Image Classification

Linear and DNN Models

Intro to CNNs

Dealing with Data Scarcity

Going Deeper Faster

Google Cloud

Linear and DNN Models for Image Classification

Learn how to...

Understand how image data is represented as floating point numbers that can be flattened

Compare functions for model confidence in image classification (Softmax)

Train and evaluate a Linear model for image classification using TensorFlow

Train and evaluate a Deep Neural Network (DNN) model for image classification using TensorFlow

Understand how to apply dropout as a regularization technique for DNNs

Agenda

Introduction

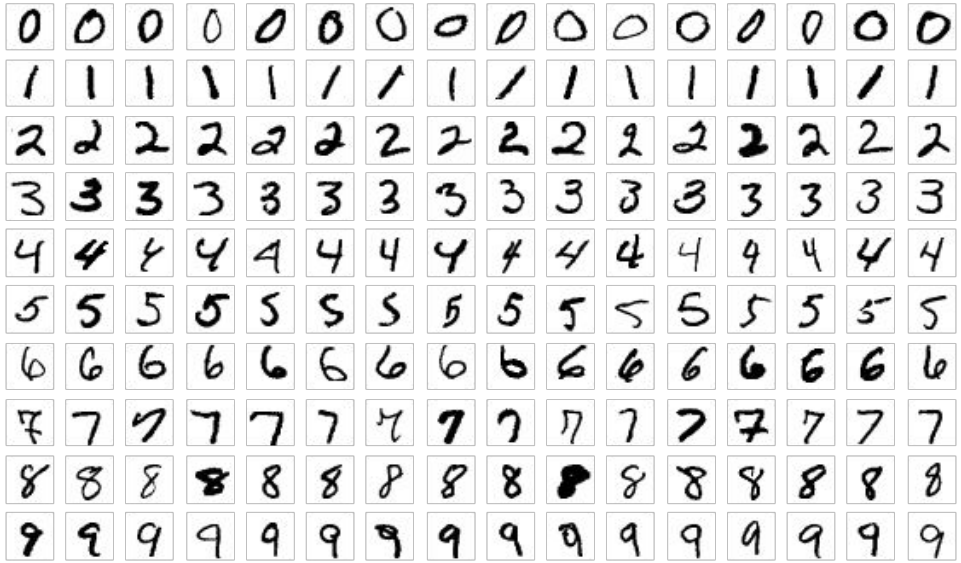
Linear models

Deep neural network models

DNN dropout

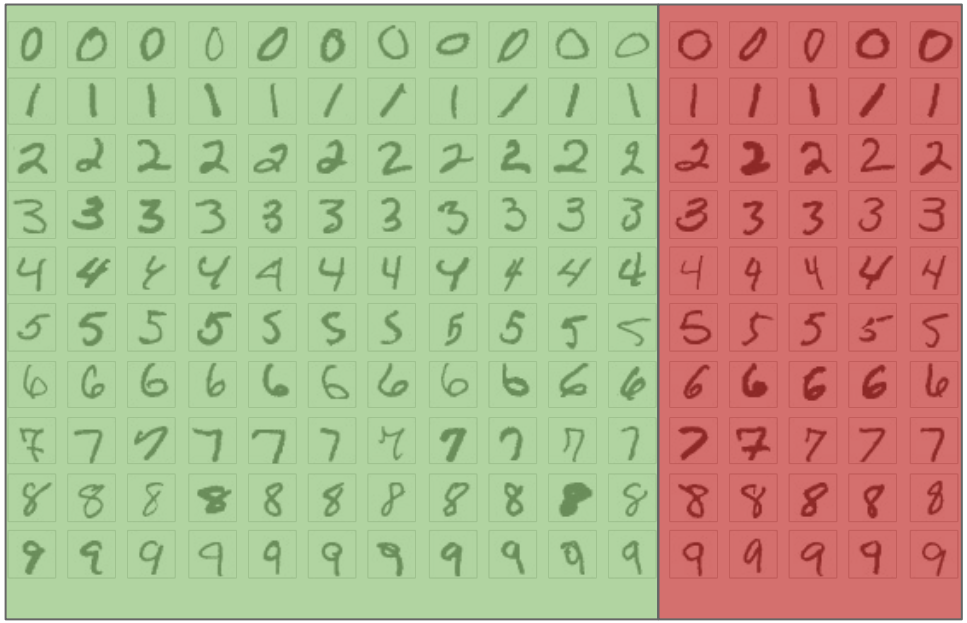
Problem: Recognizing handwritten digits

Introducing the MNIST dataset of labeled images



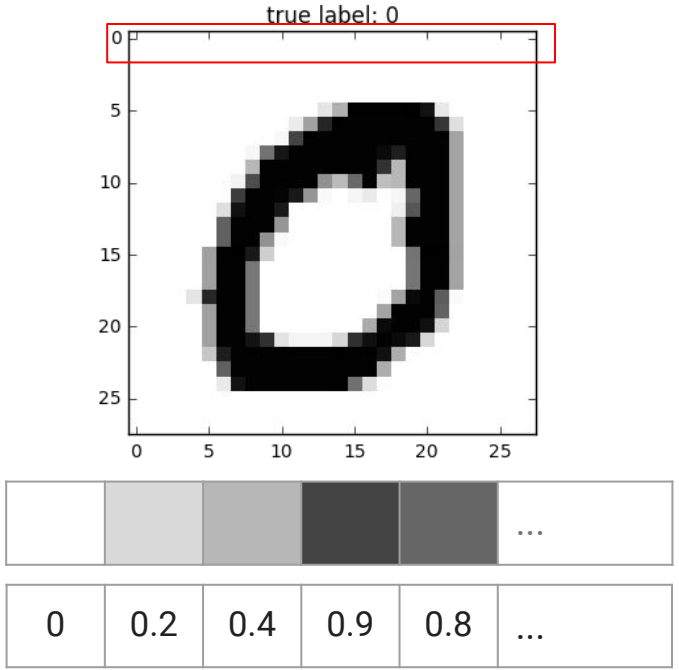
Introducing the MNIST dataset of labeled images

Training
50K

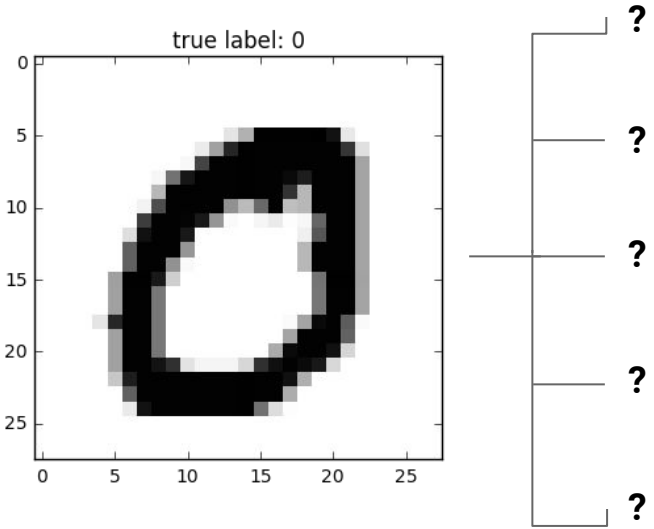


Test
10K

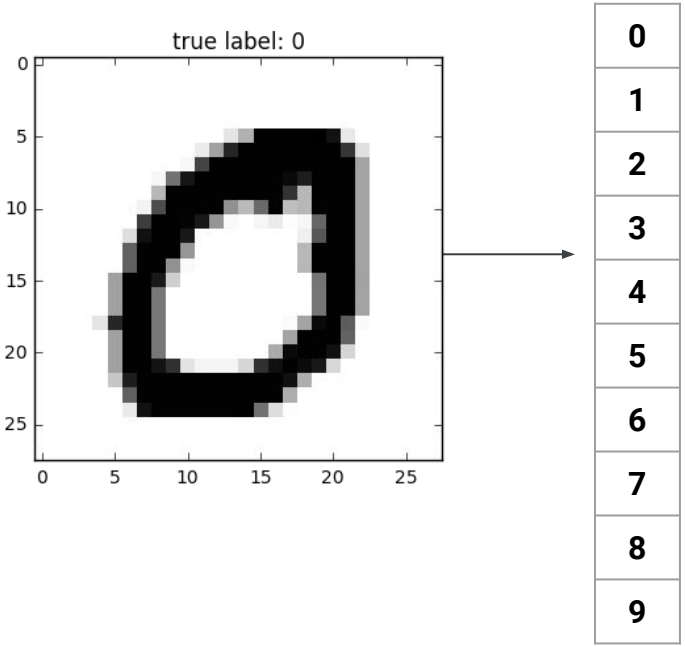
The flattened image is represented as an array



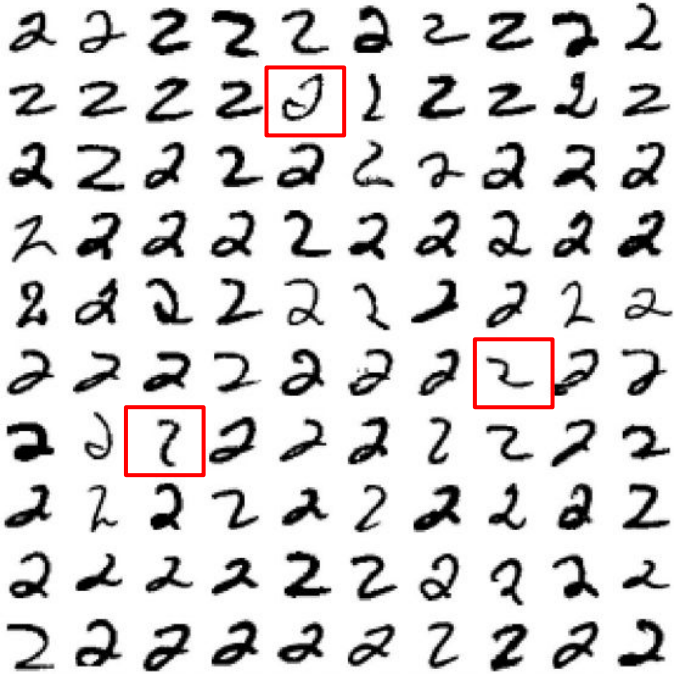
How many output classes do we have?



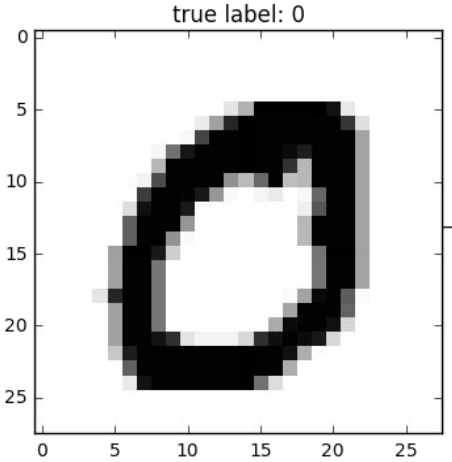
How many output classes do we have?



What if the model is unsure?

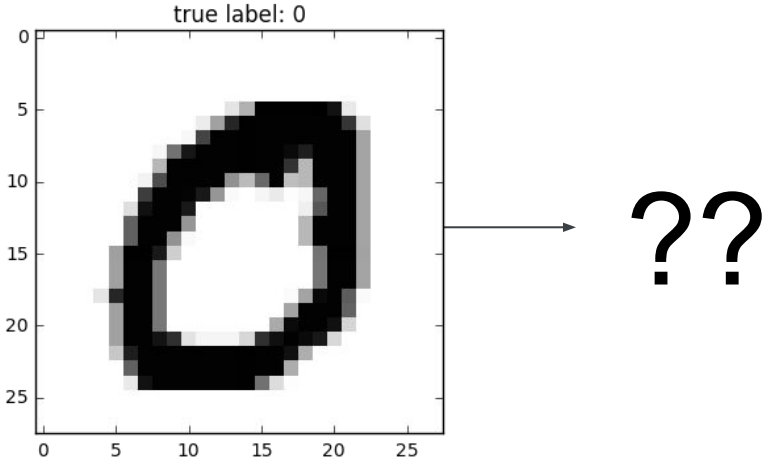


Assessing the model's confidence with a function

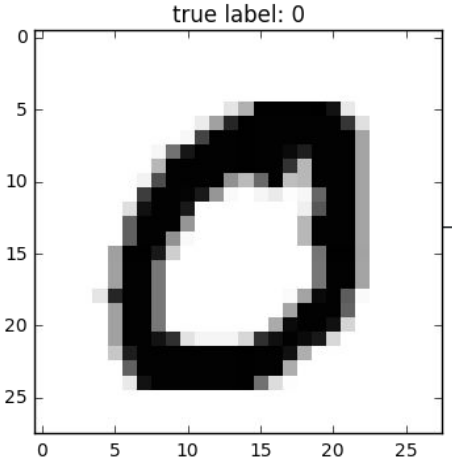


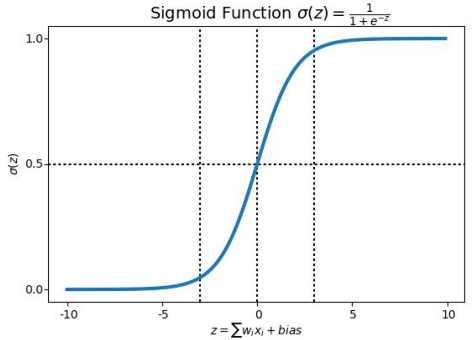
0	20%
1	70%
2	
3	
4	
5	
6	
7	
8	10%
9	

Assessing the model's confidence with a function



Assessing the model's confidence with a function

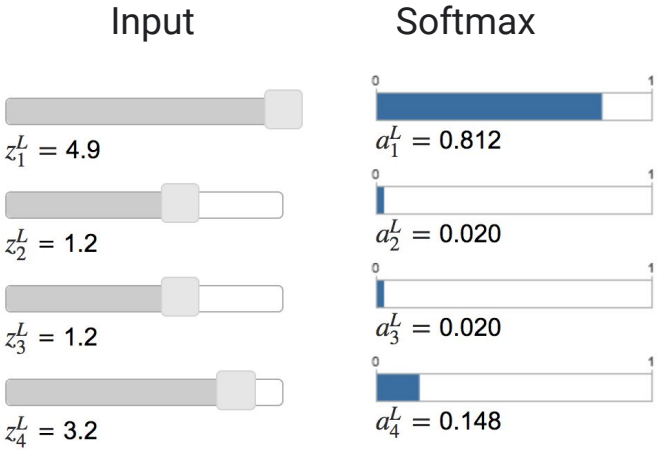




~~Sigmoid?~~
Nope.

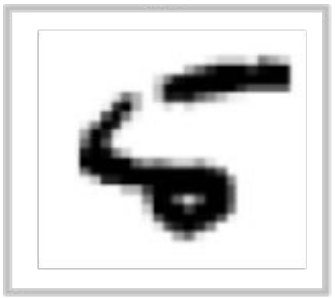
Softmax exponentiating its inputs and then normalizing them

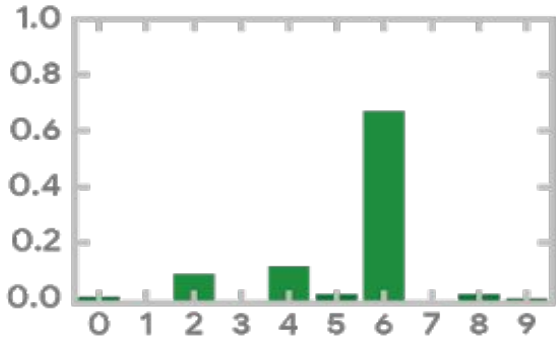
$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$



Softmax exponentiating its inputs and then normalizing them

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$



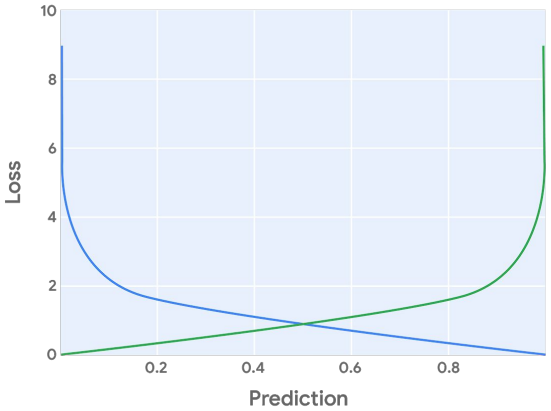


0	2%
1	
2	10%
3	
4	11%
5	3%
6	70%
7	
8	3%
9	1%

X

Training and evaluating our image classification model

Minimize cross-entropy error



$$\frac{-1}{N} \times \sum_{1}^{N} y_i \times log(\hat{y}_i) + (1 - y_i) \times log(1 - \hat{y}_i)$$

Numerically more stable:

`softmax_cross_entropy_with_logits`

Accuracy for model performance

$$\frac{\text{\# correctly classified images}}{\text{\# number total images}}$$

8 → 8
0 → 0
1 → 1
9 → 9
4 → 4
6 → 1 ✖
7 → 7

$$\frac{7}{8} = 88\%$$

Agenda

Introduction

Linear models

Deep neural network models

DNN dropout

Training and evaluating our image classification model

```
def train_and_evaluate(
          model, num_epochs, steps_per_epoch, output_dir):
    """Compiles keras model and loads data into it for training."""
    mnist = tf.keras.datasets.mnist.load_data()
    train_data = util.load_dataset(mnist)
    validation_data = util.load_dataset(mnist, training=False)

    history = model.fit(
        train_data,
        validation_data=validation_data,
        epochs=num_epochs,
        steps_per_epoch=steps_per_epoch)

    if output_dir:
        export_path = os.path.join(output_dir, 'keras_export')
        model.save(export_path, save_format='tf')

    return history
```

Making our image classification model

```
def linear_model():
    model = Sequential([
        Flatten(),
        Dense(NCLASSES),
        Softmax()
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Reading the Data

```
def load_dataset(training=True):
    """Loads MNIST dataset into a tf.data.Dataset"""
    mnist = tf.keras.datasets.mnist.load_data()
    (x_train, y_train), (x_test, y_test) = mnist
    x = x_train if training else x_test
    y = y_train if training else y_test
    # One-hot encode the classes
    y = tf.keras.utils.to_categorical(y, NCLASSES)
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.map(scale).batch(BATCH_SIZE)
    if training:
        dataset = dataset.shuffle(BUFFER_SIZE).repeat()
    return dataset
```

Reading the Data

```
def load_dataset(training=True):
    """Loads MNIST dataset into a tf.data.Dataset"""
    mnist = tf.keras.datasets.mnist.load_data()
    (x_train, y_train), (x_test, y_test) = mnist
    x = x_train if training else x_test
    y = y_train if training else y_test
    # One-hot encode the classes
    y = tf.keras.utils.to_categorical(y, NCLASSES)
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.map(scale).batch(BATCH_SIZE)
    if training:
        dataset = dataset.shuffle(BUFFER_SIZE).repeat()
    return dataset
```

Training our image classification model

```
def scale(image, label):
    image = tf.cast(image, tf.float32)
    image /= 255
    return image, label
```

```
                                    f.data.Dataset"""
                                    .load_data()
                                    test) = mnist
                                    test
    y = y_train if training else y_test
    # One-hot encode the classes
    y = tf.keras.utils.to_categorical(y, NCLASSES)
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.map(scale).batch(BATCH_SIZE)
    if training:
        dataset = dataset.shuffle(BUFFER_SIZE).repeat()
    return dataset
```

Lab

MNIST Image Classification with TensorFlow

- Know how to read and display image data
- Know how to find incorrect predictions to analyze the model
- Visually see how computers see images

https://github.com/GoogleCloudPlatform/training-data-analyst/blob/master/courses/machine_learning/deepdive2/image_classification/labs/mnist_linear.ipynb

Lab Steps

1. Import the training dataset of MNIST handwritten images.
2. Reshape and preprocess the image data.
3. Setup your linear classifier model with 10 classes (one for each possible digit 0 through 9).
4. Train and predict with the linear model
5. Reshape the model weights to visually see what it learned.

Agenda

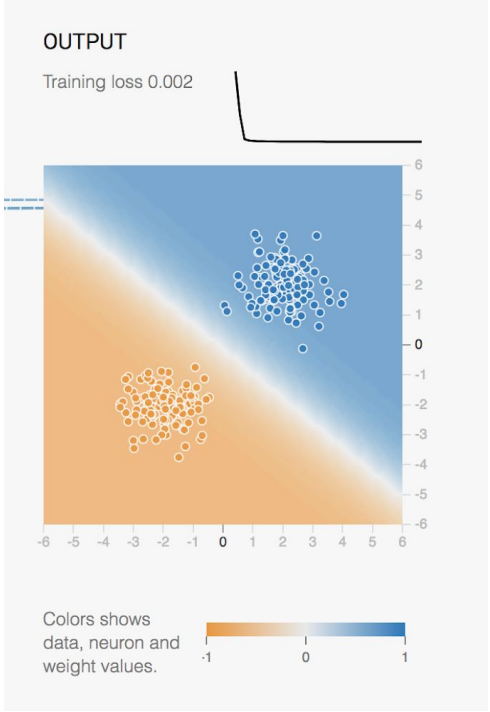
Introduction

Linear models

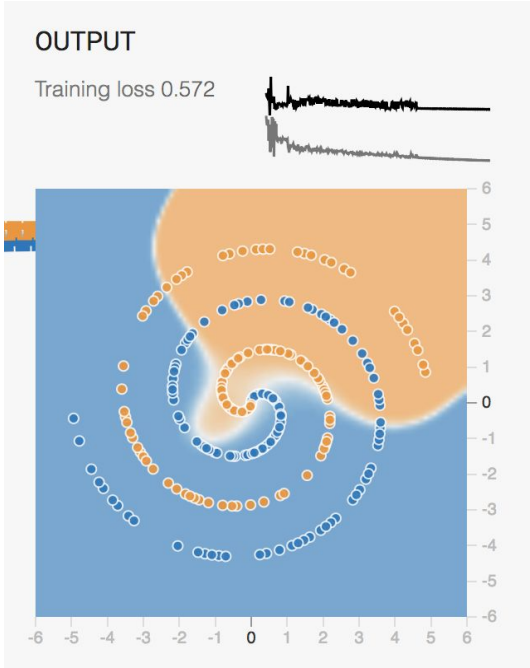
Deep neural network models

DNN dropout

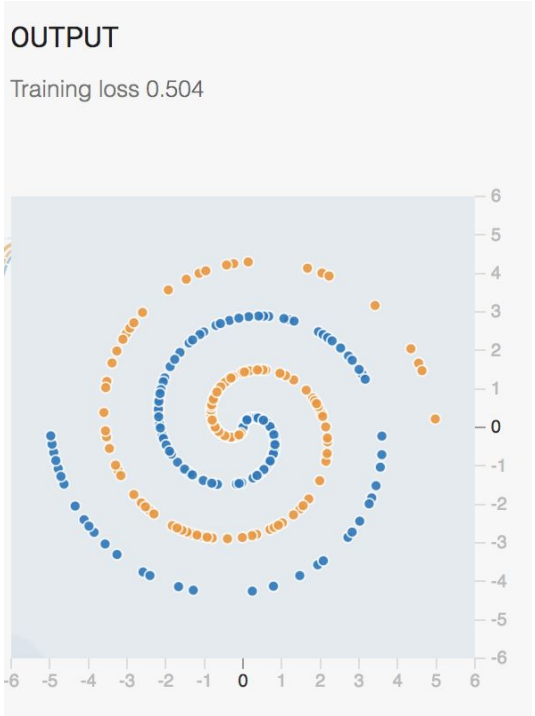
Linear classification models in the TensorFlow playground



For more complex input features, like the data swirl, modelling might not be so easy

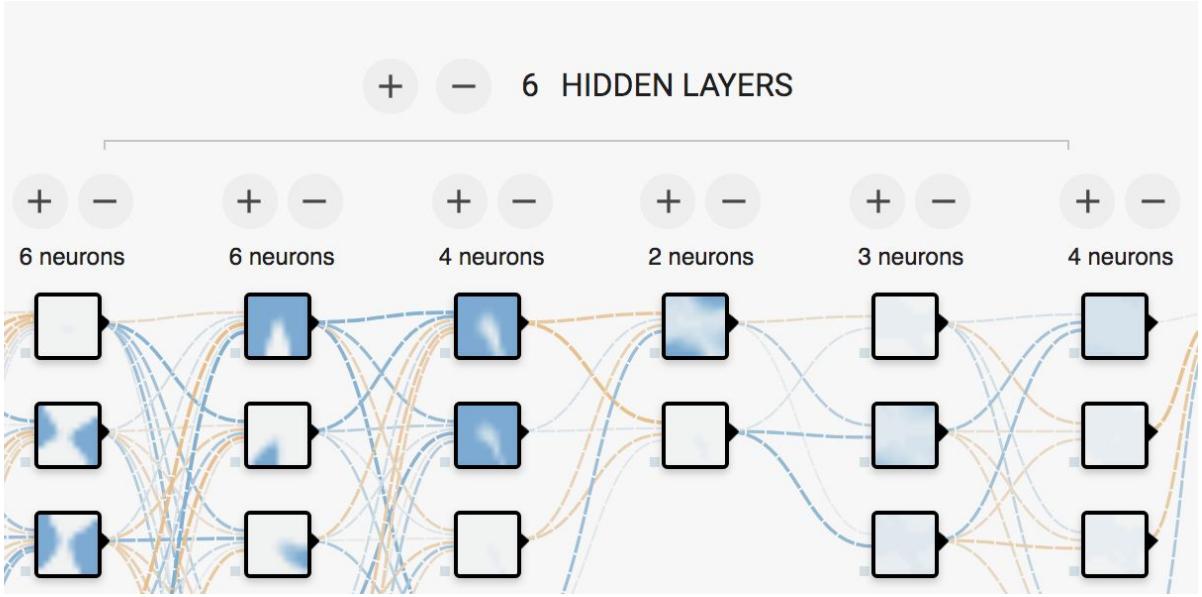


Let’s see how a neural network does

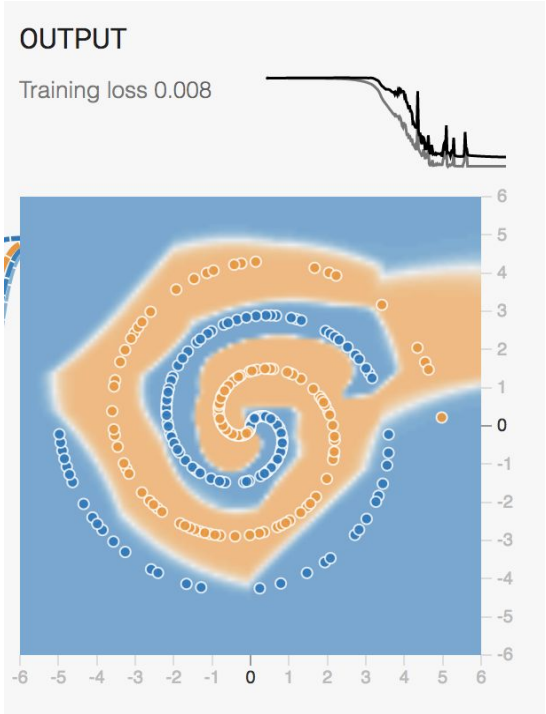


The choice of Activation Function is what separates linear models from neural networks

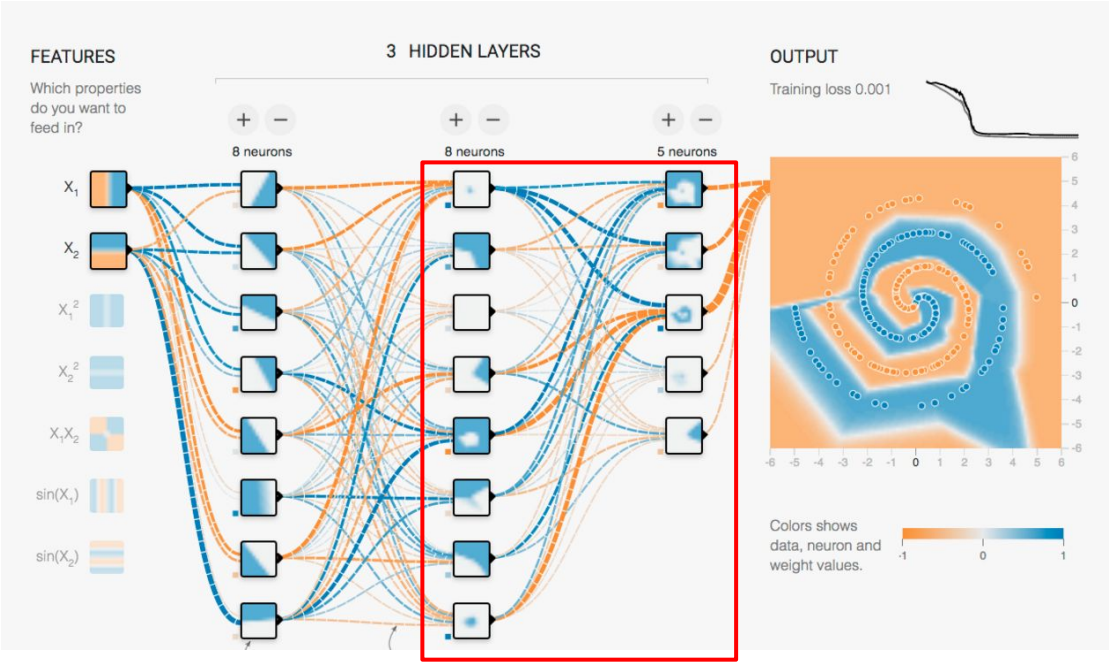
The hidden layers feature allows you to change the number of hidden layers and the number of neurons within each hidden layer



Let’s see how a neural network does



The relationship between the first hidden layer and those that come after it



Agenda

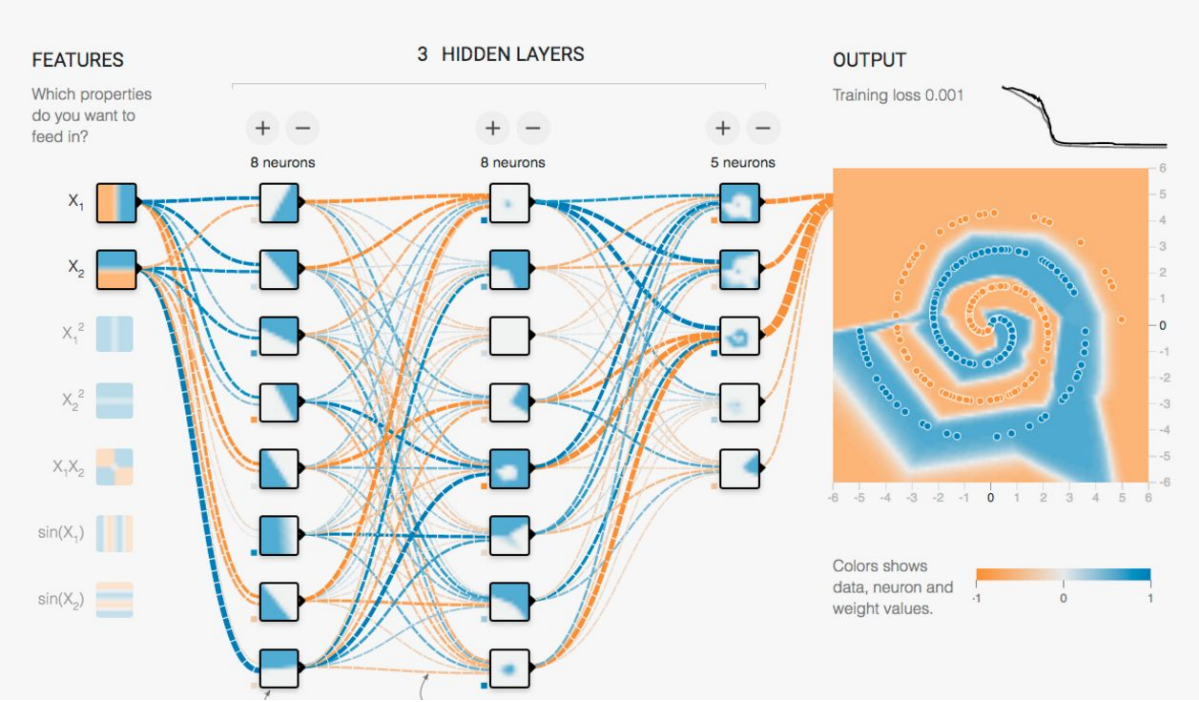
Introduction

Linear models

Deep neural network models

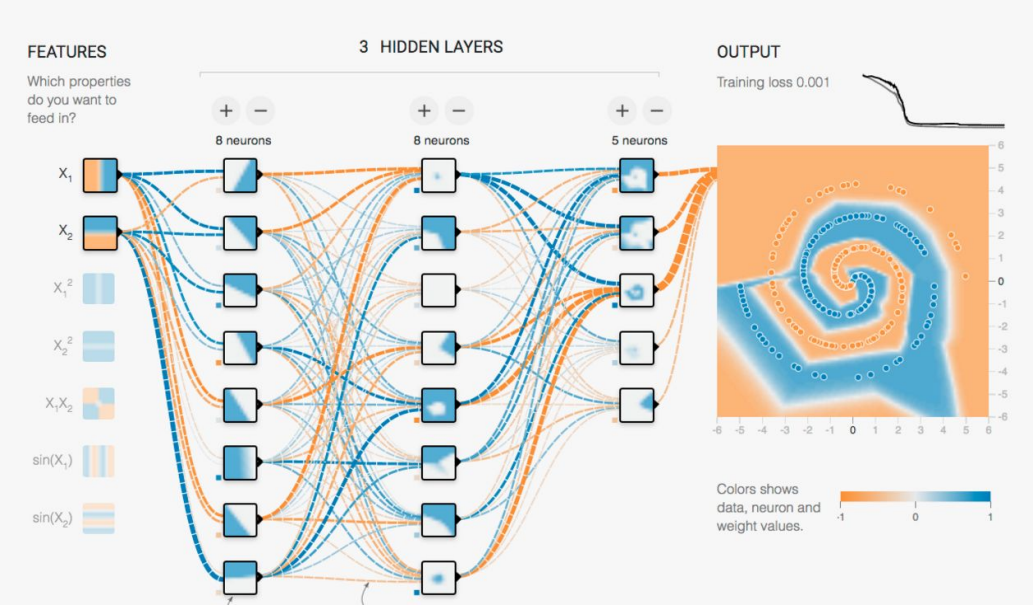
DNN dropout

An infinitely large DNN could classify anything right?



An infinitely large DNN could ~~classify~~ *memorize* anything

Why not have a really large DNN?



1. Computational power
2. Training time
3. Likely to overfit

Recall: How can we combat overfitting in a DNN?

One of the best ways we have of mitigating overfitting is through the use of regularization.

Quiz: Which form of regularization is only used with neural networks?

A. Dropout
B. L1
C. L2

Quiz: Which form of regularization is only used with neural networks?

A. Dropout

B. L1

C. L2

L1 and L2 are implemented using terms added to the loss function

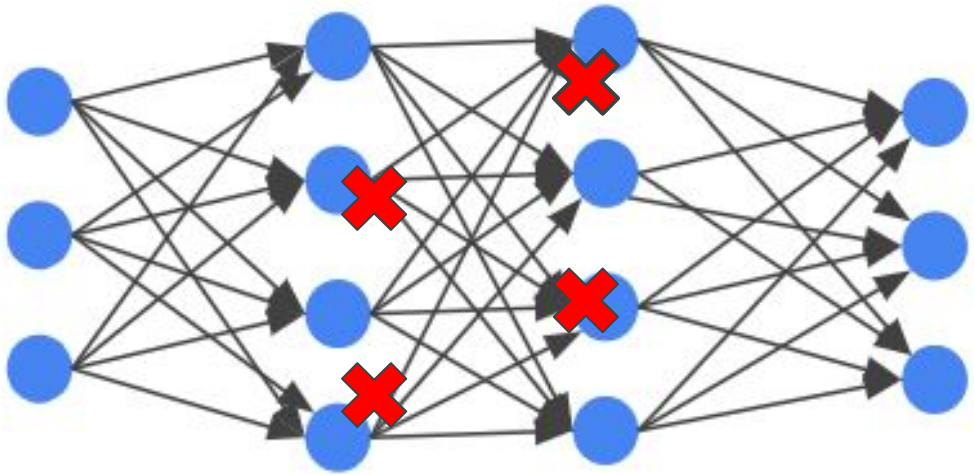
$$\lambda \sum_{i=1}^{k} |\omega_i|$$

L1 regularization term

$$\lambda \sum_{i=1}^{k} \omega_i^2$$

L2 regularization term

Use dropout to avoid overfitting in a DNN

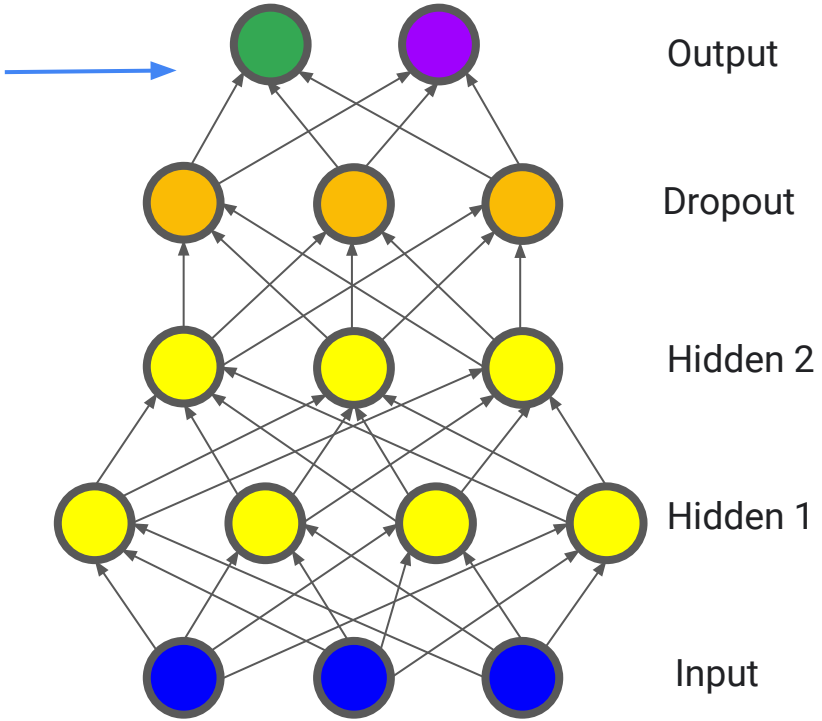


Dropout layers are a form of regularization

Dropout works by randomly “dropping out” unit activations in a network for a single gradient step.

During training only!
In prediction all nodes are kept.

Helps learn “multiple paths” -- think: ensemble models, random forests.



Adding a dropout layer to your DNN

```
model = Sequential([
    ...
    Dense(dropout_neurons, activation='relu'),

    Dropout(dropout_rate),

    Dense(after_dropout_neurons),
    ...
])
```

Google Cloud

Introduction to Convolutional Neural Networks

Agenda

Convolutional neural networks

Understanding convolutions

Parameters (padding, stride)

Pooling layers

Implementing CNNs

Architecture walkthrough

Neural networks for image recognition can get complex

← **3264px** Width →

←**2448px** Height →

8 Megapixel resolution

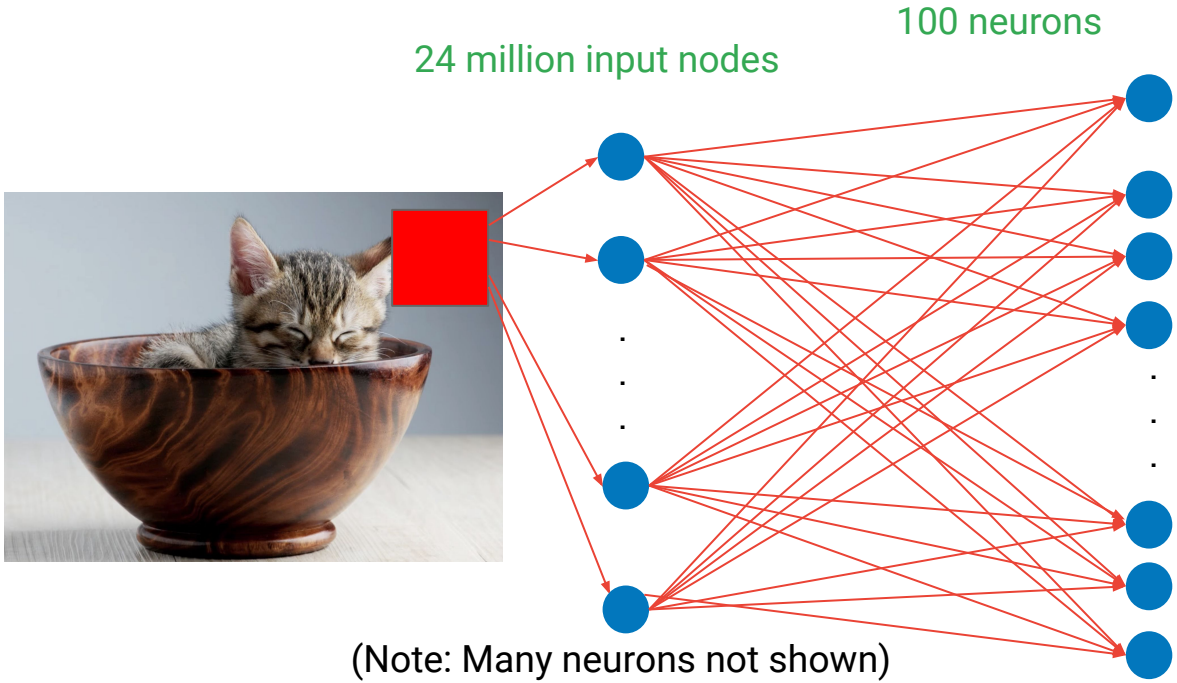
3264 (w) x **2448** (h) x **3** (RGB) =

23,970,816 per image

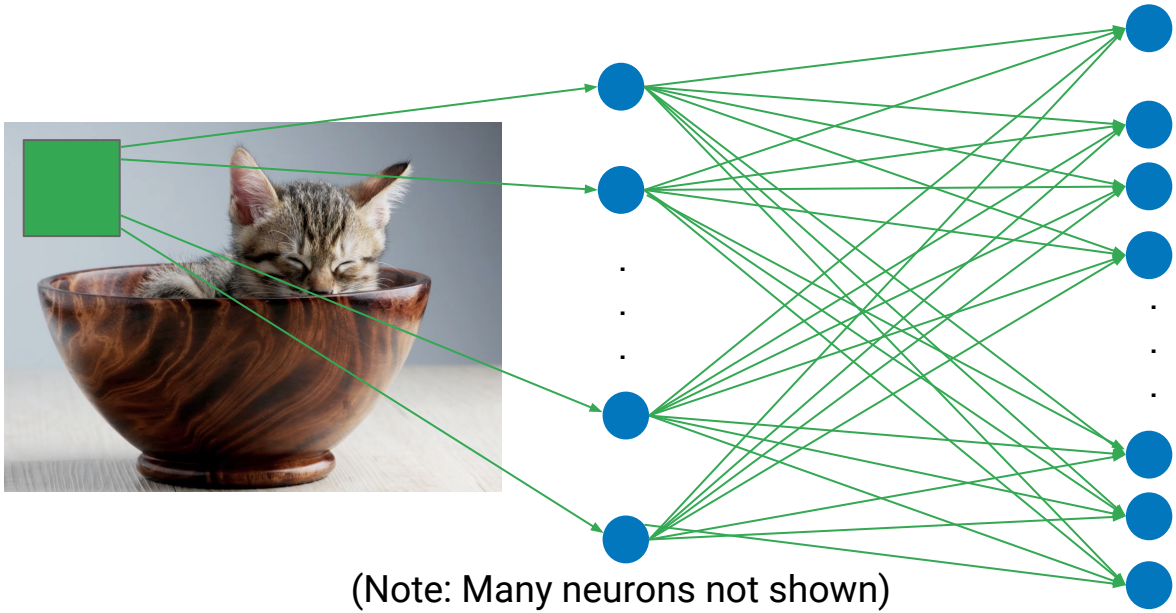
23,970,816 * 100 neurons =

2.3 Billion weights

Using only a DNN for image classification could have billions of weights

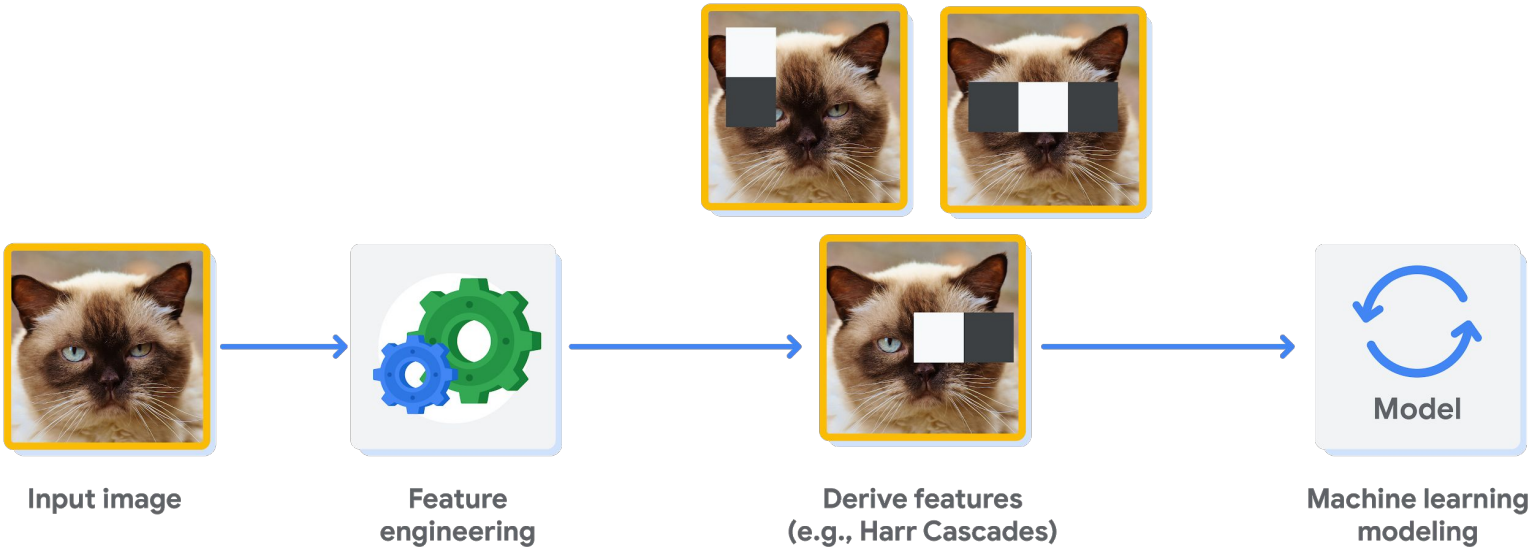


Using only a DNN for image classification could have billions of weights

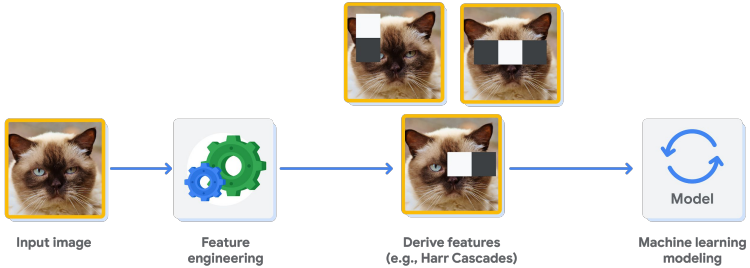


(Note: Many neurons not shown)

Traditionally, image classification tasks utilize a feature engineering step



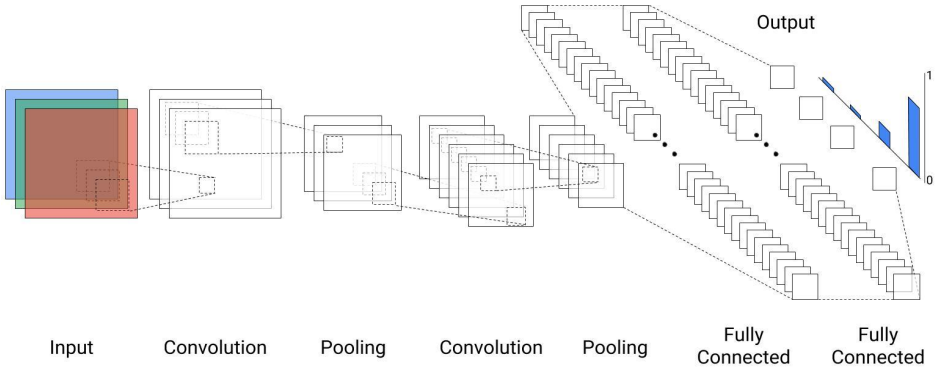
Pre-2012 versus Post-2012



Pre-2012

Post-2012

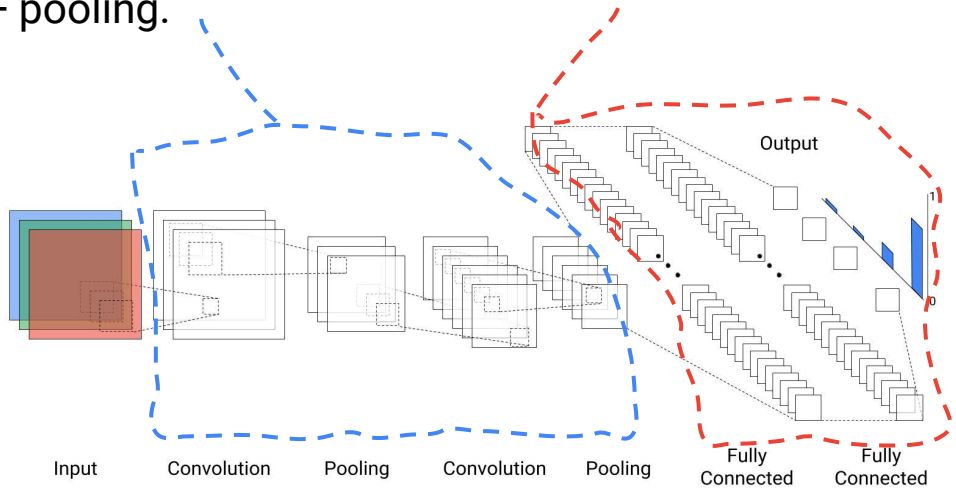
CNN model learns features itself during training.



Pre-2012 versus Post-2012

CNN feature engineering with convolutions + pooling.

Traditional deep neural network for classification.



Post-2012

CNN model learns features itself during training.

Agenda

Convolutional neural networks

Understanding convolutions

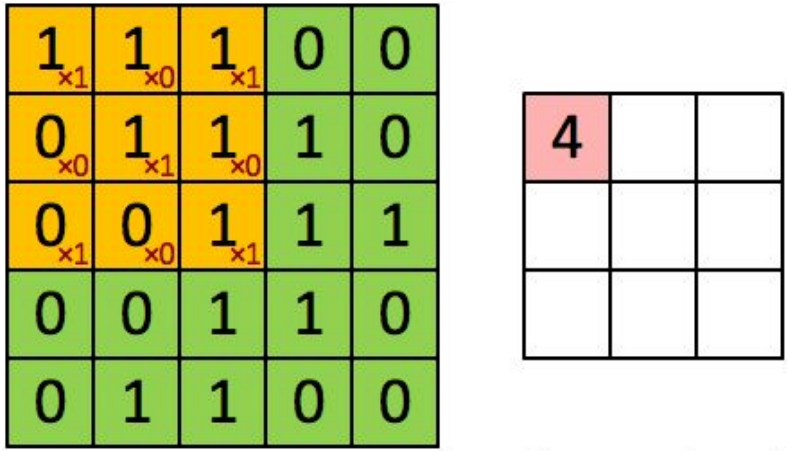
Parameters (padding, stride)

Pooling layers

Implementing CNNs

Architecture walkthrough

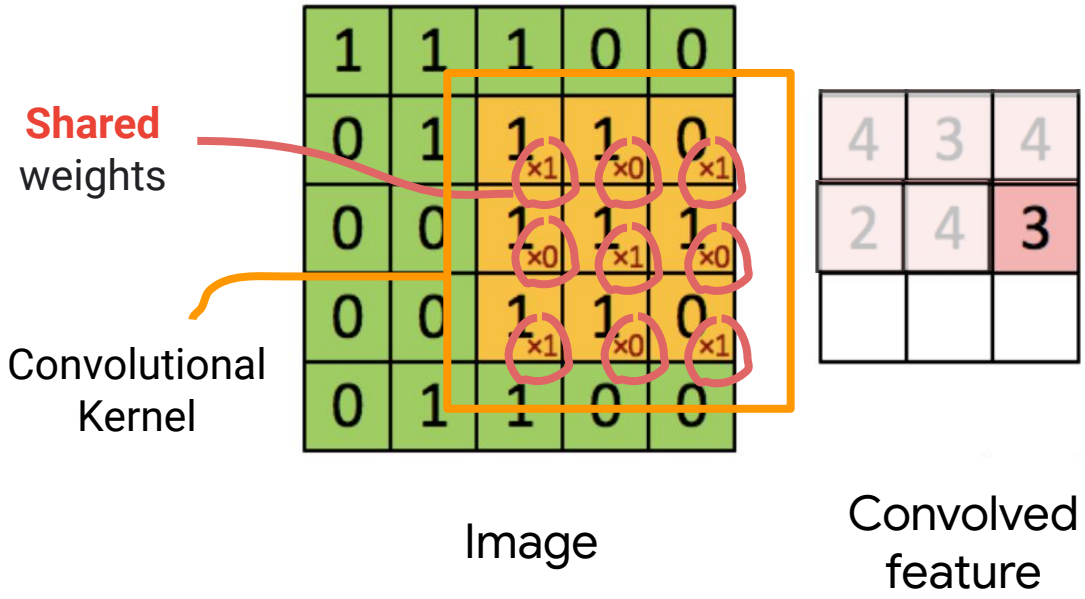
A convolution is an operation that processes groups of nearby pixels



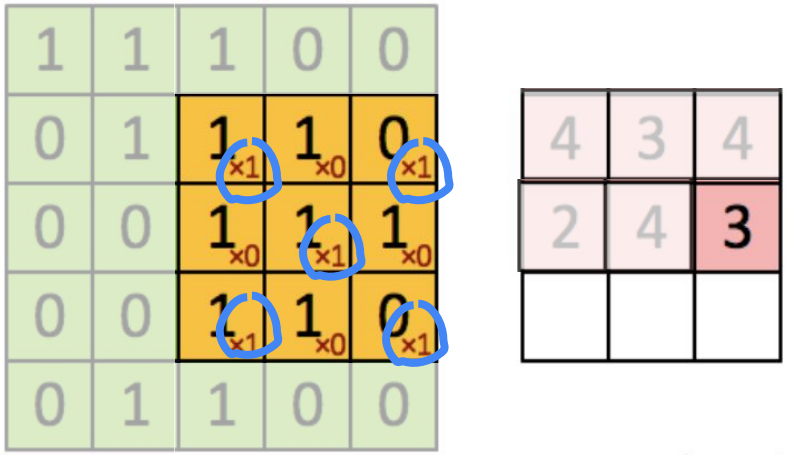
Image

Convolved feature

A convolution is an operation that processes groups of nearby pixels



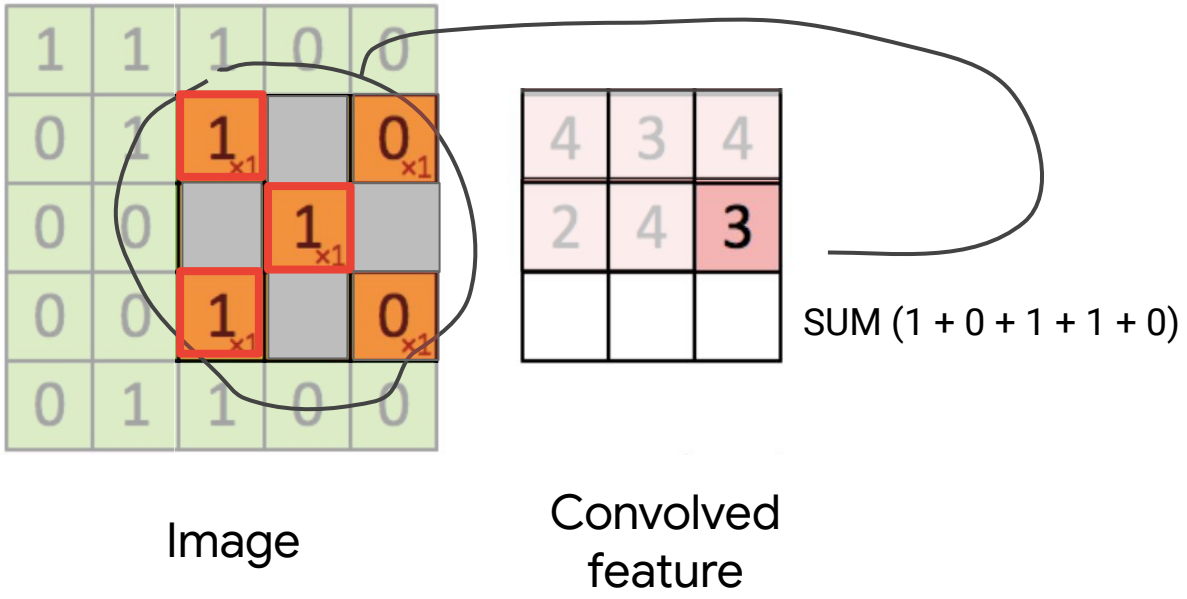
A convolved feature map is created by multiplying the kernel weights by the features



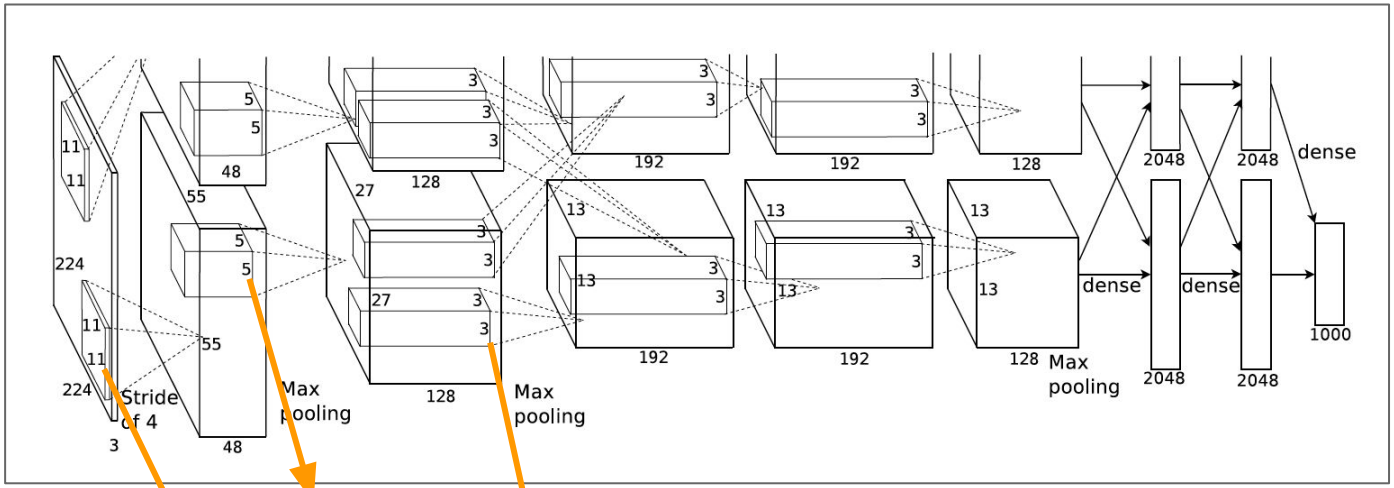
Image

Convolved feature

A convolved feature map is created by multiplying the kernel weights by the features



Convolutional neural networks commonly use multiple kernels



5x5

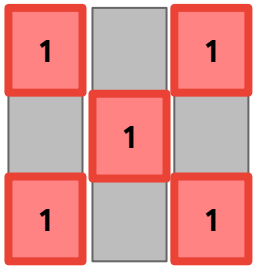
3x3

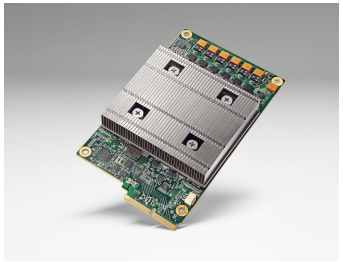
11x11

AlexNet CNN Architecture 2012

The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches https://arxiv.org/abs/1803.01164

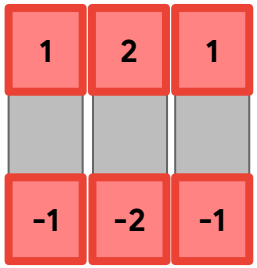
Convolutional neural networks commonly use multiple kernels



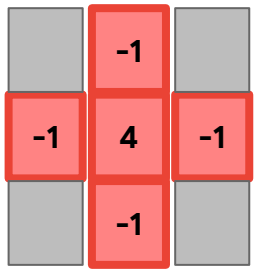


TPU

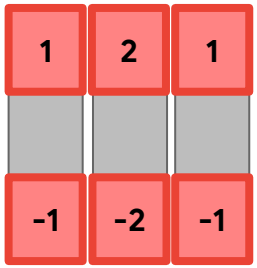
Convolutional neural networks commonly use multiple kernels



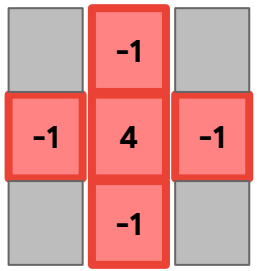
Detects
horizontal edges



Convolutional neural networks commonly use multiple kernels

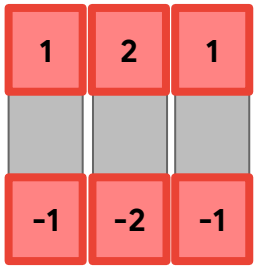


Detects
horizontal edges

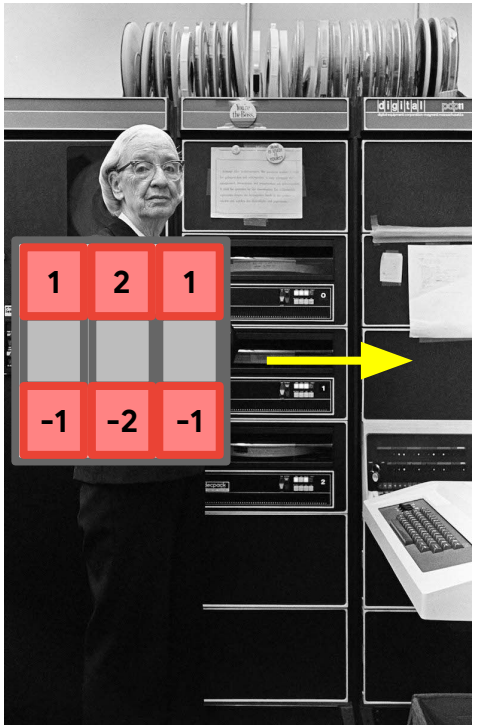


Detects bright spots

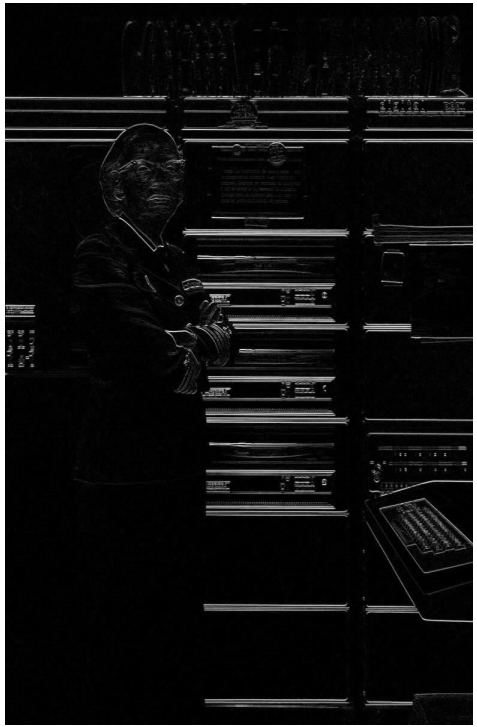
Different weights detect different features



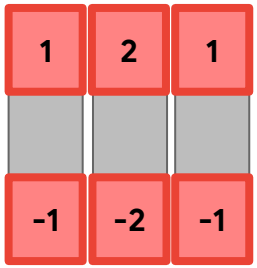
Detects
horizontal edges



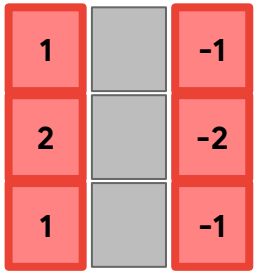
Different weights detect different features



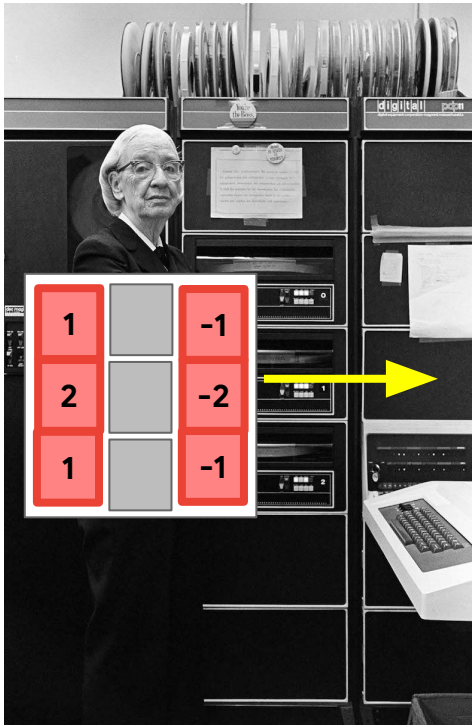
Different weights detect different features



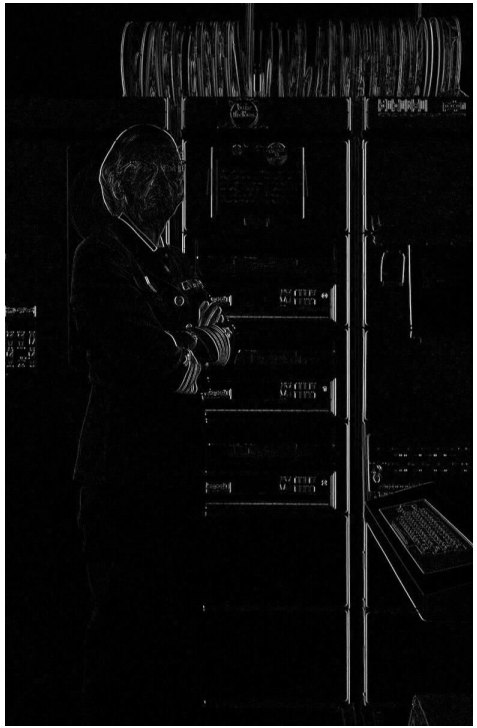
Detects
horizontal edges



Detects
vertical edges



Different weights detect different features



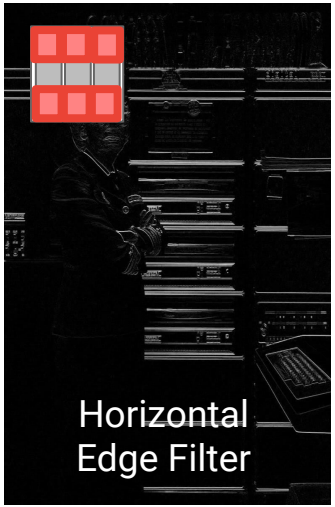
Different weights detect different features

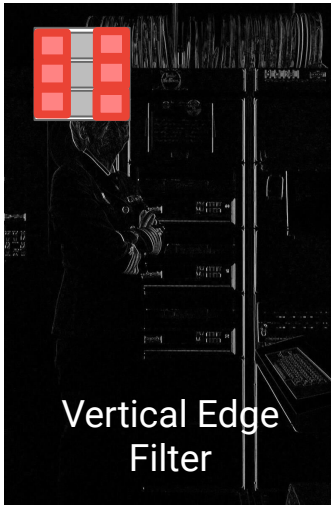
Kernel 1

1	2	1
-1	-2	-1

Kernel 2

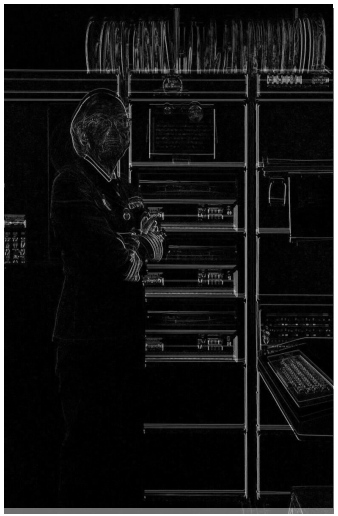
1		-1
2		-2
1		-1



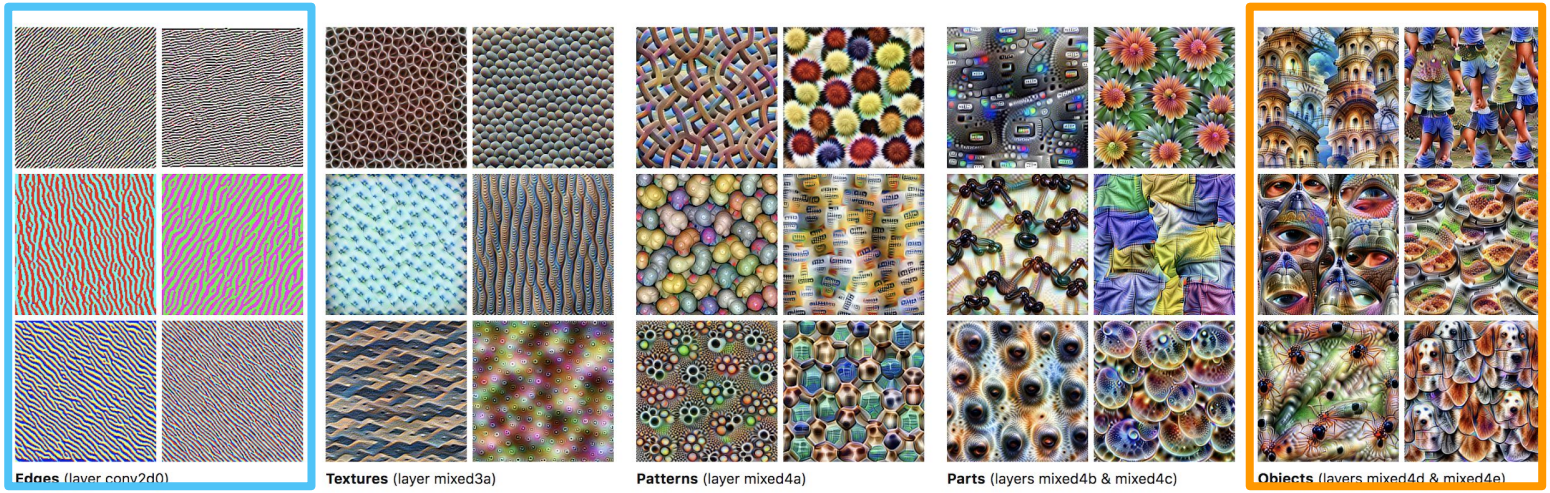


=

Sum of the two filter outputs



CNNs can learn a hierarchy of features



https://distill.pub/2017/feature-visualization/

Quiz: What is the process of "sliding" a kernel across an image called?

A. Convolution
B. Confusion matrix
C. Contortion
D. Curved detection

Quiz: What is the process of "sliding" a kernel across an image called?

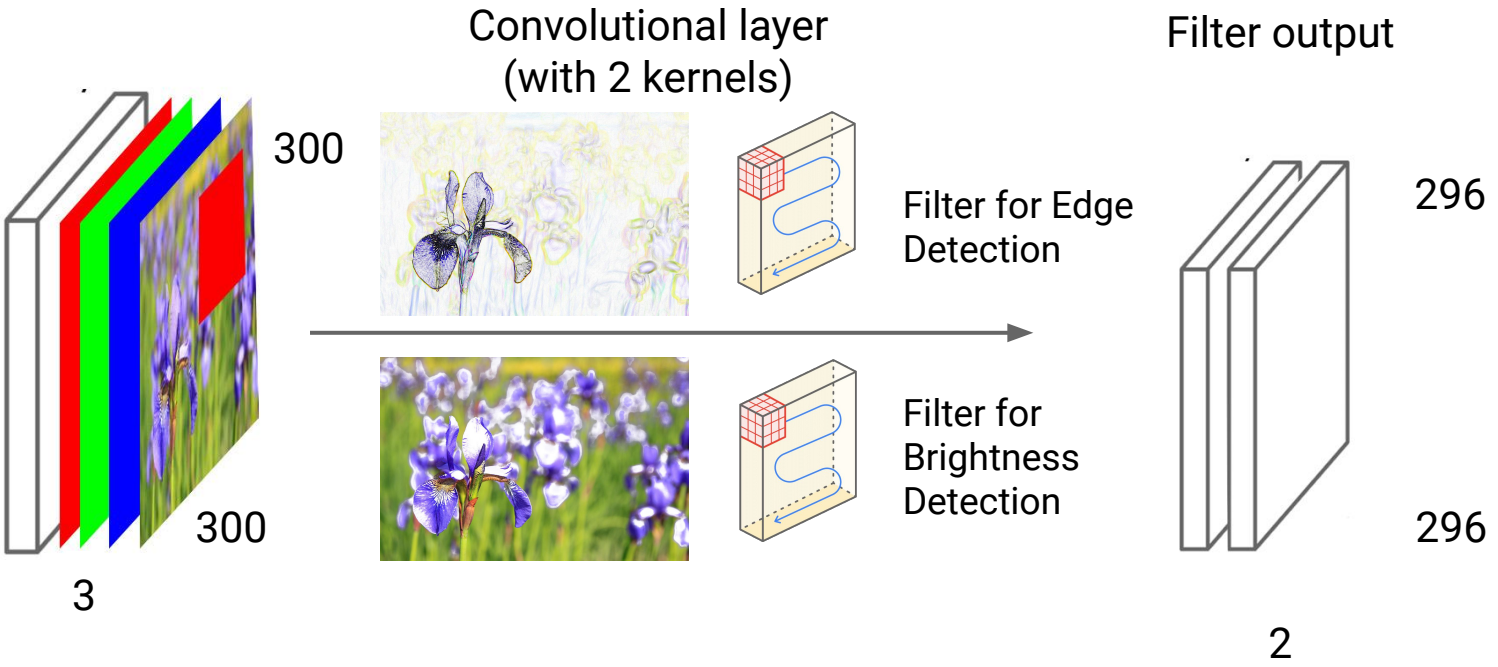
A. Convolution

B. Confusion matrix

C. Contortion

D. Curved detection

Convolutional layers are collections of filters



Agenda

Convolutional Neural Networks

Understanding Convolutions

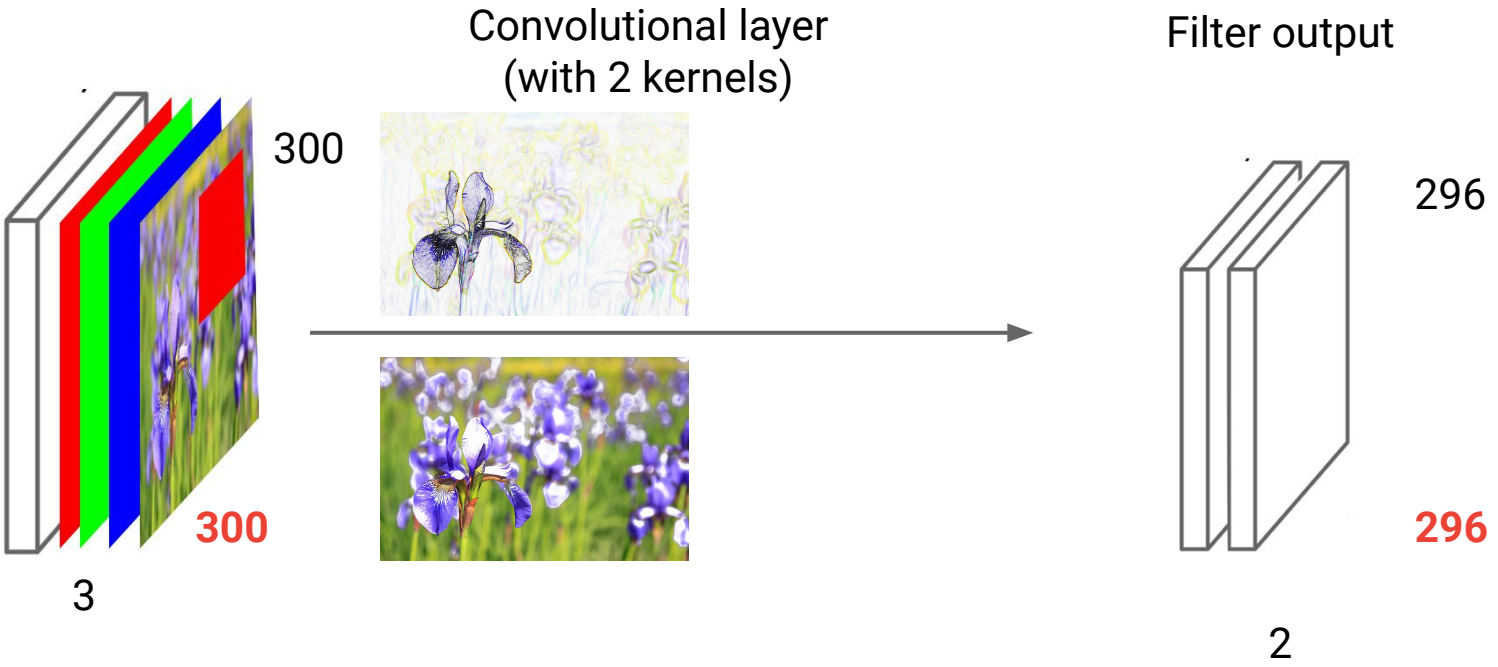
Parameters (padding, stride)

Pooling layers

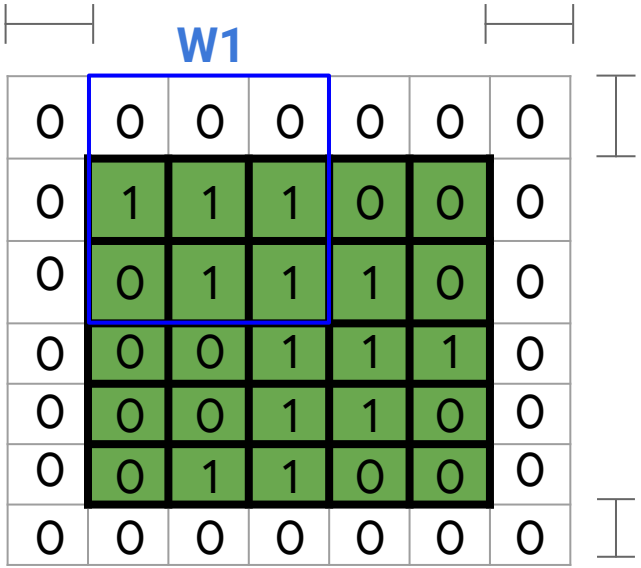
Implementing CNNs

Architecture walkthrough

Convolutional layers are collections of filters



Padding preserves the shape of the input after the convolution

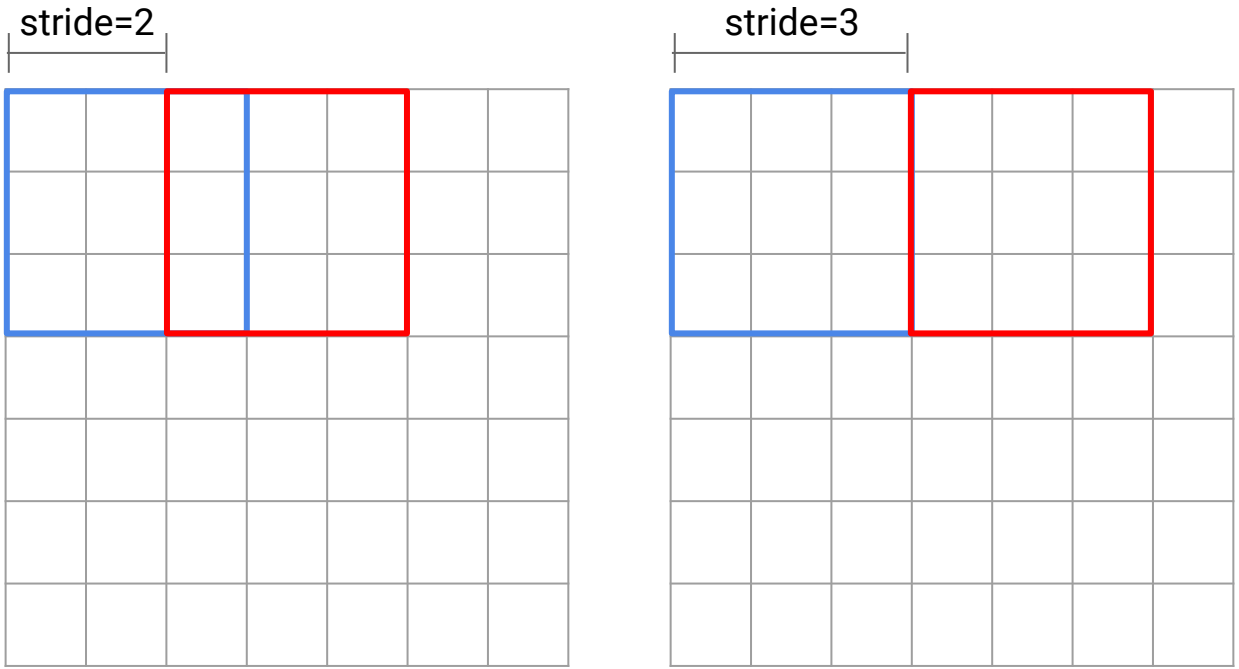


"Same" padding for 3x3 kernel size.

Keras provides a high level API to set up a convolutional layer

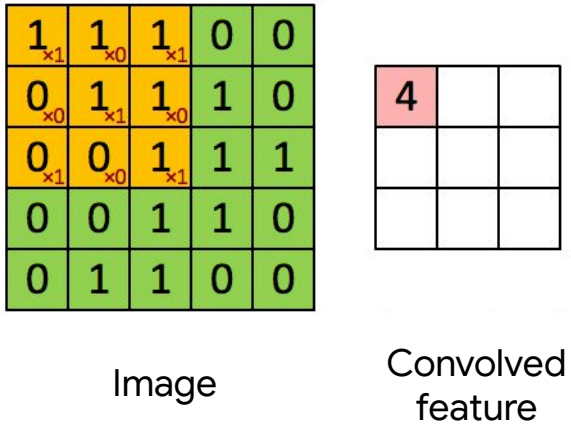
```
tensorflow.keras.layers.Conv2D(
    filters,     # number of filters, i.e. out_channels
    kernel_size=3,   # size of the kernel, e.g. 3 for a 3x3 kernel
    padding='same'    # maintain the same shape across the input and output
    input_shape=batch_shape      # (batch_size, height, width, in_channels)
    …
 )
```

Dimensionality reduction decreases the total # of parameters and computation time



Quiz: What does the green part of the diagram below represent?

A. The kernel convolving
B. The filter output
C. The original image

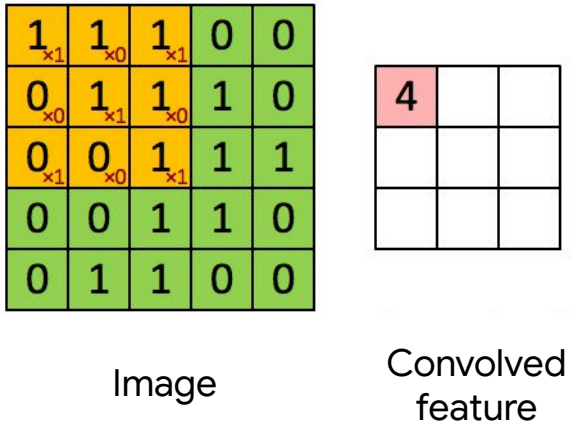


Quiz: What does the green part of the diagram below represent?

A. The kernel convolving

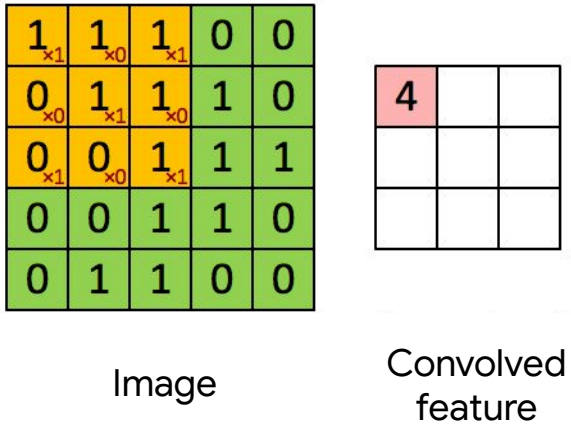
B. The filter output

C. The original image



Quiz: What does the yellow part of the diagram below represent?

A. The kernel convolving
B. The filter output
C. The original image

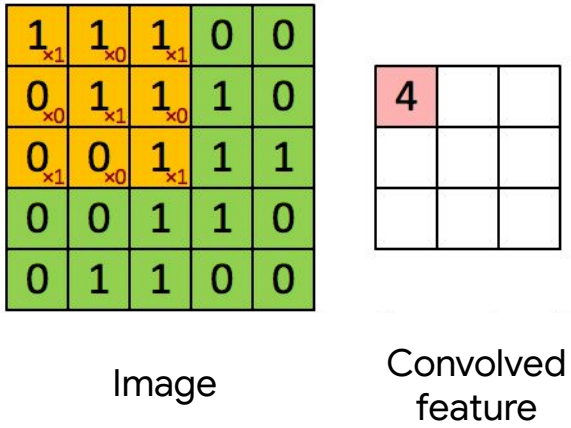


Quiz: What does the yellow part of the diagram below represent?

A. The kernel convolving

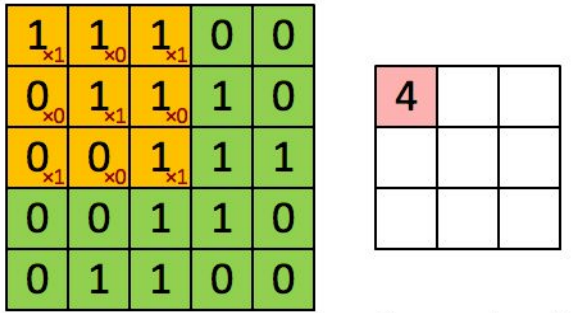
B. The filter output

C. The original image



Quiz: What is the size of the stride step in the diagram below?

A. Stride of 3
B. Stride of 2
C. Stride of 1
D. Unknown



Image

Convolved feature

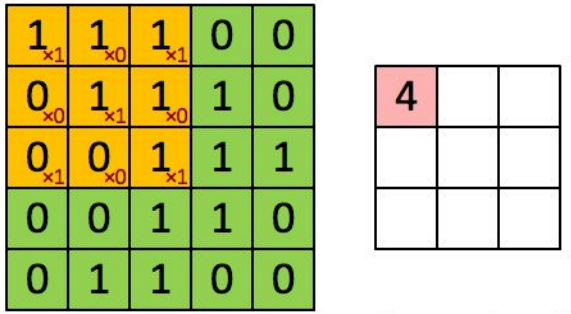
Quiz: What is the size of the stride step in the diagram below?

A. Stride of 3

B. Stride of 2

C. Stride of 1

D. Unknown



Image

Convolved feature

Agenda

Convolutional neural networks

Understanding convolutions

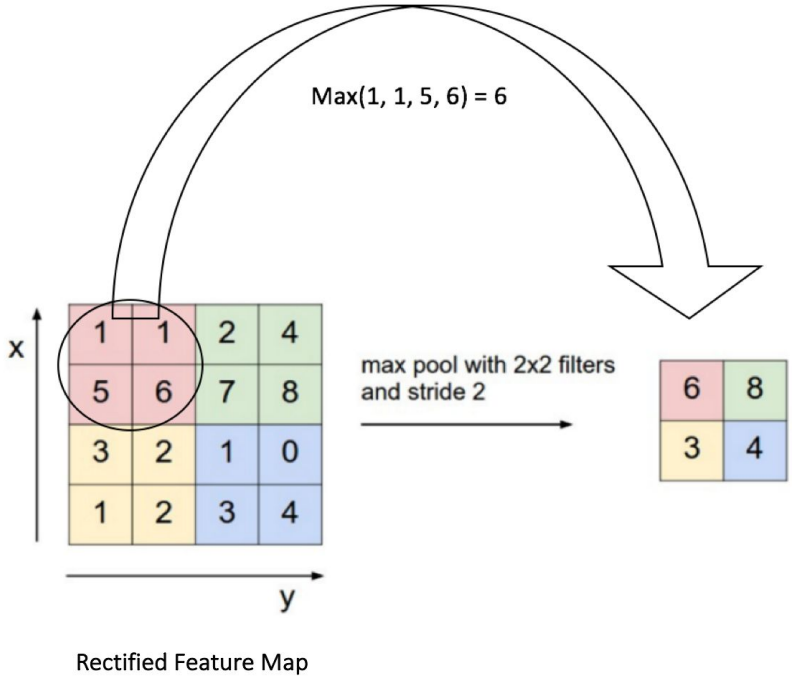
Parameters (padding, stride)

Pooling layers

Implementing CNNs

Architecture walkthrough

Maxpooling also reduces dimensionality



Maxpooling example

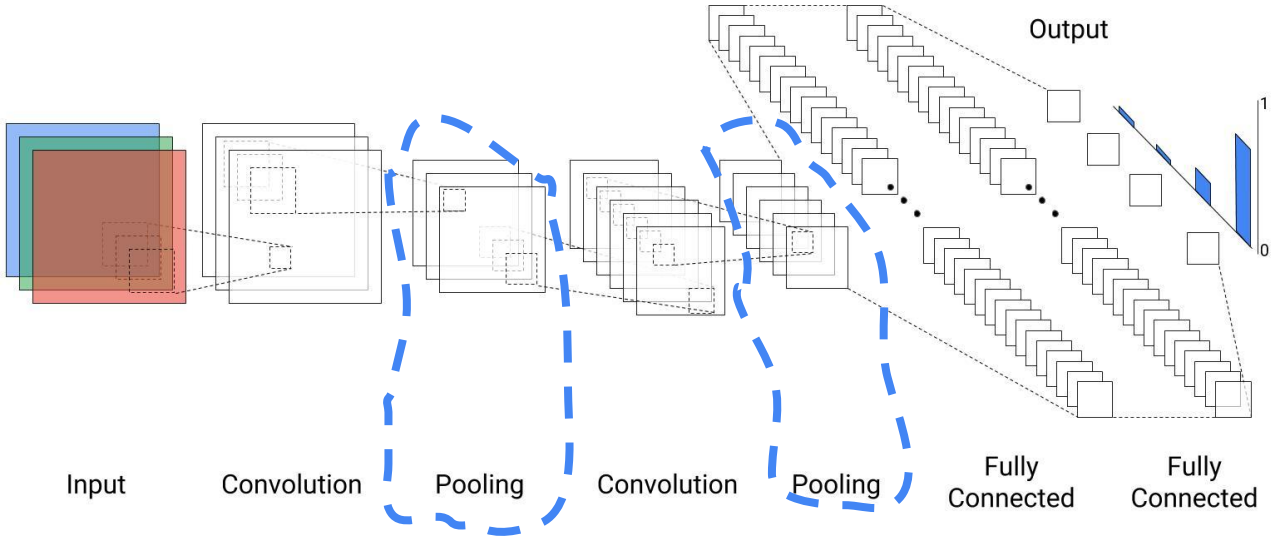
Maxpool

Kernel: 2x2
Stride: 2

Brighter values correspond to larger pixel values.

Dimensionality is decreased by a factor of 2 both horizontal and vertically.

Maxpooling operations are just additional layers in our network



TensorFlow provides the tf.layers.max_pooling2d API

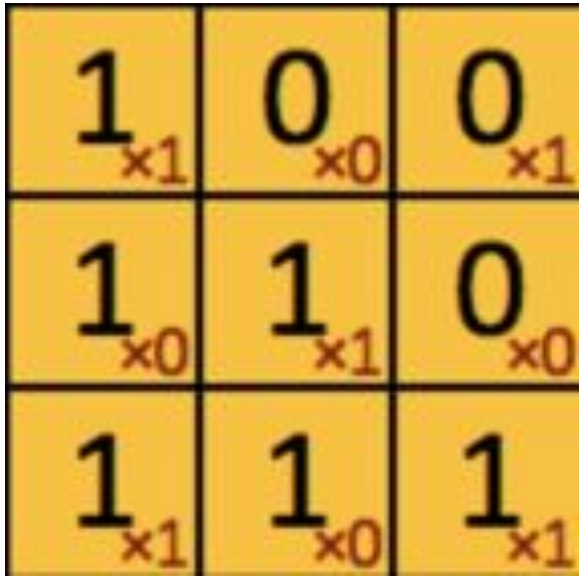
```
tf.keras.layers.MaxPool2D(
           pool_size=2,     # size of the maxpooling kernel
           strides=2        # size of the stride step
)
```

Quiz: What do the smaller red numbers represent in this image you've seen before?

A. The weights of the particular kernel we are applying (i.e. what feature we are detecting).

A. The intensity of the pixel for that area of the original image.

A. The channel depth of the image.

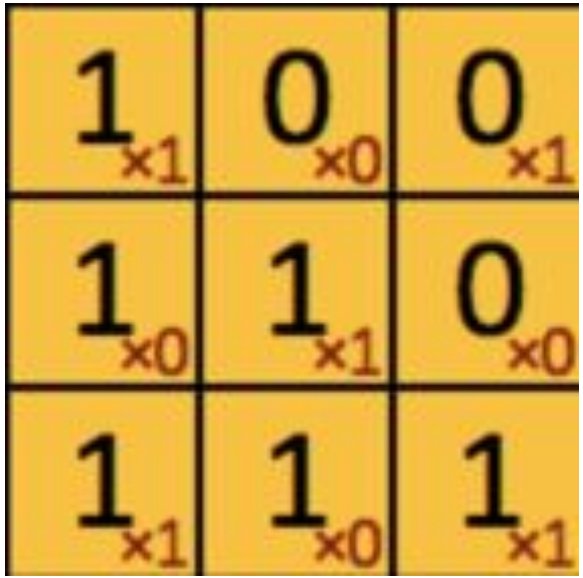


Quiz: What do the smaller red numbers represent in this image you've seen before?

A. The weights of the particular kernel we are applying (i.e. what feature we are detecting).

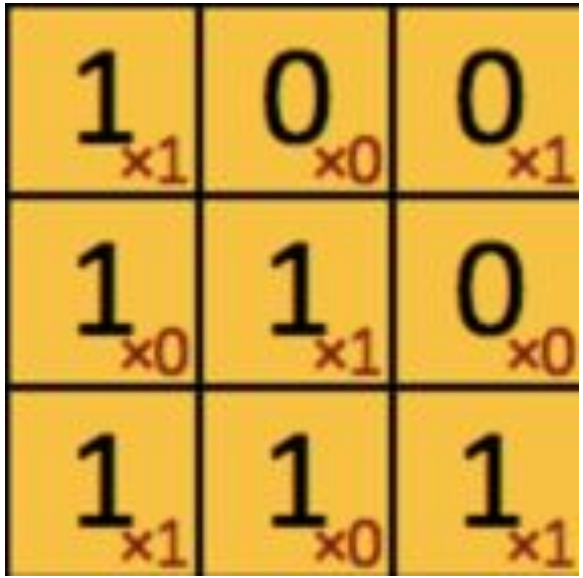
A. The intensity of the pixel for that area of the original image.

A. The channel depth of the image.



Quiz: Following on from the previous question, what is the value of the output that the kernel will generate for this part of the image?

A. 0
B. 1
C. 2
D. 3
E. 4
F. 5



Quiz: Following on from the previous question, what is the value of the output that the kernel will generate for this part of the image?

A. 0

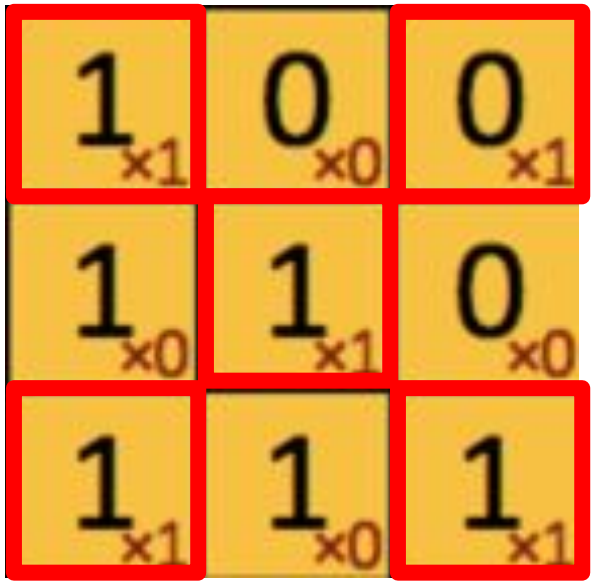
B. 1

C. 2

D. 3

E. 4

F. 5



Agenda

Convolutional neural networks

Understanding convolutions

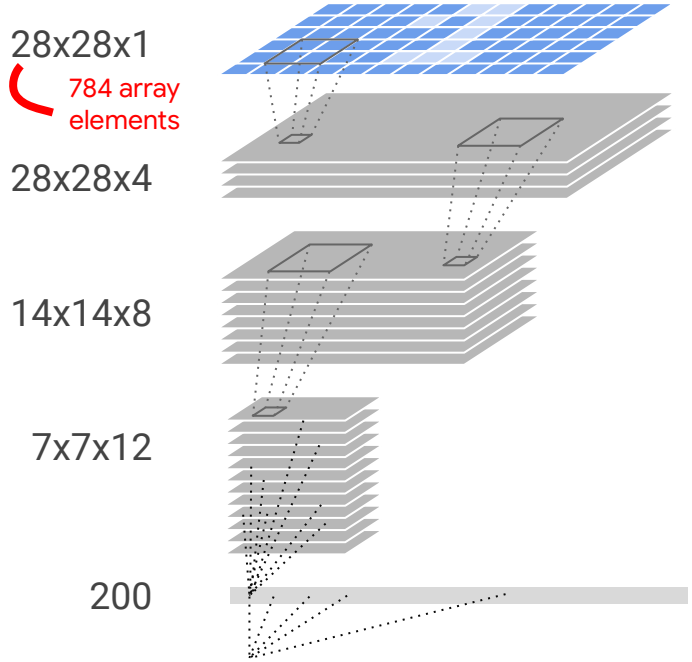
Parameters (padding, stride)

Pooling layers

Implementing CNNs

Architecture walkthrough

Successive convolution layers apply filters to increasing scales



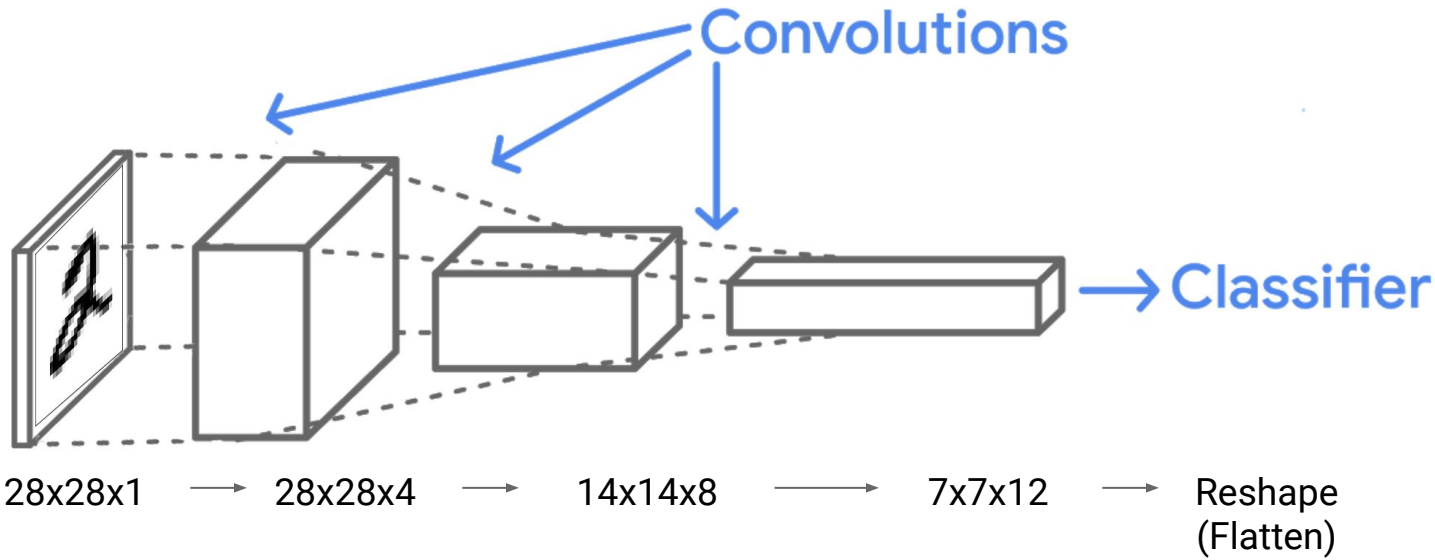
convolutional layer, 4 channels deep
W1[5, 5, 1, 4] stride 1

convolutional layer, 8 channels deep
W2[3, 3, 4, 8] stride 2

convolutional layer, 12 channels deep
W3[3, 3, 8, 12] stride 2

fully connected layer W4[7x7x12, 200]
softmax readout layer W5[200, 10]

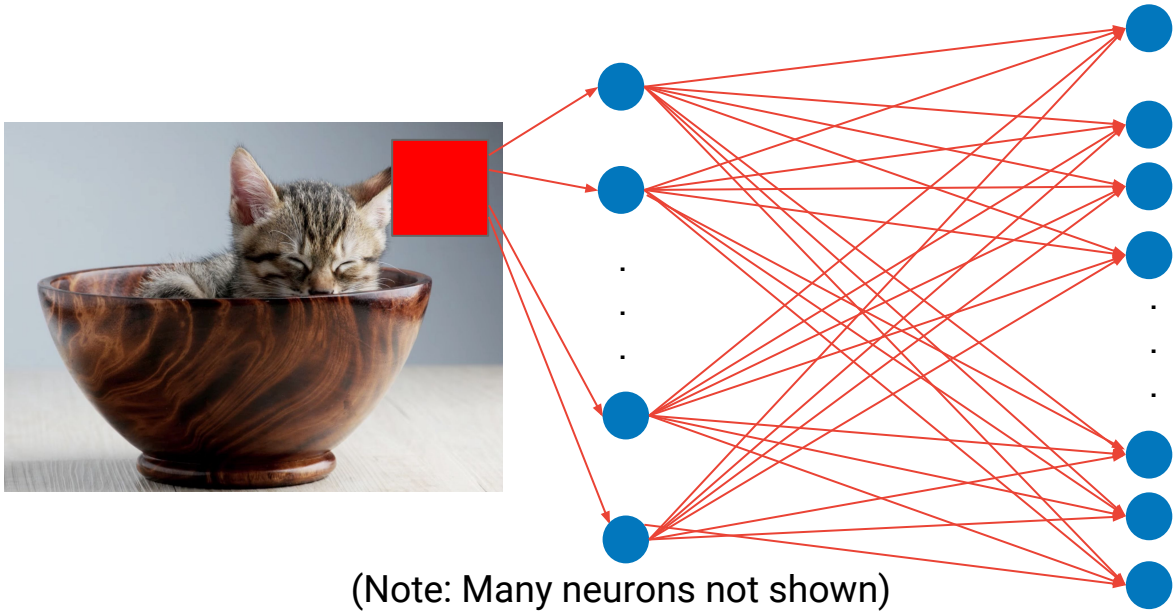
CNN model architecture for MNIST image



Setting up a model with convolutional layers

```
model = Sequential([
    Conv2D(num_filters_1, kernel_size=8,
           activation='relu', input_shape=(WIDTH, HEIGHT, 1)),
    MaxPooling2D(pool_size=2),
    Conv2D(num_filters_2, kernel_size=4,
           activation='relu'),
    MaxPooling2D(pool_size=2),
    Flatten(),
    Dense(units_1, activation='relu'),
    Dense(units_2, activation='relu'),
    Dropout(dropout_rate),
    Dense(nclasses),
    Softmax()
])
```

Using only a DNN for image classification could have billions of weights



(Note: Many neurons not shown)

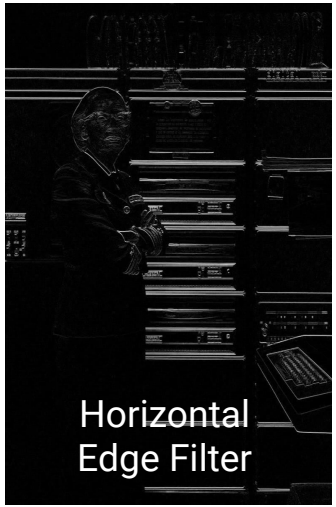
Kernels detect patterns and output filters

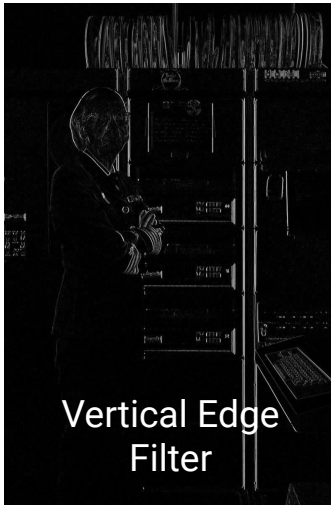
Kernel 1

1	2	1
-1	-2	-1

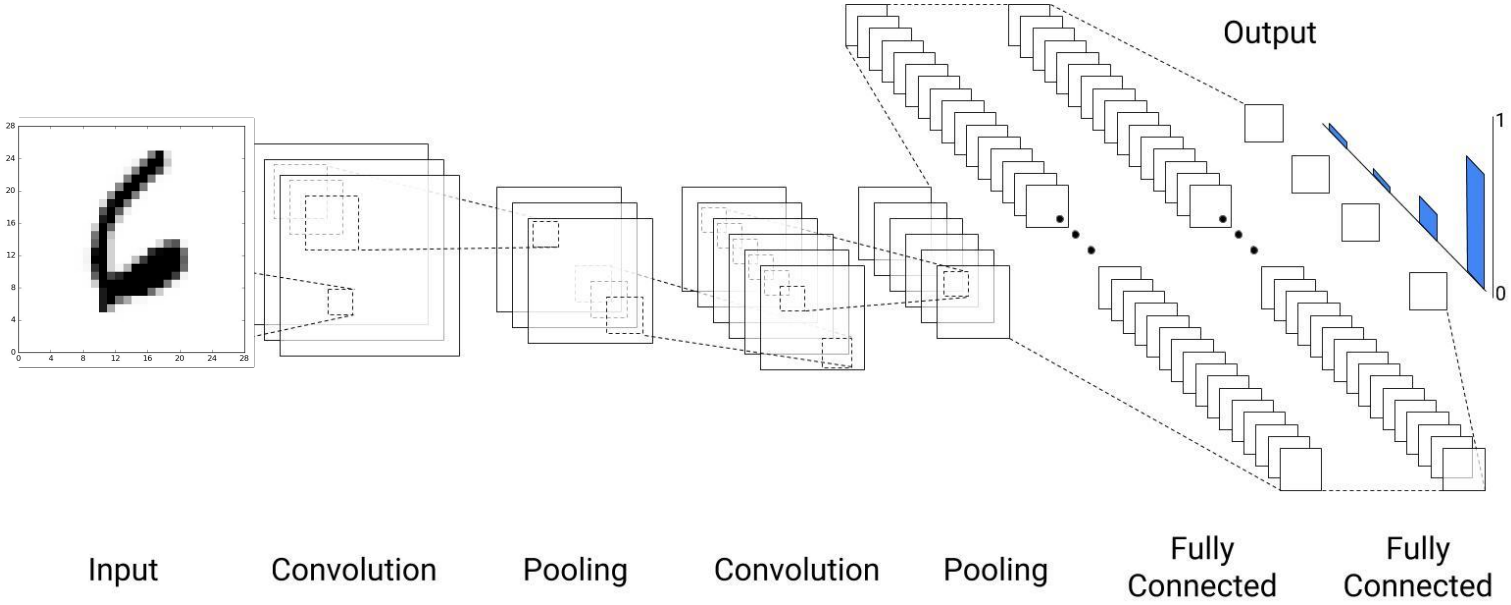
Kernel 2

1		-1
2		-2
1		-1





Convolutional neural networks for image classification



Quiz: Choose the advantage(s) of processing a small patch of the image at a time using a convolution kernel instead of using a dense layer for the entire image

A. Fewer weights
B. Faster processing
C. Weights are shared across the input
D. All of the above

Quiz: Choose the advantage(s) of processing a small patch of the image at a time using a convolution kernel instead of using a dense layer for the entire image

A. Fewer weights

B. Faster processing

C. Weights are shared across the input

D. All of the above

Agenda

Convolutional neural networks

Understanding convolutions

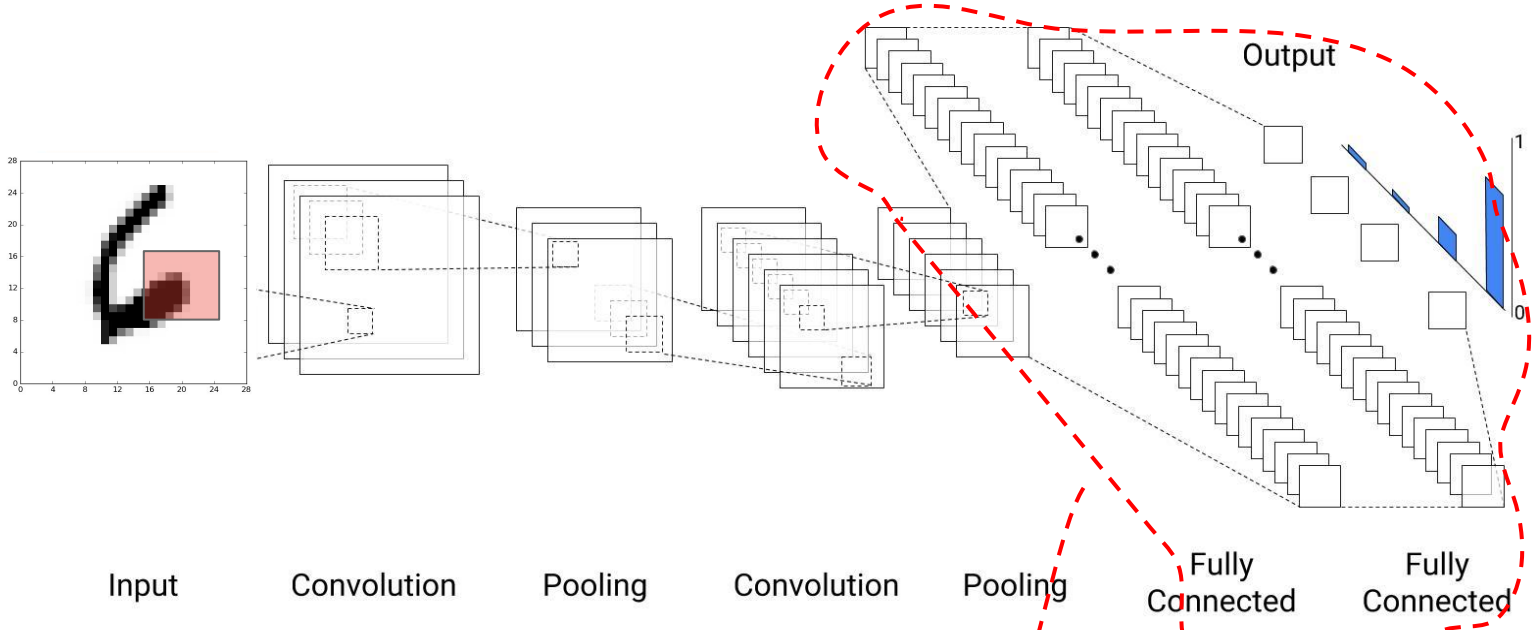
Parameters (padding, stride)

Pooling layers

Implementing CNNs

Architecture walkthrough

CNN architecture review



Fully-connected DNN for classification.

Lab

MNIST Image Classification with TensorFlow on Cloud AI Platform

- Understand how to build a Dense Neural Network (DNN) for image classification
- Understand how to use dropout (DNN) for image classification
- Understand how to use Convolutional Neural Networks (CNN)
- Know how to deploy and use an image classification model using Google Cloud's AI Platform

.../machine_learning/deepdive2/image_classification/labs/mnist_models.ipynb

Lab Steps

1. Import the training dataset of MNIST handwritten images.
2. Reshape and preprocess the image data.
3. Setup your DNN classifier model with 10 classes (one for each possible digit 0 through 9).
4. Define and run your train_and_evaluate function to train against the input dataset and evaluate your model's performance.
5. Try the classification with CNN and a DNN with Dropout

Google Cloud

Dealing with Data Scarcity

Agenda

The data scarcity problem

Data augmentation

Transfer learning

No data, no problem

Cloud AutoML

Learn how to...

Calculate the number of parameters for ML models and understand why the data needs grow with the number of parameters

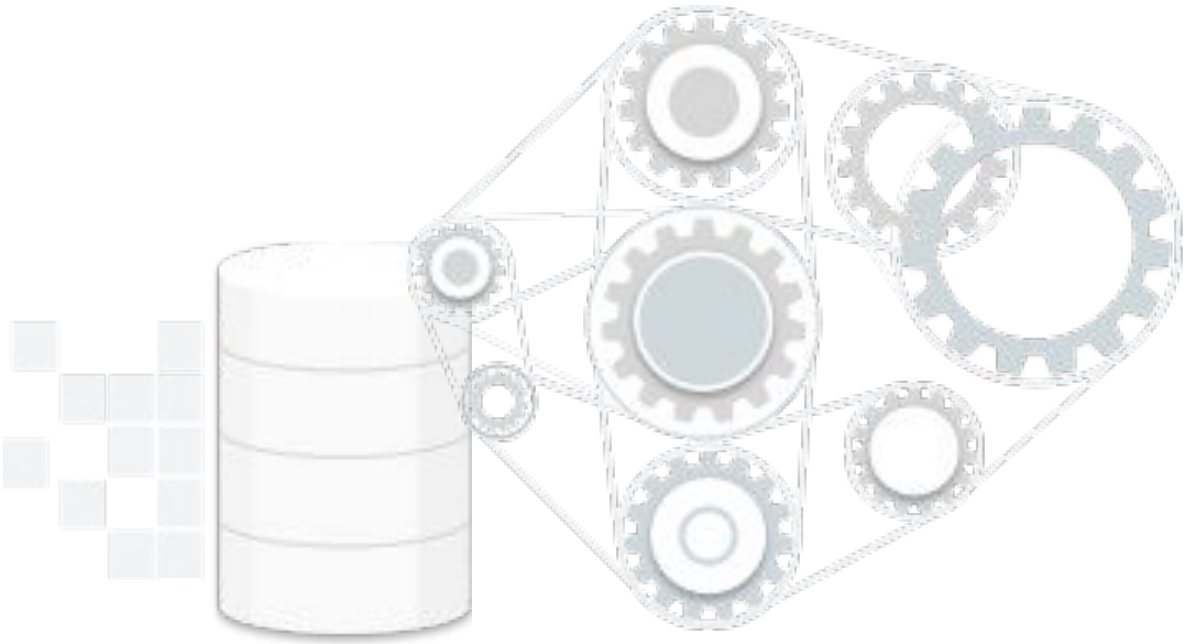
Understand how data augmentation can improve performance if done thoughtfully

Understand how to add data augmentation to a model

Apply understanding of CNNs to the concept of task dependence

Understand how transfer learning decreases the need for data

Data need grows with model complexity



A simple linear model

```
def linear_model():
    model = Sequential([
        Flatten(),
        Dense(NCLASSES),
        Softmax()
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

  return model
```

Quiz: How many parameters are in our linear model?

1. HEIGHT * WIDTH * batch_size
2. HEIGHT * WIDTH * NCLASSES + NCLASSES
3. HEIGHT * WIDTH * NCLASSES
4. HEIGHT * WIDTH + NCLASSES

Quiz: How many parameters are in our linear model?

1. HEIGHT * WIDTH * batch_size
2. HEIGHT * WIDTH * NCLASSES + NCLASSES
3. HEIGHT * WIDTH * NCLASSES
4. HEIGHT * WIDTH + NCLASSES

A simple DNN

```
units_1 = 400
units_2 = 100

dnn_model = Sequential([
    Flatten(),
    Dense(units_1, activation='relu'),
    Dense(units_2, activation='relu'),
    Dense(NCLASSES),
    Softmax()
])
```

Quiz: How many parameters are in the h1 layer?

```
X = tf.reshape(img, [-1, HEIGHT*WIDTH])
h1 = tf.layers.dense(X, 300)
```

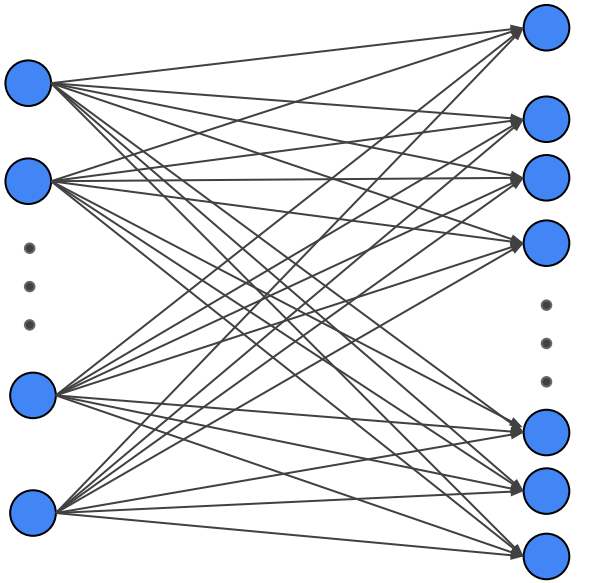
1. HEIGHT * WIDTH * 300
2. HEIGHT * WIDTH * 100 + 100
3. HEIGHT * WIDTH * 300 * NCLASSES + 300
4. HEIGHT * WIDTH * 300 + 300

Quiz: How many parameters are in the h1 layer?

```
X = tf.reshape(img, [-1, HEIGHT*WIDTH])
h1 = tf.layers.dense(X, 300)
```

1. HEIGHT * WIDTH * 300
2. HEIGHT * WIDTH * 100 + 100
3. HEIGHT * WIDTH * 300 * NCLASSES + 300
4. HEIGHT * WIDTH * 300 + 300

Each neuron also has its own bias term



Height * Width 300

Model Type	Number of Parameters
Toy linear model	7,850
Toy DNN	270,000

A simple CNN model

```
dnn_model = Sequential([
    Conv2D(num_filters_1, kernel_size=8,
           activation='relu',
input_shape=(WIDTH, HEIGHT, 1)),
    MaxPooling2D(2),
    Conv2D(num_filters_2, kernel_size=4,
           activation='relu'),
    MaxPooling2D(2),
    Flatten(),
    Dense(units_1, activation='relu'),
    Dense(units_2, activation='relu'),
    Dropout(dropout_rate),
    Dense(nclasses),
    Softmax()
])
```

Quiz: How many parameters are in the c1 layer?

```
Conv2D(num_filters_1, kernel_size=8,
       activation='relu',
       input_shape=(WIDTH, HEIGHT, 1))
```

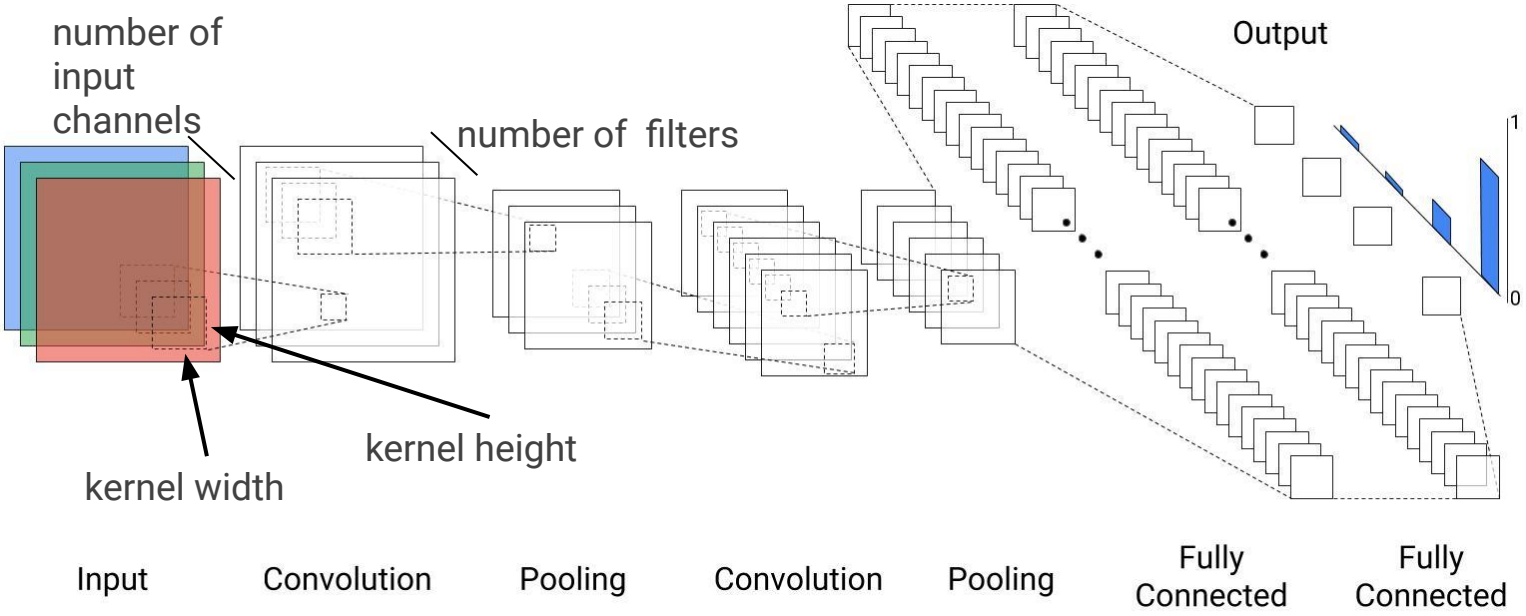
1. HEIGHT * WIDTH * 10 * 3 + 10
2. HEIGHT * WIDTH * 10 * 9 + 90
3. 20 * 9 + 20
4. 10 * 9 + 10

Quiz: How many parameters are in the c1 layer?

```
Conv2D(num_filters_1, kernel_size=8,
       activation='relu',
       input_shape=(WIDTH, HEIGHT, 1))
```

1. HEIGHT * WIDTH * 10 * 3 + 10
2. HEIGHT * WIDTH * 10 * 9 + 90
3. 20 * 9 + 20
4. 10 * 9 + 10

Computing the number of parameters in a CNN



Parameters by model type for MNIST

Model Type	Number of Parameters
Toy linear model	7,850
Toy DNN	270,000
Toy CNN	300,000

Real-world models have even more parameters

Model	Year	Number of Parameters
Alexnet	2012	60 million
VGGNet	2014	138 million
GoogLeNet	2014	23 million
ResNet	2015	25 million

What to do when you don’t have enough labeled data, but you do have some data

1. Data augmentation
2. Transfer learning

Agenda

The data scarcity problem

Data augmentation

Transfer learning

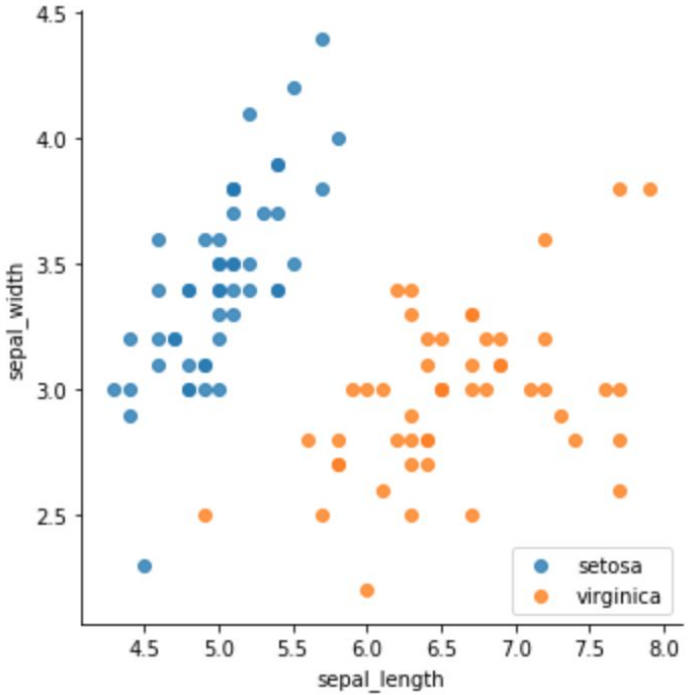
No data, no problem

Cloud AutoML

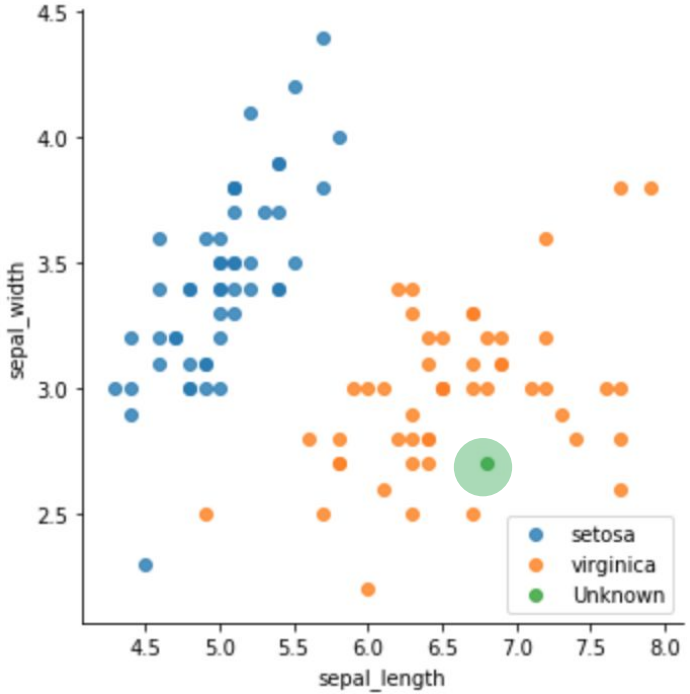
The iris dataset is structured

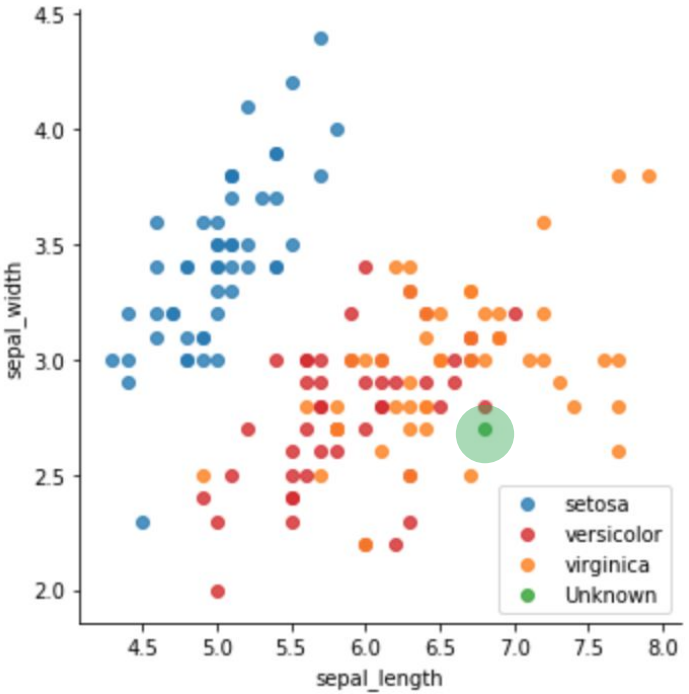
sepal_length	sepal_width	species
5.1	3.5	setosa
4.9	3.0	setosa
4.7	3.2	setosa
4.6	3.1	setosa
5.0	3.6	setosa

A typical labeled, structured dataset

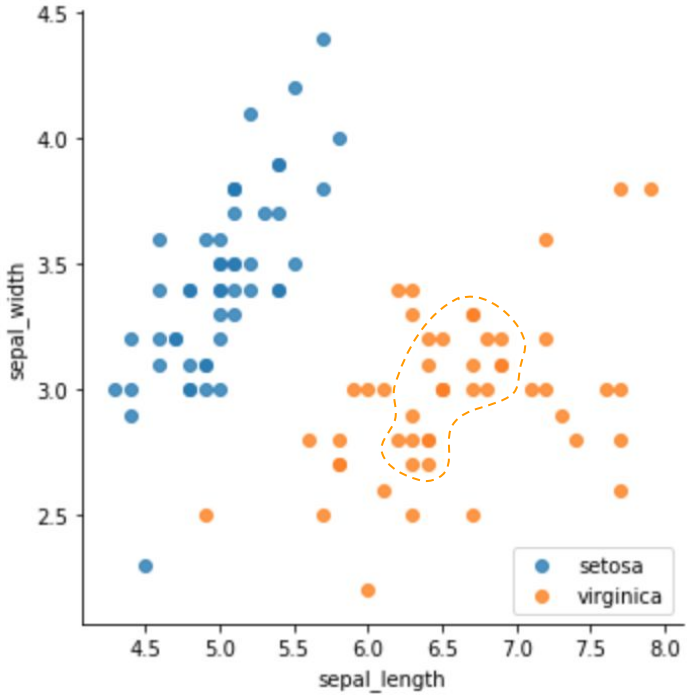


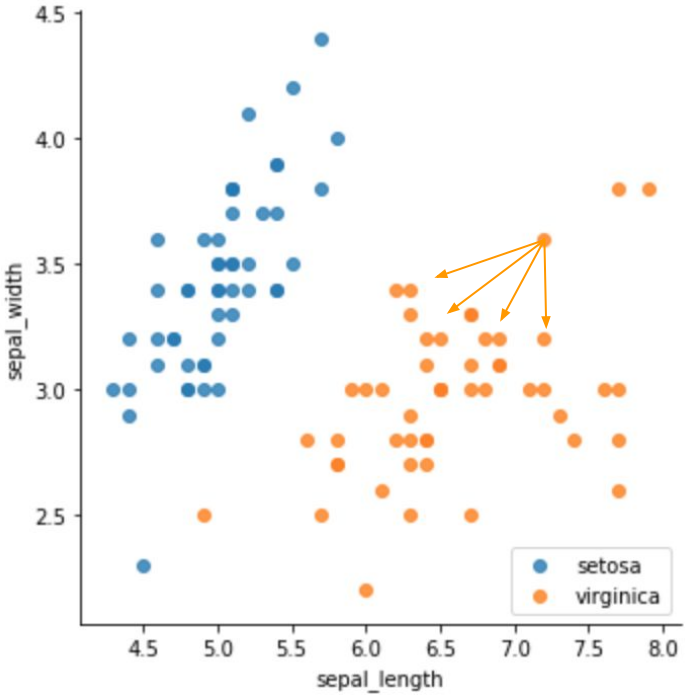
What about these new points?





Strategies to make the task easier

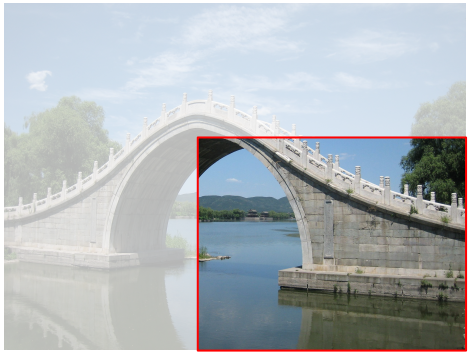




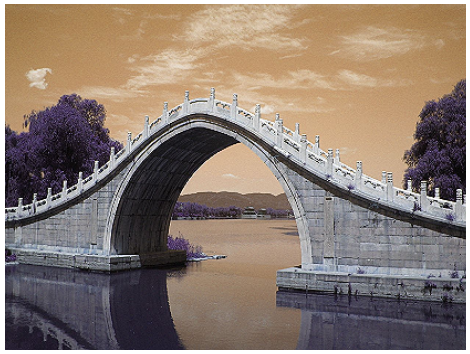
Common image augmentation techniques



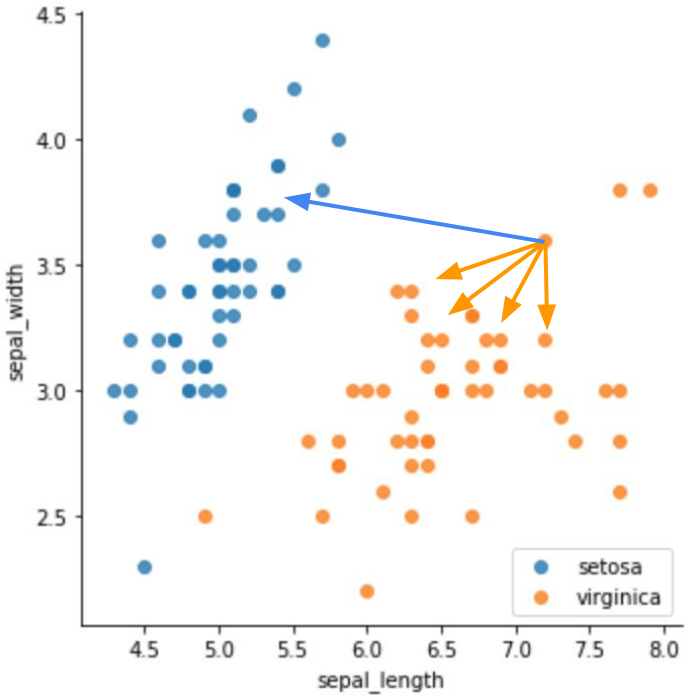





Common image augmentation techniques



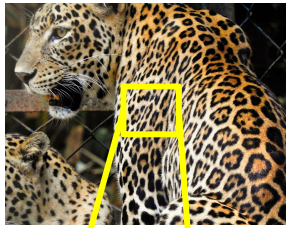
Data augmentation requires care

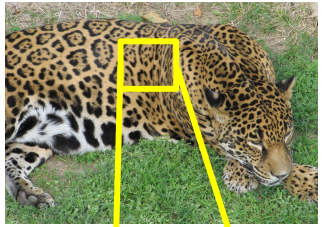


Information is often in the detail

Sometimes color is informative

Sometimes orientation is informative





Sometimes small detail is informative

Don’t make the problem harder than it needs to be

Quiz: What would happen if you augmented data at decision time? Would this be a good idea?

Let’s say that our task is to accept a professionally photographed flower and classify it by species. Would randomly changing the brightness and contrast during data augmentation likely improve performance?

1. Yes
2. No

Quiz: What would happen if you augmented data at decision time? Would this be a good idea?

Let's say that our task is to accept a professionally photographed flower and classify it by species. Would randomly changing the brightness and contrast during data augmentation likely improve performance?

1. Yes
2. No

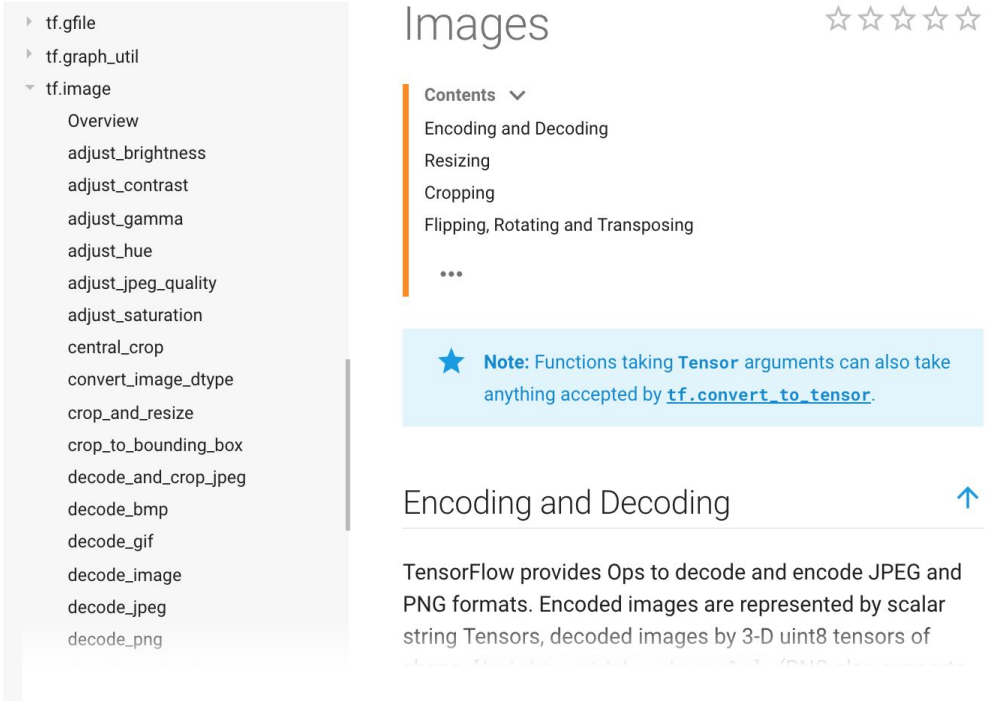
Implementing image augmentation

```
def load_dataset(csv_of_filenames, batch_size, training=True):
    dataset = tf.data.TextLineDataset(filenames=csv_of_filenames) \
        .map(decode_csv).cache()

    if training:
        dataset = dataset \
            .map(read_and_preprocess_with_augment) \
            .shuffle(SHUFFLE_BUFFER) \
            .repeat(count=None)  # Indefinitely.
    else:
        dataset = dataset \
            .map(read_and_preprocess) \
            .repeat(count=1)  # Each photo used once.

    return dataset.batch(batch_size=batch_size) \
        .prefetch(buffer_size=AUTOTUNE)
```

TensorFlow has a robust set of image functions



Implementing image augmentation

```
def load_dataset(csv_of_filenames, batch_size, training=True):
    dataset = tf.data.TextLineDataset(filenames=csv_of_filenames) \
        .map(decode_csv).cache()

    if training:
        dataset = dataset \
            .map(read_and_preprocess_with_augment) \
            .shuffle(SHUFFLE_BUFFER) \
            .repeat(count=None)  # Indefinitely.
    else:
        dataset = dataset \
            .map(read_and_preprocess) \
            .repeat(count=1)  # Each photo used once.

    return dataset.batch(batch_size=batch_size) \
        .prefetch(buffer_size=AUTOTUNE)
```

Implementing image augmentation

JPEG URL	Label
gs://path/to/image_1.jpg	daisy
gs://path/to/image_2.jpg	dandelion
gs://path/to/image_3.jpg	dandelion

Implementing decode_csv()

```
def load_dataset(csv_of_filenames, batch_size, training=True):
    dataset = tf.data.TextLineDataset(filenames=csv_of_filenames) \
        .map(decode_csv).cache()

    if training:
        dataset = dataset \
            .map(read_and_preprocess_with_augment) \
            .shuffle(SHUFFLE_BUFFER) \
            .repeat(count=None)  # Indefinitely.
    else:
        dataset = dataset \
            .map(read_and_preprocess) \
            .repeat(count=1)  # Each photo used once.

    return dataset.batch(batch_size=batch_size) \
        .prefetch(buffer_size=AUTOTUNE)
```

Implementing decode_csv()

```
def decode_csv(csv_row):
    record_defaults = ["path", "flower"]
    filename, label_string = tf.io.decode_csv(
        csv_row, record_defaults)
    image_bytes = tf.io.read_file(filename=filename)
    label = tf.math.equal(CLASS_NAMES, label_string)
    return image_bytes, label
```

Implementing `decode_img()`

```
def decode_img(img, reshape_dims):
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)
    return tf.image.resize(img, reshape_dims)
```

Implementing read_and_preprocess()

```
def load_dataset(csv_of_filenames, batch_size, training=True):
    dataset = tf.data.TextLineDataset(filenames=csv_of_filenames) \
        .map(decode_csv).cache()

    if training:
        dataset = dataset \
            .map(read_and_preprocess_with_augment) \
            .shuffle(SHUFFLE_BUFFER) \
            .repeat(count=None)  # Indefinitely.
    else:
        dataset = dataset \
            .map(read_and_preprocess) \
            .repeat(count=1)  # Each photo used once.

    return dataset.batch(batch_size=batch_size) \
        .prefetch(buffer_size=AUTOTUNE)
```

Implementing `read_and_preprocess()`

```
def read_and_preprocess(image_bytes, label, random_augment=False):
    if random_augment:
        img = decode_img(image_bytes, [IMG_HEIGHT + 10, IMG_WIDTH + 10])
        img = tf.image.random_crop(
            img, [IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS])
        img = tf.image.random_flip_left_right(img)
        img = tf.image.random_brightness(img, MAX_DELTA)
        img = tf.image.random_contrast(img, CONTRAST_LOWER, CONTRAST_UPPER)
    else:
        img = decode_img(image_bytes, [IMG_WIDTH, IMG_HEIGHT])

    img = tf.subtract(x=img, y=0.5)
    img = tf.multiply(x=img, y=2.0)
    return img, label
```

`random` functions apply different effects when called.

Implementing read_and_preprocess_with_augment()

```
def read_and_preprocess_with_augment(image_bytes, label):
    return read_and_preprocess(
        image_bytes, label, random_augment=True)
```

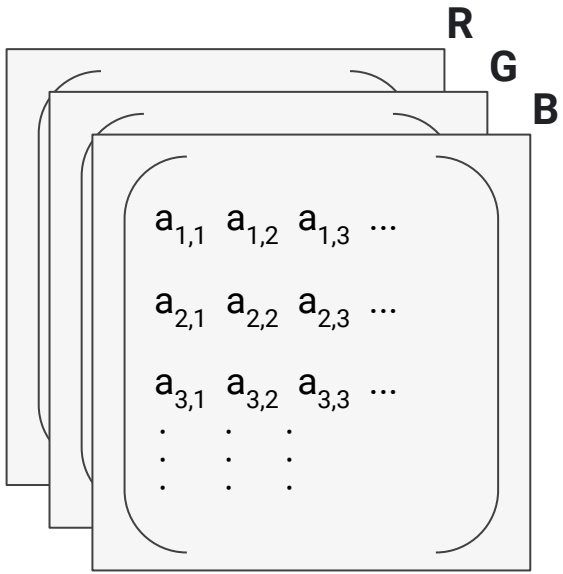
Working with color images

← 299px Width →

←299px Height →

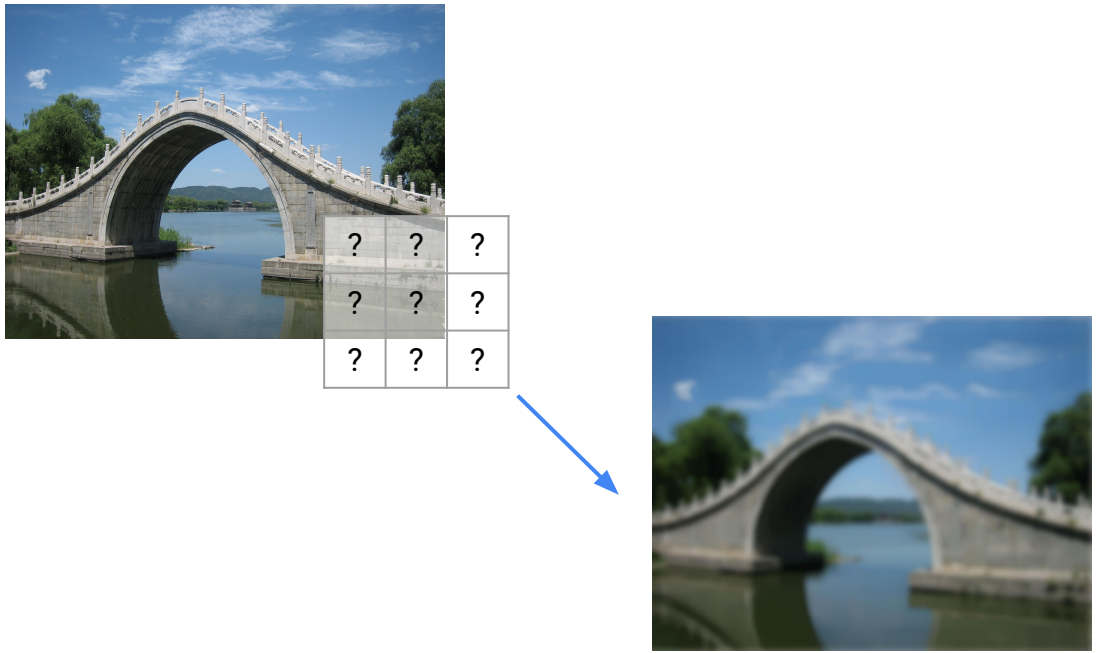
3 “Layers” in depth for Red Blue Green

Color images are still just tensors



JPEG URL	Label
gs://path/to/image_1.jpg	daisy
gs://path/to/image_2.jpg	dandelion
gs://path/to/image_3.jpg	dandelion

Extra Credit: Implement blur using convolution



Agenda

The data scarcity problem

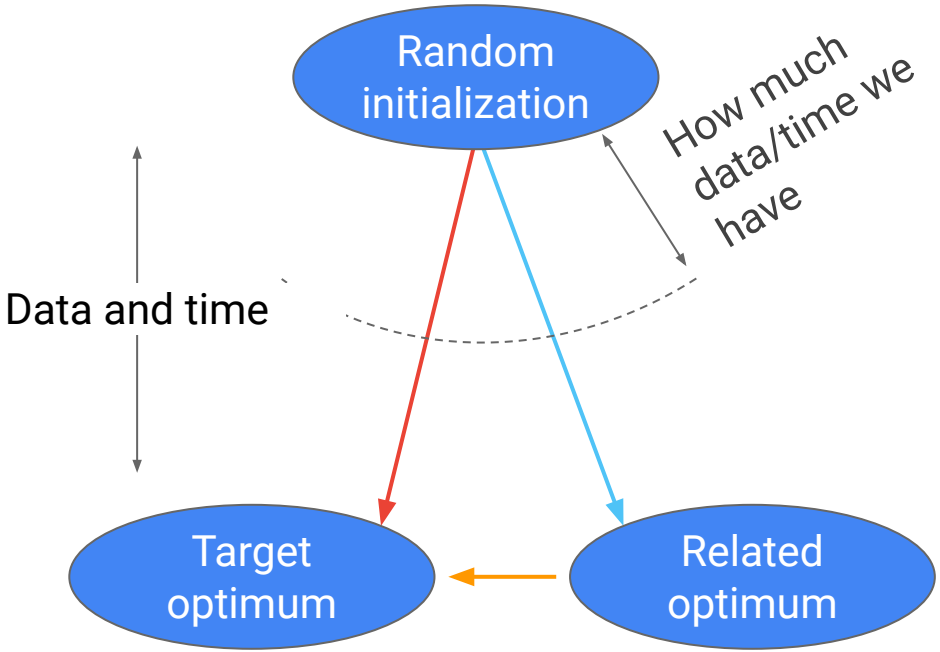
Data augmentation

Transfer learning

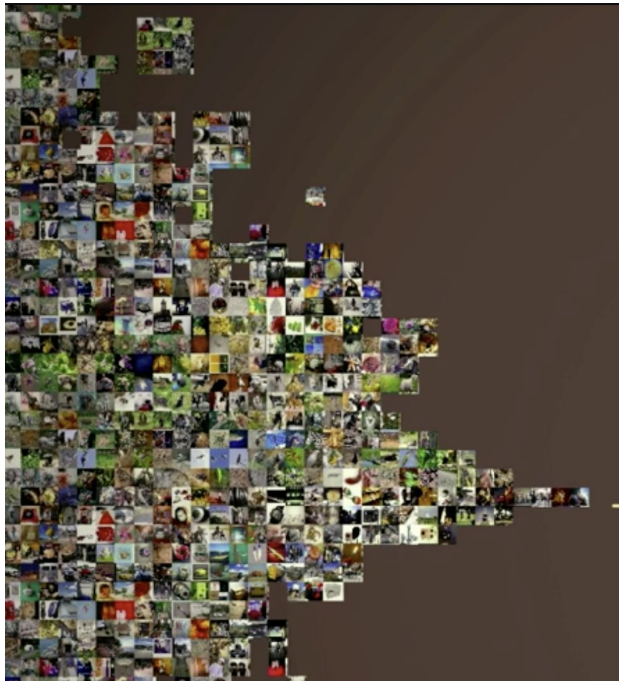
No data, no problem

Cloud AutoML

Transfer learning is like a shortcut



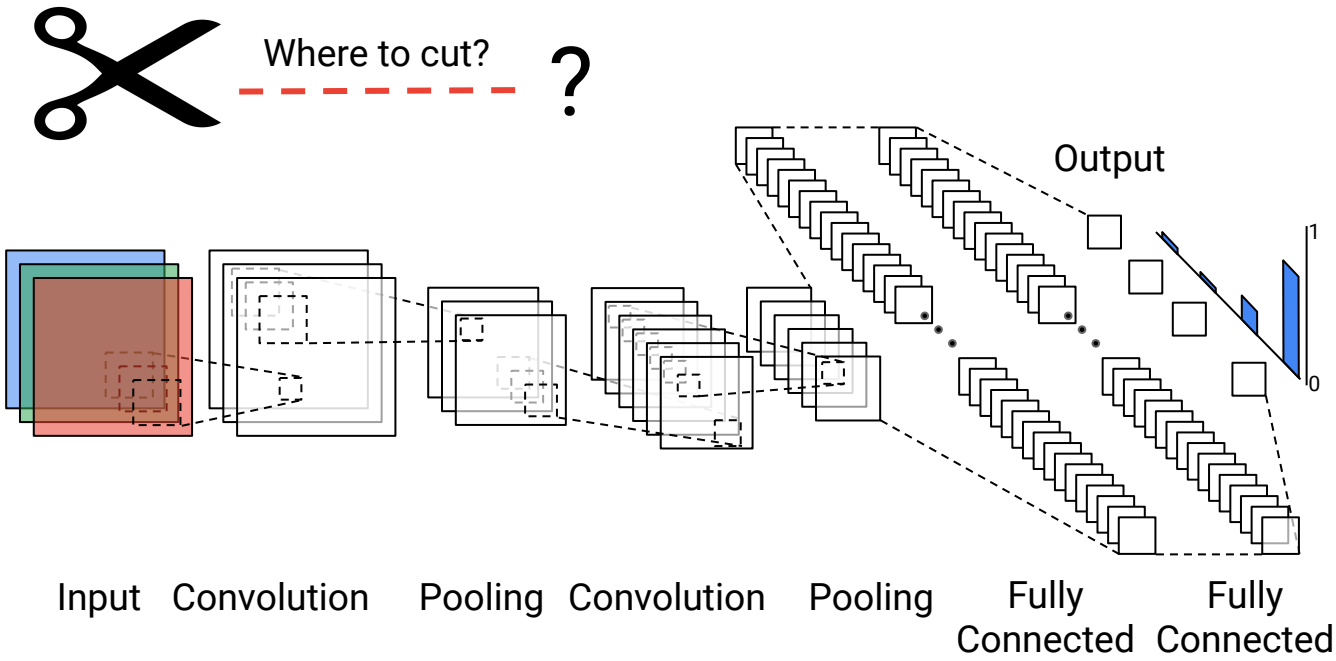
Predicting the categories of images in ImageNet remains an important benchmark in computer vision



Naive transfer learning would simply take this model and then use it to predict on our task

ImageNet classes		Your classes
Domestic dog	←	Domestic dog
Domestic cat	?	Cirrus
Field corn	←	Field corn
Sand verbena	?	Cumulonimbus
...		...

Take a subtler approach to transferring the source model



Quiz: Which parts of the source image model are most closely aligned to the source task?

A. The input to the model
B. The convolutional layers
C. The layers near the output

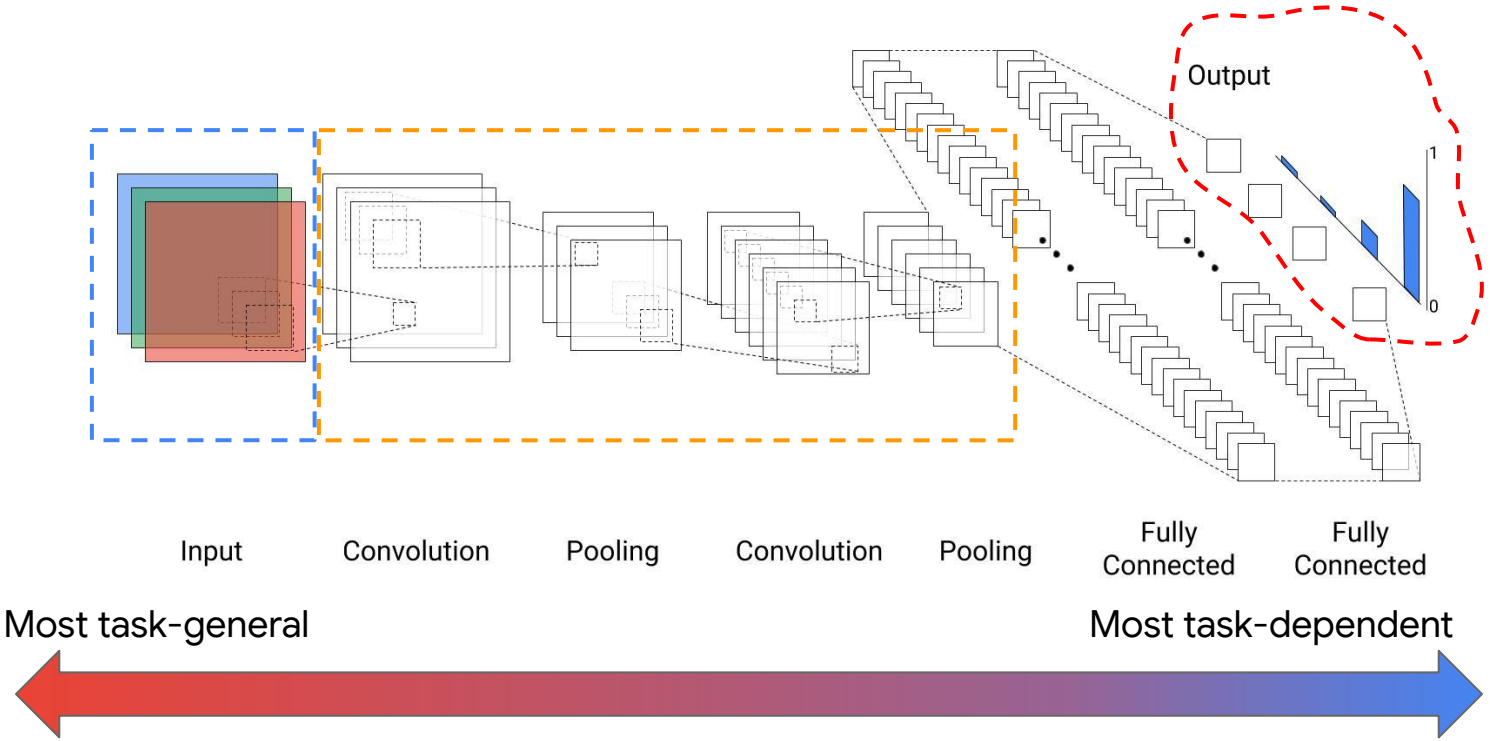
Quiz: Which parts of the source image model are most closely aligned to the source task?

A. The input to the model

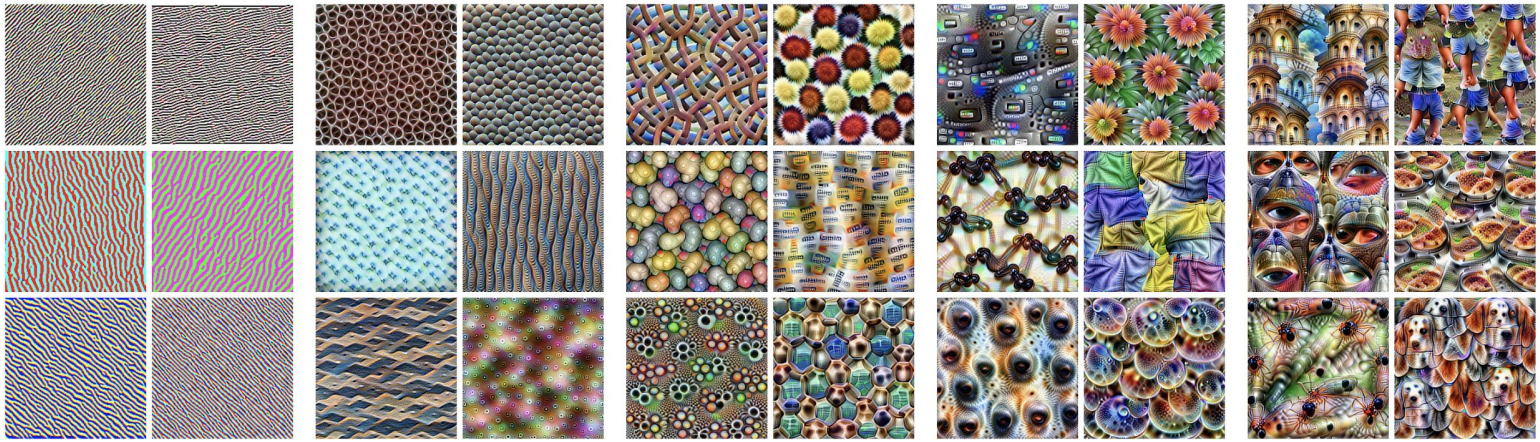
B. The convolutional layers

C. The layers near the output

CNNs and the spectrum of task-dependence



CNNs learn a hierarchy of features

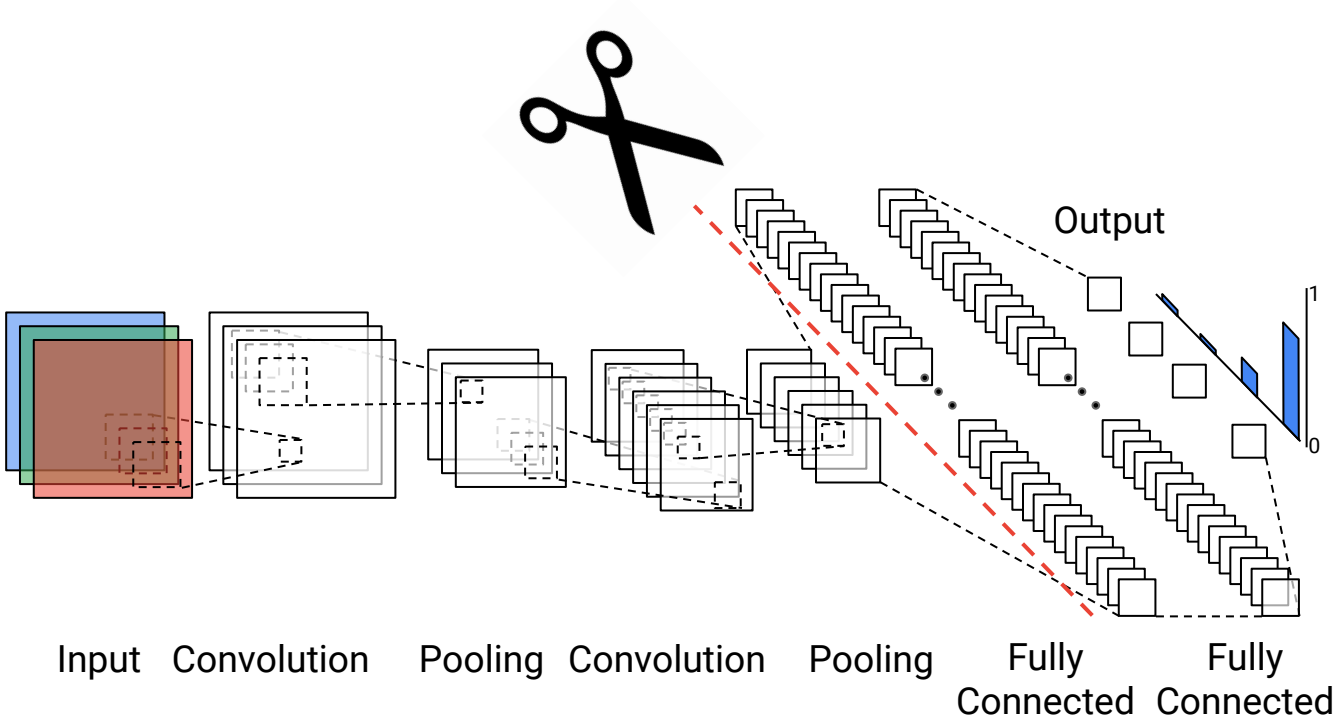


Edges (layer conv2d0) **Textures** (layer mixed3a) **Patterns** (layer mixed4a) **Parts** (layers mixed4b & mixed4c) **Objects** (layers mixed4d & mixed4e)

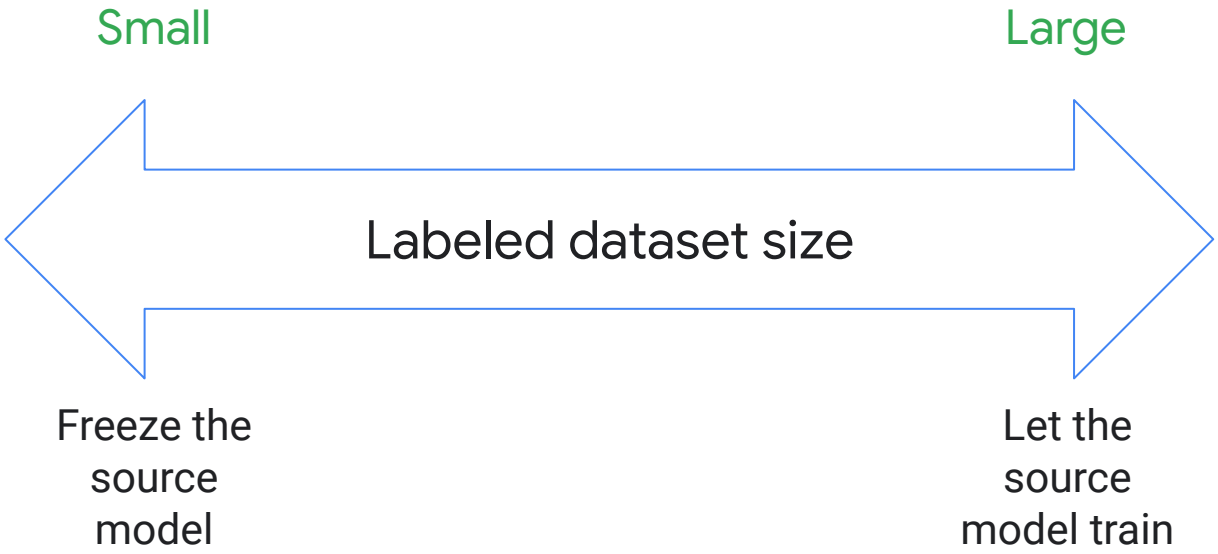
Flow of data in the network

https://distill.pub/2017/feature-visualization/

The source network is cut after the convolutional layers



To freeze or not to freeze?



Lab

TensorFlow Transfer Learning

- Know how to apply image augmentation
- Know how to download and use a TensorFlow Hub module as a layer in Keras.

.../machine_learning/deepdive2/image_classification/labs/3_tf_hub_transfer_learning.ipynb

Agenda

Introduction

Batch normalization

Residual networks

Learn how to...

Train deeper, more accurate neural networks faster

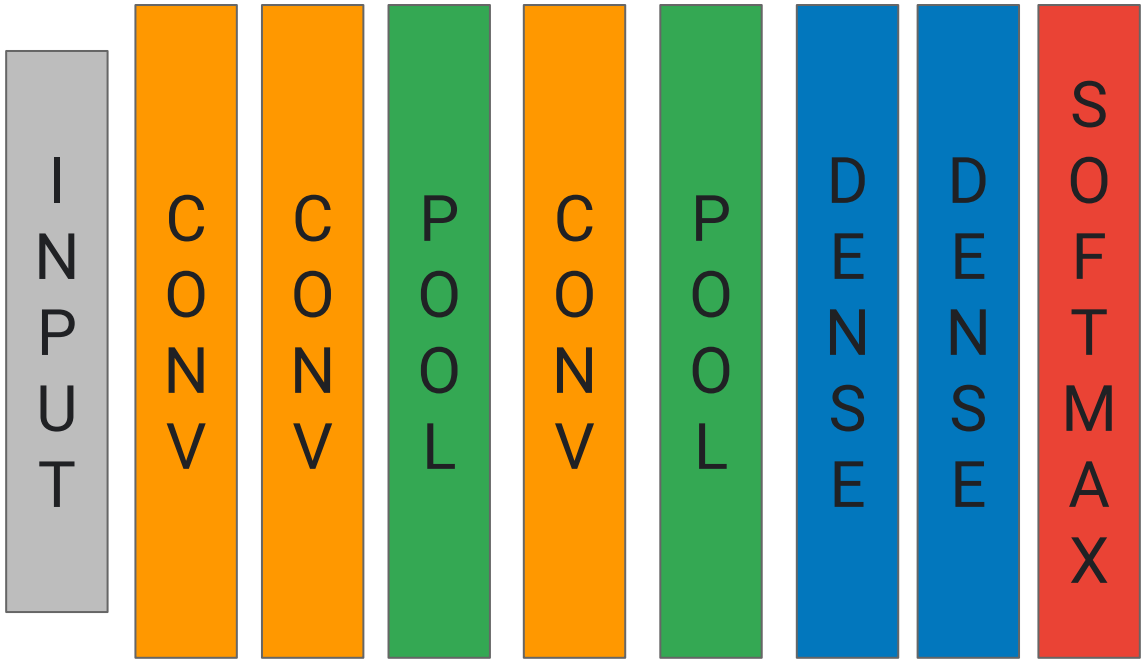
Address internal covariate shift using batch normalization

Add shortcut connections to increase network depth

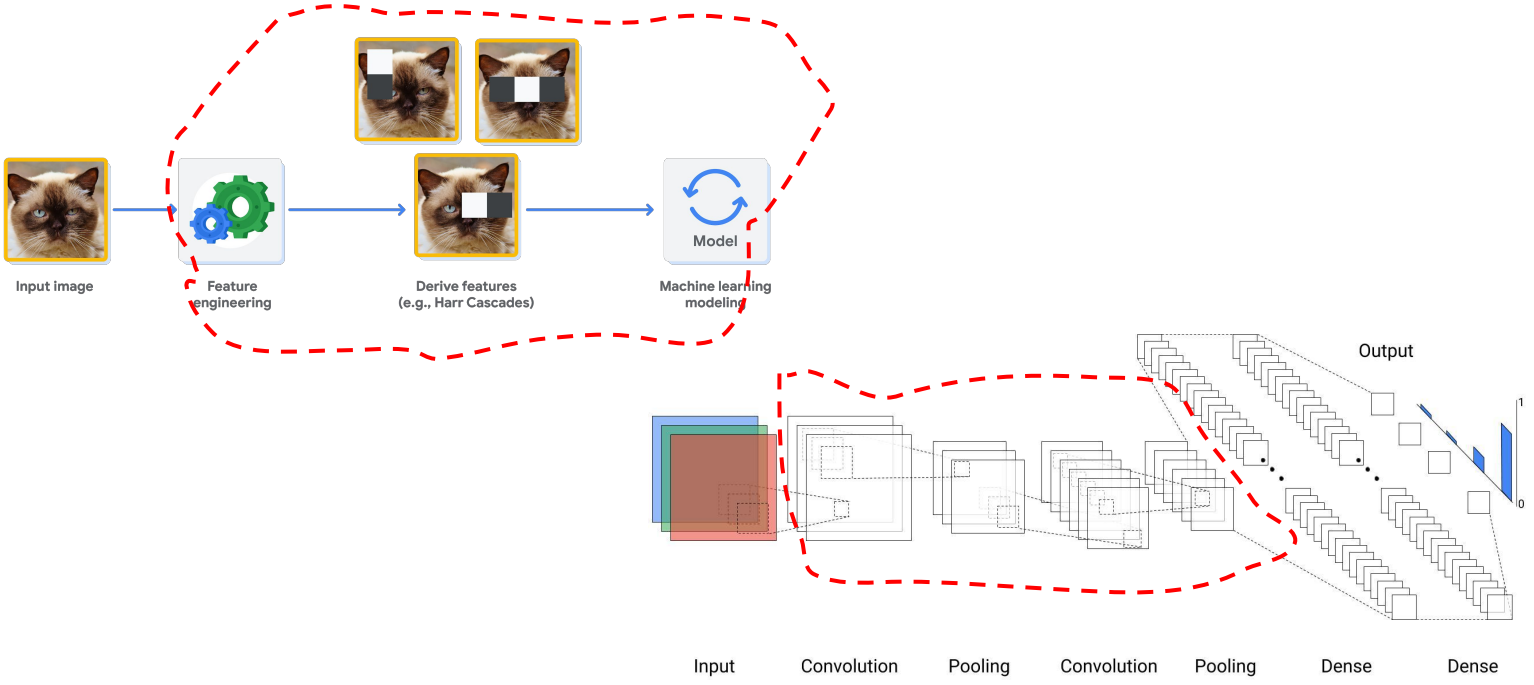
Take advantage of Tensor Processing Units

Automate network design

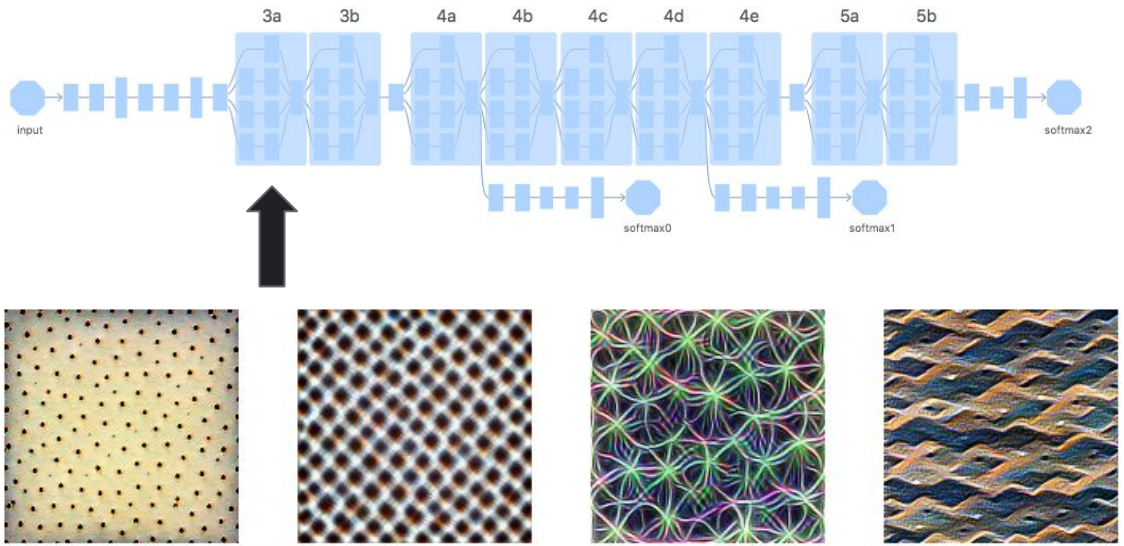
AlexNet: A neural network with 8 layers



AlexNet was fundamentally different

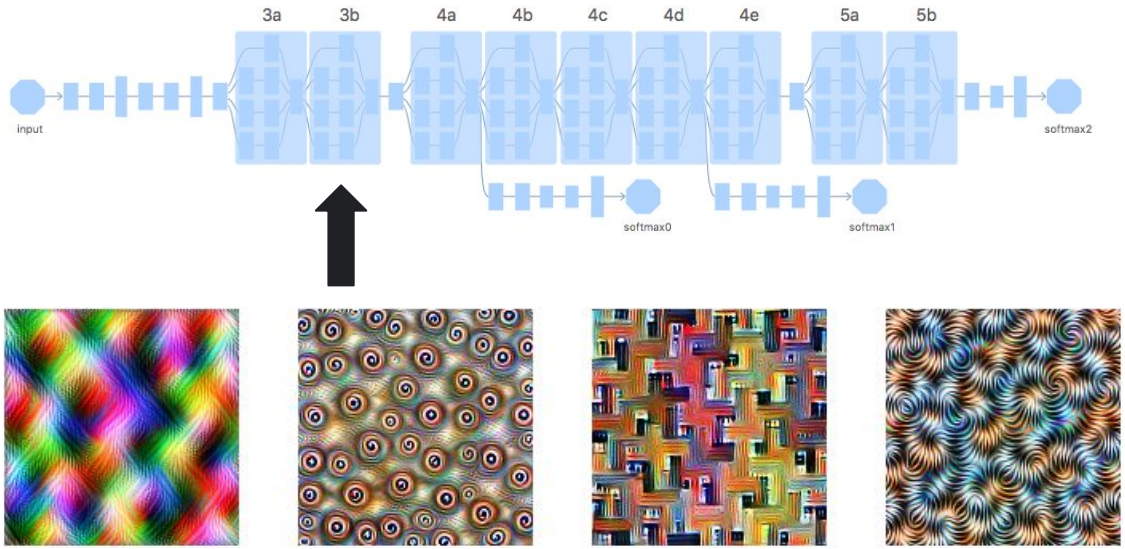


Visualizing layer activations



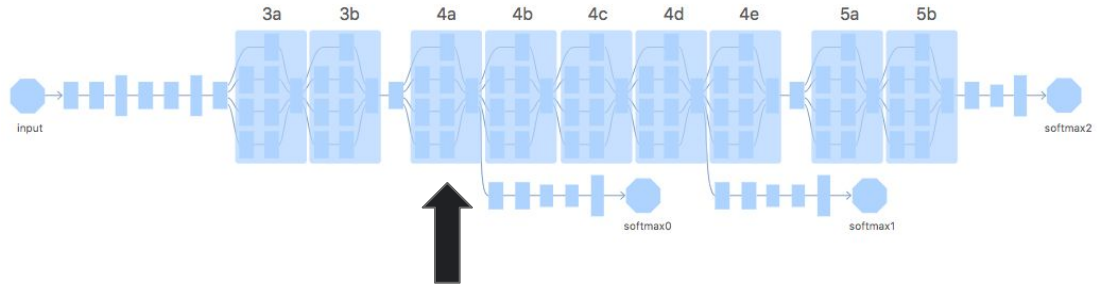
Diagrams by Chris Olah
https://distill.pub/2017/feature-visualization/appendix/

Visualizing layer activations

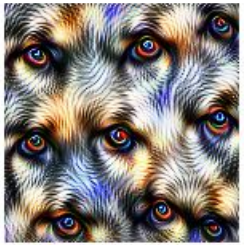


Diagrams by Chris Olah
https://distill.pub/2017/feature-visualization/appendix/

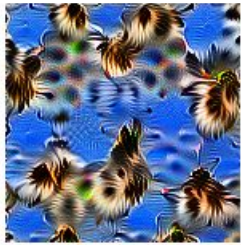
Visualizing layer activations




Bookshelves

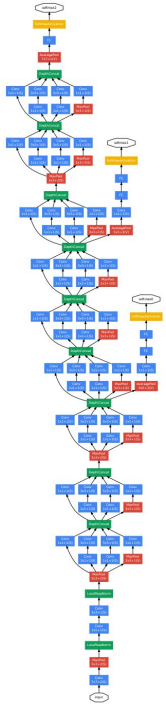

Dog eyes


Text, rivets


Birds

Diagrams by Chris Olah
https://distill.pub/2017/feature-visualization/appendix/

GoogLeNet showed that deeper models yielded higher accuracy



More is better, right?

2012 AlexNet: 8

2013 ZFNet: 8

2014 VGGNet: 19
2014 GoogLeNet: 22

...

150?

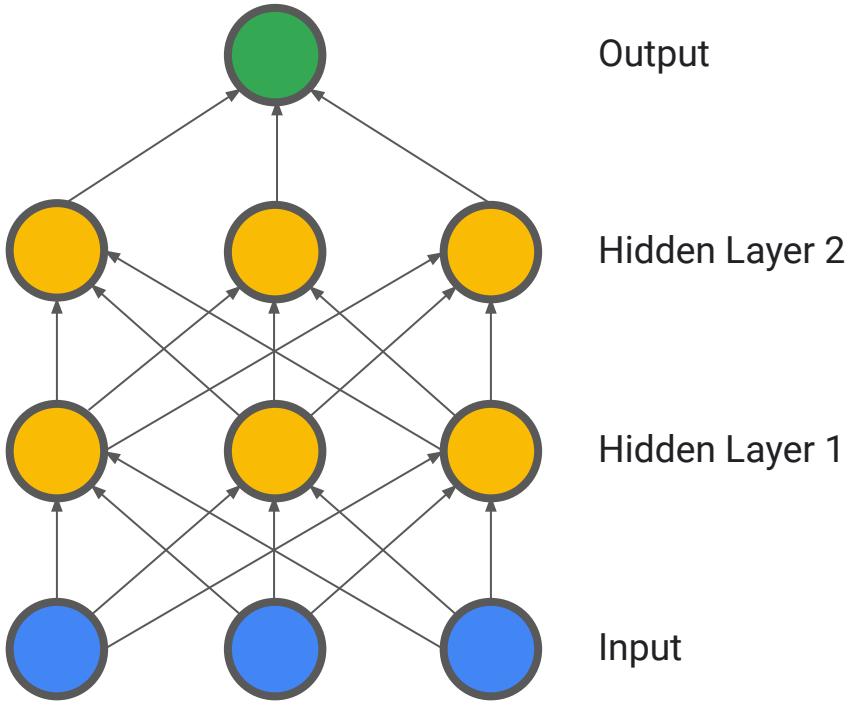
Agenda

Introduction

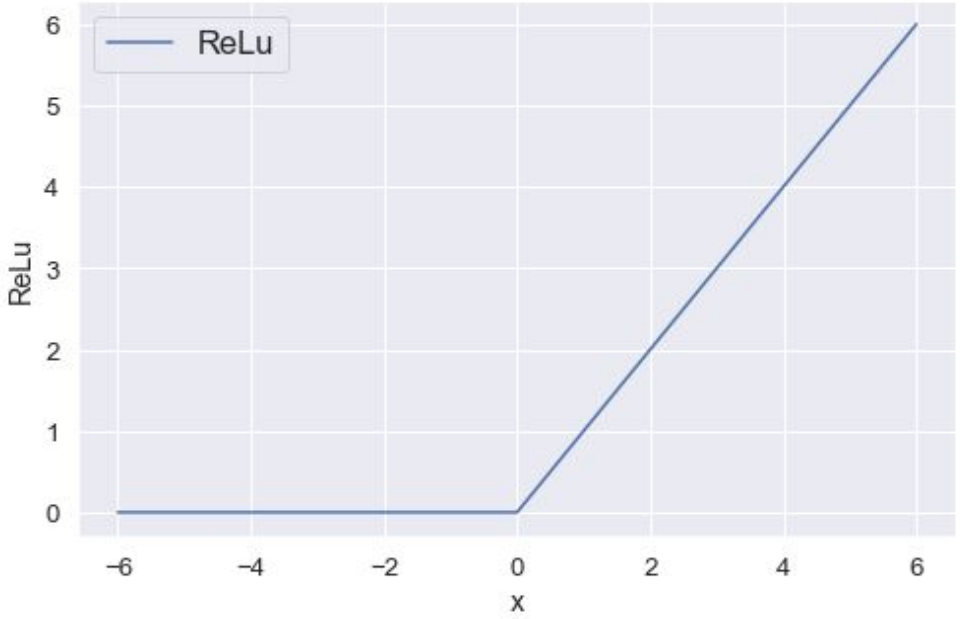
Batch normalization

Residual networks

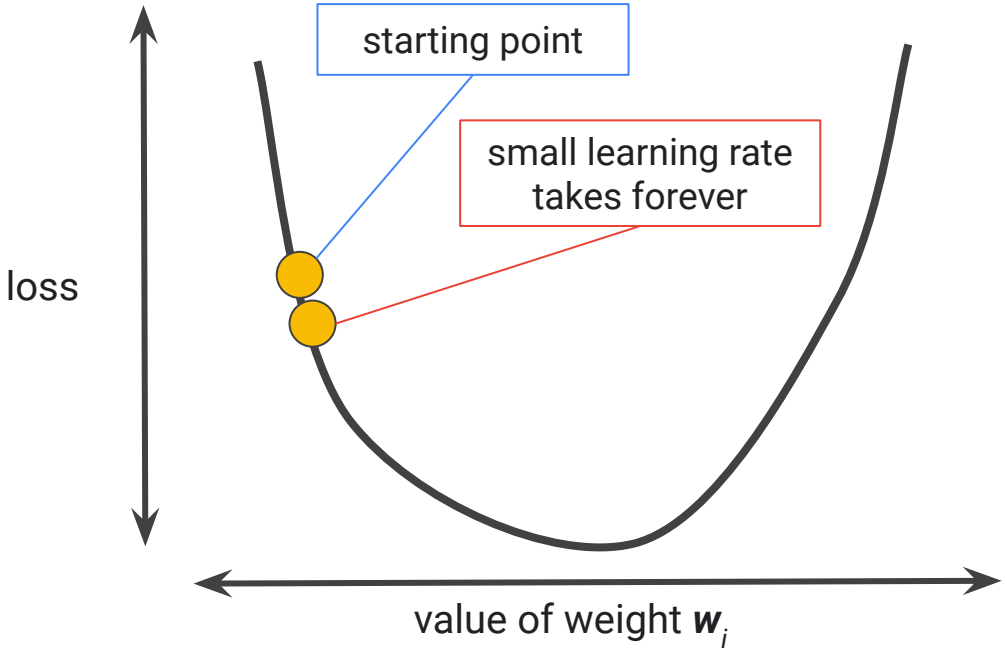
Internal covariate shift



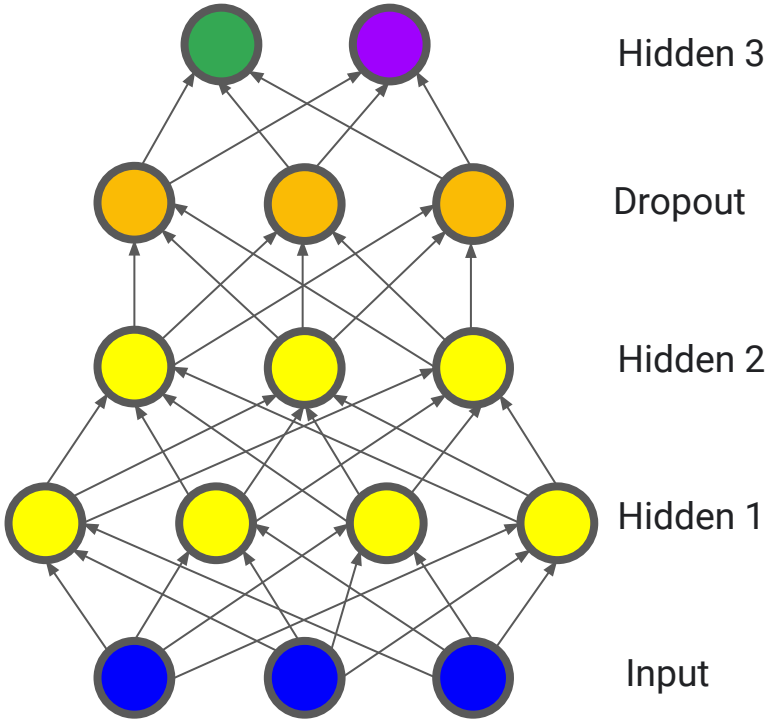
Neurons may stop learning



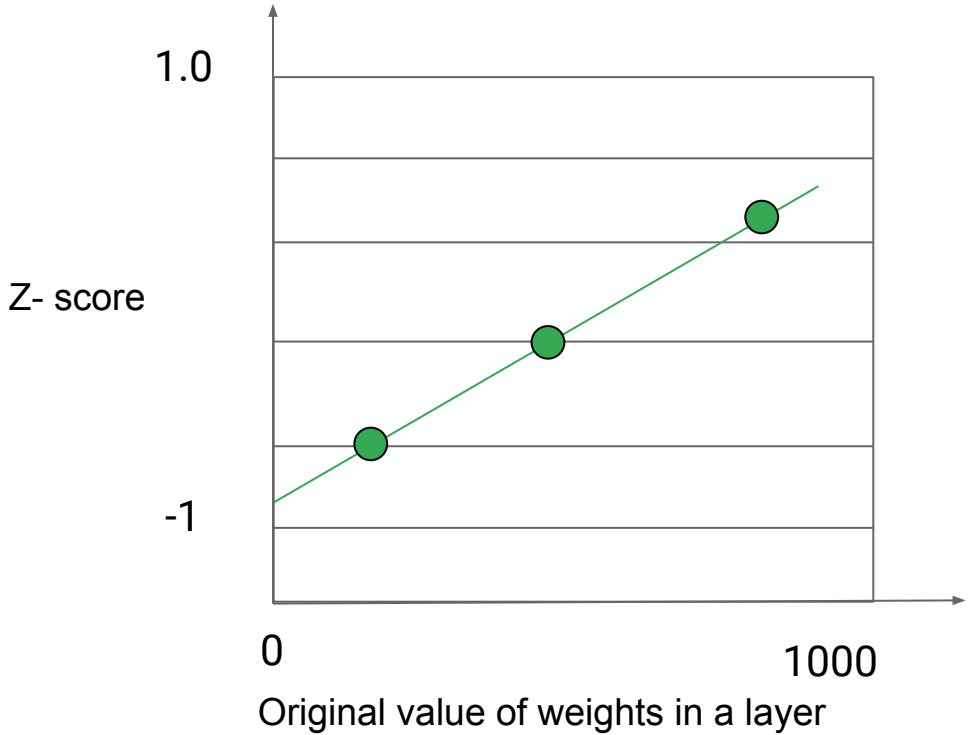
Lower the learning rate



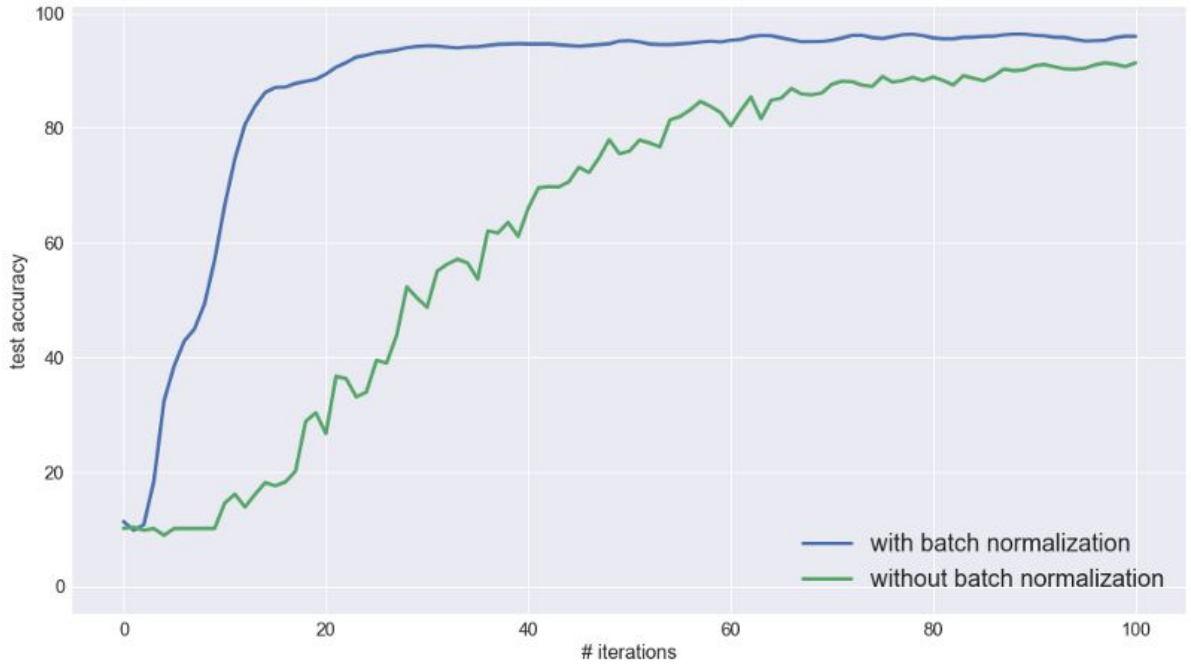
Use dropout



Batch normalization scales the weights between layers



Batch normalization helps you train faster



Batch normalization in a custom Estimator with a Keras model function

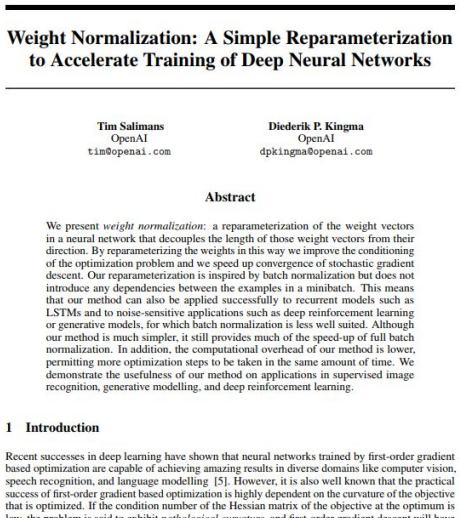
```
layer2 = tf.keras.layers.Dense(...)
bn = tf.keras.layers.BatchNormalization(momentum=0.99)
layer3 = bn(layer2, training=training)
```

Batch normalization in a custom Estimator

```
layer3 = tf.layers.batch_normalization(
          layer2, training=(mode == TRAIN))
```

```
update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(update_ops):
    train_op = optimizer.minimize(loss)
```

Subsequent work with normalization



Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks

Tim Salimans
OpenAI
tim@openai.com

Diederik P. Kingma
OpenAI
dpkingma@openai.com

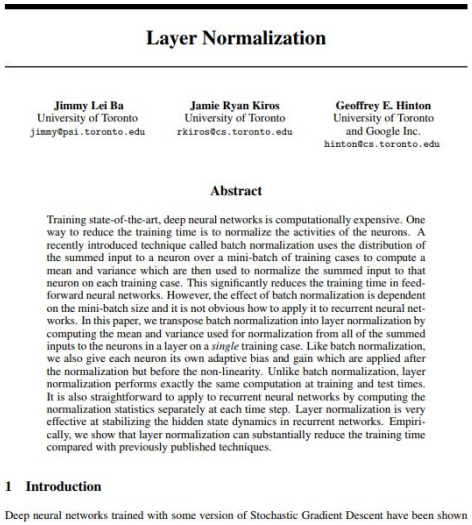
Abstract

We present *weight normalization*: a reparameterization of the weight vectors in a neural network that decouples the length of those weight vectors from their direction. By reparameterizing the weights in this way we improve the conditioning of the optimization problem and we speed up convergence of stochastic gradient descent. Our reparameterization is inspired by batch normalization but does not introduce any dependencies between the examples in a minibatch. This means that our method can also be applied successfully to recurrent models such as LSTMs and to noise-sensitive applications such as deep reinforcement learning or generative models, for which batch normalization is less well suited. Although our method is much simpler, it still provides much of the speed-up of full batch normalization. In addition, the computational overhead of our method is lower, permitting more optimization steps to be taken in the same amount of time. We demonstrate the usefulness of our method on applications in supervised image recognition, generative modelling, and deep reinforcement learning.

1 Introduction

Recent successes in deep learning have shown that neural networks trained by first-order gradient based optimization are capable of achieving amazing results in diverse domains like computer vision, speech recognition, and language modelling [5]. However, it is also well known that the practical success of first-order gradient based optimization is highly dependent on the curvature of the objective that is optimized. If the condition number of the Hessian matrix of the objective at the optimum is

[Weight Normalization]



Layer Normalization

Jimmy Lei Ba
University of Toronto
jimmy@psi.toronto.edu

Jamie Ryan Kiros
University of Toronto
rkiros@cs.toronto.edu

Geoffrey E. Hinton
University of Toronto
and Google Inc.
hinton@cs.toronto.edu

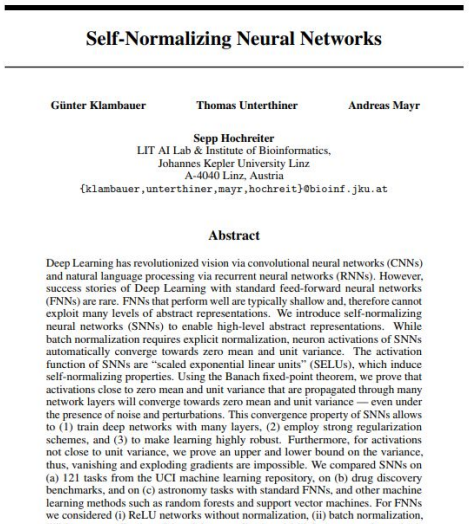
Abstract

Training state-of-the-art, deep neural networks is computationally expensive. One way to reduce the training time is to normalize the activities of the neurons. A recently introduced technique called batch normalization uses the distribution of the summed input to a neuron over a mini-batch of training cases to compute a mean and variance which are then used to normalize the summed input to that neuron on each training case. This significantly reduces the training time in feed-forward neural networks. However, the effect of batch normalization is dependent on the mini-batch size and it is not obvious how to apply it to recurrent neural networks. In this paper, we transpose batch normalization into layer normalization by computing the mean and variance used for normalization from all of the summed inputs to the neurons in a layer on a *single* training case. Like batch normalization, we also give each neuron its own adaptive bias and gain which are applied after the normalization but before the non-linearity. Unlike batch normalization, layer normalization performs exactly the same computation at training and test times. It is also straightforward to apply to recurrent neural networks by computing the normalization statistics separately at each time step. Layer normalization is very effective at stabilizing the hidden state dynamics in recurrent networks. Empirically, we show that layer normalization can substantially reduce the training time compared with previously published techniques.

1 Introduction

Deep neural networks trained with some version of Stochastic Gradient Descent have been shown

[Layer Normalization]



Self-Normalizing Neural Networks

Günter Klambauer **Thomas Unterthiner** **Andreas Mayr**

Sepp Hochreiter
LIT AI Lab & Institute of Bioinformatics,
Johannes Kepler University Linz
A-4040 Linz, Austria
{klambauer,unterthiner,mayr,hochreit}@bioinf.jku.at

Abstract

Deep Learning has revolutionized vision via convolutional neural networks (CNNs) and natural language processing via recurrent neural networks (RNNs). However, success stories of Deep Learning with standard feed-forward neural networks (FNNs) are rare. FNNs that perform well are typically shallow and, therefore cannot exploit many levels of abstract representations. We introduce self-normalizing neural networks (SNNs) to enable high-level abstract representations. While batch normalization requires explicit normalization, neuron activations of SNNs automatically converge towards zero mean and unit variance. The activation function of SNNs are "scaled exponential linear units" (SELUs), which induce self-normalizing properties. Using the Banach fixed-point theorem, we prove that activations close to zero mean and unit variance that are propagated through many network layers will converge towards zero mean and unit variance — even under the presence of noise and perturbations. This convergence property of SNNs allows to (1) train deep networks with many layers, (2) employ strong regularization schemes, and (3) to make learning highly robust. Furthermore, for activations not close to unit variance, we prove an upper and lower bound on the variance, thus, vanishing and exploding gradients are impossible. We compared SNNs on (a) 121 tasks from the UCI machine learning repository, on (b) drug discovery benchmarks, and on (c) astronomy tasks with standard FNNs, and other machine learning methods such as random forests and support vector machines. For FNNs we considered (i) ReLU networks without normalization, (ii) batch normalization,

[Self-Normalizing Networks]

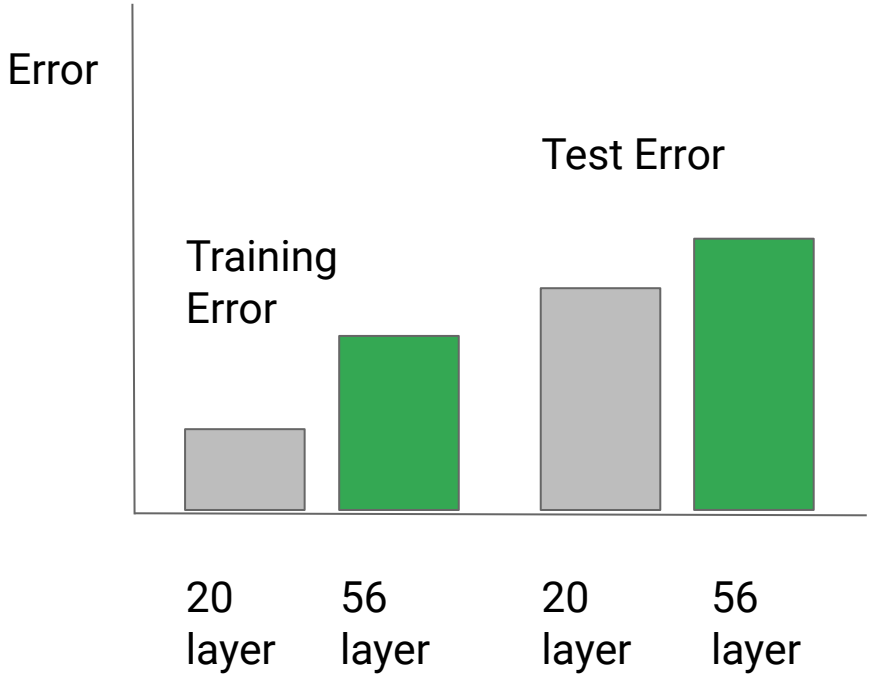
Agenda

Introduction

Batch normalization

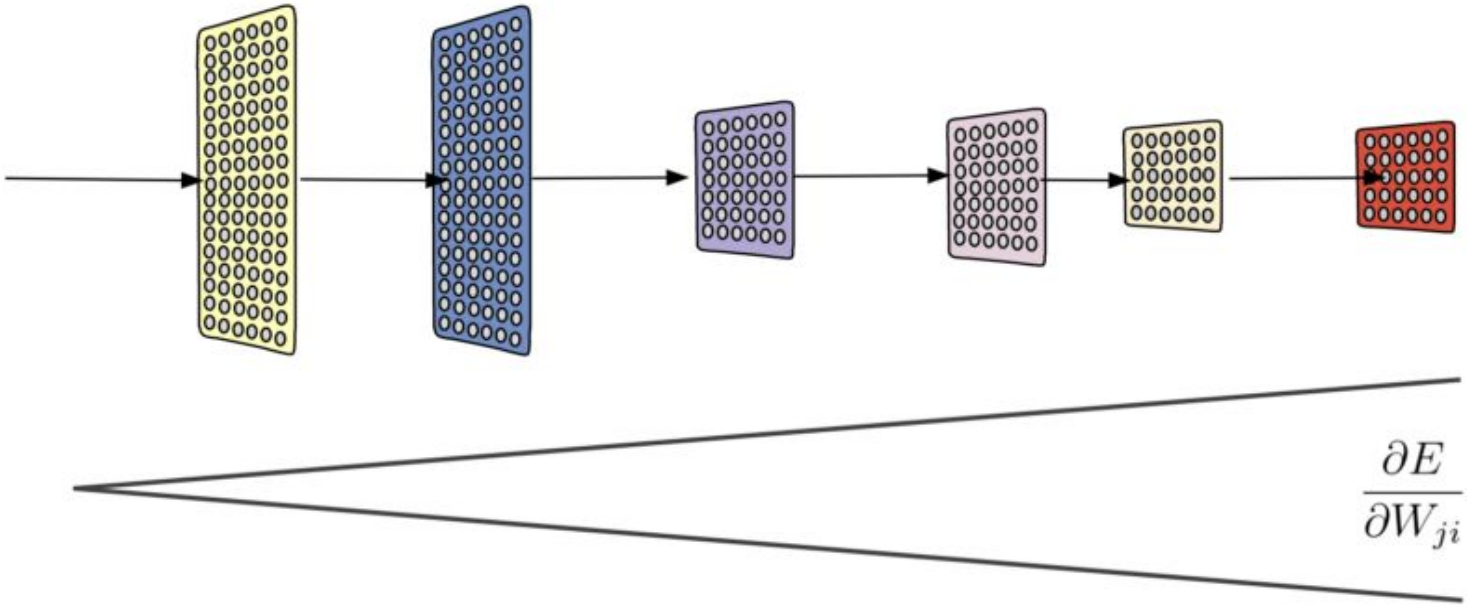
Residual networks

Depth didn’t always help

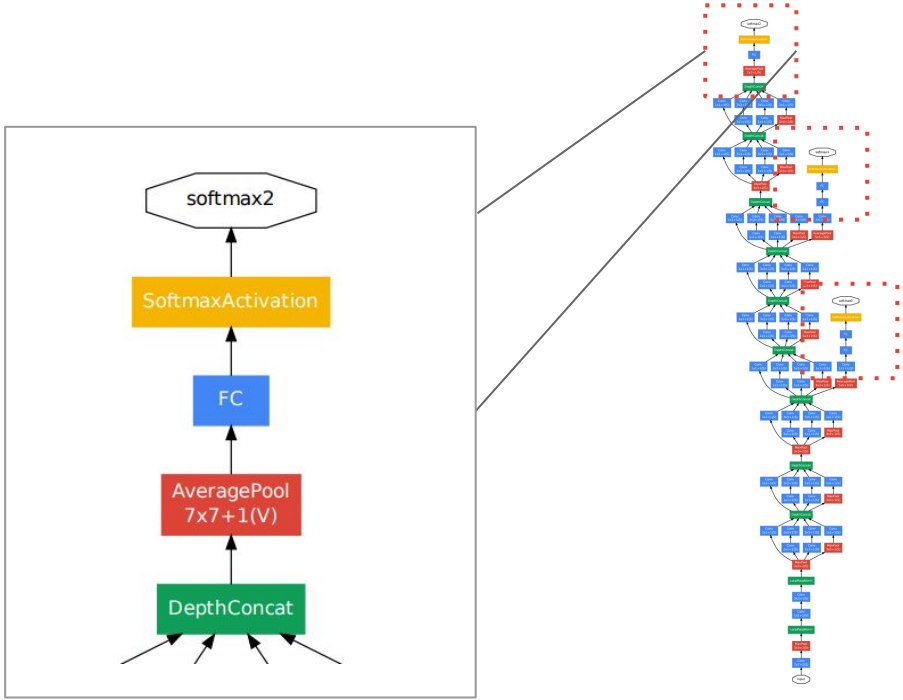


Adapted from
https://arxiv.org/pdf/1512.03385.pdf

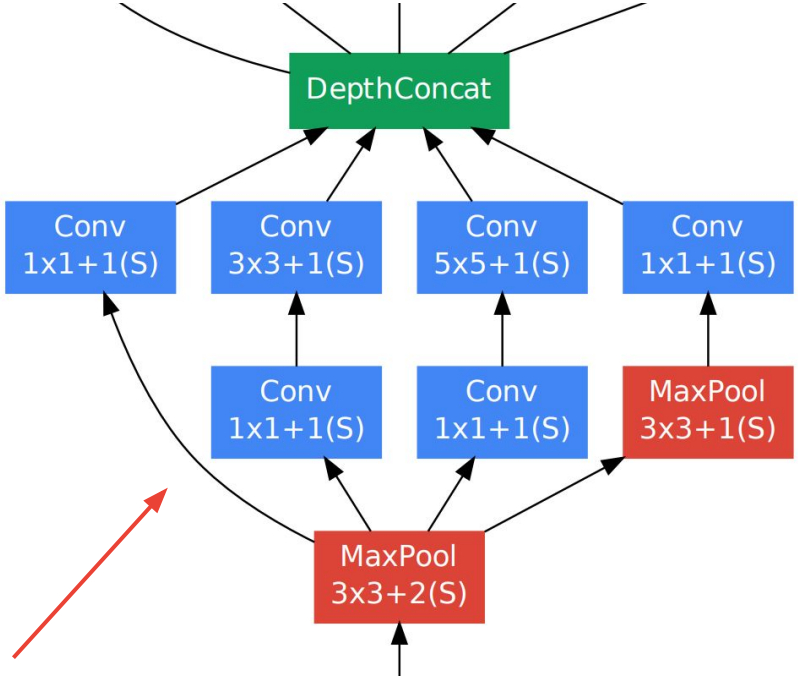
Vanishing gradients



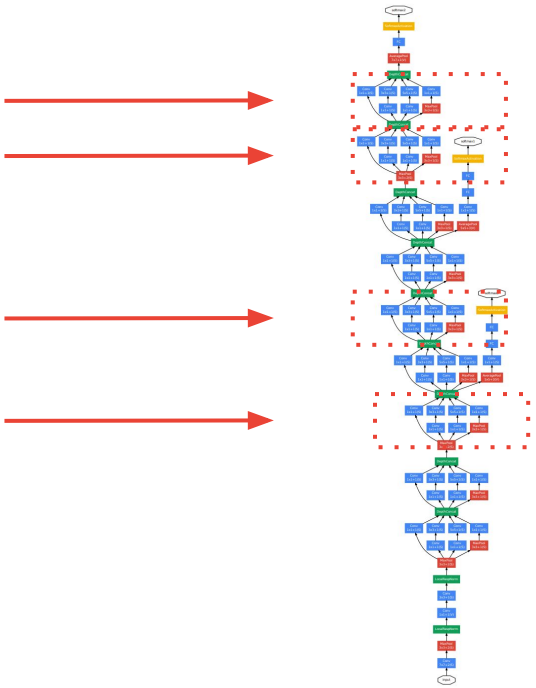
GoogLeNet tried to tackle the problem of vanishing gradients using two novel ideas



Parallel paths and shortcuts

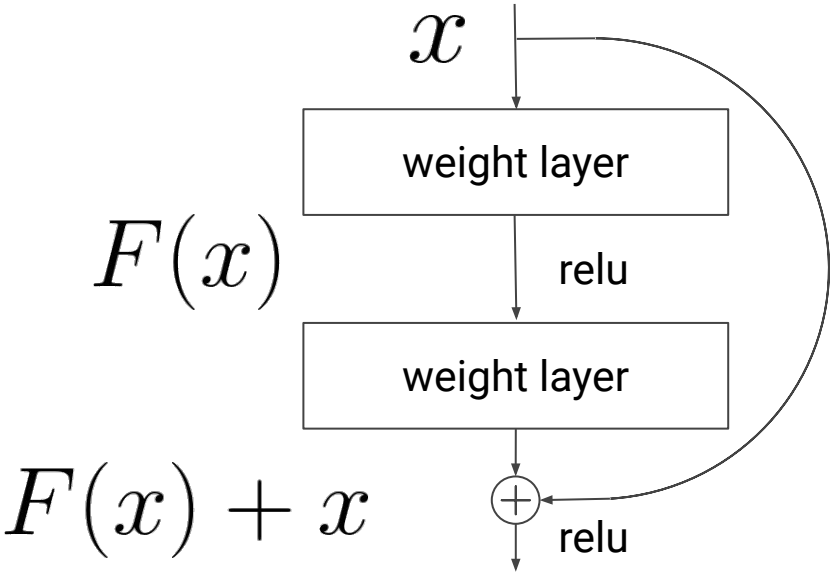


GoogLeNet used a repeating structure



Residual shortcuts decrease the effective depth

Residual is smaller



x
identity

Shallower

Deep Residual Learning for Image Recognition
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 2015
https://arxiv.org/abs/1512.03385

ImageNet dataset has one million images

2012 AlexNet: 8

2013 ZFNet: 8

2014 VGGNet: 19
2014 GoogLeNet: 22

2015 ResNet-152

Subsequent work on residual connections

1. ResNext
2. DenseNet
3. FractalNet
4. SqueezeNet
5. Stochastic Depth