File Name: T-ICML-O_3_I1_the_data_scarcity_problem

Format: Presenter in Studio

Presenter: Max Lotstein

Google Cloud

Dealing with Data Scarcity

Max Lotstein

Agenda

The Data Scarcity Problem

Data Augmentation

Transfer Learning

No Data, No Problem

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(img):
  X = tf.reshape(img,[-1,HEIGHT*WIDTH])
  W = tf.get_variable("W",
      [HEIGHT*WIDTH, NCLASSES])
  b = tf.get_variable("b",NCLASSES)
  ylogits = tf.matmul(X,W)+b
  return ylogits, NCLASSES
```

How many parameters in our linear model?

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

How many parameters 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

```
def dnn_model(img, mode, hparams):
  X = tf.reshape(img, [-1, HEIGHT*WIDTH])
  h1 = tf.layers.dense(X, 300)
  h2 = tf.layers.dense(h1,100)
  h3 = tf.layers.dense(h2, 30)
  ylogits = tf.layers.dense(h3, NCLASSES,
              activation= None)
  return ylogits, NCLASSES
```

How many parameters are in the h1 layer?

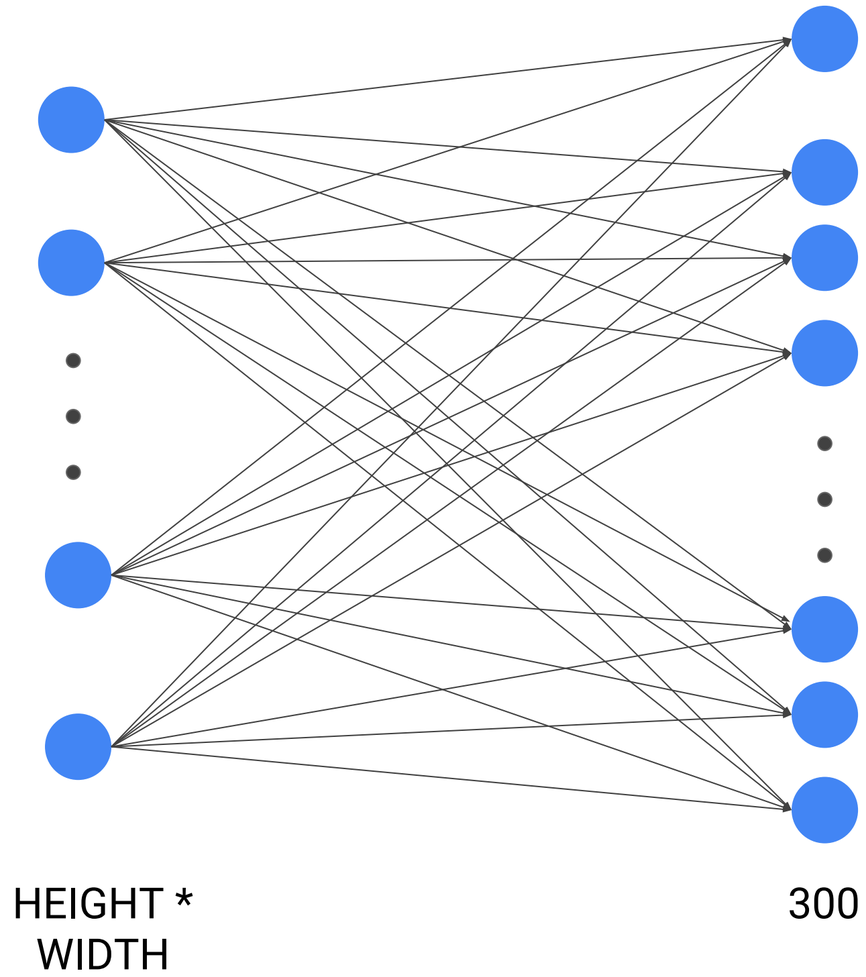
```
X = tf.reshape(img, [-1, HEIGHT*WIDTH])
h1 = tf.layers.dense(X, 300)
```

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

How many parameters are in the h1 layer?

```
# X.shape = [-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**


HEIGHT *
WIDTH
300

Parameters by model type for MNIST

Model Type	Number of Parameters
Toy linear model	7850
Toy DNN	270,000

A simple CNN model

```
import tensorflow.layers
def cnn_model(img, mode, params):
  c1 = conv2d(img, filters=10,
              kernel_size=3, strides=1,
              padding='same')
  p1 = max_pooling2d(c1,pool_size=2, strides=2)
  c2 = conv2d(p1, filters=20,
              kernel_size=3, strides=1,
              padding='same')
  p2 = max_pooling2d(c2,pool_size=2, strides=2)

  outlen = p2.shape[1]*p2.shape[2]*p2.shape[3]
  p2flat = tf.reshape(p2, [-1, outlen]) # flattened

  h3 = dense(p2flat, 300)
  ylogits = dense(h3, NCLASSES, activation=None)
  return ylogits, NCLASSES
```

How many parameters are in the c1 layer?

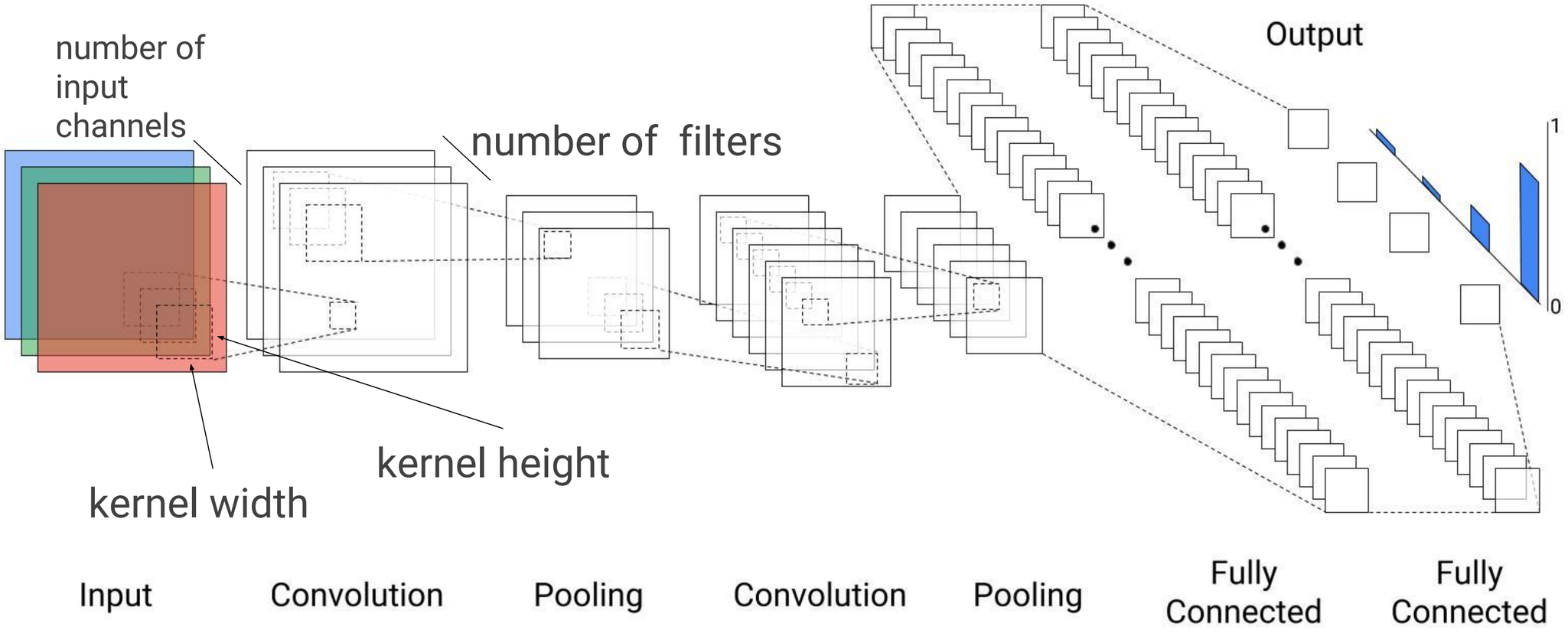
```
# img.shape = [-1, HEIGHT, WIDTH]
c1 = conv2d(img, filters=10,
            kernel_size=3, strides=1,
            padding='same')
```

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

How many parameters are in the `c1` layer?

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?

When you have some data:

- Data augmentation
- Transfer learning

File Name: T-ICML-O_3_I2_data_augmentation

Format: Presenter in Studio

Presenter: Max Lotstein

Agenda

The Data Scarcity Problem

Data Augmentation

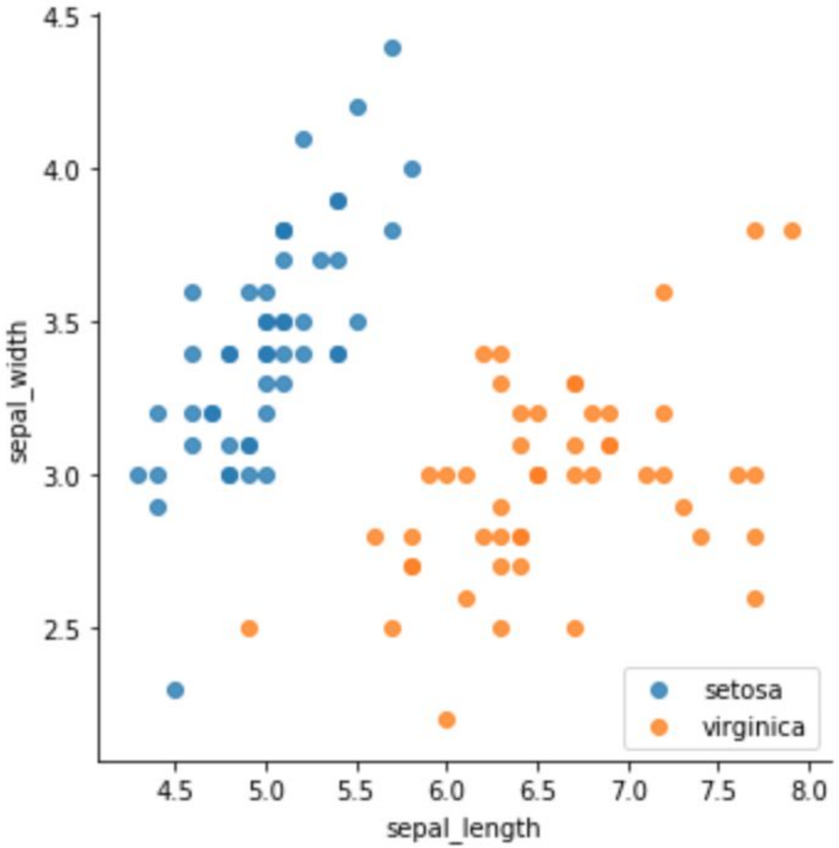
Transfer Learning

No Data, No Problem

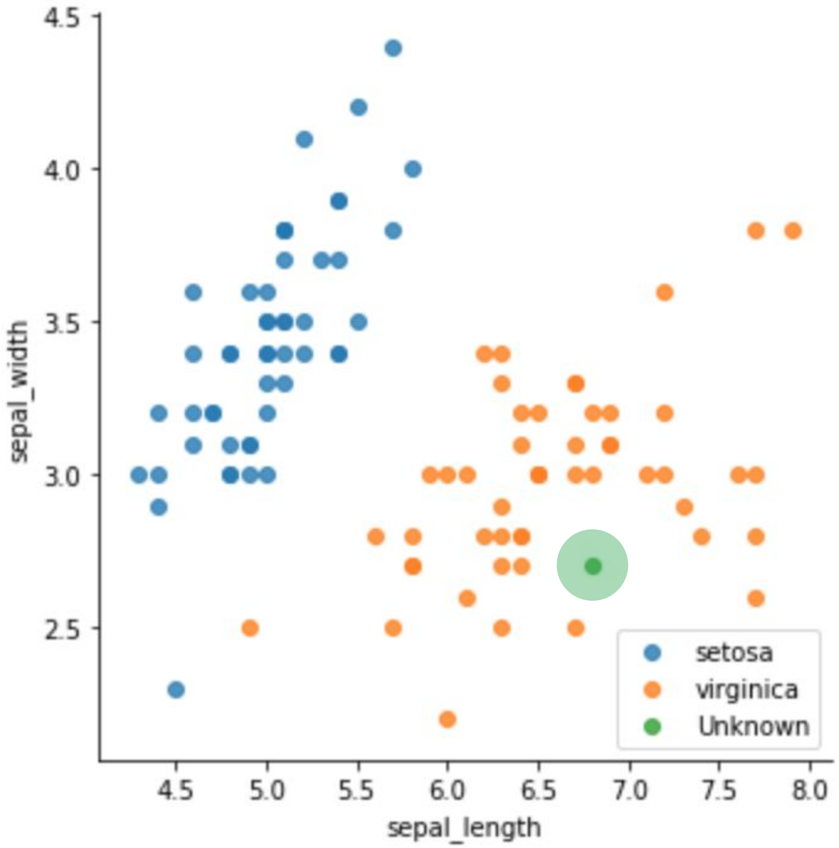
The iris dataset is structured

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

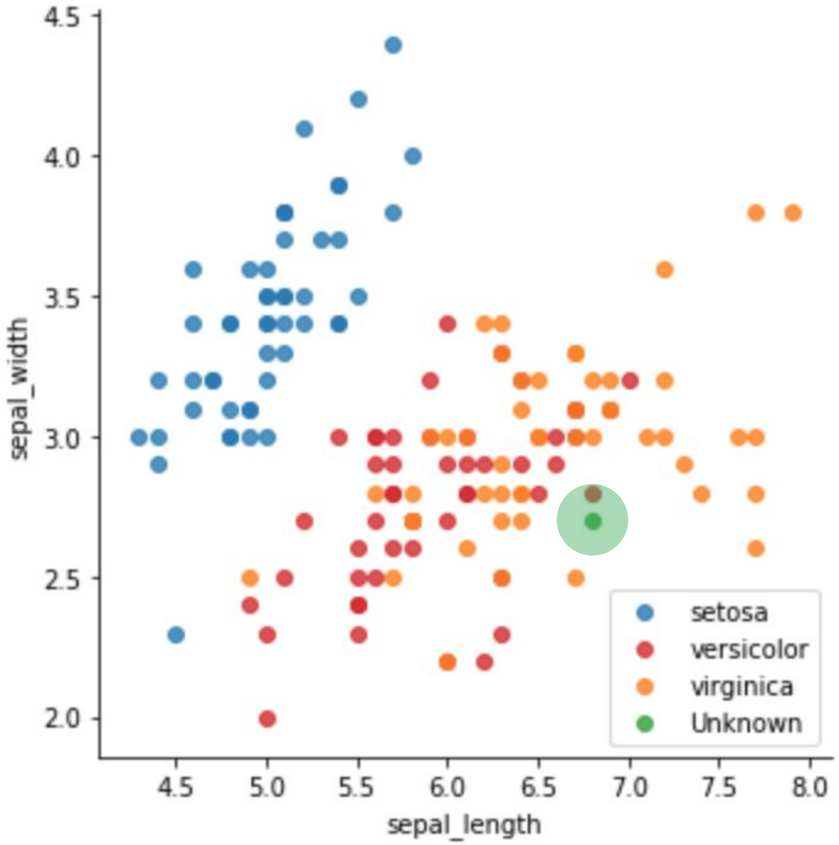
A typical labeled, structured dataset



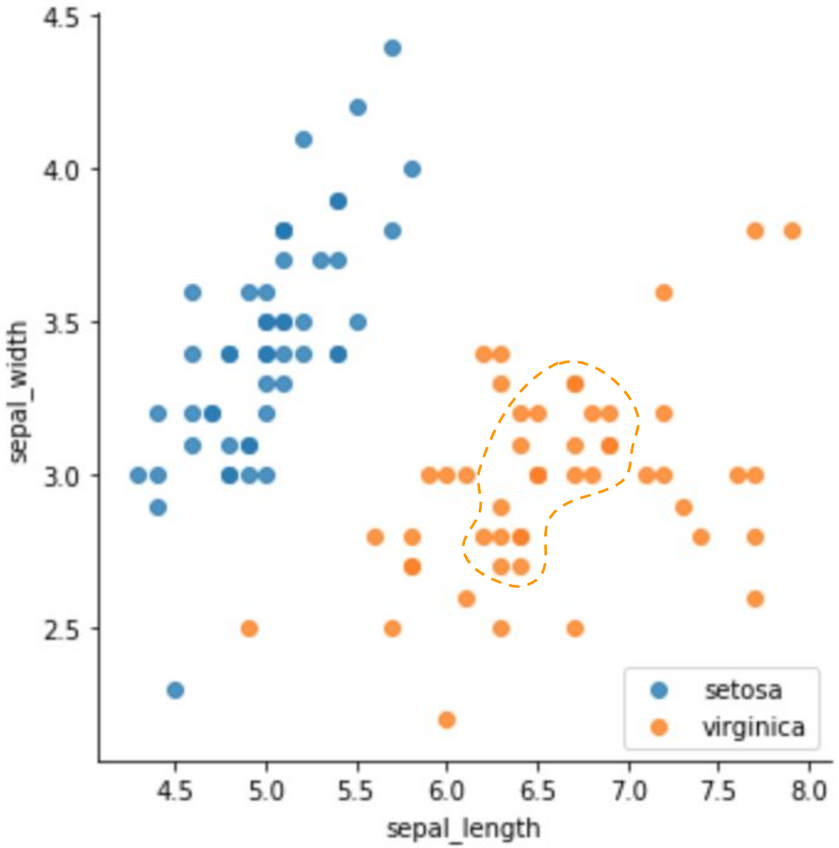
What about this new point?



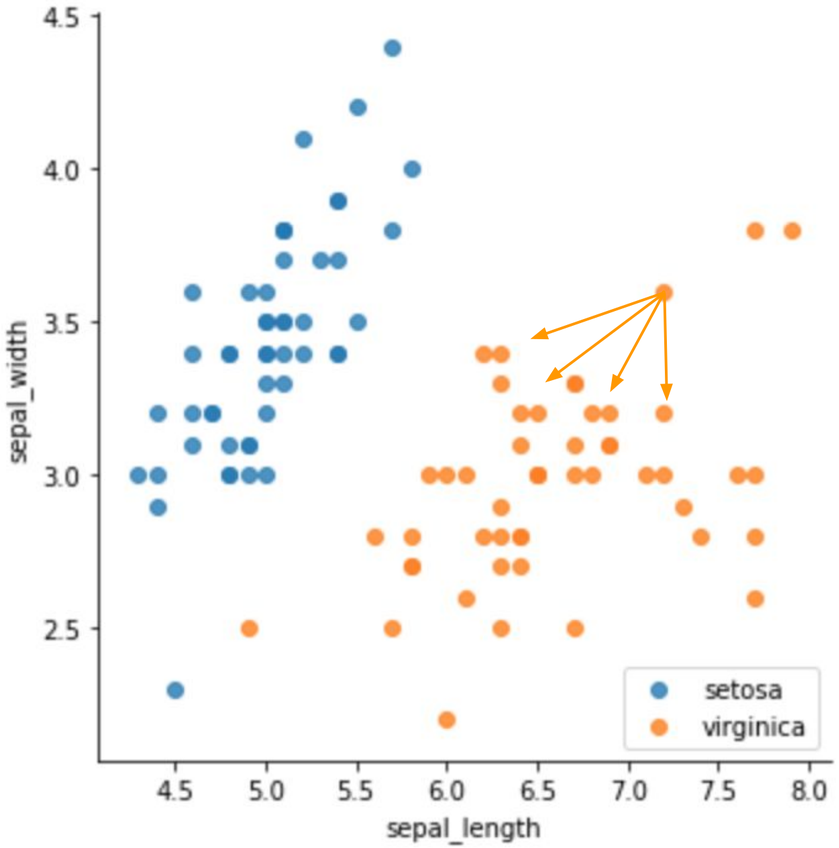
What about now?



One strategy is took look for clusters



Another is to perturb individual data points



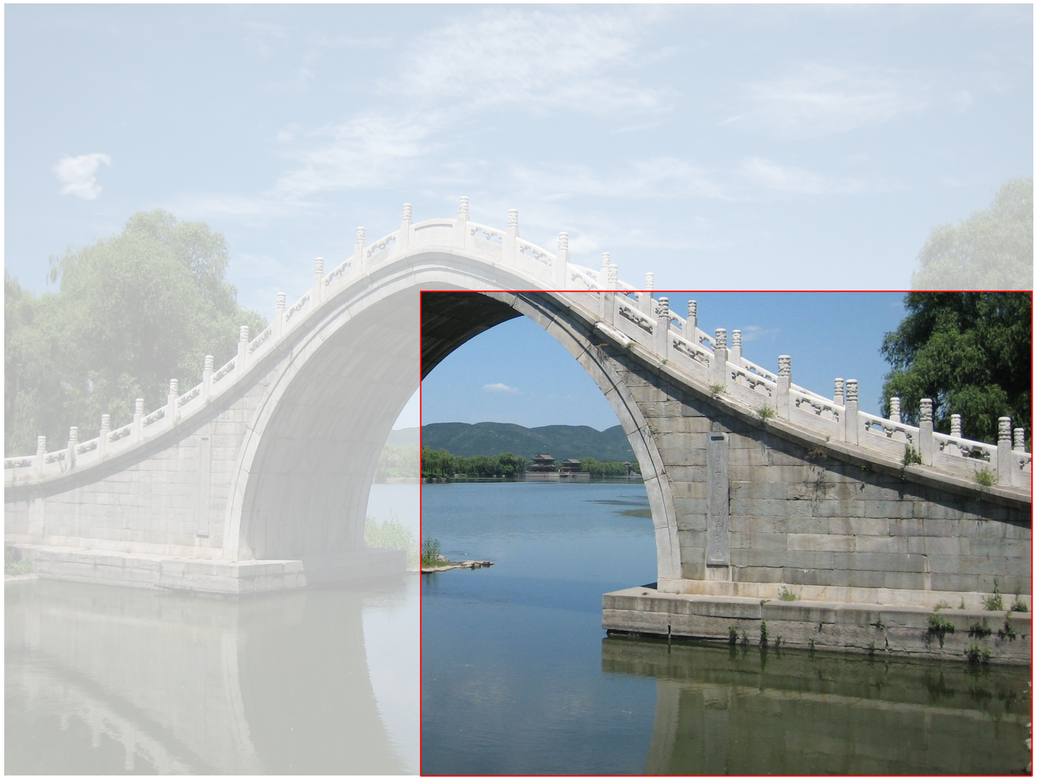
Common image augmentation techniques

Common image augmentation techniques

Common image augmentation techniques

Common image augmentation techniques

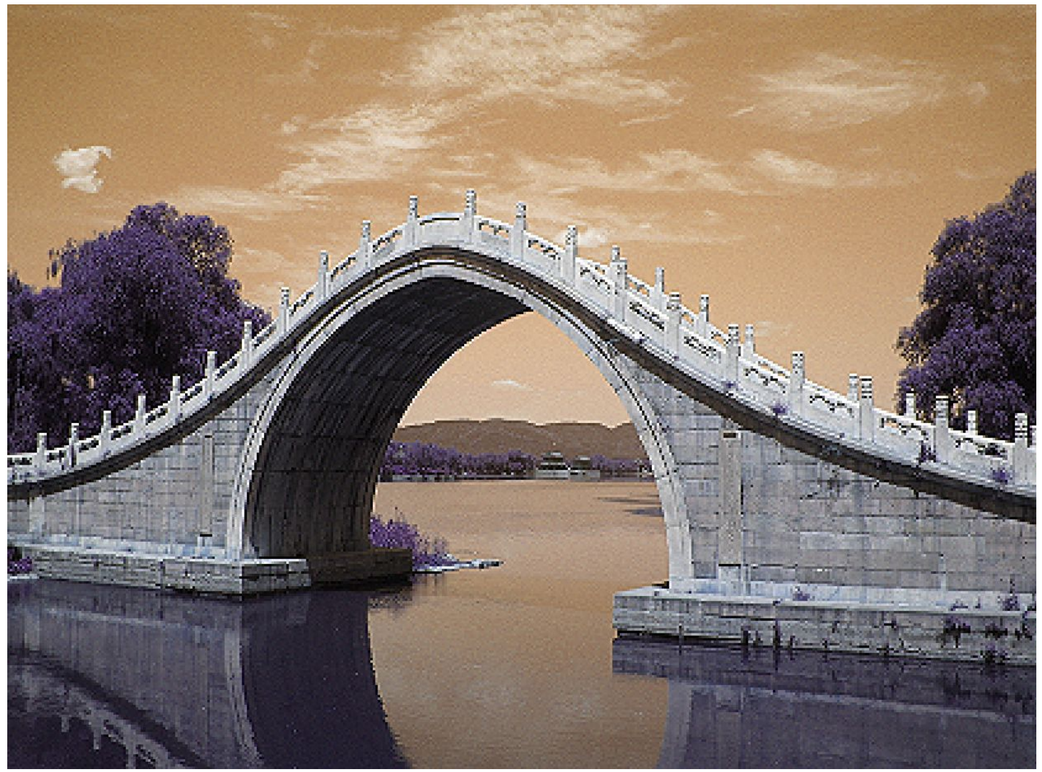
Common image augmentation techniques



Common image augmentation techniques

Common image augmentation techniques

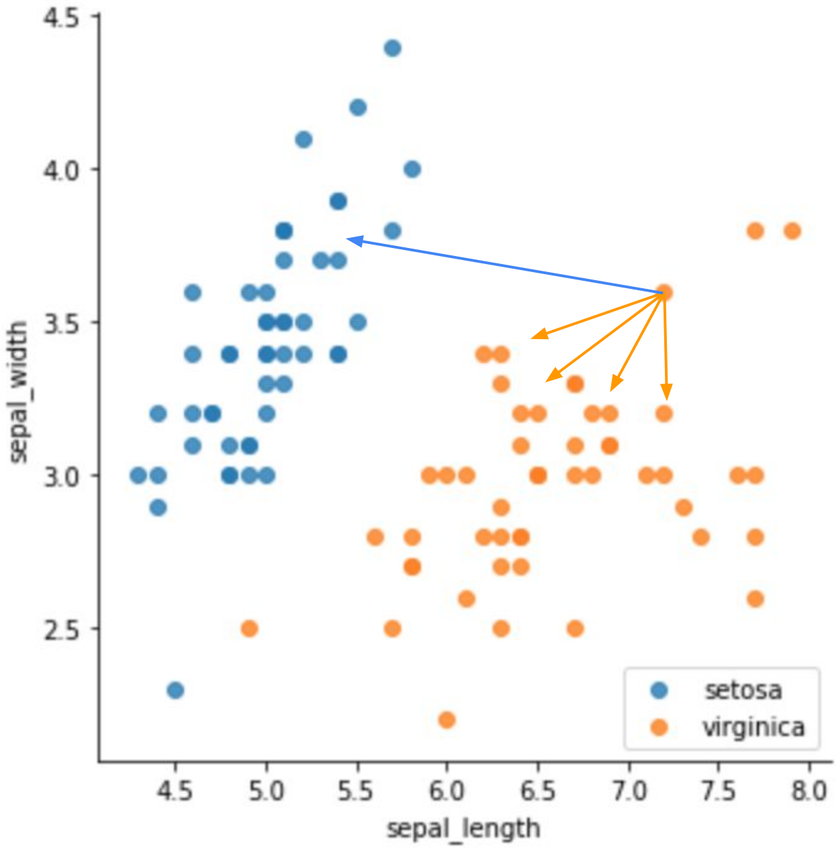
Common image augmentation techniques



Common image augmentation techniques

Common image augmentation techniques

Data augmentation requires care

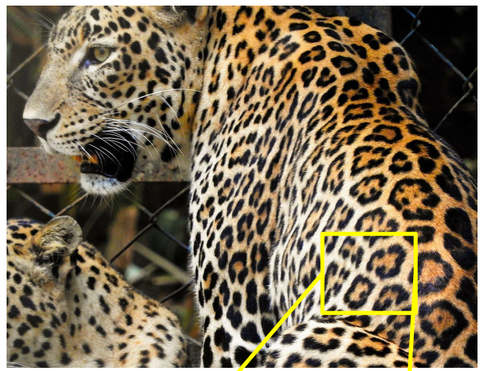


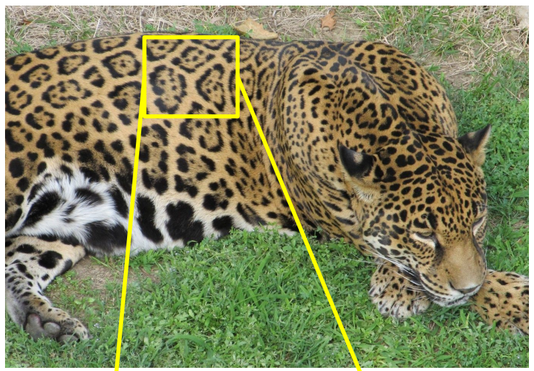
Sometimes color is informative

Sometimes color is informative

Sometimes orientation is informative

Sometimes small detail is informative





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

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

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?

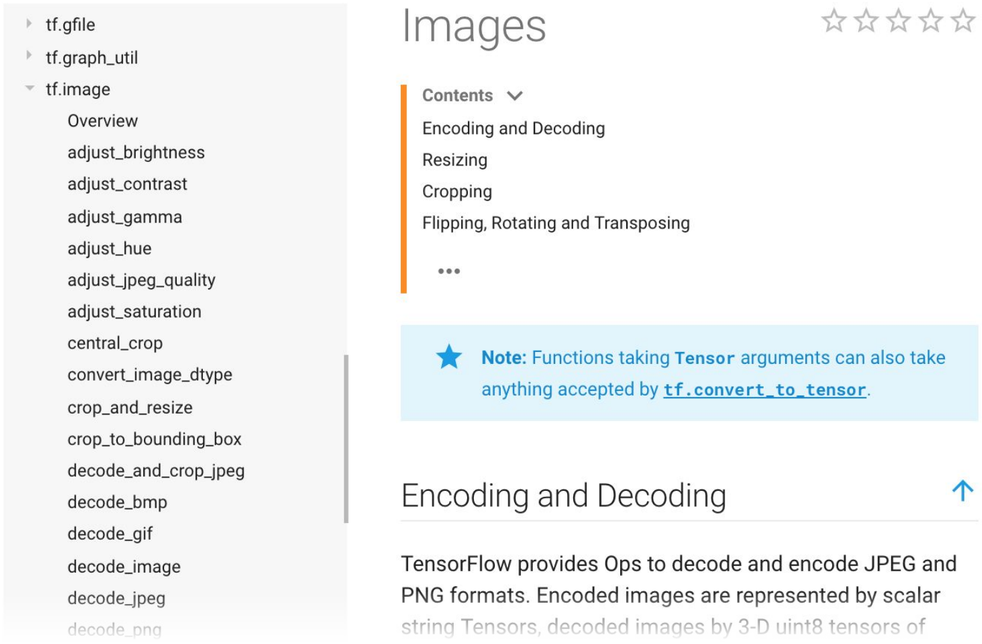
1. Yes
2. **No**

Implementing image augmentation

```
def make_input_fn(csv_of_filenames, batch_size, mode,augment):
  def _input_fn():
    def decode_csv(csv_row):
      filename, label = tf.decode_csv(
          csv_row, record_defaults = [[''],['']])
    image_bytes = tf.read_file(filename)
    return image_bytes, label

  dataset = tf.data.TextLineDataset(csv_of_filenames)
    .map(decode_csv).map(decode_jpeg).map(resize_image)
   if augment:
      dataset = dataset.map(augment_image)
  dataset = dataset.map(postprocess_image)
  ...
  return dataset.make_one_shot_iterator().get_next()
  return _input_fn
```

TensorFlow has a robust set of image functions



Implementing image augmentation

```
def make_input_fn(csv_of_filenames, batch_size, mode,augment):
  def _input_fn():
    def decode_csv(csv_row):
      filename, label = tf.decode_csv(
          csv_row, record_defaults = [[''],['']])
    image_bytes = tf.read_file(filename)
    return image_bytes, label

  dataset = tf.data.TextLineDataset(csv_of_filenames)
    .map(decode_csv).map(decode_jpeg).map(resize_image)
   if augment:
      dataset = dataset.map(augment_image)
  dataset = dataset.map(postprocess_image)
  ...
  return dataset.make_one_shot_iterator().get_next()
  return _input_fn
```

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 make_input_fn(csv_of_filenames, batch_size, mode,augment):
  def _input_fn():
    def decode_csv(csv_row):
      filename, label = tf.decode_csv(
          csv_row, record_defaults = [[''],['']])
    image_bytes = tf.read_file(filename)
    return image_bytes, label

  dataset = tf.data.TextLineDataset(csv_of_filenames)
    .map(decode_csv).map(decode_jpeg).map(resize_image)
   if augment:
      dataset = dataset.map(augment_image)
  dataset = dataset.map(postprocess_image)
  ...
  return dataset.make_one_shot_iterator().get_next()
  return _input_fn
```

Implementing `decode_jpeg()`

```
def make_input_fn(csv_of_filenames, batch_size, mode,augment):
  def _input_fn():
    def decode_csv(csv_row):
      filename, label = tf.decode_csv(
          csv_row, record_defaults = [[''],['']])
    image_bytes = tf.read_file(filename)
    return image_bytes, label

  dataset = tf.data.TextLineDataset(csv_of_filenames)
    .map(decode_csv).map(decode_jpeg).map(resize_image)
   if augment:
      dataset = dataset.map(augment_image)
  dataset = dataset.map(postprocess_image)
  ...
  return dataset.make_one_shot_iterator().get_next()
  return _input_fn
```

Implementing `decode_jpeg()`

```
import tensorflow.image as tfi
def decode_jpeg(image_bytes, label):
    image = tfi.decode_jpeg(image_bytes, channels=NUM_CHANNELS)
    image = tfi.convert_image_dtype(image, dtype=tf.float32) # 0-1
    return image, label
```

Implementing `resize_image()`

```
def make_input_fn(csv_of_filenames, batch_size, mode,augment):
  def _input_fn():
    def decode_csv(csv_row):
      filename, label = tf.decode_csv(
          csv_row, record_defaults = [[''],['']])
    image_bytes = tf.read_file(filename)
    return image_bytes, label

  dataset = tf.data.TextLineDataset(csv_of_filenames)
    .map(decode_csv).map(decode_jpeg).map(resize_image)
   if augment:
      dataset = dataset.map(augment_image)
  dataset = dataset.map(postprocess_image)
  ...
  return dataset.make_one_shot_iterator().get_next()
  return _input_fn
```

Implementing `resize_image()`

```
import tensorflow.image as tfi
def resize_image(image, label):
    image = tf.expand_dims(image, 0) # add batch dimension
    image = tfi.resize_bilinear(image, [HEIGHT, WIDTH],
            align_corners=False)
    image = tf.squeeze(image) # throw away batch dimension
    return image, label
```

Implementing `augment_image()`

```
def make_input_fn(csv_of_filenames, batch_size, mode,augment):
  def _input_fn():
    def decode_csv(csv_row):
      filename, label = tf.decode_csv(
          csv_row, record_defaults = [[''],['']])
    image_bytes = tf.read_file(filename)
    return image_bytes, label

  dataset = tf.data.TextLineDataset(csv_of_filenames)
    .map(decode_csv).map(decode_jpeg).map(resize_image)
   if augment:
      dataset = dataset.map(augment_image)
  dataset = dataset.map(postprocess_image)
  ...
  return dataset.make_one_shot_iterator().get_next()
  return _input_fn
```

Implementing `augment_image()`

```
def augment_image(image_dict, label=None):
  image = image_dict['image']
  image = tf.expand_dims(image, 0) # resize_bilinear needs batches
  image = tfi.resize_bilinear(
    image, [HEIGHT+10, WIDTH+10], align_corners=False)
  image = tf.squeeze(image) #remove batch dimension
  image = tf.random_crop(image, [HEIGHT, WIDTH, NUM_CHANNELS])
  image = tfi.random_flip_left_right(image)
  image = tfi.random_brightness(image, max_delta=63.0/255.0)
  image = tfi.random_contrast(image, lower=0.2, upper=1.8)

  return image, label
```

Implementing `augment_image()`

```
def augment_image(image_dict, label=None):
  image = image_dict['image']
  image = tf.expand_dims(image, 0) # resize_bilinear needs batches
  image = tfi.resize_bilinear(
     image, [HEIGHT+10, WIDTH+10], align_corners=False)
  image = tf.squeeze(image) #remove batch dimension
  image = tf.random_crop(image, [HEIGHT, WIDTH,
NUM_CHANNELS])
  image = tfi.random_flip_left_right(image)
  image = tfi.random_brightness(image, max_delta=63.0/255.0)
  image = tfi.random_contrast(image, lower=0.2, upper=1.8)

  return image, label
```

Implementing `postprocess_image()`

```
def make_input_fn(csv_of_filenames, batch_size, mode,augment):
  def _input_fn():
    def decode_csv(csv_row):
      filename, label = tf.decode_csv(
          csv_row, record_defaults = [[''],['']])
    image_bytes = tf.read_file(filename)
    return image_bytes, label

  dataset = tf.data.TextLineDataset(csv_of_filenames)
    .map(decode_csv).map(decode_jpeg).map(resize_image)
   if augment:
      dataset = dataset.map(augment_image)
  dataset = dataset.map(postprocess_image)
  ...
  return dataset.make_one_shot_iterator().get_next()
  return _input_fn
```

Implementing `postprocess_image()`

```
def postprocess_image(image, label):
    #pixel values are in range [0,1], convert to [-1,1]
    image = tf.subtract(image, 0.5)
    image = tf.multiply(image, 2.0)
    return {‘image’ : image}, label
```

File Name:
T-ICML-O_3_I3_lab_intro:_implementing_image_augmentation

Format: Presenter in Studio

Presenter: Max Lotstein

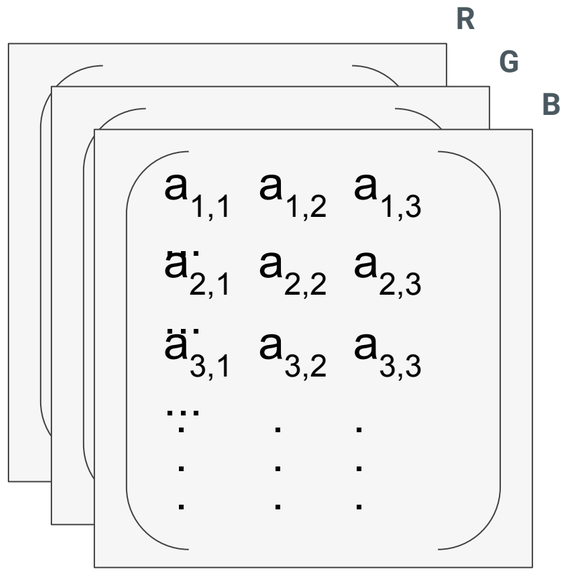
Lab

Implementing image augmentation

Max Lotstein

Working with color images

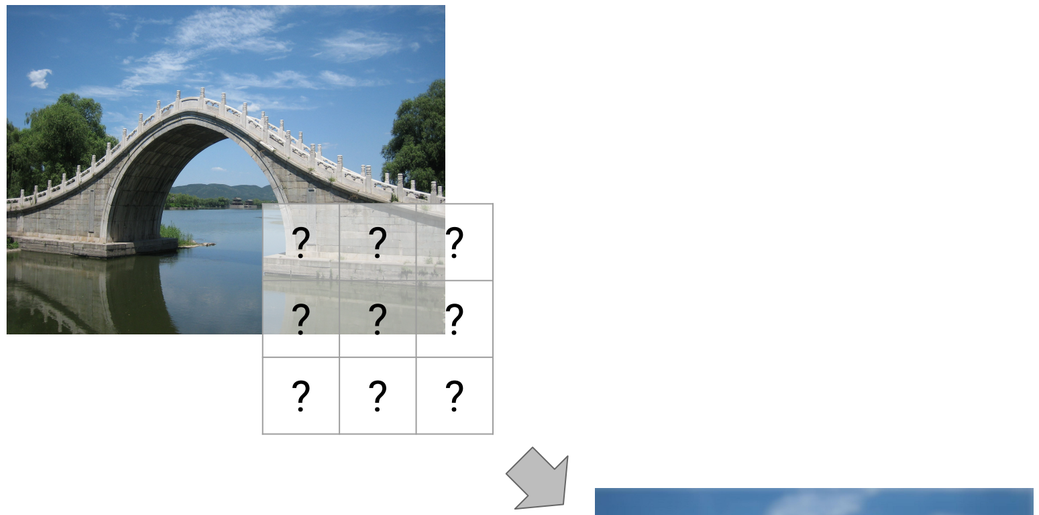
Color images are still just tensors



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



File Name: T-ICML-O_3_l9_image_augmentation_lab_intro

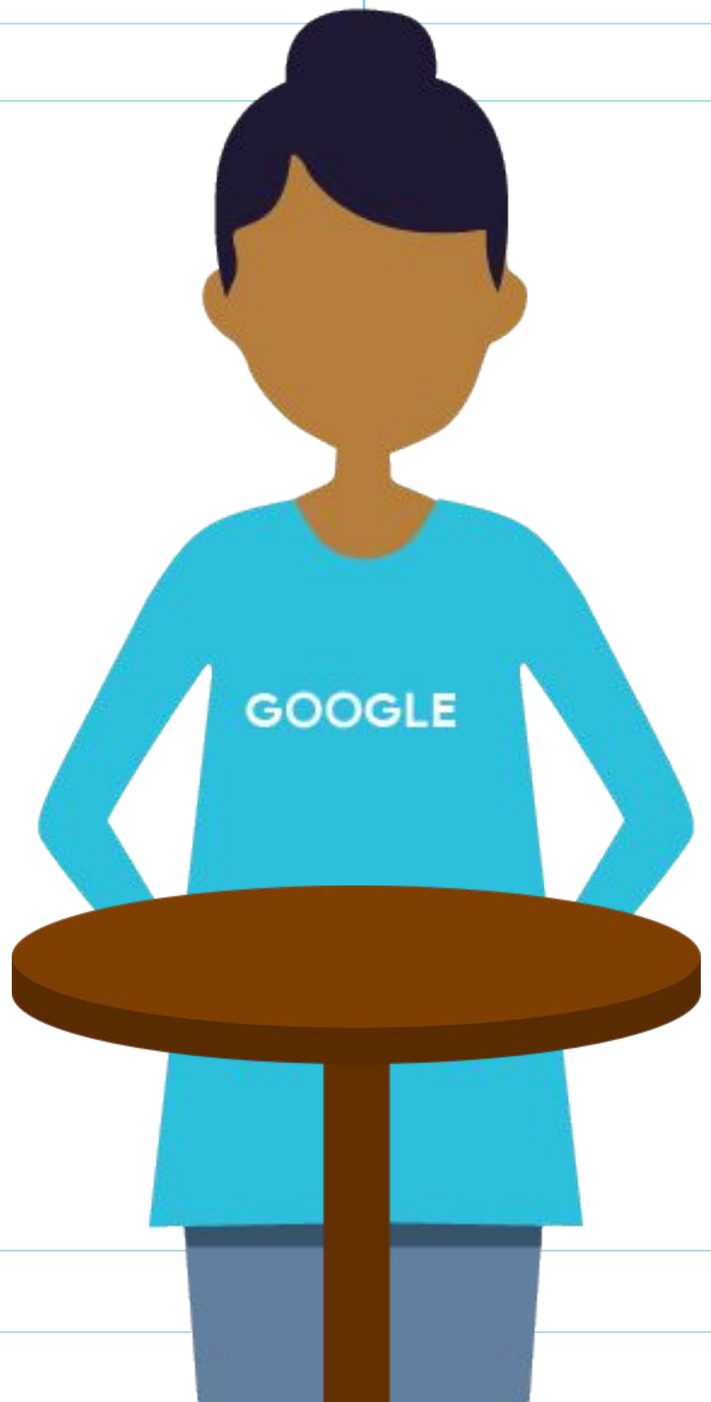
Format: Presenter in Studio

Presenter: Max Lotstein

File Name: T-ICML-O_3_l4_lab_solution:_implementing_image_augmentation

Format: Presenter in Studio

Presenter: Max Lotstein



Title Safe >

< Action Safe

File Name: T-ICML-O_3_l5_transfer_learning

Format: Presenter in Studio

Presenter: Max Lotstein

Agenda

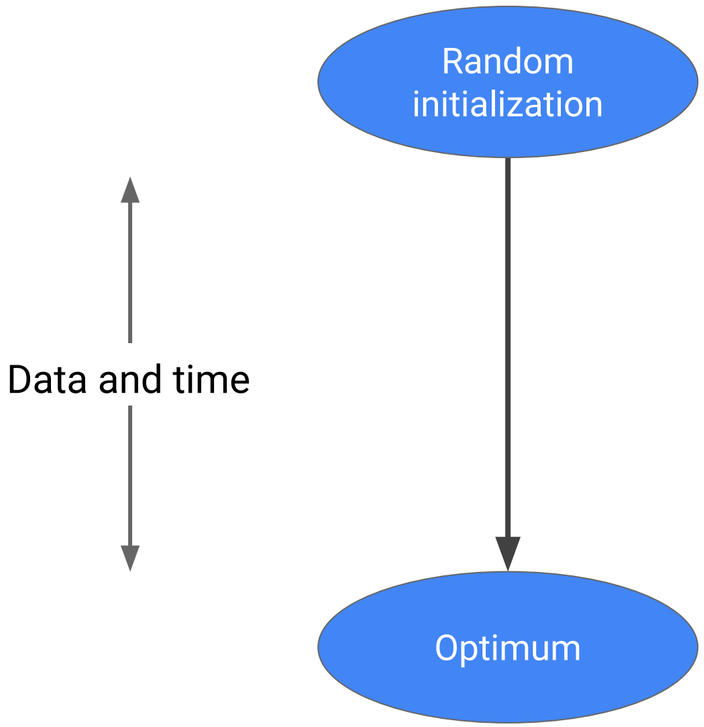
The Data Scarcity Problem

Data Augmentation

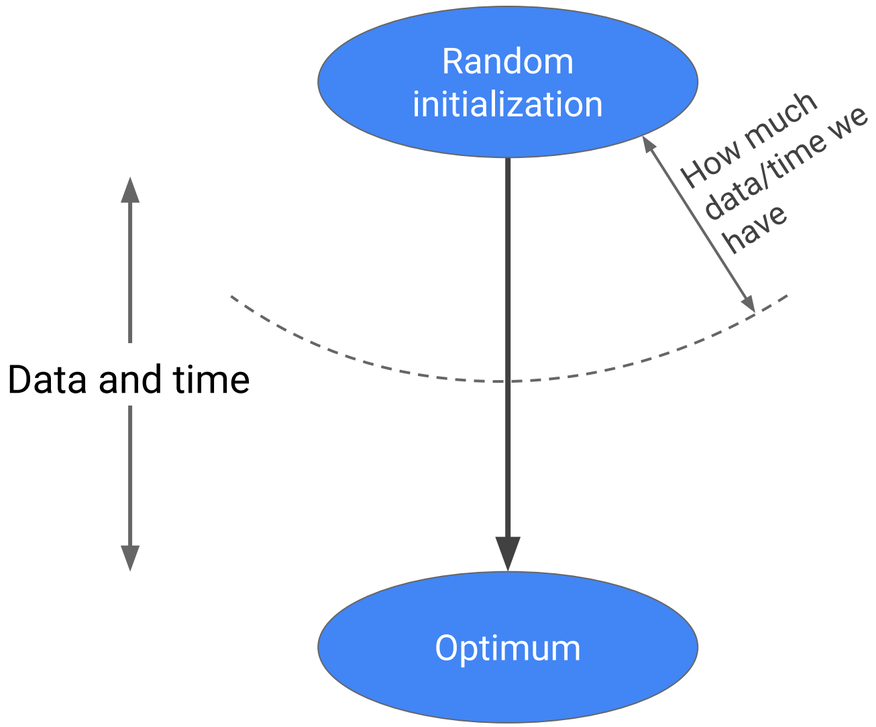
Transfer Learning

No Data, No Problem

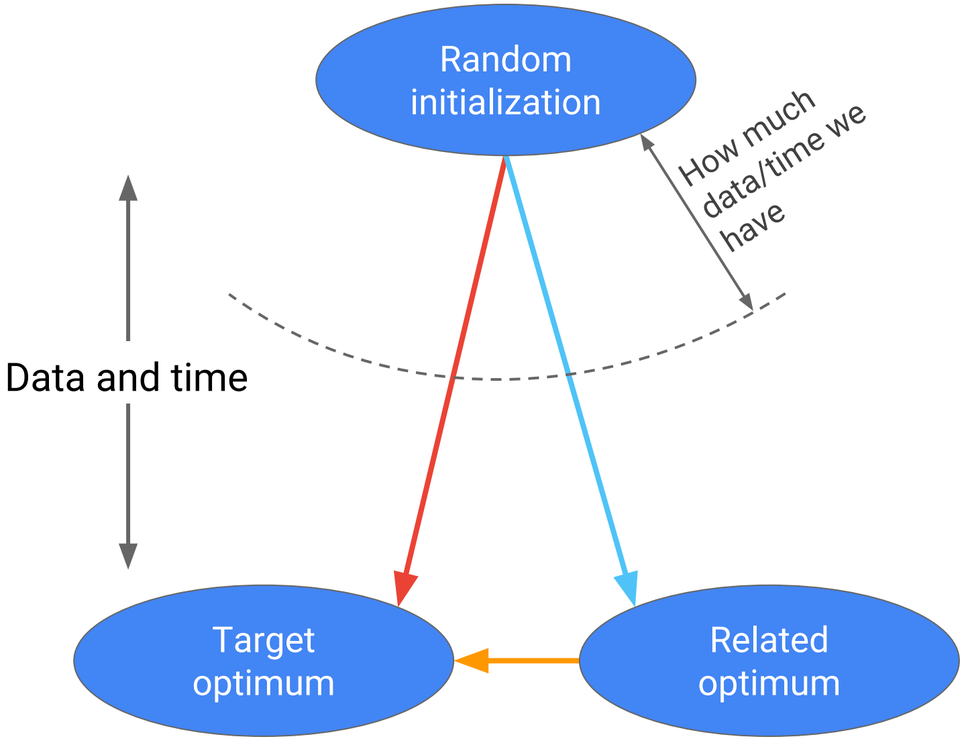
Optimization is a journey through weight space



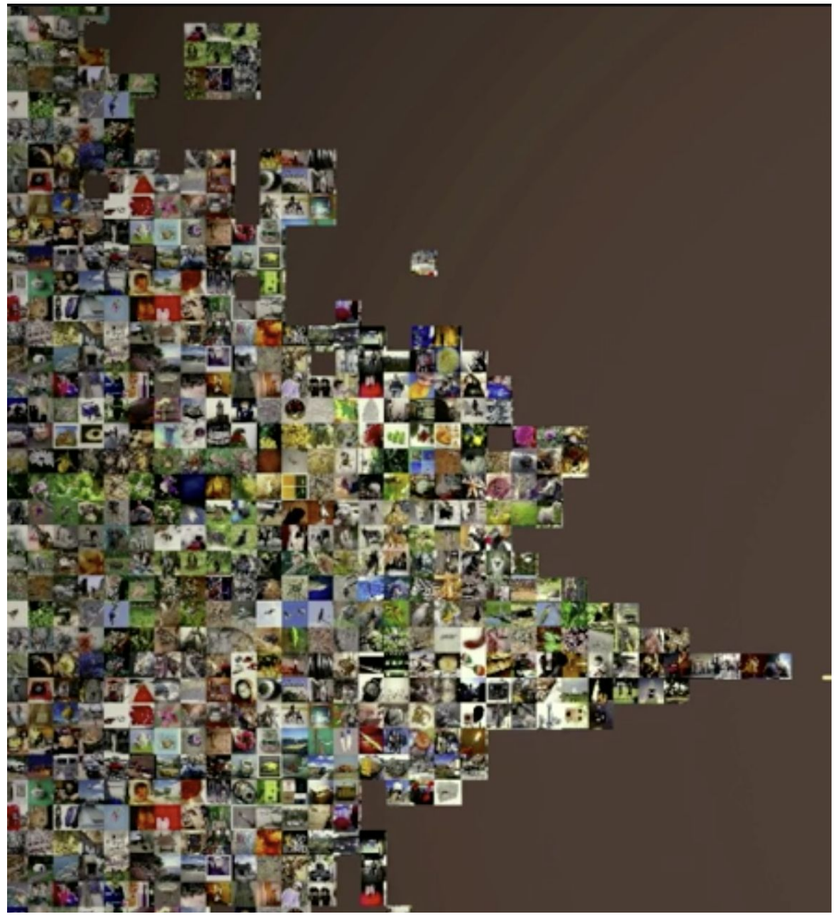
Unstructured domains limit our optimization range



Transfer learning is like a shortcut

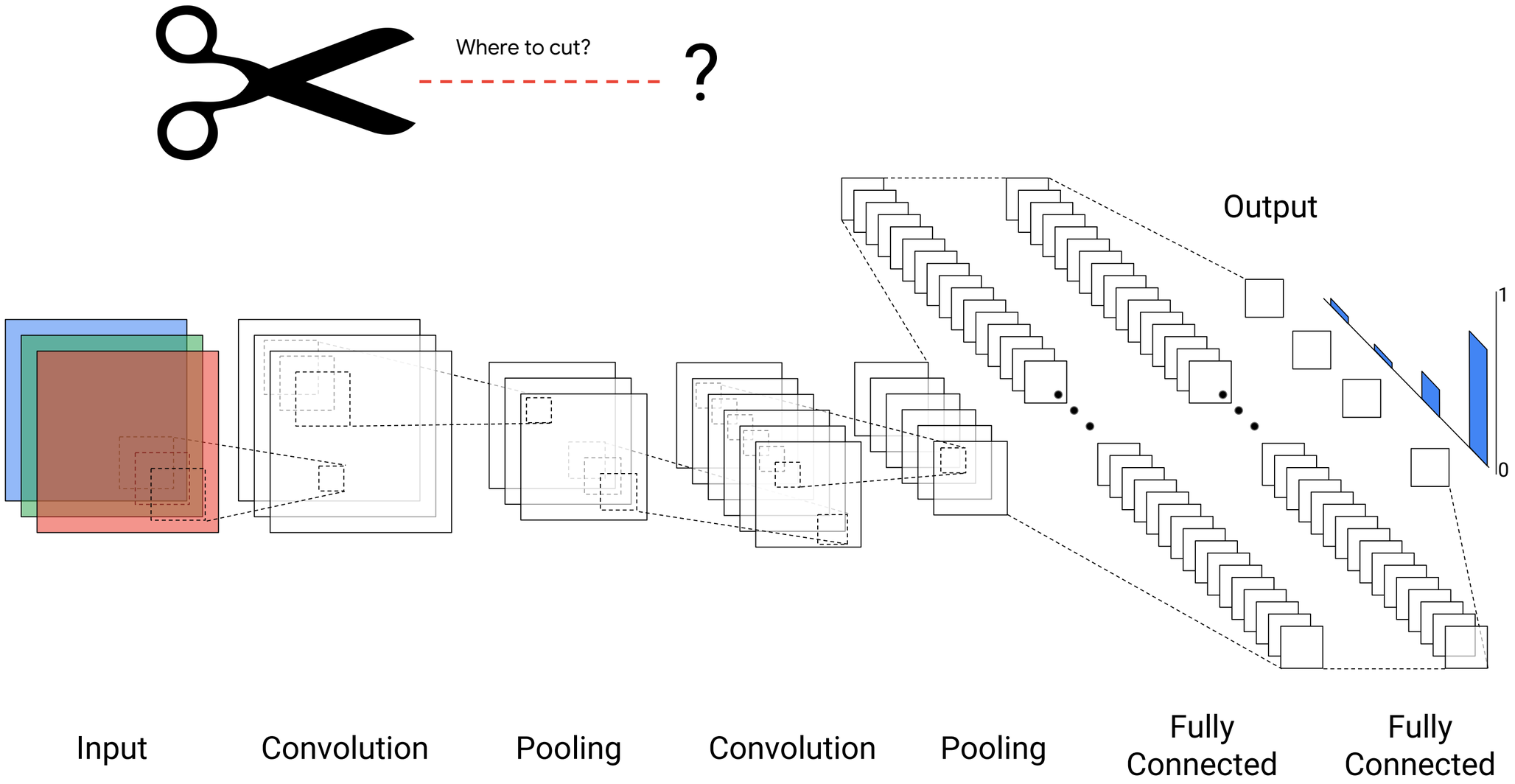


ImageNet



Naive Transfer Learning

ImageNet Classes		**Your Classes**
Domestic Dog	←	Domestic Dog
Domestic Cat	?	Cirrus
Field Corn	←	Field Corn
Sand Verbena	?	Cumulo Nimbus
...		....


Where to cut?
?
Output
1
0
Input
Convolution
Pooling
Convolution
Pooling
Fully Connected
Fully Connected

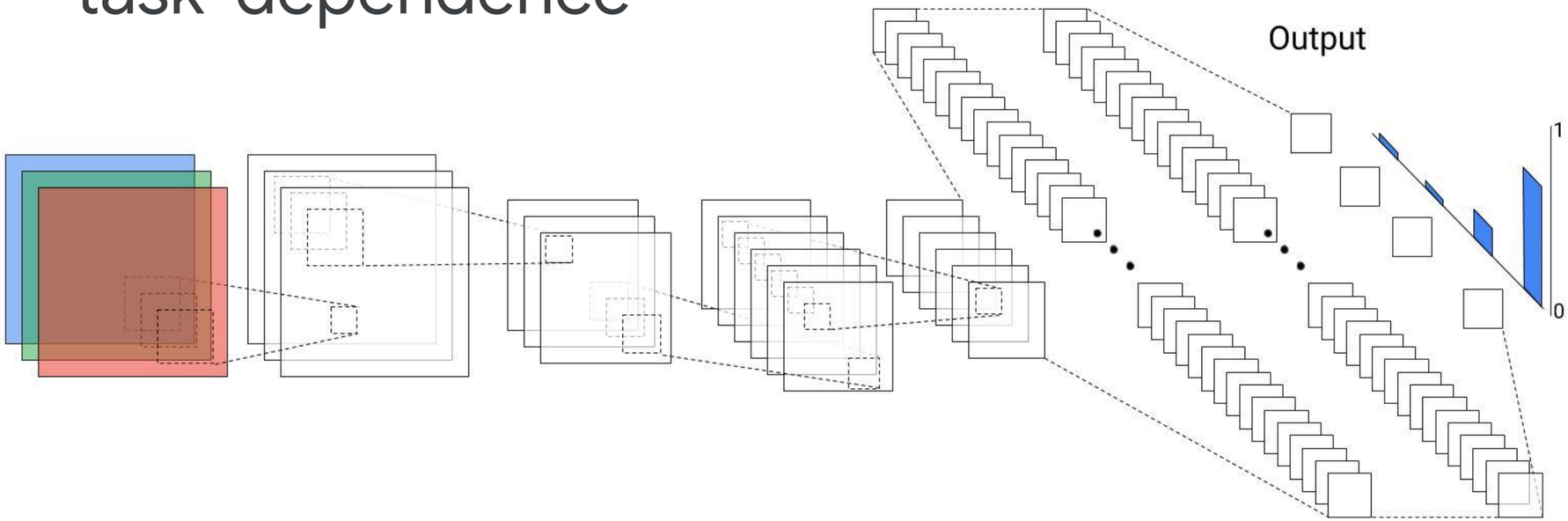
Which parts of the source image model are most closely aligned to the source task?

1. The input to the model
2. The convolutional layers
3. The layers near the output

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

1. The input to the model
2. The convolutional layers
3. **The layers near the output**

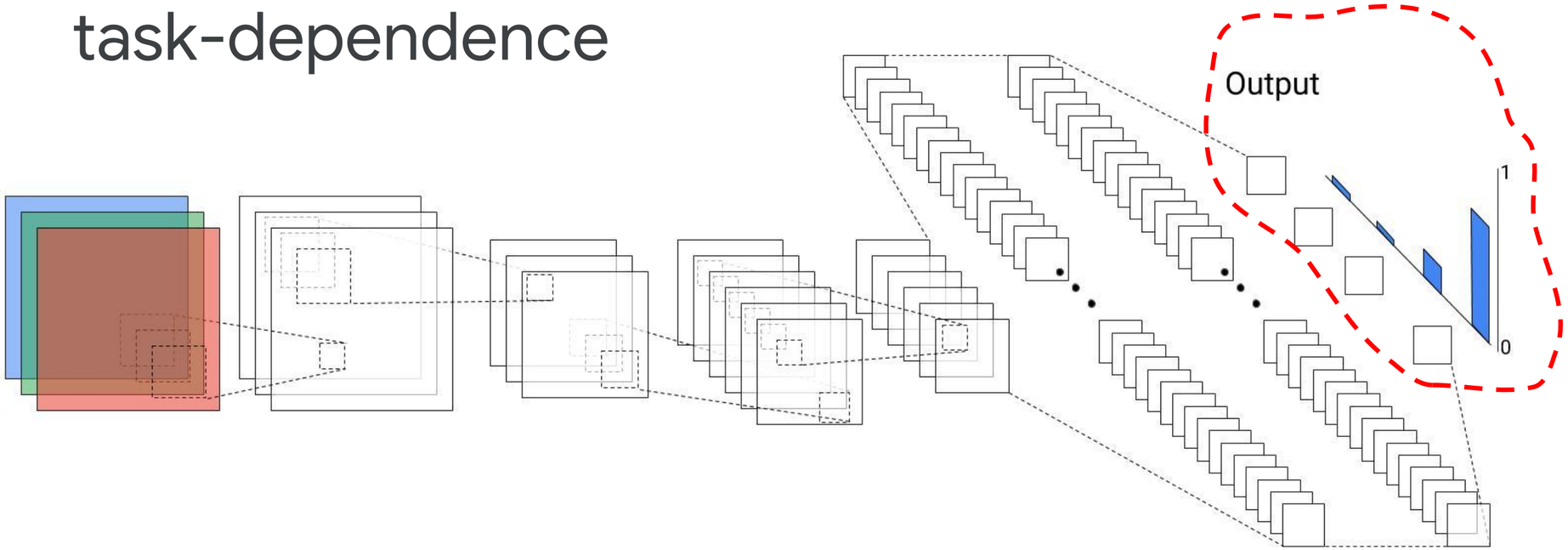
CNNs and the spectrum of task-dependence



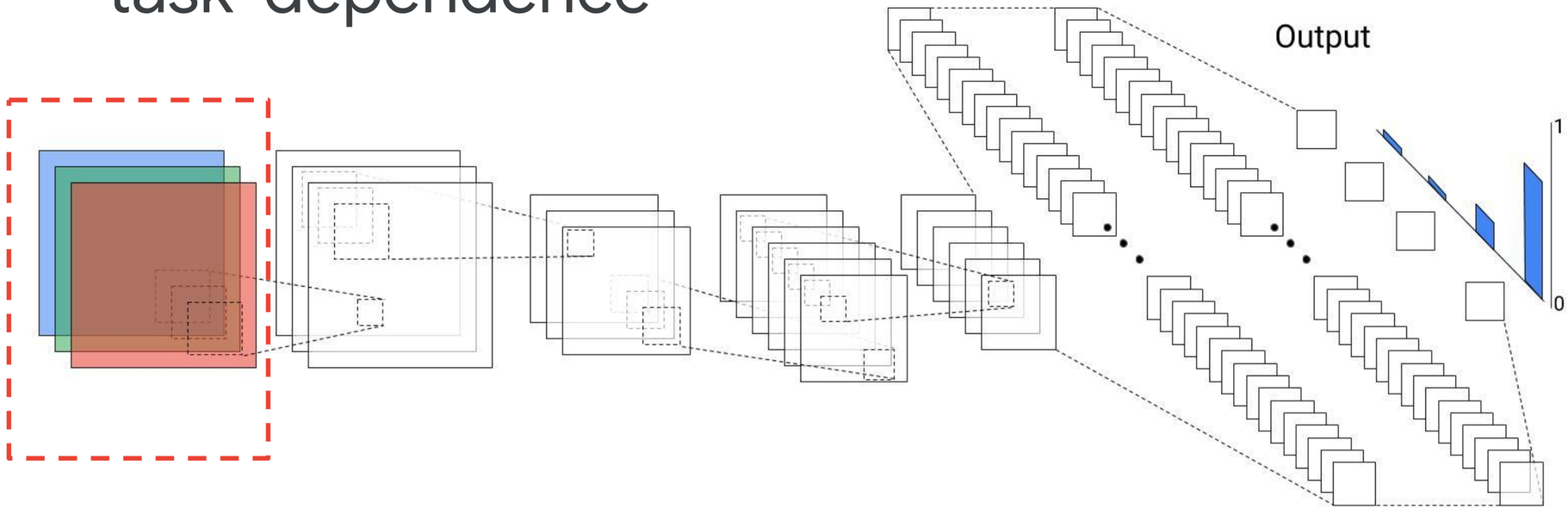
Most task-general

Most task-dependent

CNNs and the spectrum of task-dependence



CNNs and the spectrum of task-dependence



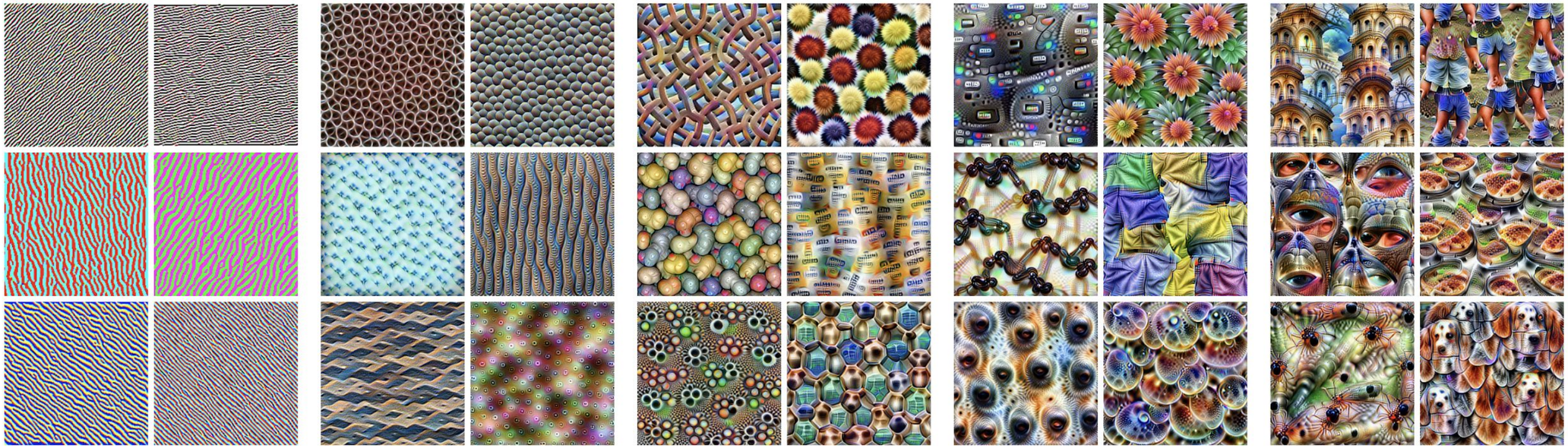
CNNs and the spectrum of task-dependence

Input | Convolution | Pooling | Convolution | Pooling | Fully Connected | Fully Connected

Most task-general

Most task-dependent

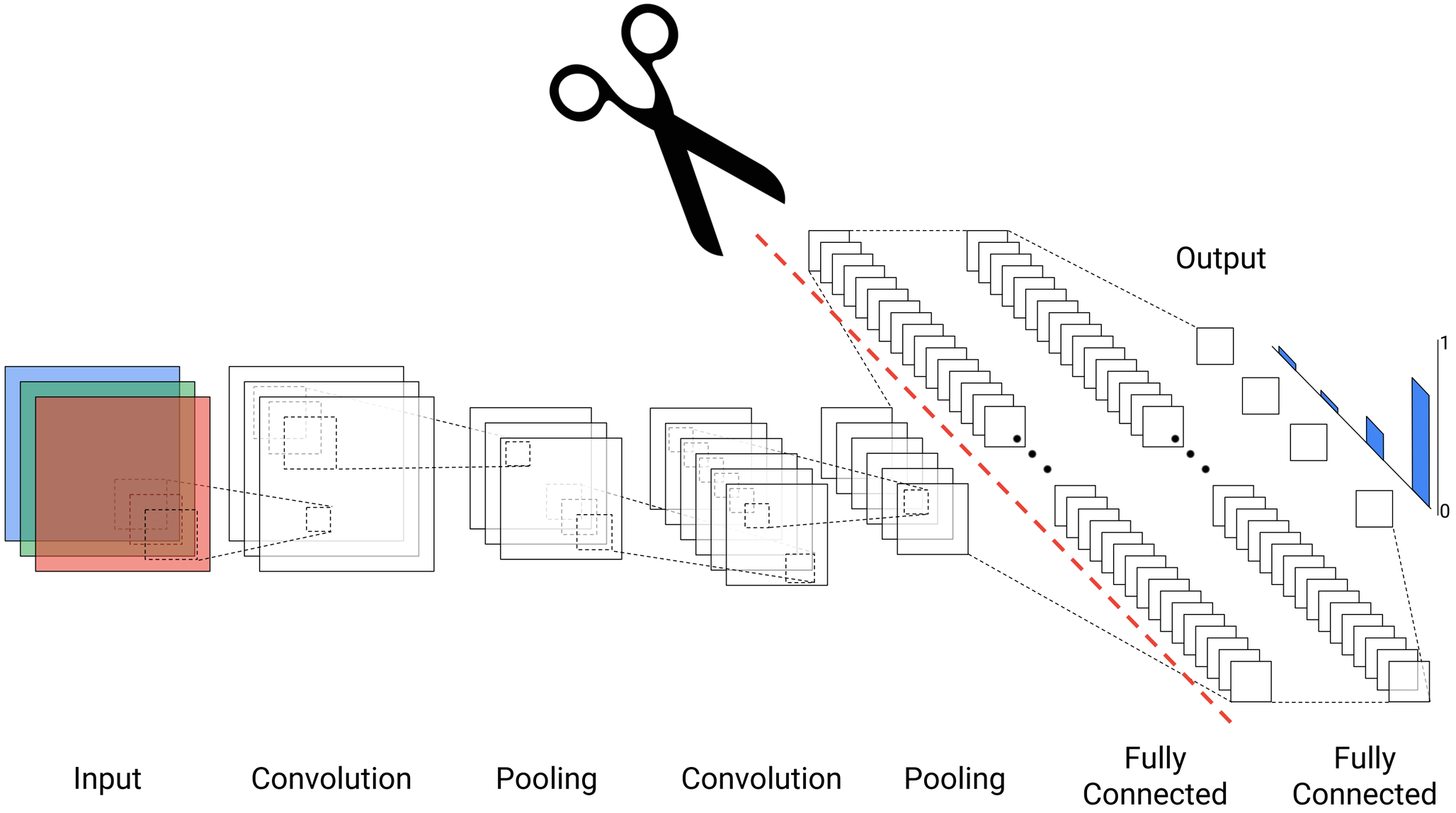
CNNs learn a hierarchy of features



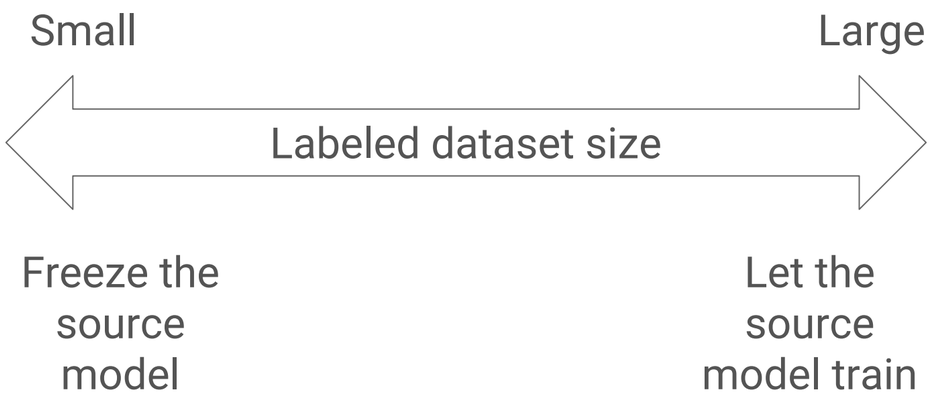
Edges (layer conv2d0) **Textures** (layer mixed3a) **Patterns** (layer mixed4a) **Parts** (layers mixed4b & mixed4c) **Objects** (layers mixed4d & mixed4e)

Flow of data in the network

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


Output
1
0
Input
Convolution
Pooling
Convolution
Pooling
Fully Connected
Fully Connected

To freeze or not to freeze?



File Name:
T-ICML-O_3_l6_lab_intro:_implementing_transfer_learning

Format: Presenter in Studio

Presenter: Max Lotstein

Lab

Implementing transfer learning

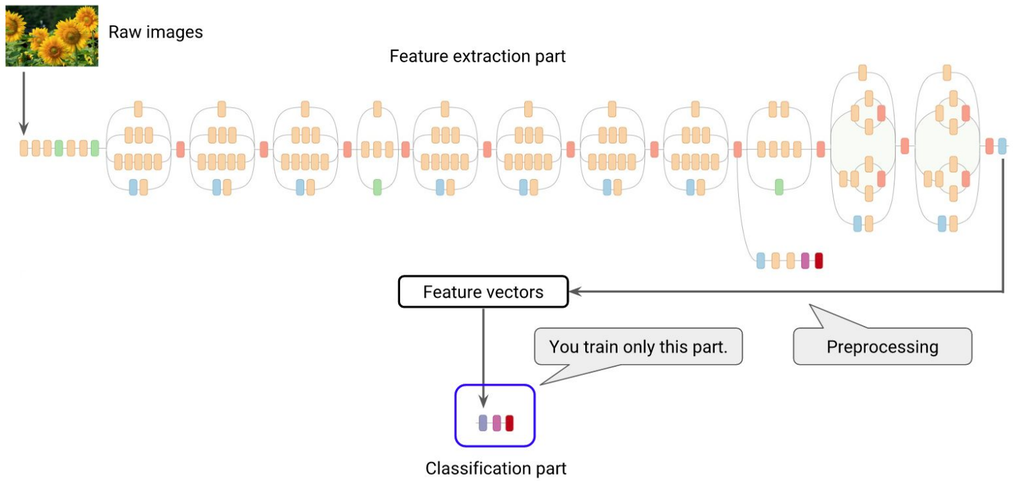
Max Lotstein


Feature
extraction
Classification
head

First, we'll preprocess using Dataflow

```
_ = (p
     | 'Read input' >> read_input_source
     | 'Parse input' >> beam.Map(lambda line:
          csv.reader([line]).next())
     | 'Extract label ids' >>
        beam.ParDo(ExtractLabelIdsDoFn(),
        beam.pvalue.AsIter(labels))
     | 'Read and convert to JPEG'
        >> beam.ParDo(ReadImageAndConvertToJpegDoFn())
     | 'Embed and make TFExample' >>
        beam.ParDo(TFExampleFromImageDoFn())
     | 'SerializeToString' >> beam.Map(lambda x:
        x.SerializeToString())
     | 'Save to disk'
        >> beam.io.WriteToTFRecord(opt.output_path,
        file_name_suffix='.tfrecord.gz'))
```

Then, we'll train the model

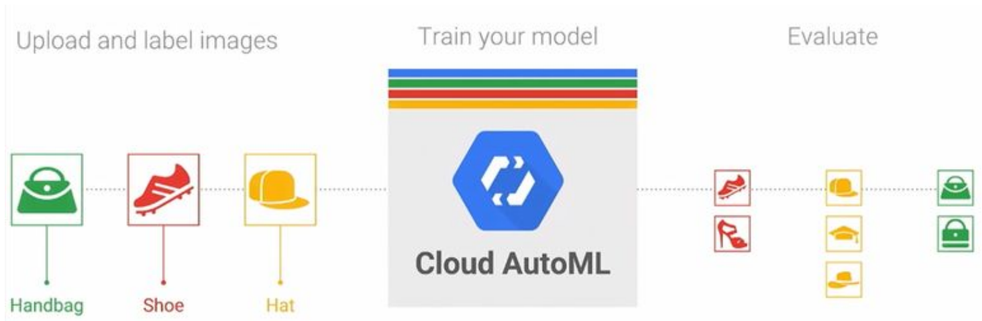


File Name:
T-ICML-O_3_I7_lab_solution:_implementing_transfer_learning

Format: Presenter in Studio

Presenter: Max Lotstein

AutoML is like codeless transfer learning



File Name: T-ICML-O_3_I8_no_data,_no_problem

Format: Presenter in Studio

Presenter: Max Lotstein

Agenda

The Data Scarcity Problem

Data Augmentation

Transfer Learning

No Data, No Problem

No Data, No Problem

1 Use a pre-trained model

Google has cutting-edge machine learning APIs

Cloud
Vision API

Cloud
Speech API

Cloud
Jobs API

Cloud
Translation API

Cloud Natural
Language API

Cloud Video
Intelligence

Image Recognition with the Cloud Vision API

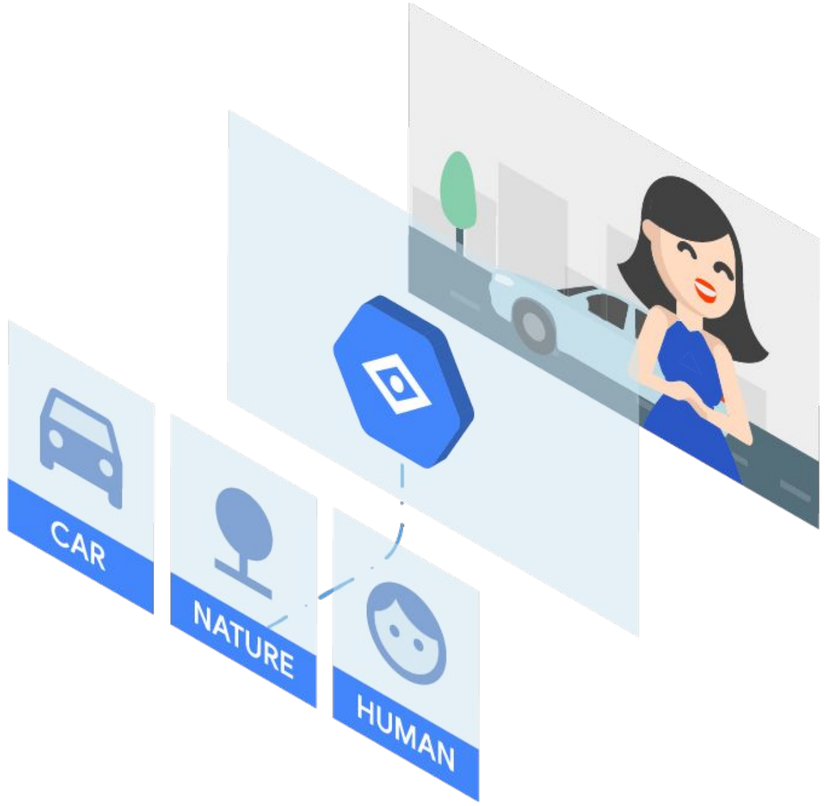
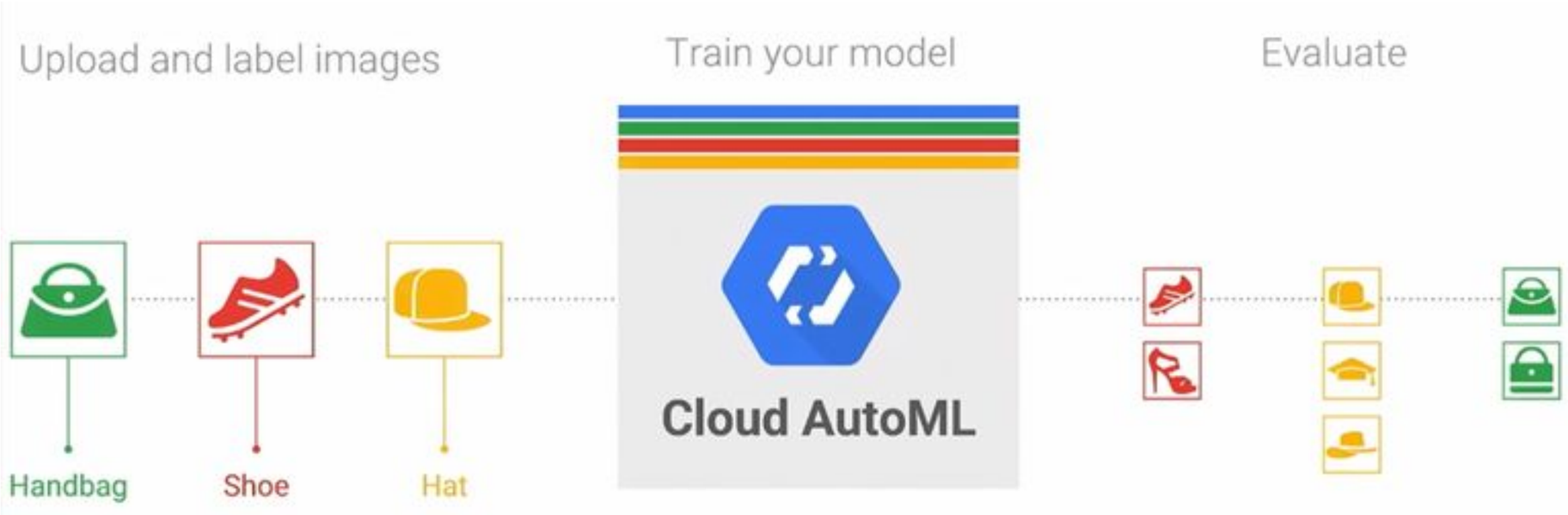


Image Recognition with the Cloud Vision API



No Data, No Problem

1. Use a pre-trained model
2. Find a partner to label your data

Data scarcity is not as bad as you might think

If you have some labeled data:

- Data augmentation helps you get more
- Transfer learning helps you need less
  - AutoML makes transfer learning codeless

Data scarcity is not as bad as you might think

If you have some labeled data:

- Data augmentation helps you get more
- Transfer learning helps you need less
  - AutoML makes transfer learning codeless

If you don’t have any labeled data:

- Machine learning APIs are state-of-the-art
- Partners can help you economically label your data