer Learning

PyTorch implementation: https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/vgg16.ipynb



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

Transfer Learning

https://pytorch.org/docs/stable/torchvision/models.html

Docs > torchvision > torchvision.models

TORCHVISION.MODELS

The models subpackage contains definitions for the following model architectures:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet

You can construct a model with random weights by calling its constructor:

```
import torchvision.models as models
resnet18 = models.resnet18()
alexnet = models.alexnet()
vgg16 = models.vgg16()
squeezenet = models.squeezenet1_0()
densenet = models.densenet161()
inception = models.inception_v3()
googlenet = models.googlenet()
```

Transfer Learning

https://pytorch.org/docs/stable/torchvision/models.html

Docs > torchvision > torchvision.models

TORCHVISION.MODELS

The models subpackage contains definitions for the following model architectures:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet

All pre-trained models expect input images normalized in the same way, i.e. mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be at least 224. The images have to be loaded in to a range of [0, 1] and then normalized using `mean = [0.485, 0.456, 0.406]` and `std = [0.229, 0.224, 0.225]`. You can use the following transform to normalize:

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
```

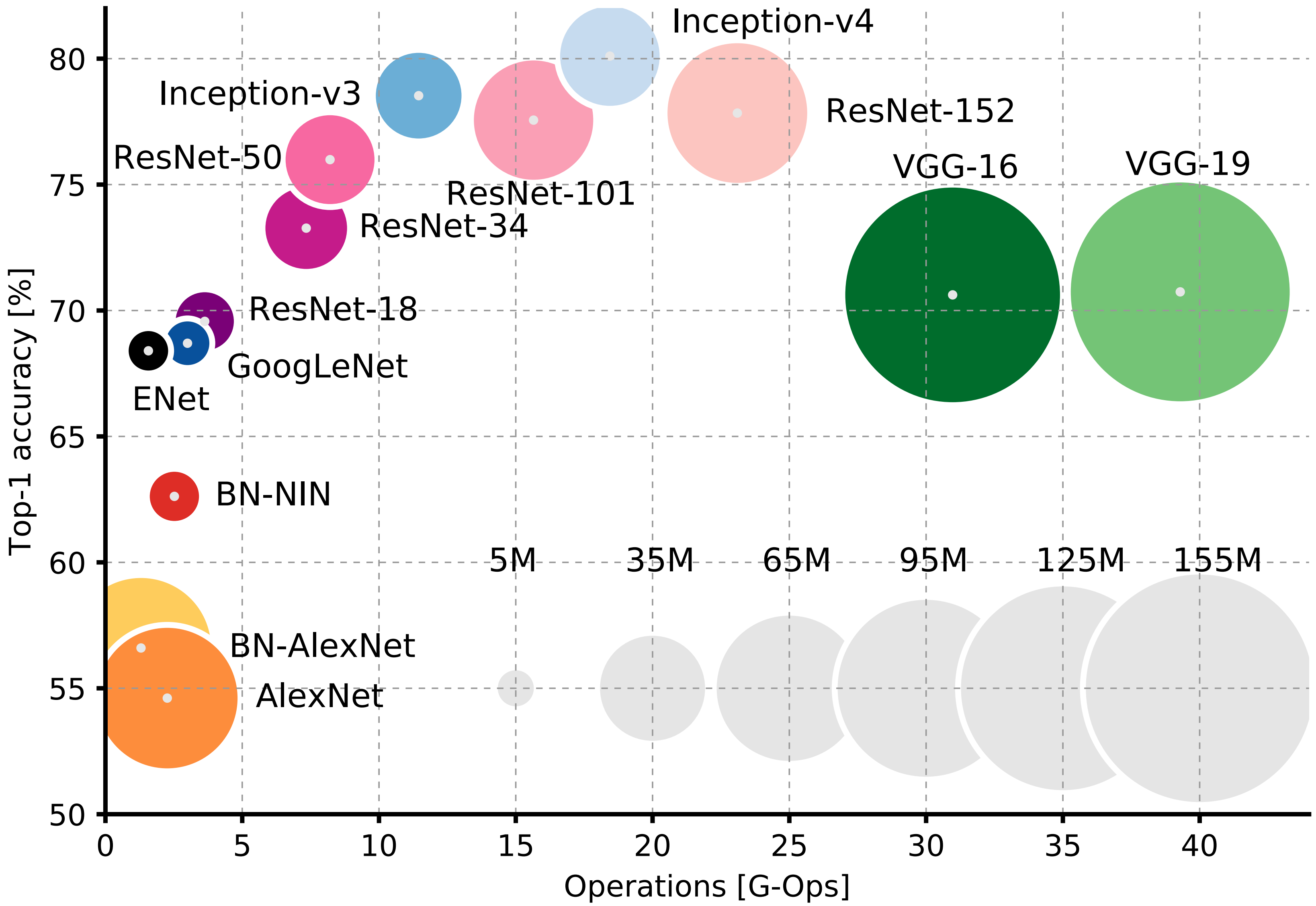
Transfer Learning Example

PyTorch example: https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/vgg16-transferlearning.ipynb

Pre-Trained Models for Text

https://modelzoo.co/model/pytorch-nlp

(Optional) News



Inception-v4
Inception-v3
ResNet-152
ResNet-50
ResNet-101
VGG-16
VGG-19
ResNet-34
ResNet-18
GoogLeNet
ENet
BN-NIN
5M
35M
65M
95M
125M
155M
BN-AlexNet
AlexNet
Top-1 accuracy [%]
80
75
70
65
60
55
50
0
5
10
15
20
25
30
35
40
Operations [G-Ops]

https://arxiv.org/abs/1904.01569

arXiv.org > cs > arXiv:1904.01569

Computer Science > Computer Vision and Pattern Recognition

Exploring Randomly Wired Neural Networks for Image Recognition

Saining Xie, Alexander Kirillov, Ross Girshick, Kaiming He

(Submitted on 2 Apr 2019 (v1), last revised 8 Apr 2019 (this version, v2))

Neural networks for image recognition have evolved through extensive manual design from simple chain-like mc
paths. The success of ResNets and DenseNets is due in large part to their innovative wiring plans. Now, neural ar
exploring the joint optimization of wiring and operation types, however, the space of possible wirings is constrai
despite being searched. In this paper, we explore a more diverse set of connectivity patterns through the lens of
this, we first define the concept of a stochastic network generator that encapsulates the entire network generatio
view of NAS and randomly wired networks. Then, we use three classical random graph models to generate rando
are surprising: several variants of these random generators yield network instances that have competitive accura
results suggest that new efforts focusing on designing better network generators may lead to new breakthrough:
spaces with more room for novel design.

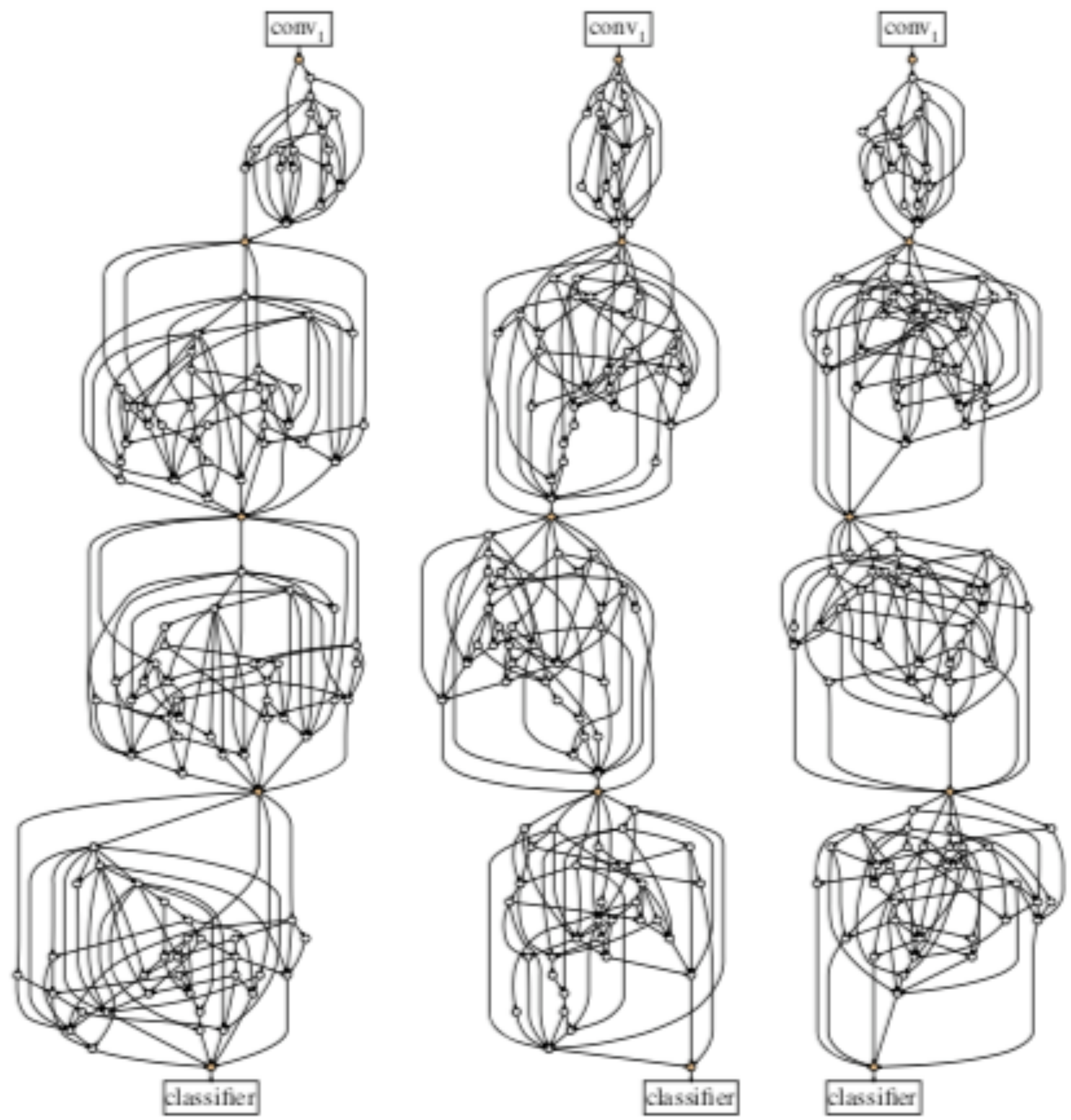
Based on neural architecture search (NAS) and stochastic network generators

https://arxiv.org/abs/1904.01569

Exploring Randomly Wired Neural Networks for Image Recognition

Saining Xie, Alexander Kirillov, Ross Girshick, Kaiming He

(Submitted on 2 Apr 2019 (v1), last revised 8 Apr 2019 (this version, v2))



Based on neural architecture search (NAS) and stochastic network generators

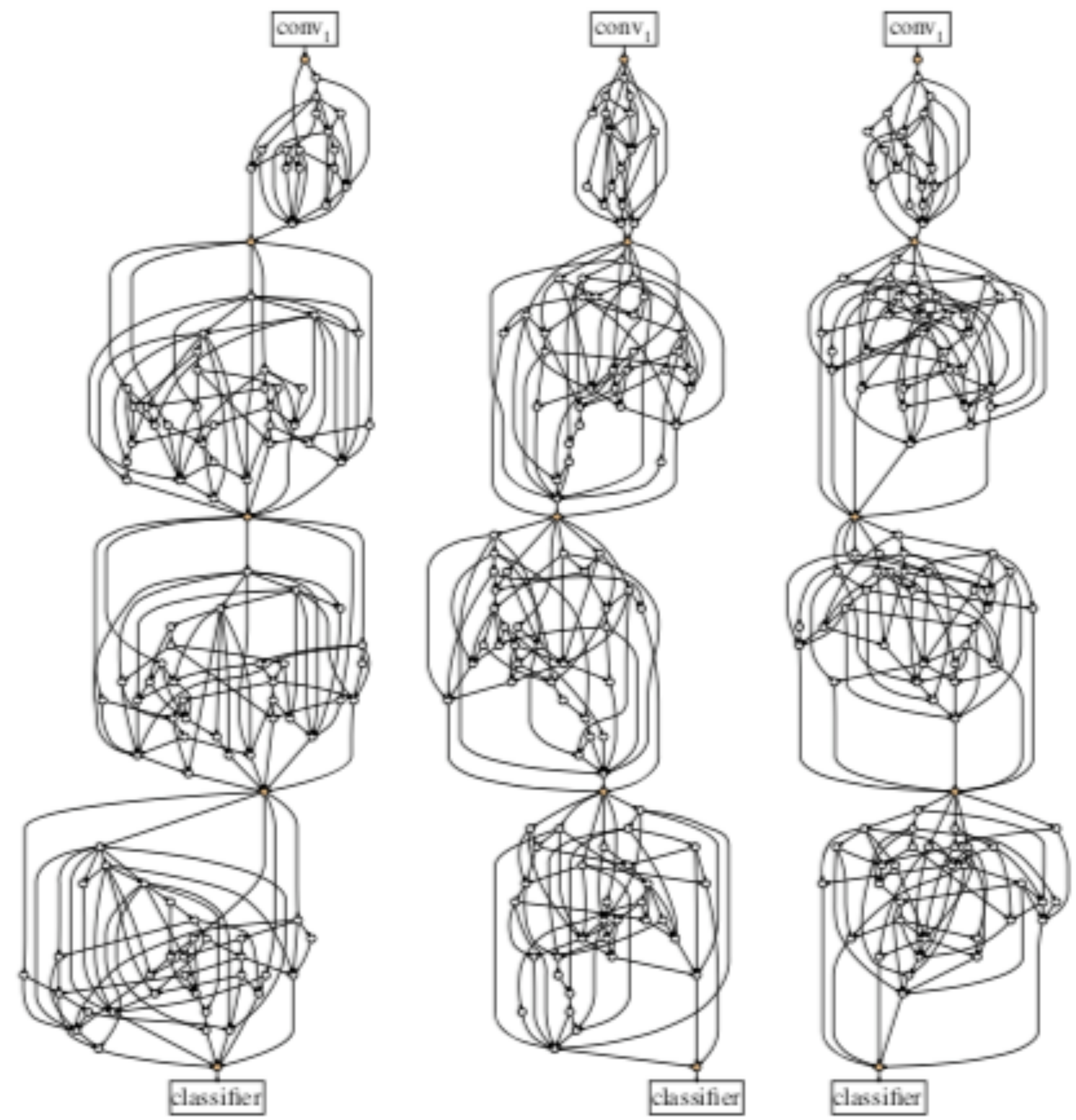
https://arxiv.org/abs/1904.01569

Exploring Randomly Wired Neural Networks for Image Recognition

Saining Xie, Alexander Kirillov, Ross Girshick, Kaiming He

(Submitted on 2 Apr 2019 (v1), last revised 8 Apr 2019 (this version, v2))

Also utilizes an LSTM controller with probabilistic behavior (will discuss LSTMs in a different context next lecture)



Based on neural architecture search (NAS) and stochastic network generators

Remaining Course Topics

http://pages.stat.wisc.edu/~sraschka/teaching/stat479-ss2019/#calendar