Intro to Neural Nets

Advanced CNNs

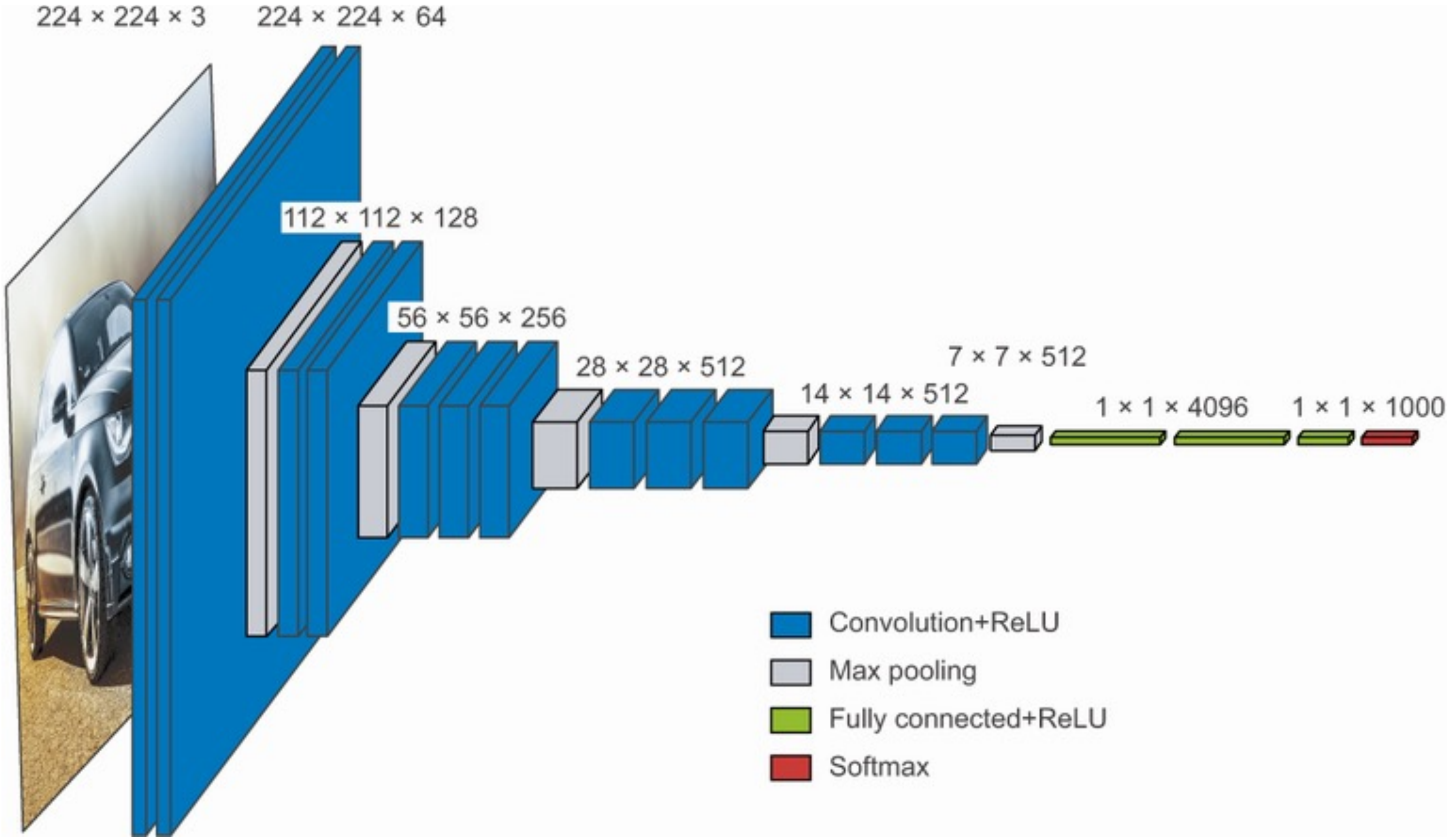
Session Agenda

More Modern Image-Network Architectures
- Mitigating vanishing gradients in deep networks.

Image-to-Image Prediction Tasks
- Image segmentation, Image super-resolution.
- Transpose convolution operations for up-sampling.

Visualizing What a CNN is Learning
- Try to visualize what components of an image your model's feature extractions are detecting.

Image-Classification Architectures



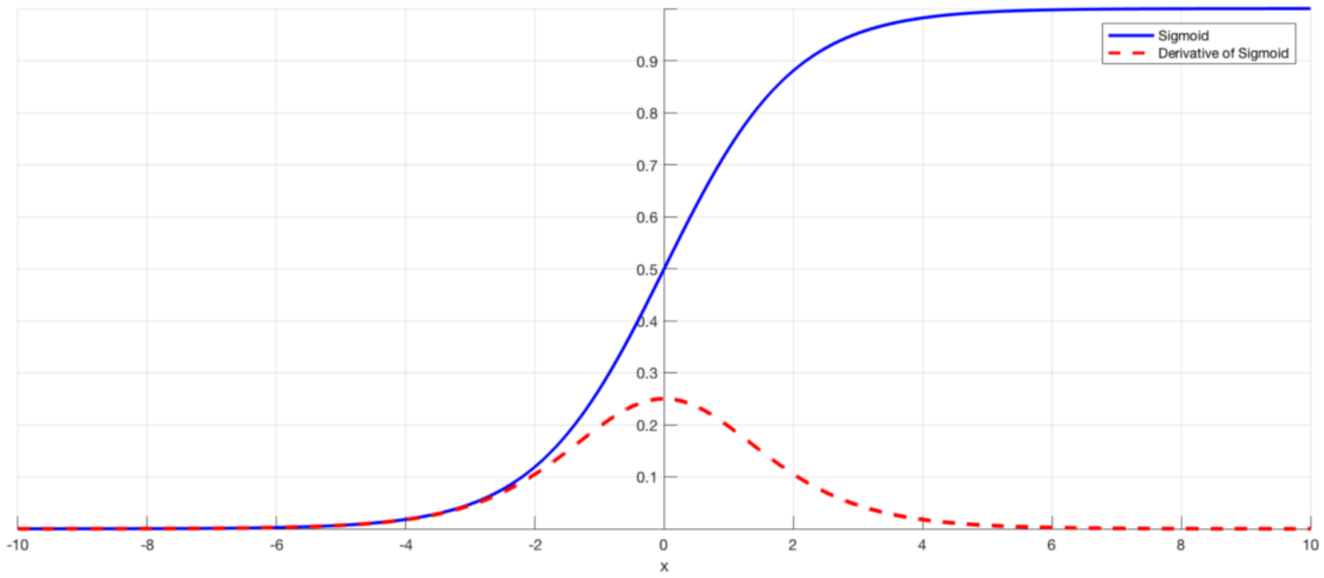
Vanishing Gradients

Recall That Certain Activations Make Vanishing Gradients More Likely

- As we approach 0 or 1 at a node's output, changes to input parameters have little impact…

However, Regardless of Activation This Can Be an Issue

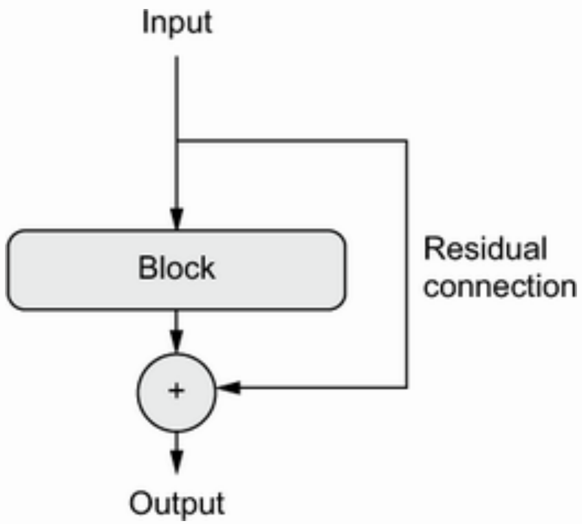
- For deep networks, parameters at front of network tend to have a much smaller influence on ultimate loss function. Accordingly, gradients for those early parameters can be very small.



Solution: Residual Connections

Provide a 'Short-cut' From Loss Function to Front-end Weights

- We include feed-forward layers as usual, but we also add short-cut connections *around* the layers.
- We typically incorporate these residual connections either via an 'Add' layer or a 'Concatenate' layer.
- Note that conformity of the tensor shapes will be very important here – you'll start encountering shape conformity errors if you are not careful!



Listing 9.2 Residual block where the number of filters changes

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(32, 32, 3))
x = layers.Conv2D(32, 3, activation="relu")(inputs)
residual = x
x = layers.Conv2D(64, 3, activation="relu", padding="same")(x)
residual = layers.Conv2D(64, 1)(residual)
x = layers.add([x, residual])
```

Other Modern CNN Innovations

Separable Convolutions = Independent Filters for Each Input Channel

- Rather than sharing the filters across input channels (e.g., R, G and B), we create different filters for each channel.

Batch Normalization → Activation → Separable Convolutions + Residual

- Rather than sharing the filters across input channels (e.g., R, G and B), we create different filters for each channel.

```
import keras

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)                                            1
x = layers.Conv2D(filters=32, kernel_size=5, use_bias=False)(x)                 2

for size in [32, 64, 128, 256, 512]:                                            3
    residual = x

    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False)(x)

    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False)(x)

    x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

    residual = layers.Conv2D(
        size, 1, strides=2, padding="same", use_bias=False)(residual)
    x = layers.add([x, residual])

x = layers.GlobalAveragePooling2D()(x)                                          4
x = layers.Dropout(0.5)(x)                                                      5
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

This is the Xception Architecture

Xception: Deep Learning with Depthwise Separable Convolutions

François Chollet
Google, Inc.
fchollet@google.com

Abstract

We present an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution). In this light, a depthwise separable convolution can be understood as an Inception module with a maximally large number of towers. This observation leads us to propose a novel deep convolutional neural network architecture inspired by Inception, where Inception modules have been replaced with depthwise separable convolutions. We show that this architecture, dubbed Xception, slightly outperforms Inception V3 on the ImageNet dataset (which Inception V3 was designed for), and significantly outperforms Inception V3 on a larger image classification dataset comprising 350 million images and 17,000 classes. Since the Xception architecture has the same number of parameters as Inception V3, the performance gains are not due to increased capacity but rather to a more efficient use of model parameters.

as GoogLeNet (Inception V1), later refined as Inception V2 [7], Inception V3 [21], and most recently Inception-ResNet [19]. Inception itself was inspired by the earlier Network-In-Network architecture [11]. Since its first introduction, Inception has been one of the best performing family of models on the ImageNet dataset [14], as well as internal datasets in use at Google, in particular JFT [5].

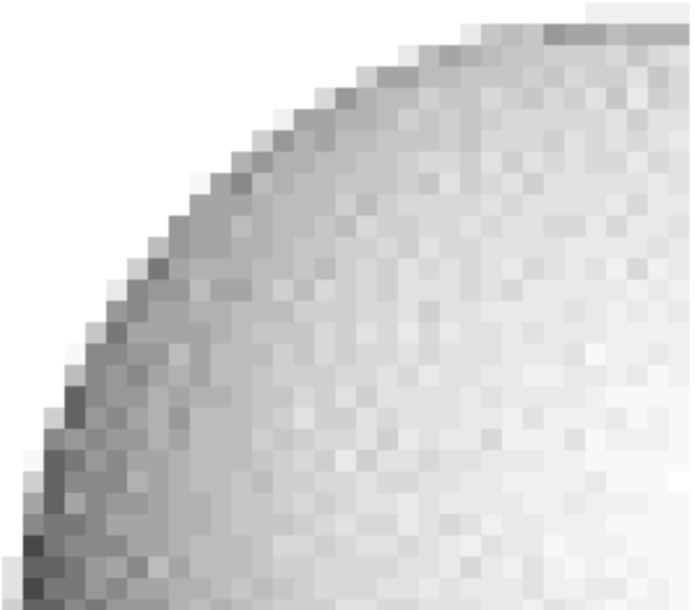
The fundamental building block of Inception-style models is the Inception module, of which several different versions exist. In figure 1 we show the canonical form of an Inception module, as found in the Inception V3 architecture. An Inception model can be understood as a stack of such modules. This is a departure from earlier VGG-style networks which were stacks of simple convolution layers.

While Inception modules are conceptually similar to convolutions (they are convolutional feature extractors), they empirically appear to be capable of learning richer representations with less parameters. How do they work, and how do they differ from regular convolutions? What design strategies come after Inception?

Image-to-Image Prediction

Super-Resolution

- Take a high-resolution image, pixelate it, then try to predict the high resolution from the pixelated version.
- Q: What kind of activation and loss function will make sense in this task?



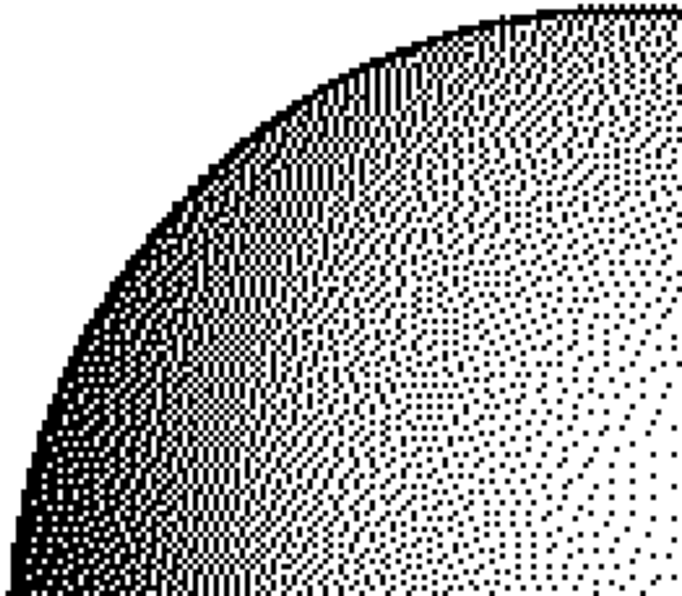
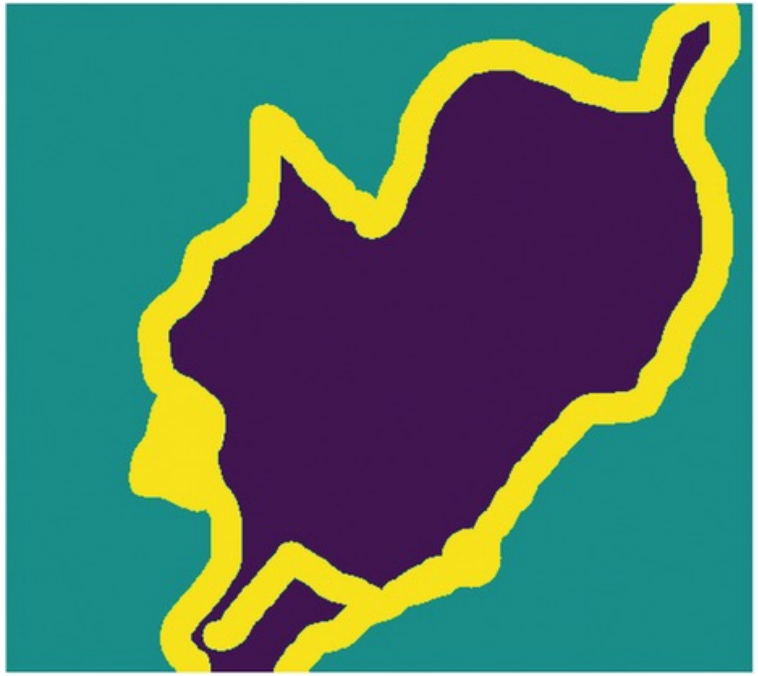


Image-to-Image Prediction

Image-Segmentation

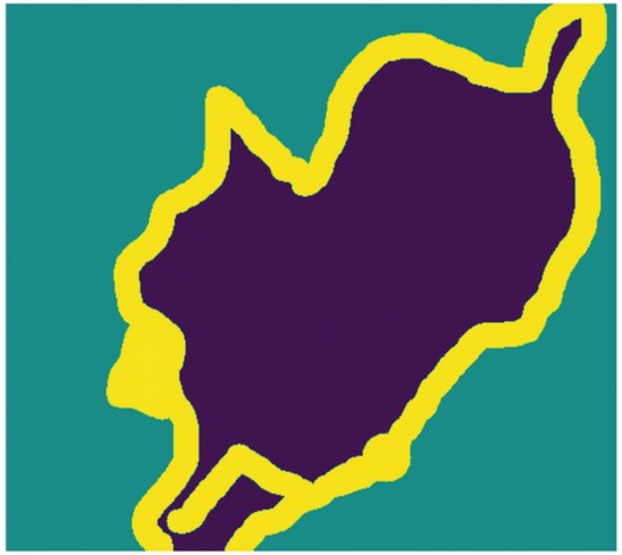
- Take an image and its segment mask, then try to predict the segment associated with each pixel from the original picture.
- Q: What kind of activation function and loss function will make sense in this task?



Topology for Image-to-Image

Auto-Encoder Architecture

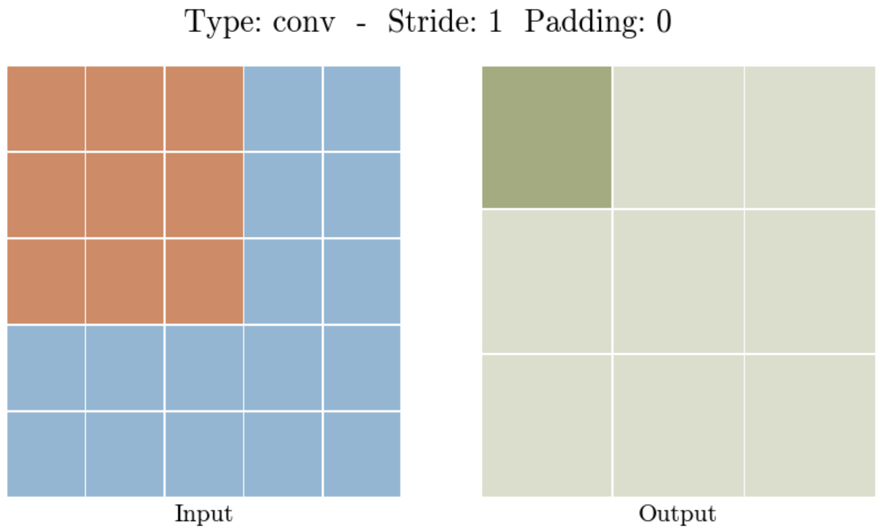
- Down-sample and then Up-sample back to same dimensionality
- We do not use down-sample using pooling (because they force attention to the whole image as we learn higher level features). Instead, we use larger strides. This enables 'dimensionality' reduction while maintaining a focus on local portions of the image.
- We then 'up-sample' back to the original dimensionality using a transpose of the convolutional operation. This is a form of autoencoder architecture.

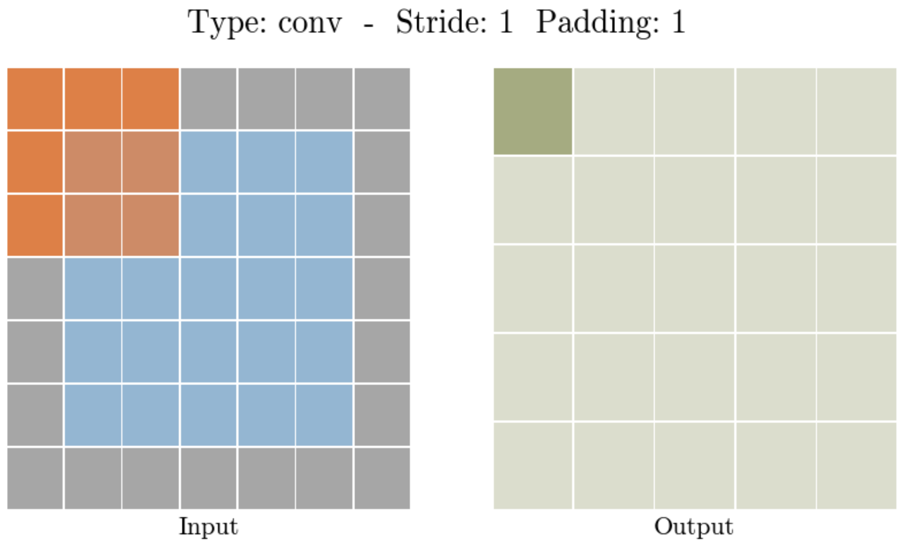


Convolution

Recall What A Convolution Is…

- It’s a down-sampling approach (it compresses information)

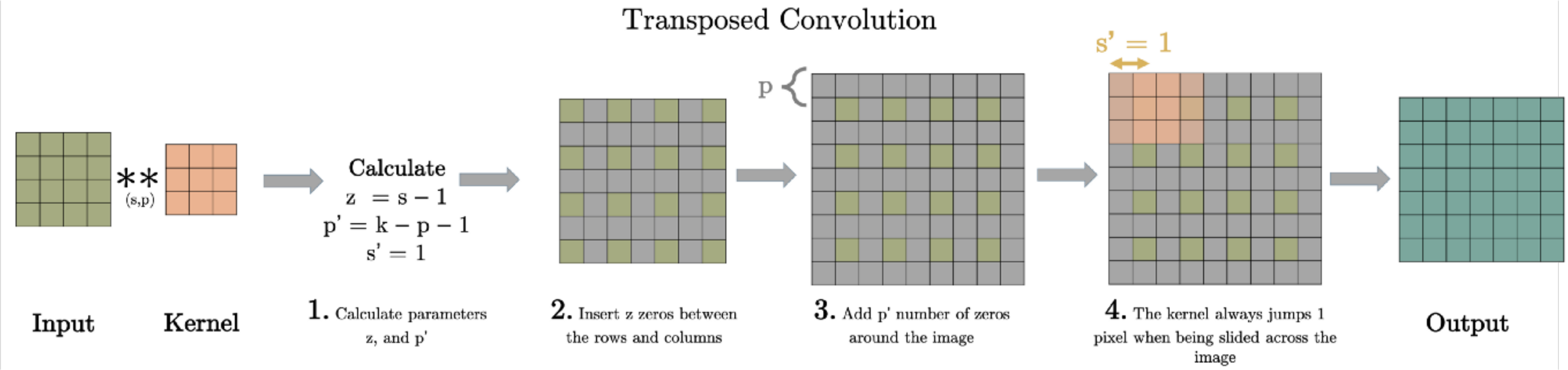




Inverse Convolution

Inverse of a Convolution

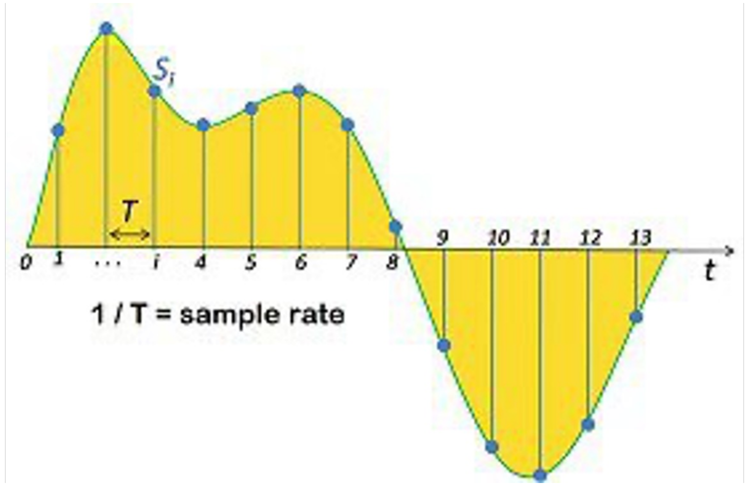
- Instead of Input (+padding) * filter (+stride) = output, this calculates the inverse operation, to up-sample.
- Here, s is the Convolution stride, p is the Convolution padding, k = is the Convolution kernel width/height; thus, s’ is the stride of the transpose convolution (always 1), z = padding around each pixel element, p’ is the final padding around the image for the transpose operation.



CNN for Audio

Same Sequence Concepts Work for Audio Data

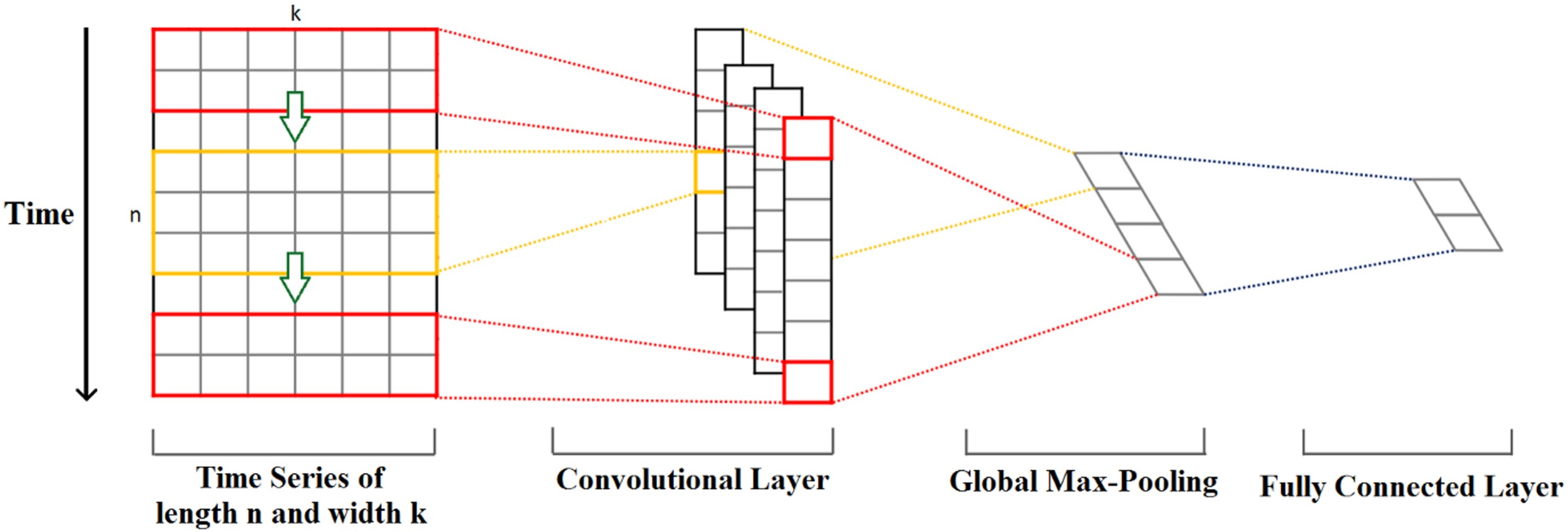
- Audio files are just sequences of numeric values (amplitude), possibly two if it was recorded in stereo.
- Once we recognize this, we realize we can predict things about audio sequences too!



(Temporal) 1D Convolution

1D Convolution Accomplishes Same Goal as 2D

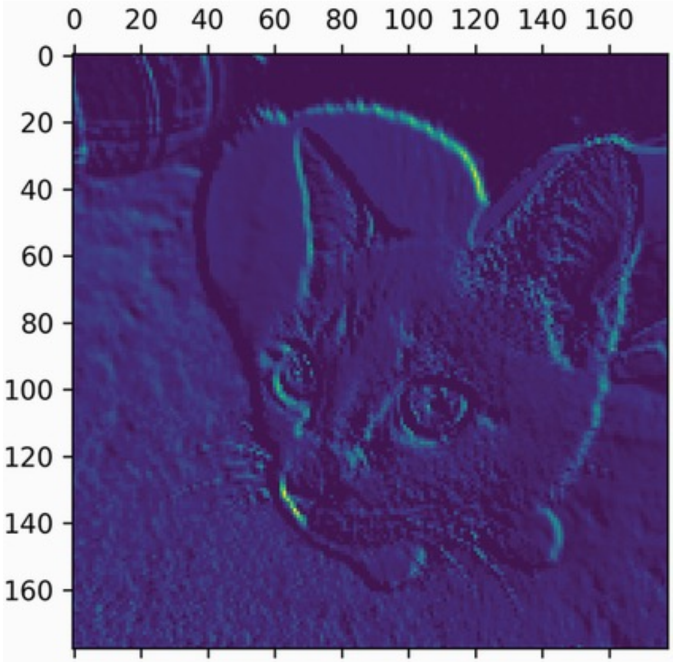
- It only considers arrangement of features in one dimension (temporal ordering).
- Compresses into shorter sequences, across the entire set of features (just as 2D Conv compresses matrices into smaller matrices, across the entire set of input channels (e.g., RGB).



Visualizing What a CNN is Learning

Visualizing Filter Activations in a Layer for a Given Image

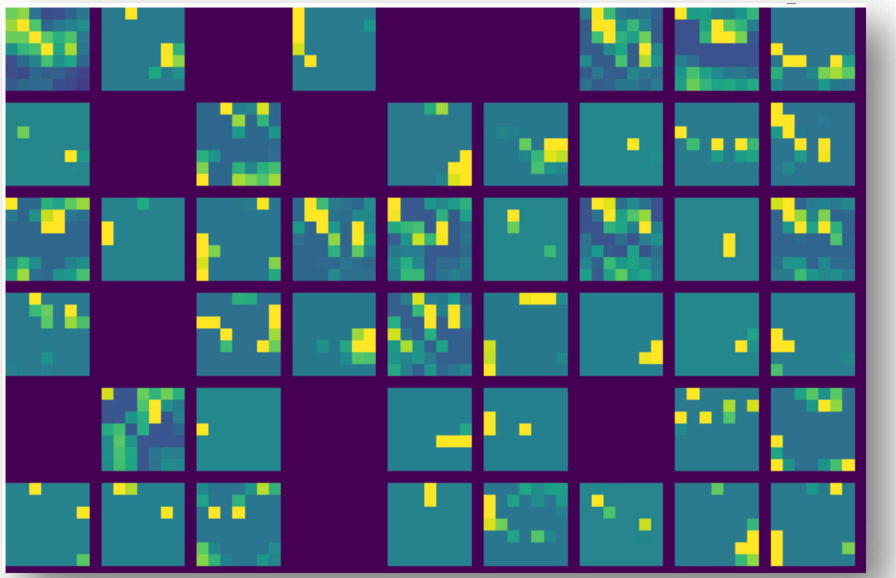
- For a given layer and input image, we plot the 2D feature map for each channel (filter). Each one will capture independent features of the image – plotting each layer's activation (output) can show us what features are being identified in the pictures to produce the final prediction.

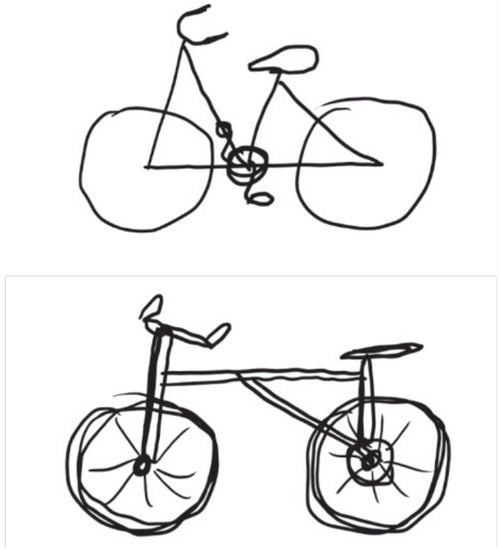


Deeper Feature Maps are Not Interpretable

Feature Maps in Deeper Layers Reflect Abstractions

- Moreover, as we go deeper, more filters are not activated at all, implying whatever concept they capture is not present in the image.

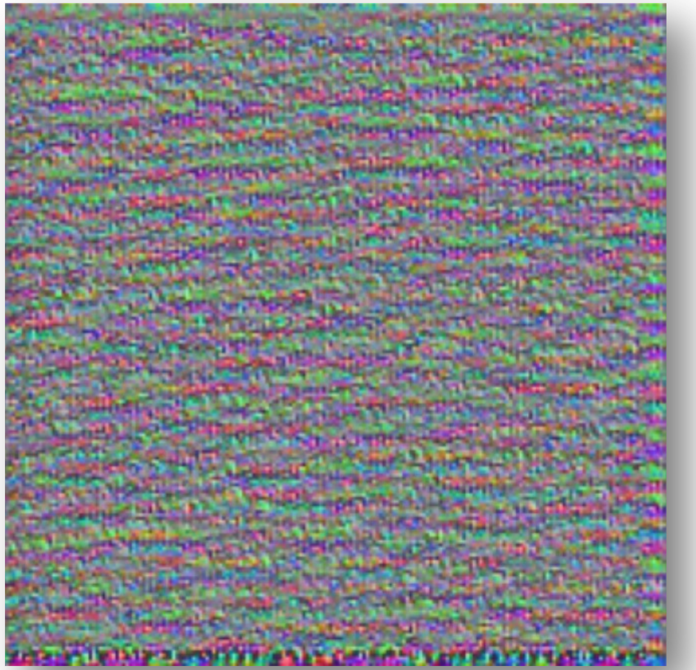




Gradient Ascent

We Learn an Image That Maximally Activates a Filter

- We can specify a function that calculates the gradient between a filter's average output activation and the pixel values of an input image.
- We can then learn an image that maximally activates the filter, by updating pixel values, starting with random noise.



Class Activation Maps

What Pixels Drove a Given Prediction?

- Generate a heatmap on the original input image that indicates what ‘components’ (pixels) of the input image most drive the final classification assignment based on ‘steepest’ gradients.
- Useful for identifying objects in a particular image too, e.g., when you have an object detector.

Questions?