acm.ai

# More about CNN's, ResNet's, and Transfer Learning

## Advanced Track Workshop #4

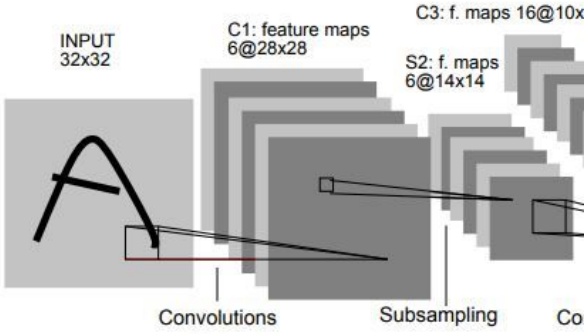
Anonymous Feedback: tinyurl.com/w21atrack4

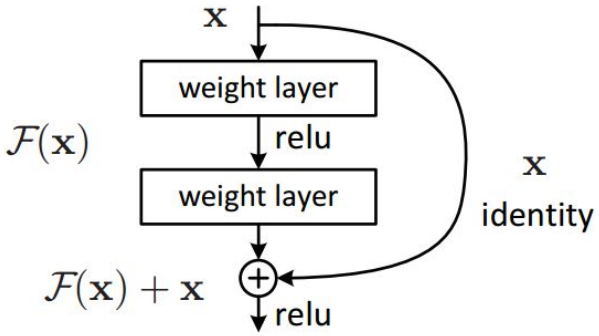
GitHub: github.com/uclaacmai/advanced-track-winter21

Attendance Code: snowboard

# Today's Contents

- Advanced ConvNet Architectures
  - Inception, ResNet, UNet
- Pretraining
- Transfer Learning




# 1. A Brief History of ConvNets

Adapted from Illustrated: 10 CNN Architectures by Raimi Karim, Towards Data Science

# Convolutional Layers and ReLu

1998: LeNet-5

- Pioneers the basic architecture of a ConvNet
  - Convolutions
  - Pooling

2012: AlexNet

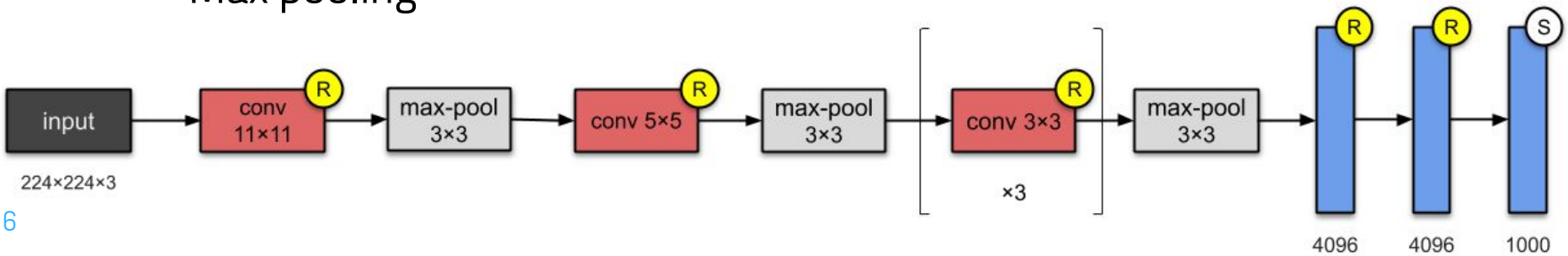
- Deeper + More Filters
- ReLU Activation
  - Faster training
  - Avoid vanishing gradient

# AlexNet

- CNN designed by Alex Krizhevsky (University of Toronto)
- Paper titled “ImageNet Classification with Deep Convolutional Networks”
  - Won the 2012 ImageNet Large-Scale Visual Recognition Challenge
  - One of the most influential publications in Computer Vision, really pioneered the use of CNNs – been cited over 30,000 times

# AlexNet Structure

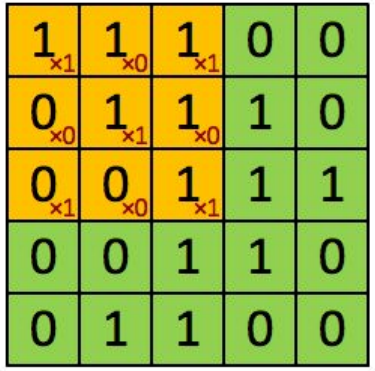
- 5 convolutional layers, 3 fully-connected layers
- ReLU is applied after each convolutional/ fully-connected layer
- Dropout is applied before each of the first two fully-connected layers
- Max pooling



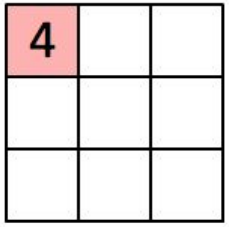
# Convolutional Layers and ReLu

## Quick Review - Basic Components of a ConvNet

- **Convolution Layers**
- Pooling Layers
  - Max vs. Average
- ReLU Activation




Image




Convolved Feature

# Convolutional Layers and ReLu

## Quick Review - Basic Components of a ConvNet

- Convolution Layers
- **Pooling Layers**
  - **Max vs. Average**
- ReLU Activation

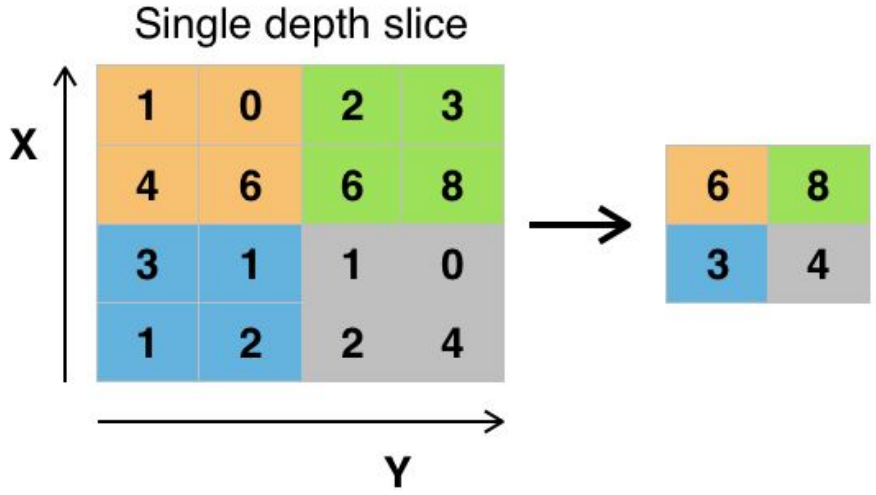


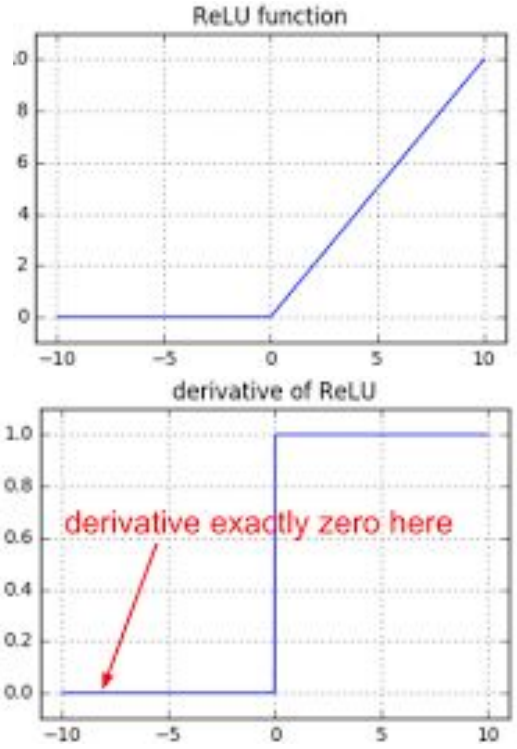

Image: Aphex34, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45673581

acm.ai

# Convolutional Layers and ReLU

## Quick Review - Basic Components of a ConvNet

- Convolution Layers
- Pooling Layers
  - Max vs. Average
- **ReLU Activation**



acm.ai

# Code Snippet (Pytorch)

```
class my_model(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(in_channels= 1, out_channels=16, kernel_size = 5, padding = (5//2), bias = True)
        self.relu1 = nn.LeakyReLU()

        self.conv2 = nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size = 5, padding = (5//2),bias=True)
        self.relu2 = nn.LeakyReLU()

        self.maxpool1 = nn.MaxPool2d(kernel_size = 2, stride = 2)

        self.conv3 = nn.Conv2d(in_channels = 32, out_channels = 64, kernel_size = 5, padding = (5//2),bias=True)
        self.relu3 = nn.LeakyReLU()

        self.maxpool2 = nn.MaxPool2d(kernel_size = 2, stride = 2)

        self.fc1 = nn.Linear(in_features = 1024, out_features = 512)    # these numbers depend
        self.fc2 = nn.Linear(in_features = 512, out_features = 256)     # on your data
        self.fc3 = nn.Linear(in_features = 256, out_features = 64)
        self.fc4 = nn.Linear(in_features = 64, out_features = 10)
        self.soft1 = nn.Softmax(dim = 1)
```

acm.ai

# Code Snippet (Pytorch) continued

```
def forward(self, x):
    x = self.conv1(x)
    x = self.relu1(x)
    x = self.conv2(x)
    x = self.relu2(x)
    x = self.maxpool1(x)

    x = self.conv3(x)
    x = self.relu3(x)
    x = self.maxpool2(x)

    x = x.reshape(x.shape[0], -1)

    x = self.fc1(x)
    x = self.fc2(x)
    x = self.fc3(x)
    x = self.fc4(x)
    x = self.soft1(x)

    return x
```

acm.ai

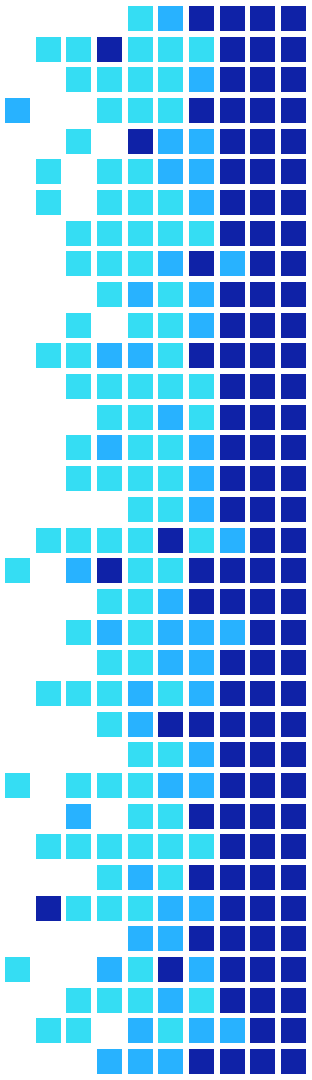
# Code Snippet (Tensorflow 2.0 & Keras)

```
import tensorflow as tf
# implementation of a CNN
cnn = tf.keras.models.Sequential()
# add the first convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=[64, 64, 3]))
# add the first maxpooling layer
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
# add the second sets of convolutional and maxpooling layers
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
# add a flattening layer
cnn.add(tf.keras.layers.Flatten())
# add a fully connected layer
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
# add the output layer
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
# compile the model
cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# train the model
cnn.fit(X_train, y_train, batch_size=32, epochs=100)
# predict
y_pred = cnn.predict(X_test)
```


# Poll question

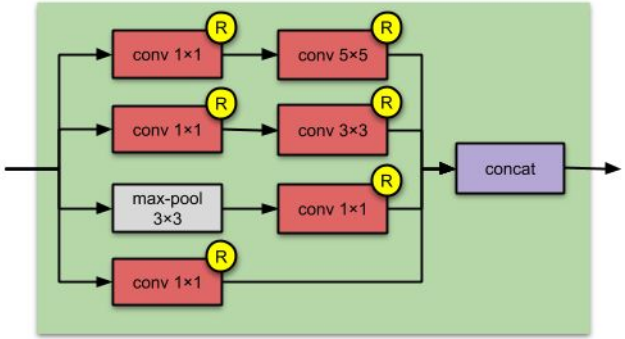
What is the shape of the input and output of the first convolutional layer of the CNN from the previous slide?

A. Input: (64, 64, 3) Output: (62, 62, 3)
B. Input: (64, 64) Output: (62, 62)
C. Input: (64, 64, 3) Output: (62, 62, 32)
D. Input: (64, 64) Output: (62, 62, 32)



# GoogLeNet / Inception

- Winner of ILSVRC 2014
- Introduces “Inception modules”: Network in Network
- Parallel pipelines of convolutional layers with different filter sizes
- Auxiliary classifiers



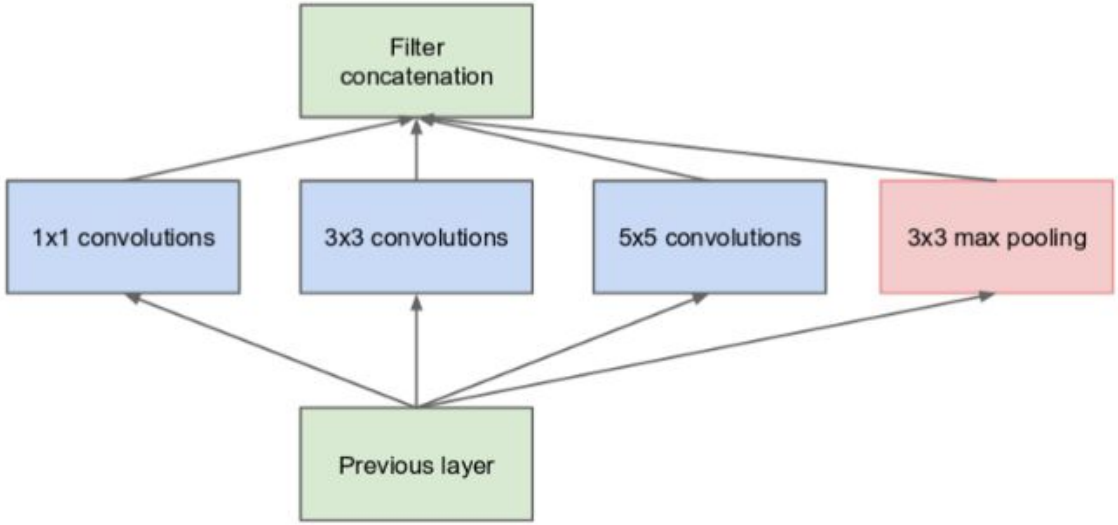

Inception module

# Inception v1: Feature Size

Example from Towards Data Science; Images from unsplash.com

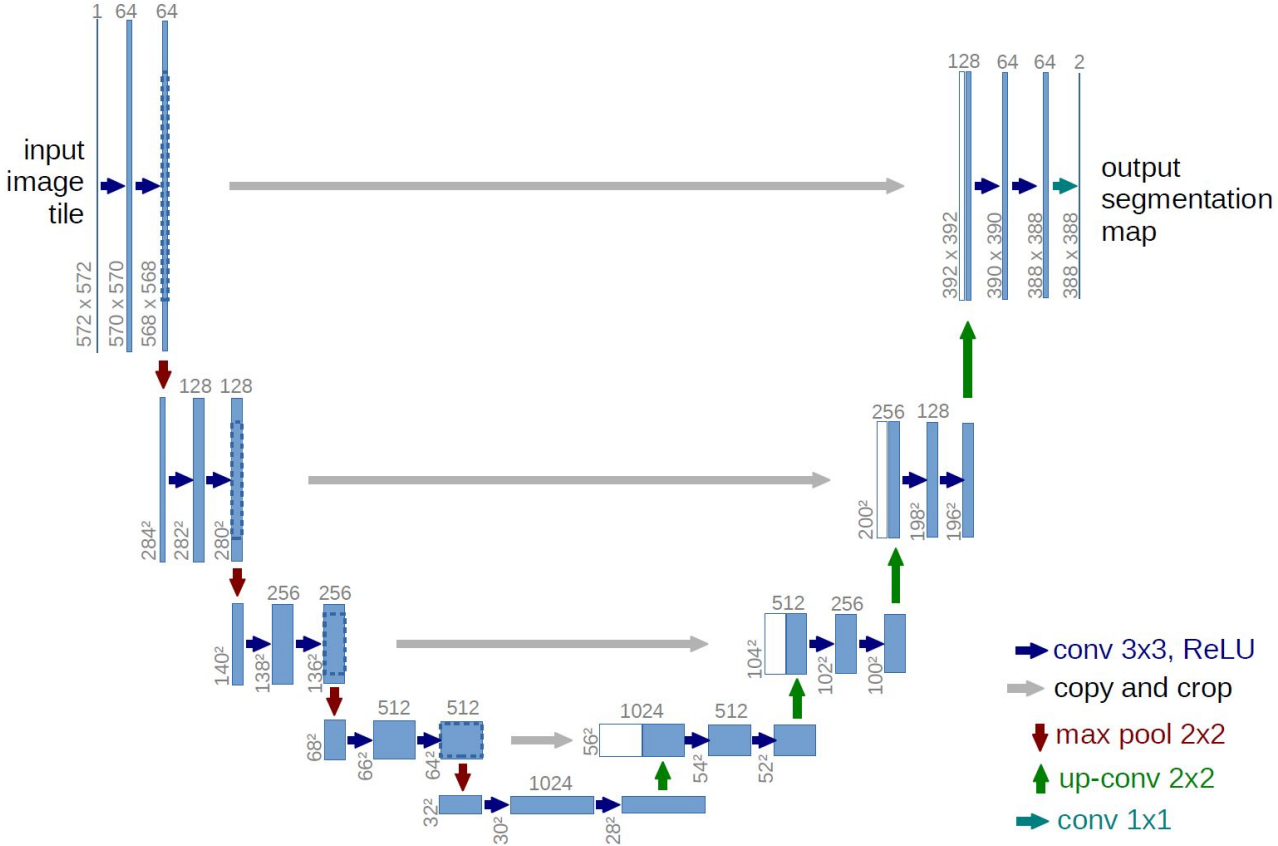
- Problem: The size of a feature can vary from image to image

# Inception v1: Feature Size




(a) Inception module, naïve version

# U-Net Architecture



# 1x1 Convolutions: Dimension Reduction

- Why?
  - Larger Kernels (e.g. 5x5 or 3x3) are expensive if the input volume has many kernels (greater depth)
  - e.g. if the previous layer outputs a volume with depth 256 and the current layer has 5x5 with 256 kernels then there are 5 x 5 x 256 x 256 total weights

# 1x1 Convolutions: Dimension Reduction

- How?
  - A 1x1 convolution kernel maps one column (depth-wise) to one “voxel” of the output
  - Using this, we can “compress” the depth of the output while preserving most of the information (i.e. an embedding)

256 depth → 256 5x5 kernel = 1,638,400 weights

256 depth → 64 1x1 kernel → 256 5x5 kernel = 425,984 weights

acm.ai

# Challenge Poll

Say we receive an input volume of spatial dimensions 1000x1000 and depth 512 to our convolutional layer. We want to reduce it's dimensions, so we use 32 filters each with a kernel size of 1x1. We then pass on our reduced volume to a convolutional layer with 256 5x5 kernels. How many weights do we end up training? (Hint: calculate the size of each filter, then multiply by the number of filters)

a. 1000 x 1000 x 512 + 1 x 1 x 32 = 512,000,032

b. 32 x 32 x 512 + 1 x 1 x 32 = 524,320

c. 1 x 1 x 32 x 32 + 5 x 5 x 32 x 512 = 410,624

d. 1 x 1 x 512 x 32 + 5 x 5 x 32 x 256 = 221,184

# 2. Classification
# Detection
# Segmentation

What's the difference?

# What's the difference?

## Classification

- What is this an image of?
- Labeling **images**


## Detection/Localization

- Where are the objects in this image?
- Labeling **regions**


## Segmentation

- Which pixels does each object occupy?
- Labeling **pixels**




Photo of Dog! by Jose Carlos Ichiro on Unsplash

# What's the difference?

## Regular Semantic Segmentation

- Label the class of each pixel

## Instance Segmentation

- Distinguish between multiple instances of the same class

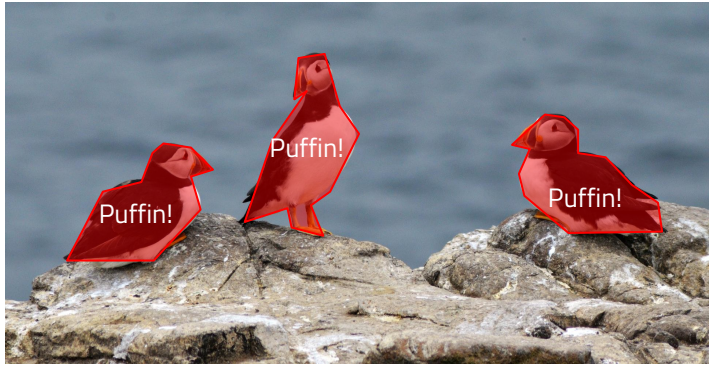


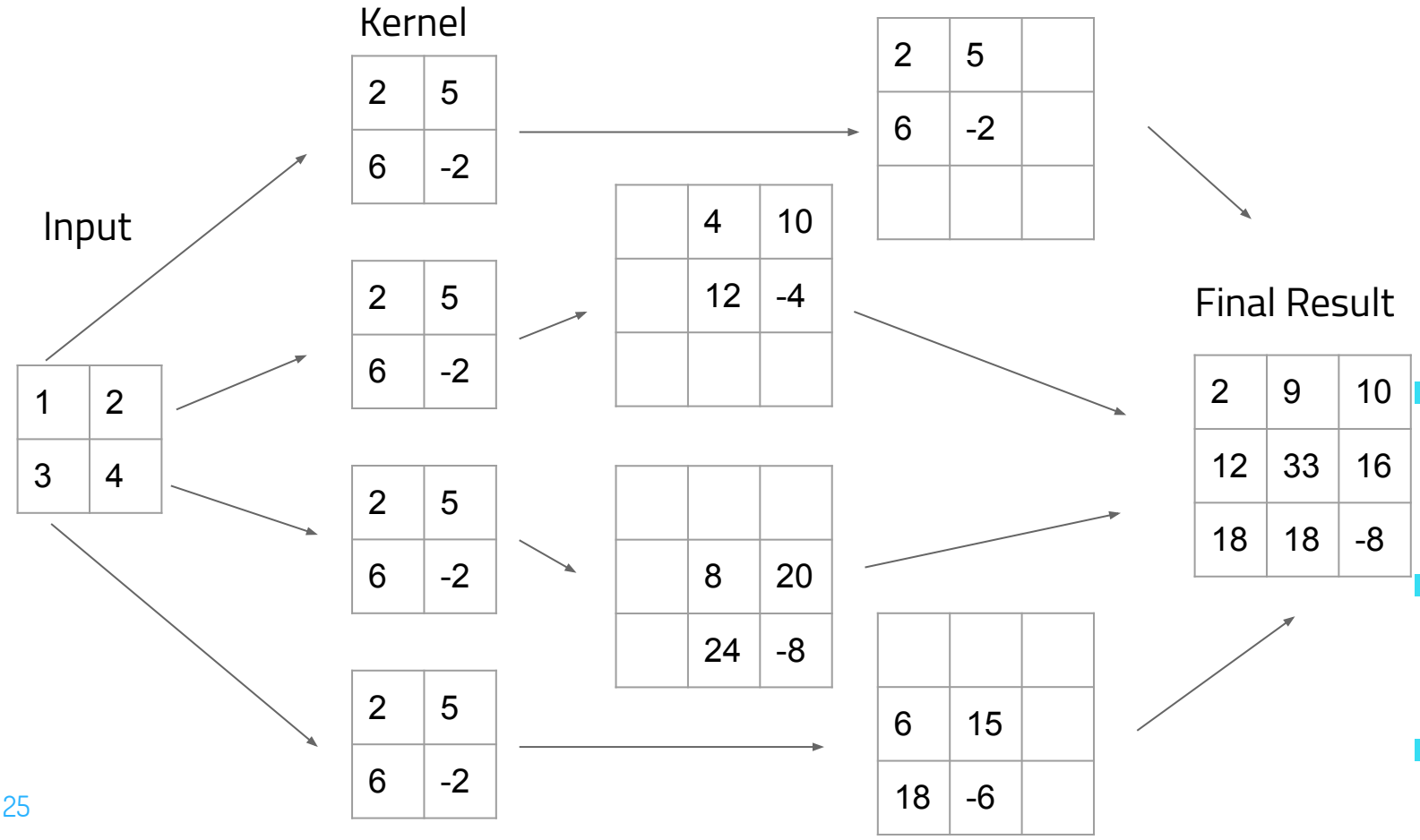


Photo of Puffins by David Heslop on Unsplash

# Transposed Convolution

- Convolutions and pooling reduce the width/height
  - How do we expand this back to a per-pixel output?
- Transposed Convolutions (aka deconvolution) allows us to "upscale" the latent space
- An example next slide

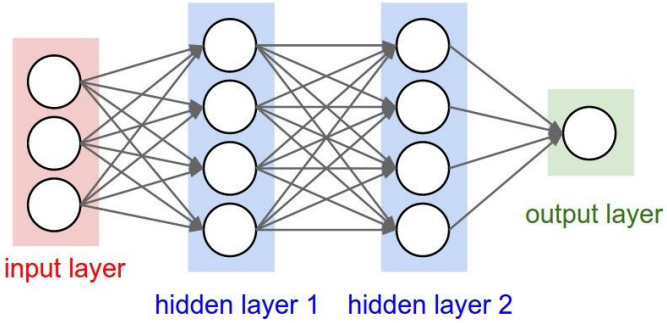
# Transposed Convolution

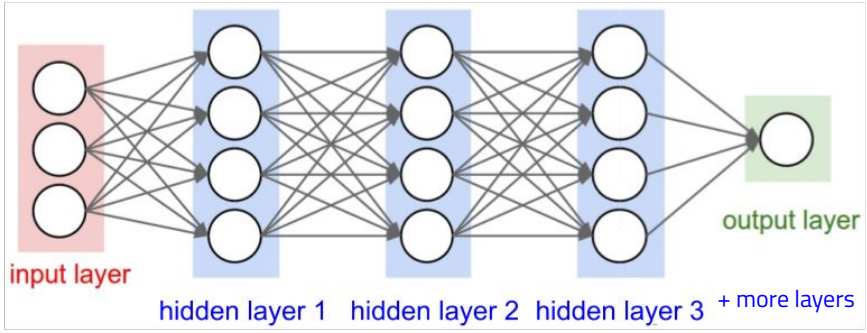



# 3. ResNets

# Motivation

- Suppose we take a shallow network and add additional layers to increase the depth.
- If we train this model, will it perform the same, better, or worse?



acm.ai

# Motivation

- Deeper networks have achieved better performance
- But we run into the "degradation problem"

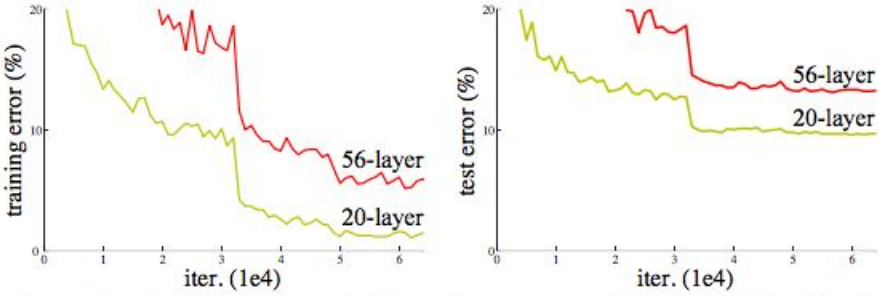



Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

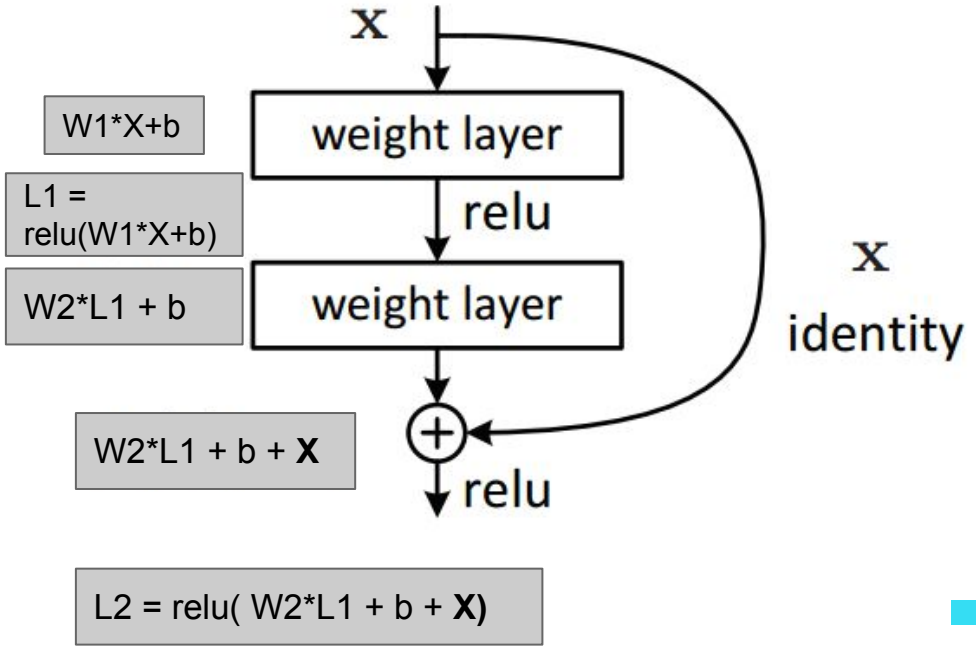
# Motivation

- Problems with traditional neural networks?
  - A deeper network isn't always better
- Why?
  - Vanishing gradient!!!!!!!!!!!!
  - Information is lost the deeper you go`

# ResNet: The Residual Block

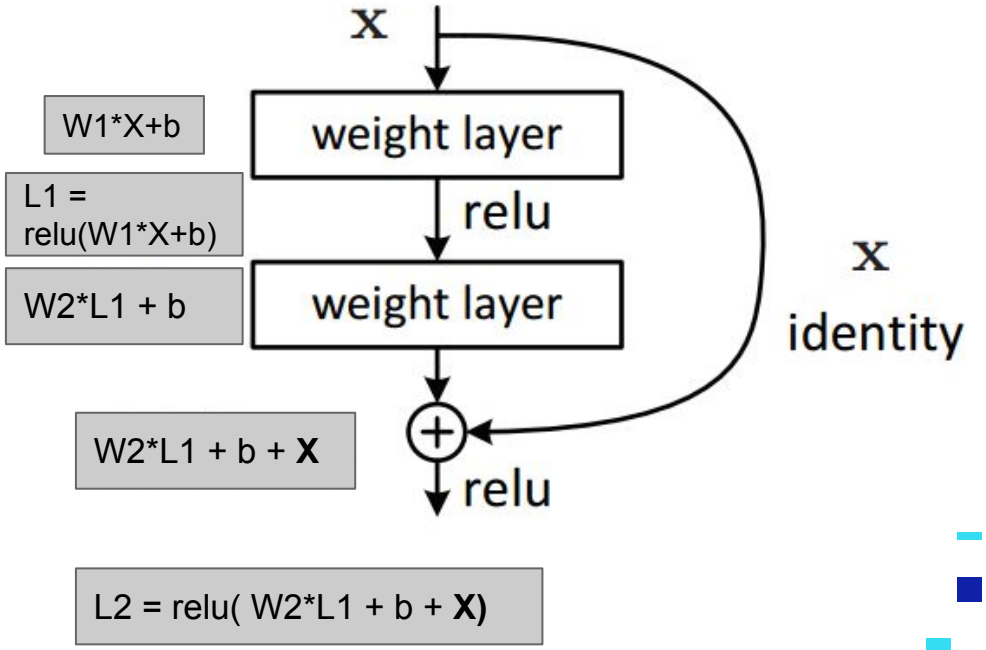
- The solution:

  Skip/Residual connections



# ResNet: The Residual Block

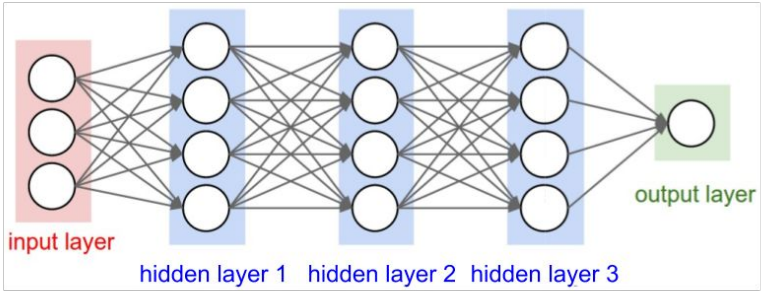
- If the weights go to zero, then the layer becomes the identity mapping



# ResNet: The Residual Block

- In theory, this should perform at least as well as the shallow network, since the additional layer can learn weights that make it the identity function.

- In practice, the extra layers seem to have trouble learning an identity mapping, but a much easier time learning a zero mapping

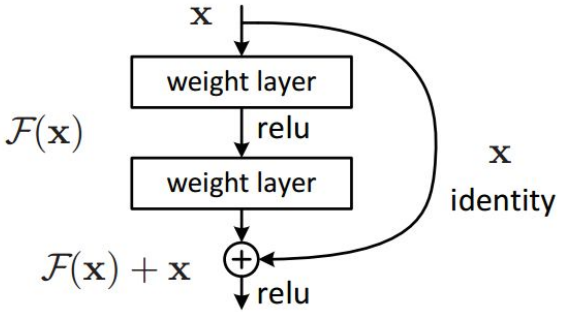


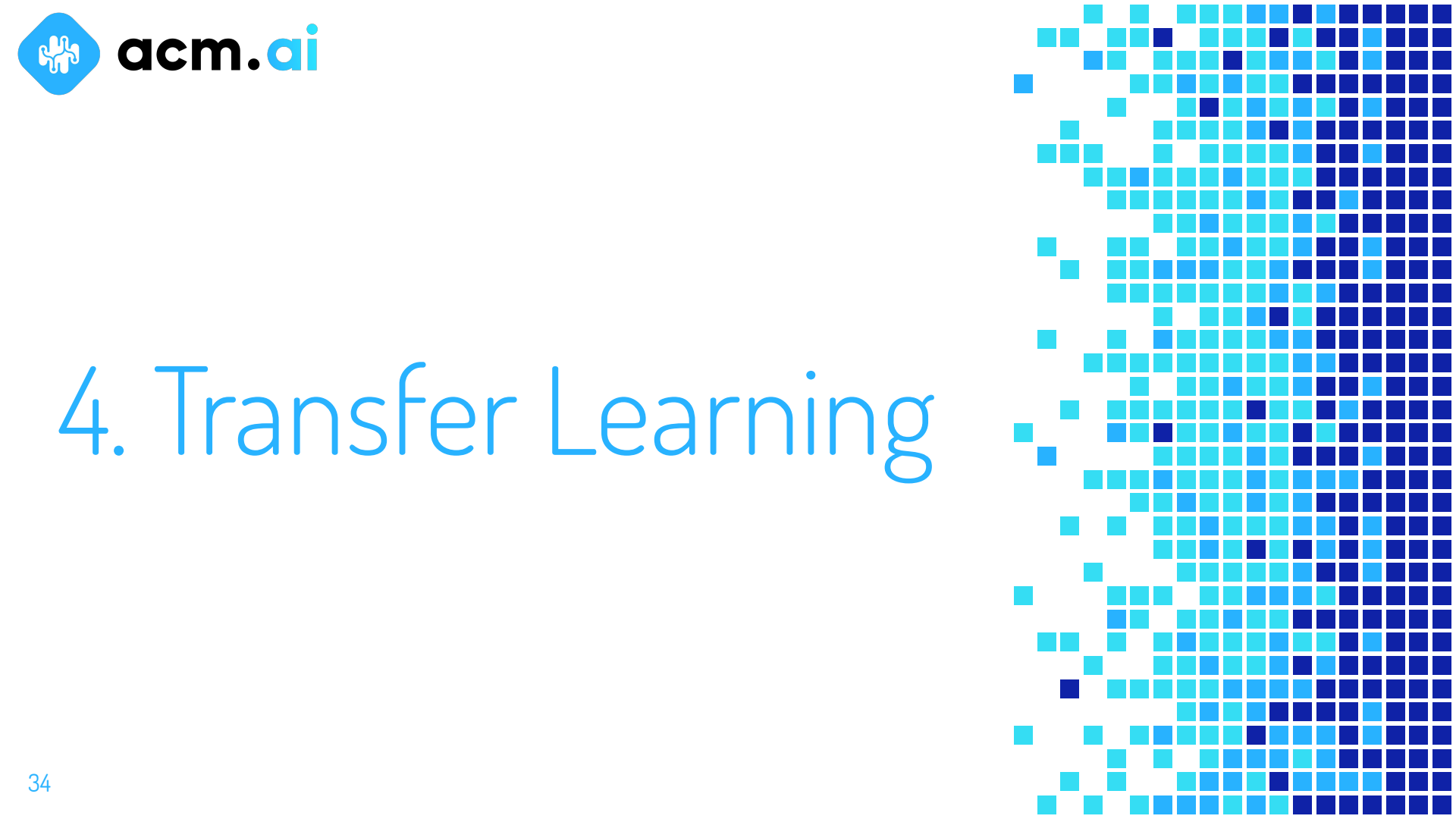
# ResNets: other interesting applications

- Neural Ordinary Differential Equations (Neural ODE's)

  This paper won the Best Paper Award at NeurIPS 2018.

  https://arxiv.org/abs/1806.07366





# 4. Transfer Learning

# Transfer Learning: Pre-training

Training from scratch with random initialization:

- Time consuming
- Requires a lot of data

The solution?

- Use a model that was **pre-trained** on a different dataset as a starting point
  - We are transferring knowledge from a network trained on one dataset to another

# Methods of Transfer Learning

Pre-training and fine-tuning

Train a model on one dataset and use the final trained weights as the initial weights when training on a new dataset

Freezing layers

Take a trained model and only re-train the final layers on a new dataset or replace the final layers with a new classifier.

e.g. freeze the convolutional layers and only train the fully connected layers.

# Discussion Time!

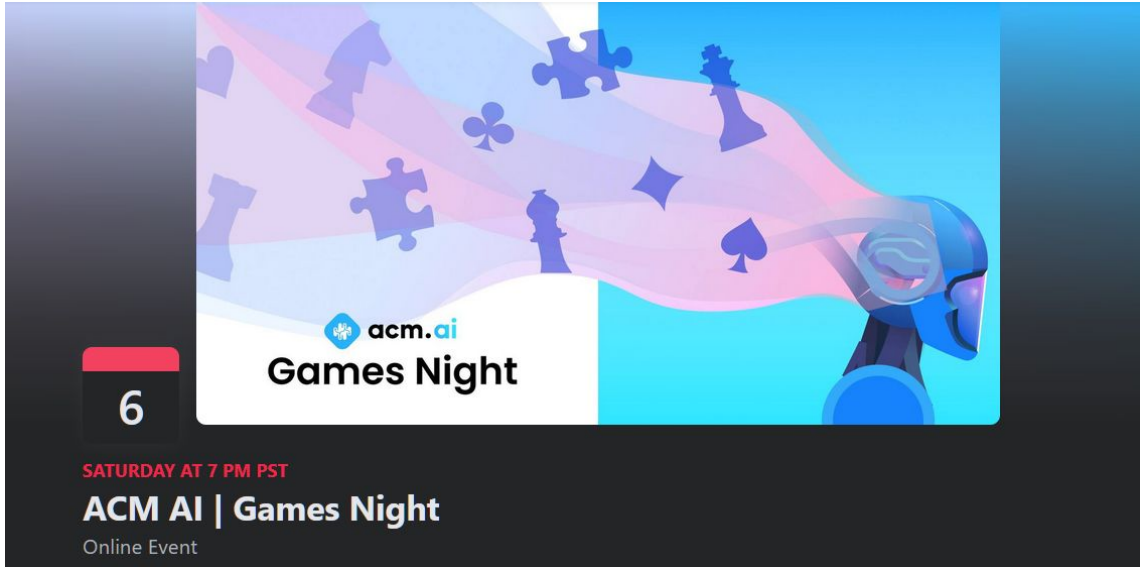
1. What problem do residual blocks address?
2. How do transposed convolutions help?

# Discussion Answers

- https://tinyurl.com/advworksheet4

acm.ai

# Join us at our social!




https://www.facebook.com/events/424485038663814/


# Thank you all for coming!

Anonymous Feedback: tinyurl.com/w21atrack4
Facebook Group: www.facebook.com/groups/uclaacmai/