Session 6: Neural Networks

2021 August 12

Dr. Richard M. Crowley
rcrowley@smu.edu.sg
http://rmc.link/

Main applications

Handwriting recognition

- MNIST: The “Hello World” of neural network design

Dependent Variable

- The number someone wrote

Independent Variables

- An image of that number
  - Treat as a vector
  - Treat as an image

> A simple introduction to building a neural network

Sentence embeddings

- Using Universal Sentence Encoder (USE)
- Try it out yourself!

> USE is available as an off-the-shelf model, which makes it easy to use

Image object detection

Given a random image, can we tell if a person is in it?

- Yes!
  - Using an off-the-shelf model, it can be done quickly and easily

Also, detecting 79 other objects in images

Frameworks for Neural networks

TensorFlow

- It can run almost ANY ML/AI/NN algorithm
- It has APIs for easier access like Keras
- Comparatively easy GPU setup
- It can deploy anywhere
  - Python & C/C++ built in
  - Swift, R Haskell, and Rust bindings
  - TensorFlow light for mobile deployment
  - TensorFlow.js for web deployment

TensorFlow Lite

TensorFlow.js

magenta

TensorFlow Hub

TensorFlow resources

- It has strong support from Google and others
  - TensorFlow Hub – Premade algorithms for text, image, and video
  - tensorflow/models – Premade code examples
    - The research folder contains an amazing set of resources
  - trax – AI research models from Google Brain

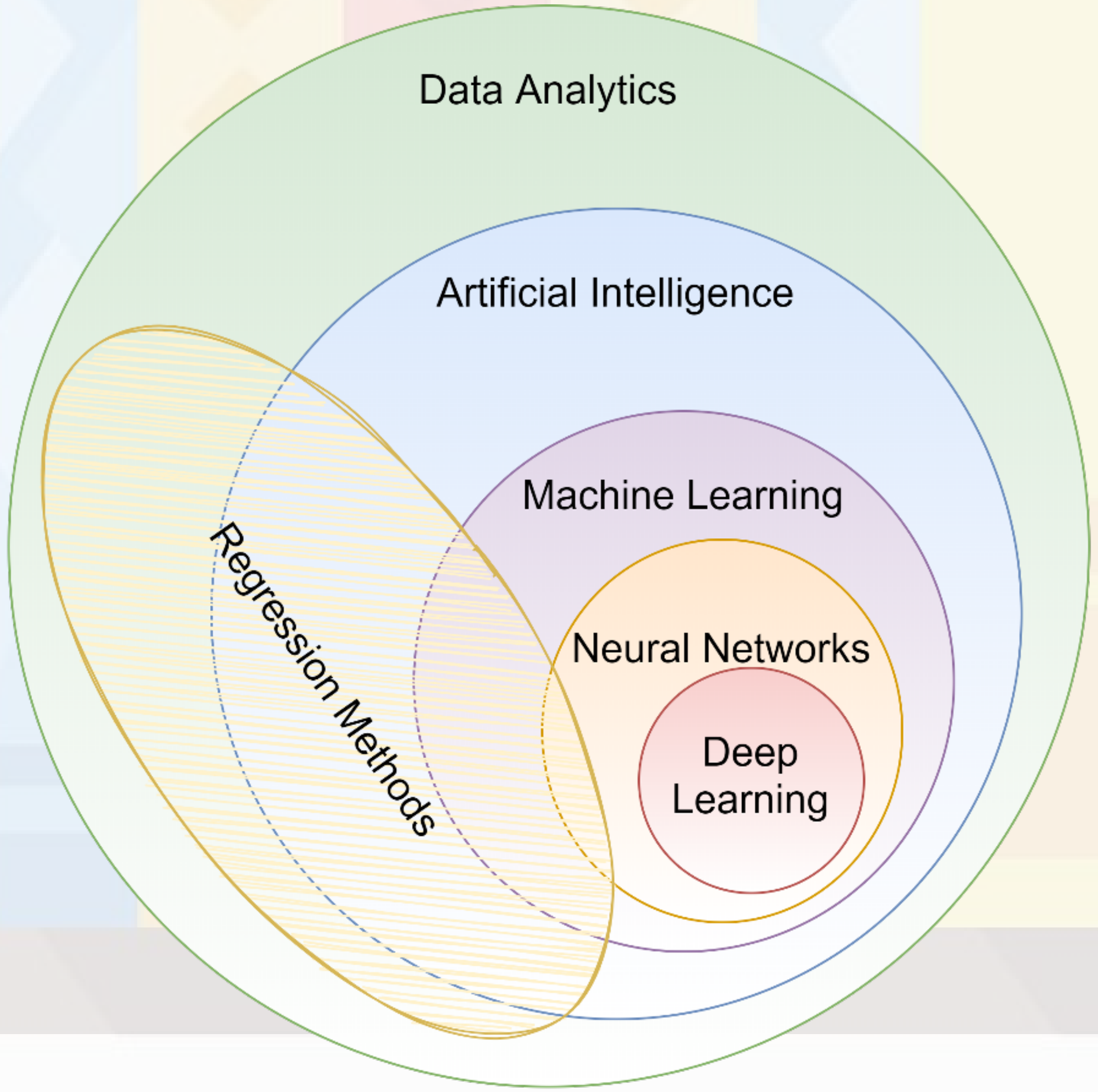
Other notable frameworks

- Caffe
  - Python, C/C++, Matlab
  - Good for image processing
- Caffe2
  - C++ and Python
  - Still largely image oriented
- Microsoft Cognitive Toolkit
  - Python, C++
  - Scales well, good for NLP
- Torch and Pytorch
  - For Lua and python
  - fast.ai, ELF, and AllenNLP
- H20
  - Python based
  - Integration with R, Scala…

Neural Networks

What are neural networks?

- The phrase *neural network* is thrown around almost like a buzz word
- *Neural networks* are actually a specific type class algorithms
  - There are many implementations with different primary uses

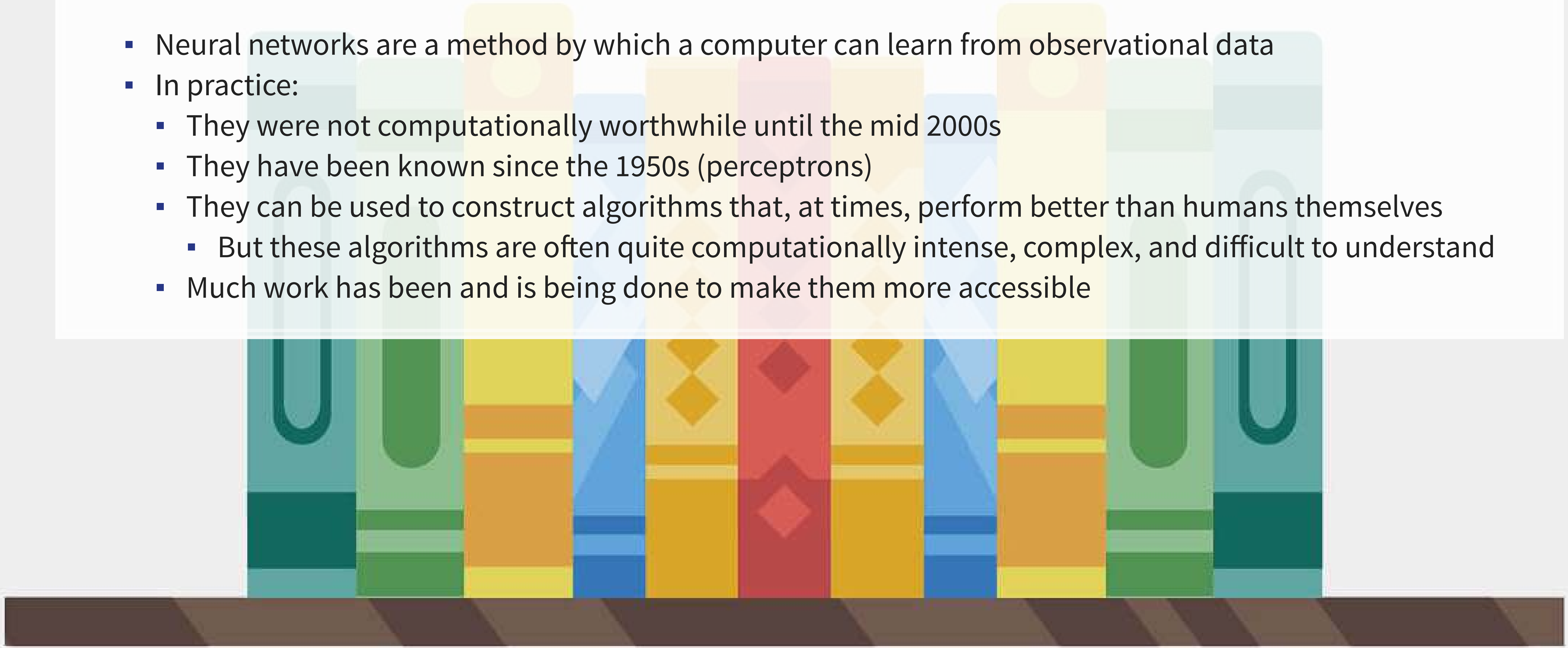


What are neural networks?

- Originally, the goal was to construct an algorithm that behaves like a human brain
  - Thus the name
- Current methods don't quite reflect human brains, however:
  1. We don't fully understand how our brains work, which makes replication rather difficult
  2. Most neural networks are constructed for specialized tasks (not general tasks)
  3. Some (but not all) neural networks use tools our brain may not have
     - I.e., **backpropogation** is potentially possible in brains, but it is not pinned down how such a function occurs (if it does occur)

What are neural networks?

- Neural networks are a method by which a computer can learn from observational data
- In practice:
  - They were not computationally worthwhile until the mid 2000s
  - They have been known since the 1950s (perceptrons)
  - They can be used to construct algorithms that, at times, perform better than humans themselves
    - But these algorithms are often quite computationally intense, complex, and difficult to understand
  - Much work has been and is being done to make them more accessible

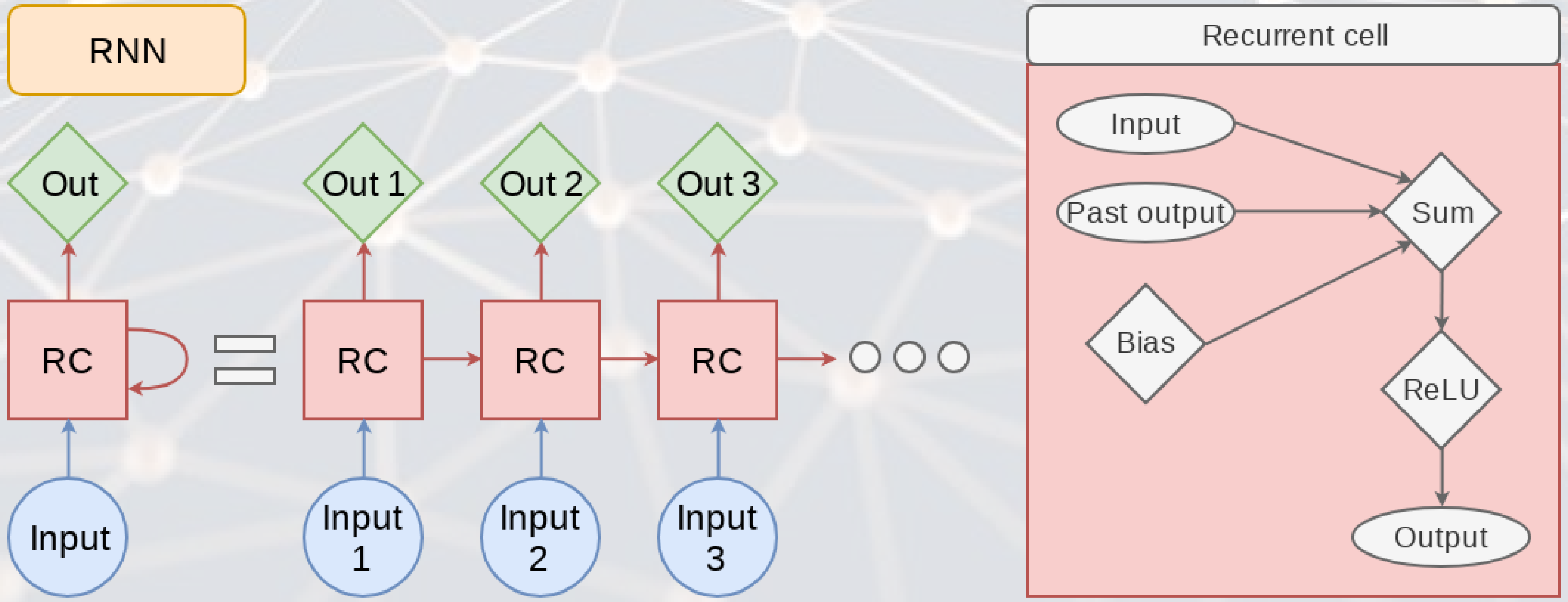


Types of neural networks

- There are *a lot* of neural network types
  - See The "Neural Network Zoo"
- Some of the more interesting ones which we will see or have seen:
  - RNN: Recurrent Neural Network
  - LSTM: Long/Short Term Memory
  - CNN: Convolutional Neural Network
  - DAN: Deep Averaging Network
  - GAN: Generative Adversarial Network
- Others worth noting
  - VAE (Variational Autoencoder): Generating *new* data from datasets
- Not in the Zoo, but of note:
  - Transformer: Networks with "attention"
    - From Attention is All You Need

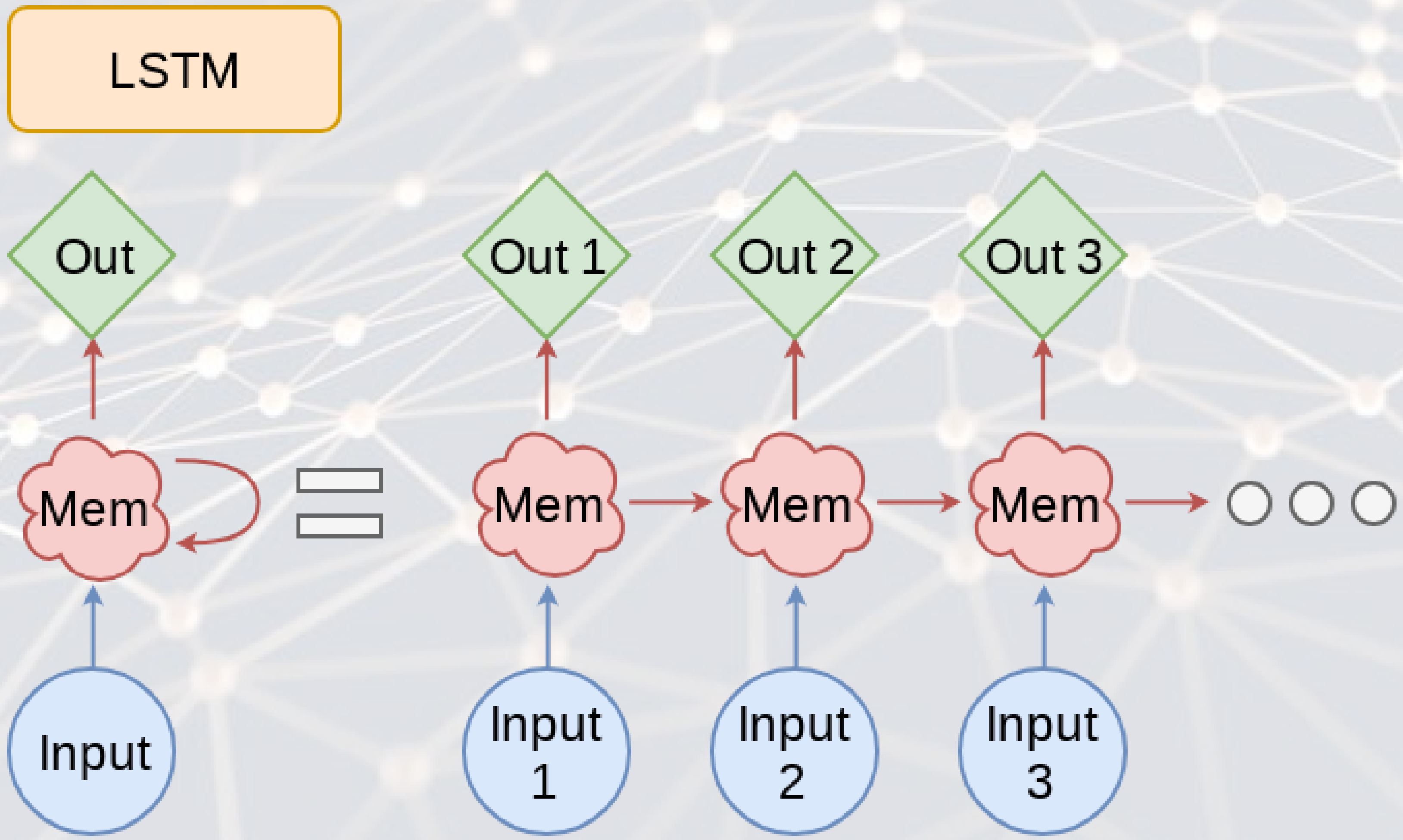
RNN: Recurrent NN

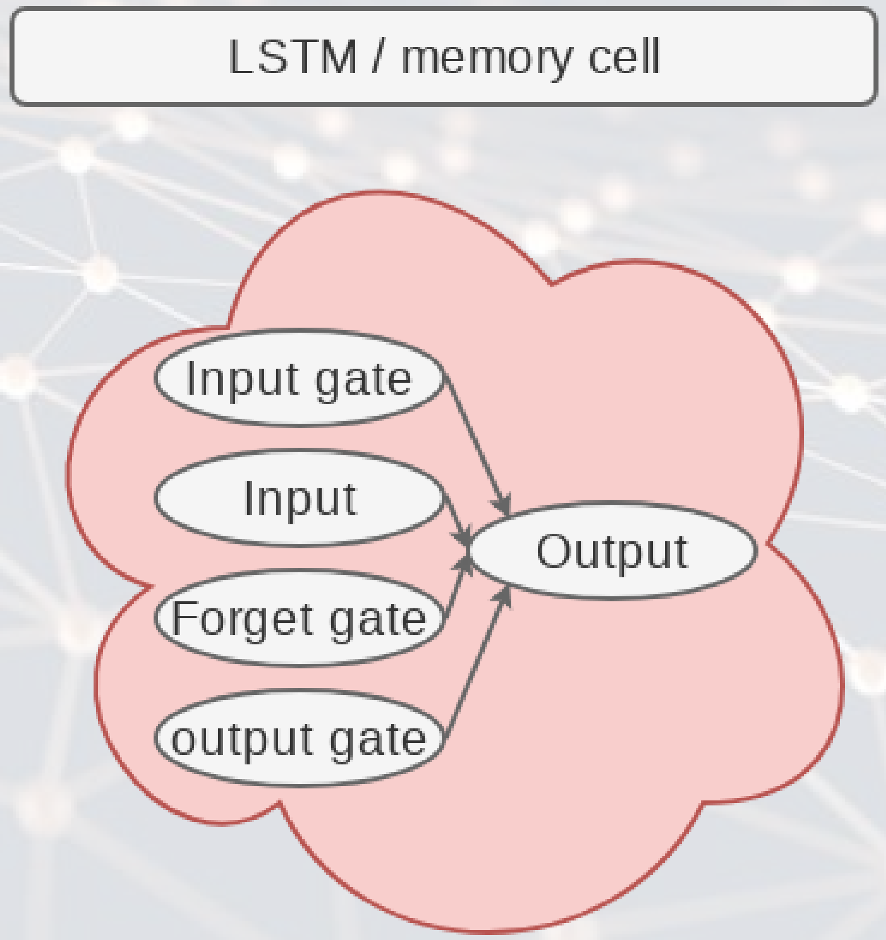
- Recurrent neural networks embed a history of information in the network
  - The previous computation affects the next one
  - Leads to a *short term memory*
- Used for speech recognition, image captioning, anomaly detection, and many others
  - Also the foundation of LSTM
  - SketchRNN (live demo)



LSTM: Long Short Term Memory

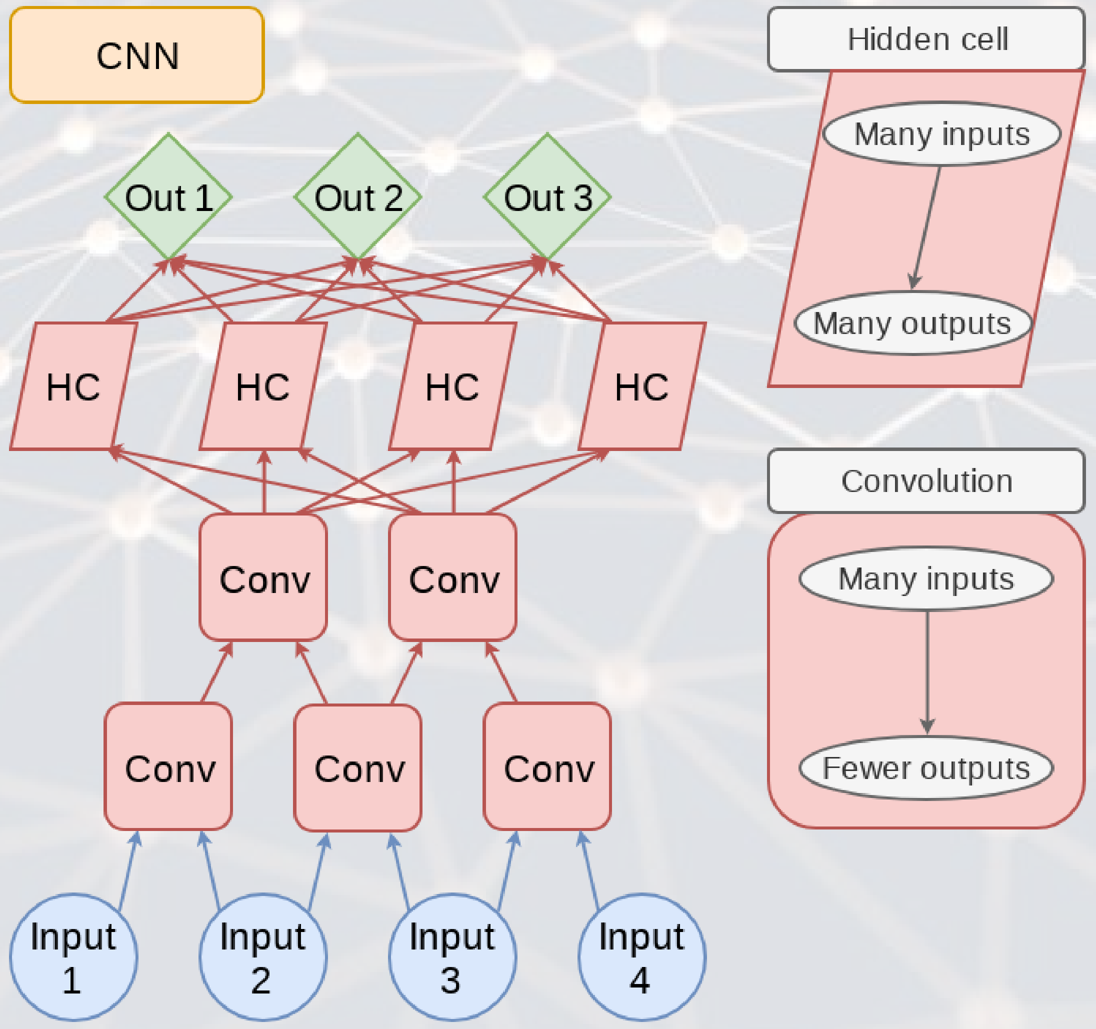
- LSTM improves the *long term memory* of the network while explicitly modeling a *short term memory*
- Used wherever RNNs are used, and then some
  - Ex.: Seq2seq (machine translation)





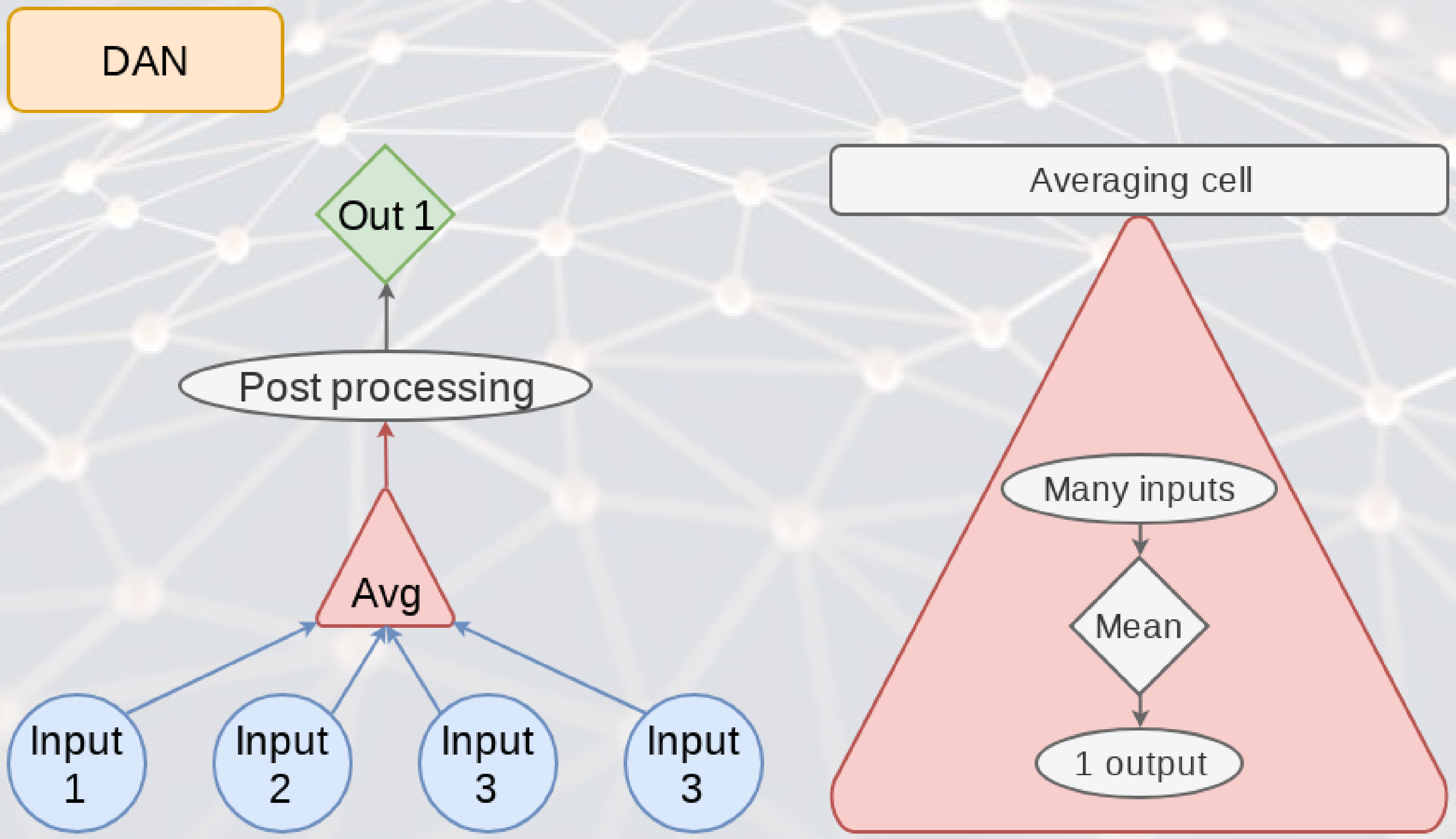
CNN: Convolutional NN

- Networks that excel at object detection (in images)
- Can be applied to other data as well
- Ex.: Inception-v3



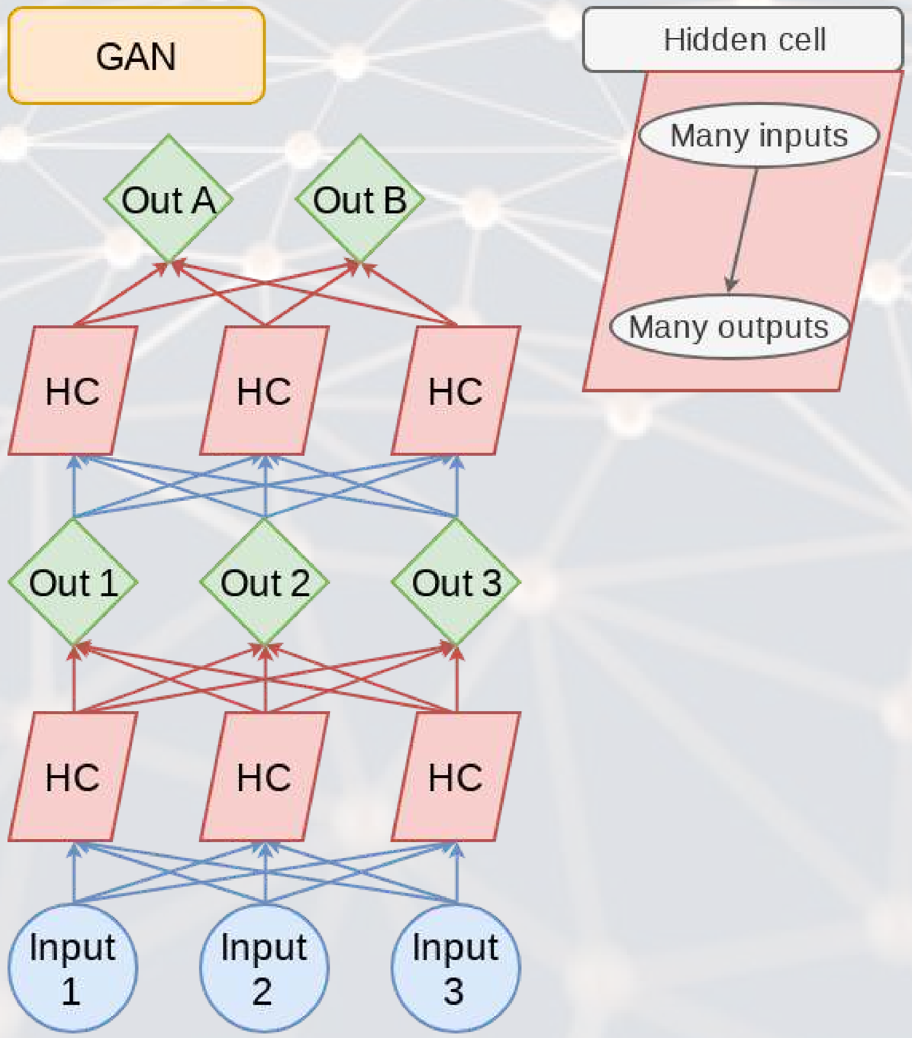
DAN: Deep Averaging Network

- DANs are simple networks that simply average their inputs
- Averaged inputs are then processed a few times
- These networks have found a home in NLP
  - Ex.: Universal Sentence Encoder



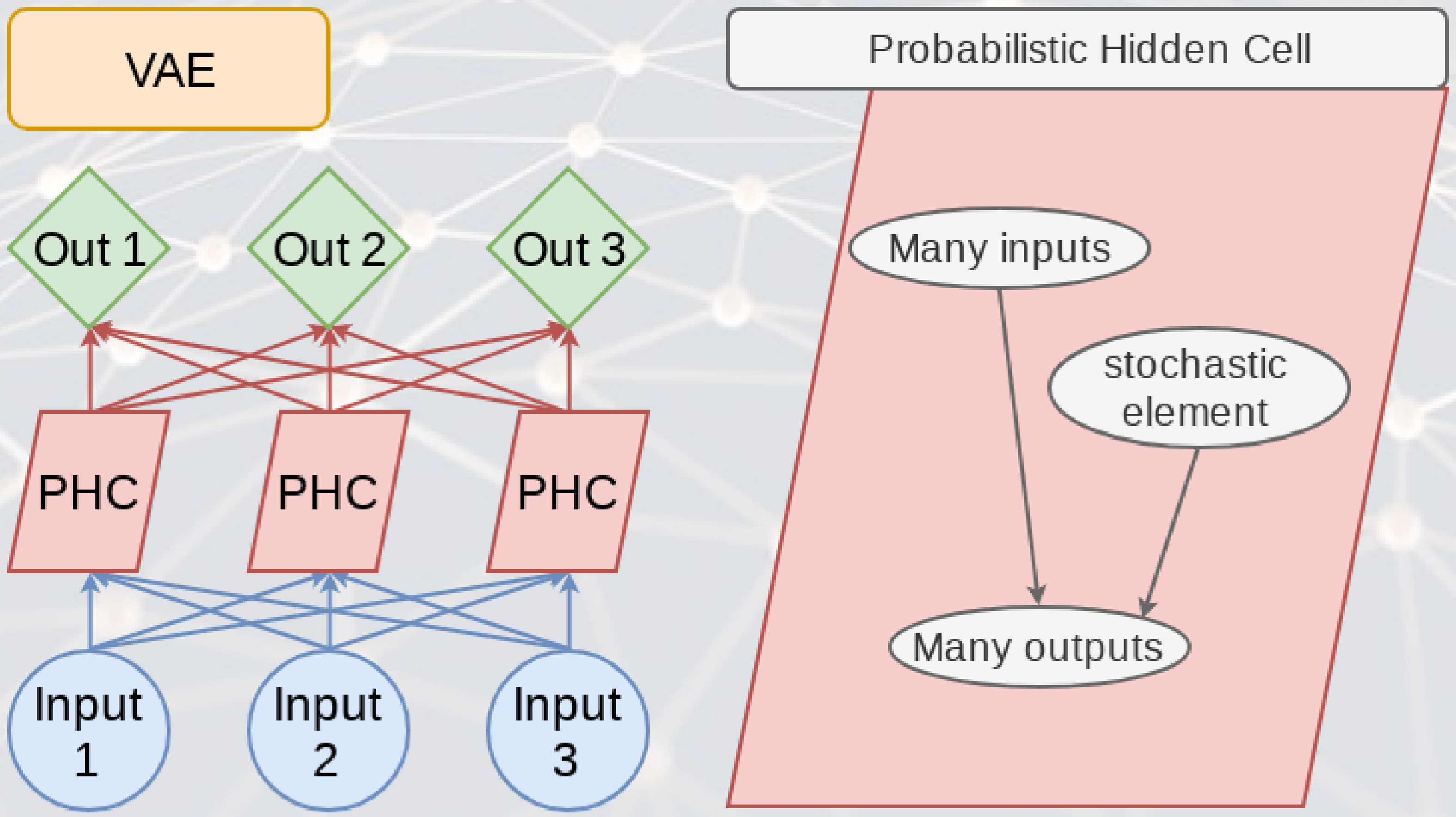
GAN: Generative Adversarial Network

- Feature two networks working against each other
- Many novel uses
  - Ex.: Anonymizing clinical trial data by simulating an attack on the dataset
  - Ex.: Aging images



VAE: Variational Autoencoder

- An autoencoder (AE) is an algorithm that can recreate input data
- Variational means this type of AE can vary other aspects to generate completely new output
  - Good for creating fake data
- Like a simpler, noisier GAN



Transformer

- Shares some similarities with RNN and LSTM: Focuses on attention
- Currently being applied to solve many types of problems
- Examples: BERT, GPT-3, XLNEt

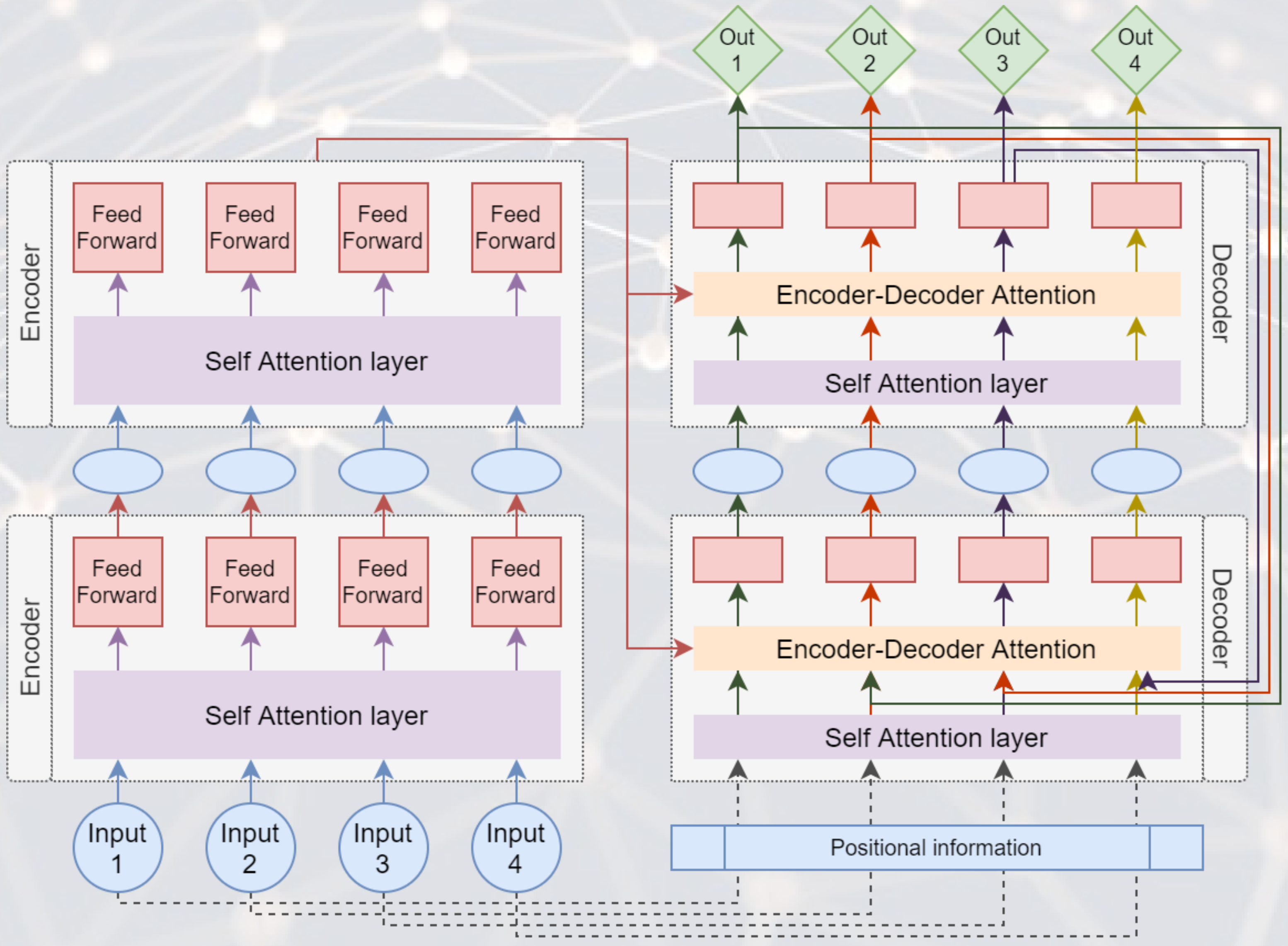


Image data

Thinking about images as data

- Images **are** data, but they are very unstructured
  - No instructions to say what is in them
  - No common grammar across images
  - Many, many possible subjects, objects, styles, etc.
- From a computer's perspective, images are just 3-dimensional matrices
  - Rows (pixels)
  - Columns (pixels)
  - Color channels (usually Red, Green, and Blue)

Using images as data

- We can definitely use numeric matrices as data
  - We did this plenty with XGBoost, for instance
- However, images have a lot of different numbers tied to each observation.

- Source: Twitter

- 798 rows
- 1200 columns
- 3 color channels
- $798 \times 1{,}200 \times 3 = 2{,}872{,}800$
  - The number of 'variables' per image like this!

Using images in practice

- There are a number of strategies to shrink images' dimensionality
  1. Downsample the image to a smaller resolution like 256x256x3
  2. Convert to grayscale
  3. Cut the image up and use sections of the image as variables instead of individual numbers in the matrix
     - Often done with convolutions in neural networks
  4. Drop variables that aren't needed, like LASSO

A simple example: MNIST

MNIST

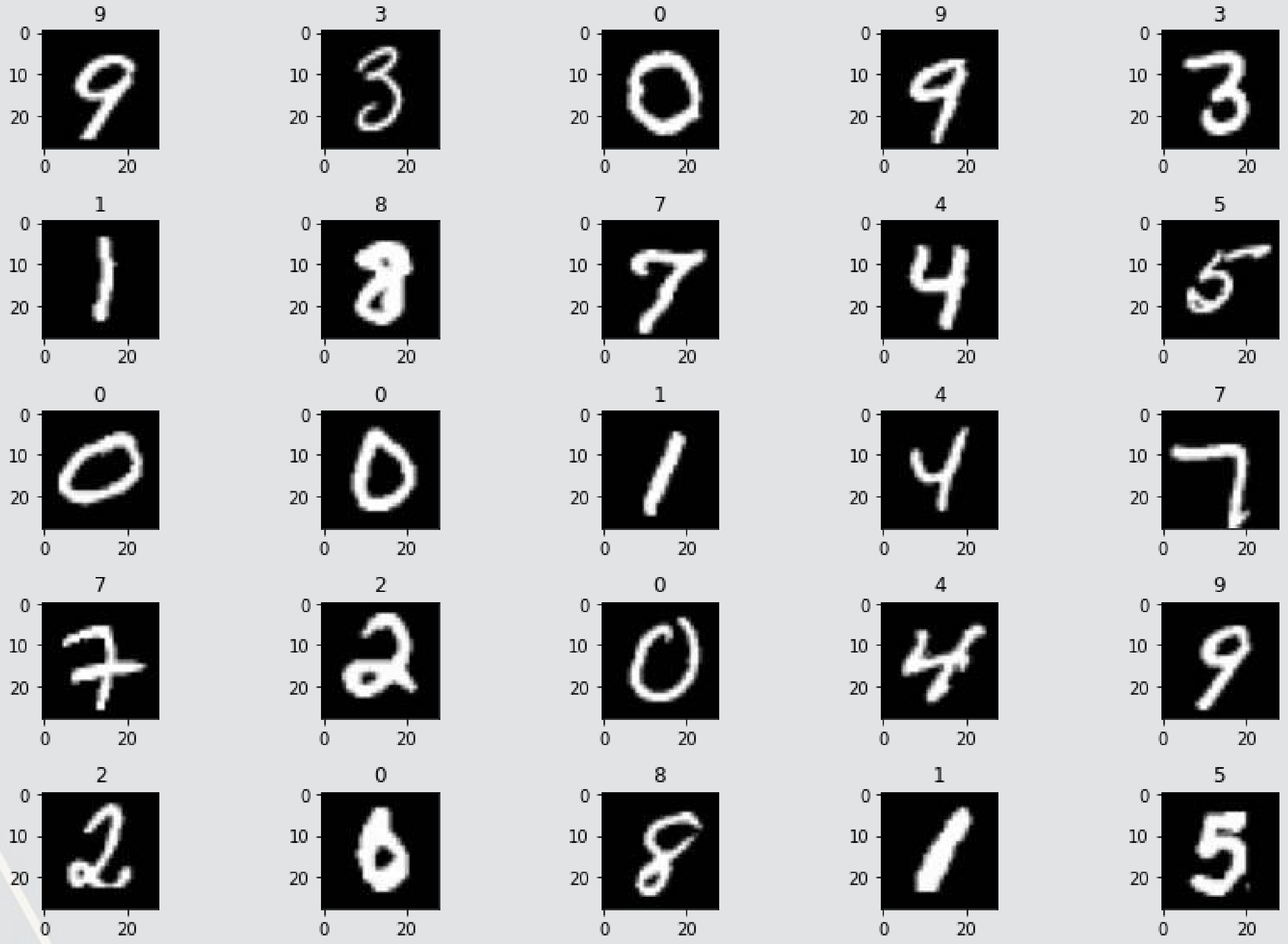
- MINST is a set of handwritten numbers with annotations
  - It has prespecified training and testing samples
    - Ensures comparability
    - 60,000 for training, 10,000 for testing
- It's available in `tensorflow`, so we will import from there

```python
(train_X, train_Y), (test_X, test_Y) = keras.datasets.mnist.load_data()

print('Train, X:%s, Y:%s' % (train_X.shape, train_Y.shape))
print('Test, X:%s, Y:%s' % (test_X.shape, test_Y.shape))
```

```
## Train, X:(60000, 28, 28), Y:(60000,)
## Test, X:(10000, 28, 28), Y:(10000,)
```

A look at the MNIST data

```python
images = np.random.randint(0, train_X.shape[0], size=25)
for i in range(0, 25):
    # define subplot
    image = images[i]
    plt.subplot(5, 5, i+1)
    # plot raw pixel data
    plt.imshow(train_X[image], cmap=plt.get_cmap('gray'))
    plt.title(train_Y[image])
plt.tight_layout()
```



Simple neural network

- We will ignore the 2D nature of the image – instead, we will treat it as a vector of values between 0 and 1
- To do this, we need to…
  1. Scale by 255 (the max value in the data/
  2. Reshape our data into vectors

```python
# Scale data
train_X = train_X.astype("float32") / 255
test_X  = test_X.astype("float32") / 255

# convert to vectors
rows = train_X.shape[0]
dim1 = train_X.shape[1]
dim2 = train_X.shape[2]
train_X = train_X.reshape((rows, dim1 * dim2))
rows = test_X.shape[0]
test_X = test_X.reshape((rows, dim1 * dim2))

print('Train, X:%s, Y:%s' % (train_X.shape, train_Y.shape))
print('Test, X:%s, Y:%s' % (test_X.shape, test_Y.shape))
```

```
## Train, X:(60000, 784), Y:(60000,)
## Test, X:(10000, 784), Y:(10000,)
```

Dealing with categorical DVs

- We need to take special care that the Y values are interpreted as categories
  - Otherwise, the default behavior would be to treat them as a continuous numeric measure
- We can use `keras.utils.to_categorical` to convert our data into the right format

```
train_Y = keras.utils.to_categorical(train_Y, 10)
test_Y = keras.utils.to_categorical(test_Y, 10)

print('Train, X:%s, Y:%s' % (train_X.shape, train_Y.shape))
print('Test, X:%s, Y:%s' % (test_X.shape, test_Y.shape))
```

```
## Train, X:(60000, 784), Y:(60000, 10)
## Test, X:(10000, 784), Y:(10000, 10)
```

Note that Y is now 10-dimensional – it is one hot encoded now

Constructing a simple neural network

- This model is a very simplistic algorithm
- The data streams in as 784-dim vectors (`InputLayer`)
- The data is compressed by 10 fully-connected neurons all in the same layer (`Dense`)
  - Each neuron will take on one category to try to pick up
- The highest probability neuron will be the category guess (`softmax`)

```python
# Parameters for the model
num_classes = 10
input_shape = (784)

model_dense = keras.Sequential(
    [
        keras.layers.InputLayer(input_shape=input_shape),
        keras.layers.Dense(num_classes, activation="softmax")
    ]
)

model_dense.summary()
```

```
## Model: "sequential_3"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_4 (Dense)              (None, 10)                7850
## =================================================================
## Total params: 7,850
## Trainable params: 7,850
## Non-trainable params: 0
## _________________________________________________________________
```

Run the neural network

- There are 2 steps to running a neural network:
  1. Compile the model: We previously described the network shape, but didn't build the network itself
  2. Fit the model to our data
- The `loss` function tells the model what to optimize in training
  - `categorical_crossentropy` corresponds to multiclass classification accuracy
- The `optimizer` is the function used for training the model – `adam` is a good default
- Metrics are what you want it to track and report back to you
- Within the fit command, note that `epochs` is the number of rounds to train the model
  - Higher is often better, but not always
- The model itself runs quickly

```python
batch_size = 128
epochs = 10

model_dense.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
history = model_dense.fit(train_X, train_Y, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

Model performance

- The model we compiled is 92.45% accurate in-sample, with 93.78% accuracy on validation data
- However, what matters most is the accuracy on the testing data
  - `model.evaluate()` will test this for us

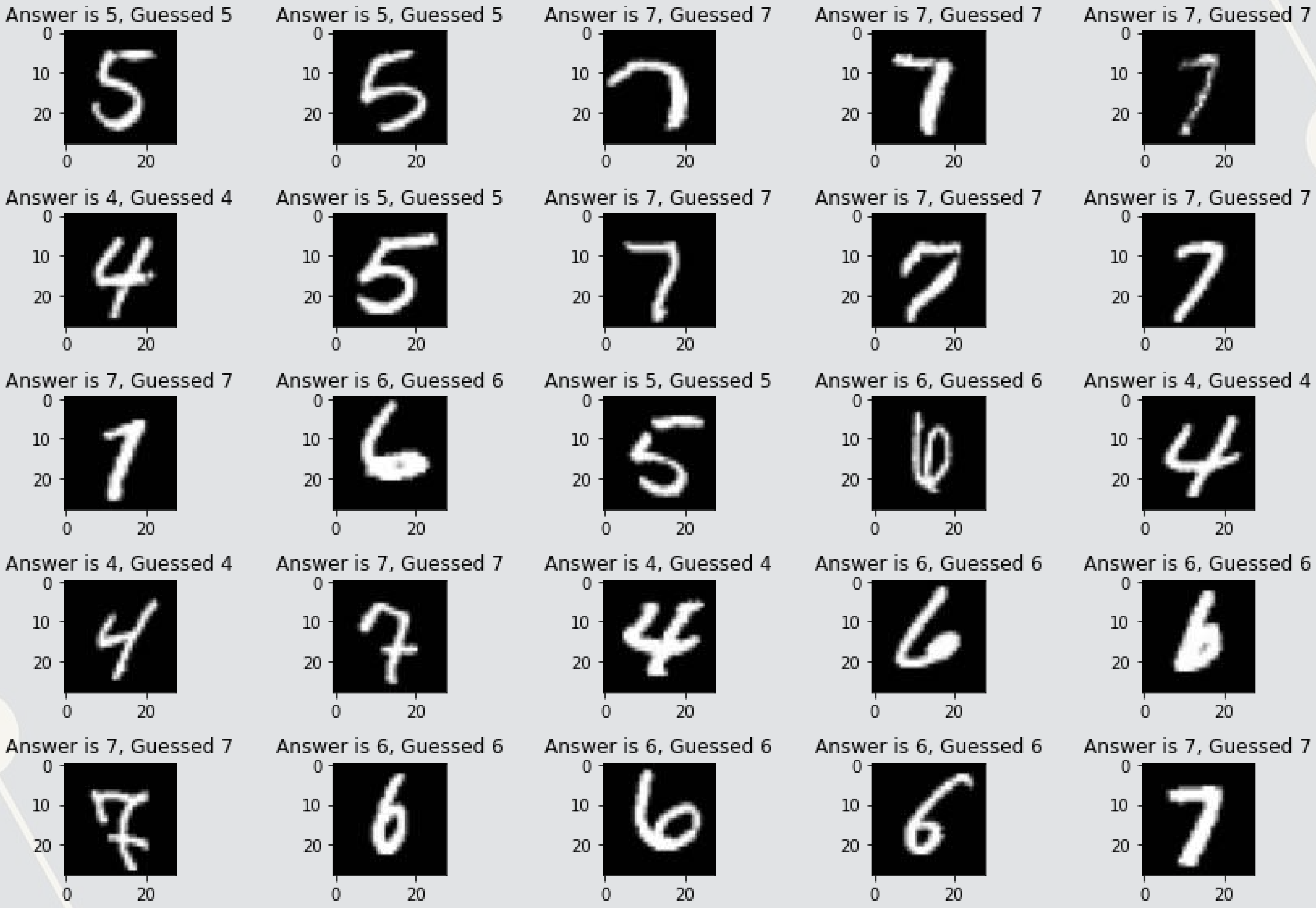
```python
score = model_dense.evaluate(test_X, test_Y, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
## Test loss: 0.26733914017677307
## Test accuracy: 0.9259999990463257
```

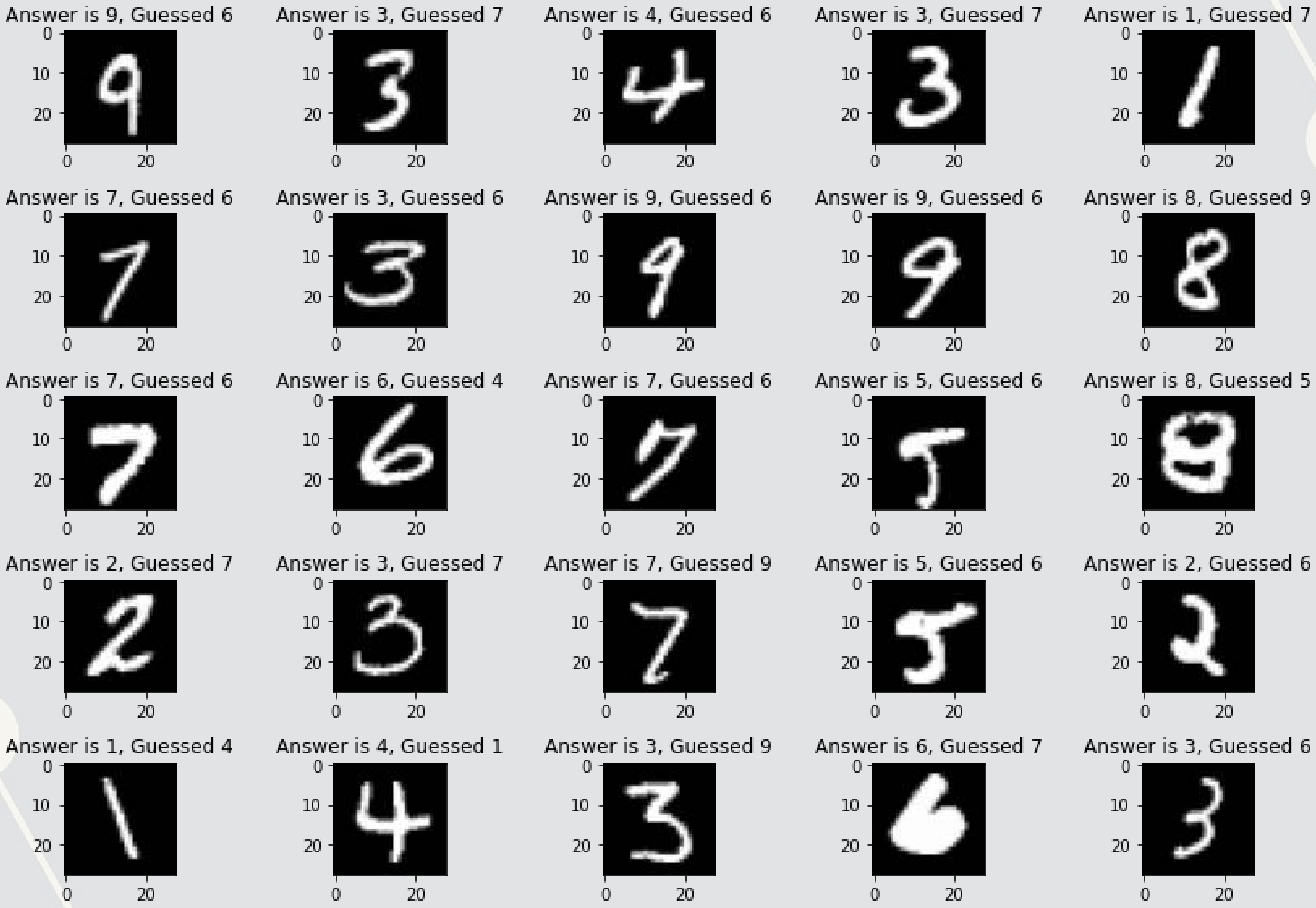
- We will also make lists of what it got right and wrong

```python
correct = np.where(np.argmax(model_dense.predict(test_X), axis=-1) == np.argmax(test_Y, axis=-1))[0]
incorrect = np.where(np.argmax(model_dense.predict(test_X), axis=-1) != np.argmax(test_Y, axis=-1))[0]
```

What does the model get right?



What does the model get wrong?



Addendum: Using R

> By R Studio: details here

- There is a port of `keras` for R made by the RStudio team
  - It calls TensorFlow in python, however
- Install with: `devtools::install_github("rstudio/keras")`
- Finish the install in one of two ways:

For those using Conda

- CPU Based, works on *any* computer

```r
library(keras)
install_keras()
```

- Nvidia GPU based
  - Install the Software requirements first

```r
library(keras)
install_keras(tensorflow = "gpu")
```

Using your own python setup

- Follow Google's install instructions for TensorFlow
- Install keras from a terminal with `pip install keras`
- R Studio's keras package will automatically find it
  - May require a reboot to work on Windows

MNIST: Extending to a CNN

Setup

- The setup is similar, except we don’t need to reshape our X data
- We do need to add an additional dimension to our images though, which `np.expand_dims()` does for us

```python
(train_X, train_Y), (test_X, test_Y) = keras.datasets.mnist.load_data()

train_X = train_X.astype("float32") / 255
test_X  = test_X.astype("float32") / 255

train_X = np.expand_dims(train_X, -1)
test_X = np.expand_dims(test_X, -1)

train_Y = keras.utils.to_categorical(train_Y, 10)
test_Y = keras.utils.to_categorical(test_Y, 10)

print('Train, X:%s, Y:%s' % (train_X.shape, train_Y.shape))
print('Test, X:%s, Y:%s' % (test_X.shape, test_Y.shape))
```

```
## Train, X:(60000, 28, 28, 1), Y:(60000, 10)
## Test, X:(10000, 28, 28, 1), Y:(10000, 10)
```

Build the model

- Here we use `Conv2D()` layers for the convolution
- The `MaxPooling2D()` layers downsample (shrink) the data
- The `Flatten()` layer reshapes the output to a vector
- `Relu` is essentially the same as a call option payoff ("hockey stick")

```python
# Parameters for the model
num_classes = 10
input_shape = (28, 28, 1)

model_cnn = keras.Sequential(
    [
        keras.layers.InputLayer(input_shape=input_shape),
        keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Flatten(),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(num_classes, activation="softmax"),
    ]
)

model_cnn.summary()
```

Build the model

```
## Model: "sequential_4"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## conv2d_2 (Conv2D)            (None, 26, 26, 32)        320
## _________________________________________________________________
## max_pooling2d_2 (MaxPooling2 (None, 13, 13, 32)        0
## _________________________________________________________________
## conv2d_3 (Conv2D)            (None, 11, 11, 64)        18496
## _________________________________________________________________
## max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
## _________________________________________________________________
## flatten_1 (Flatten)          (None, 1600)              0
## _________________________________________________________________
## dropout_2 (Dropout)          (None, 1600)              0
## _________________________________________________________________
## dense_5 (Dense)              (None, 10)                16010
## =================================================================
## Total params: 34,826
## Trainable params: 34,826
## Non-trainable params: 0
```

Fit the model and evaluate

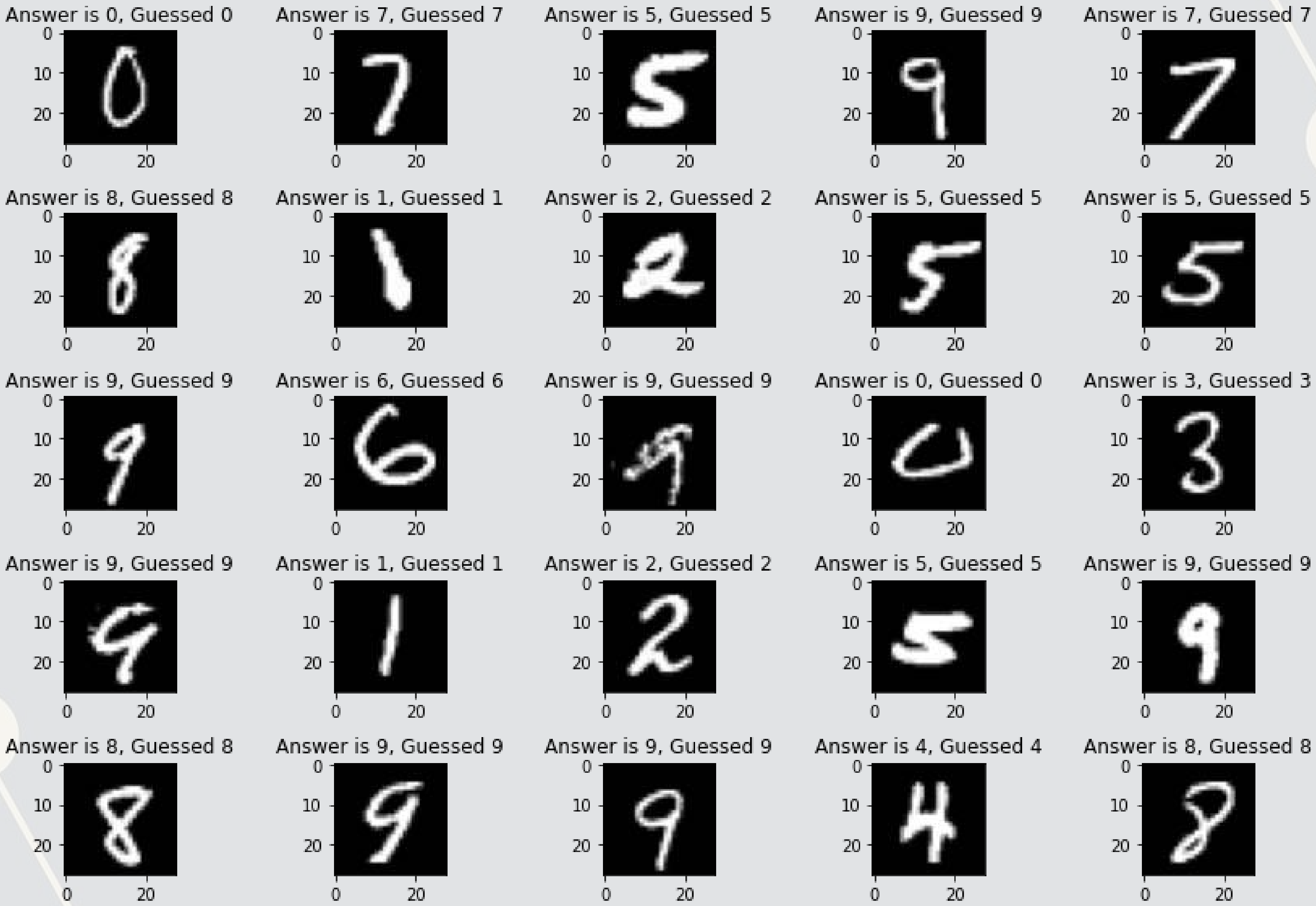
- Fitting and evaluating is the same as before

```python
batch_size = 128
epochs = 10

model_cnn.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
model_cnn.fit(train_X, train_Y, batch_size=batch_size, epochs=epochs, validation_split=0.1)

score = model_cnn.evaluate(test_X, test_Y, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

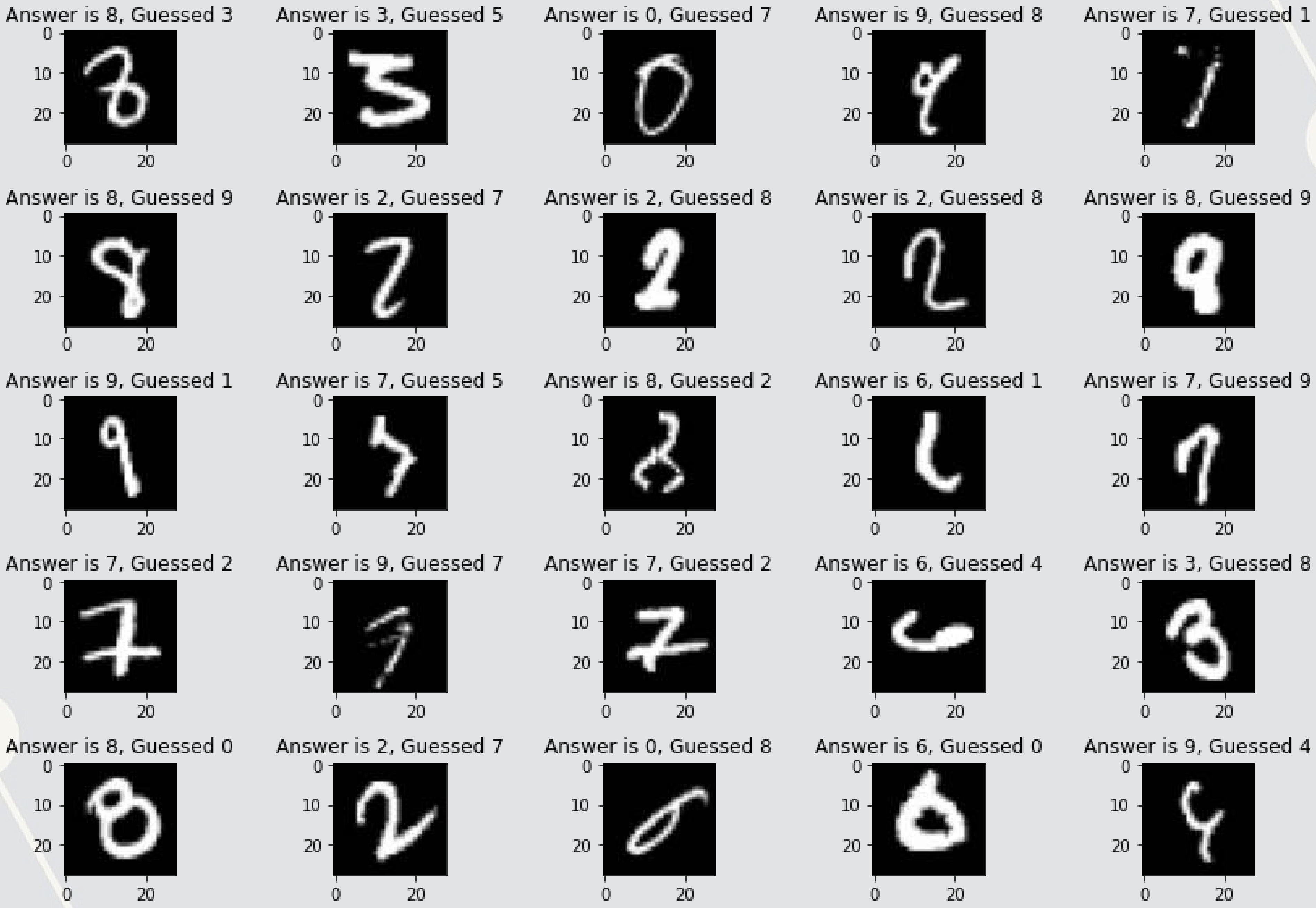
```python
print("""Test loss: 0.0291274506598711
Test accuracy: 0.9897000193595886""")
```

```
## Test loss: 0.0291274506598711
## Test accuracy: 0.9897000193595886
```

What does the model get right?



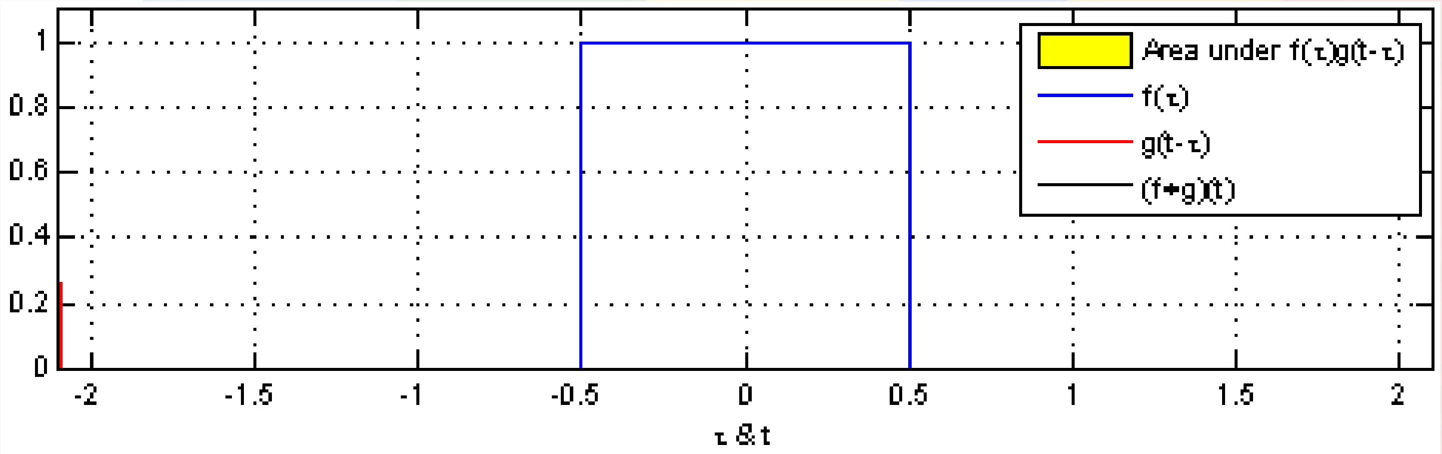
What does the model get wrong?

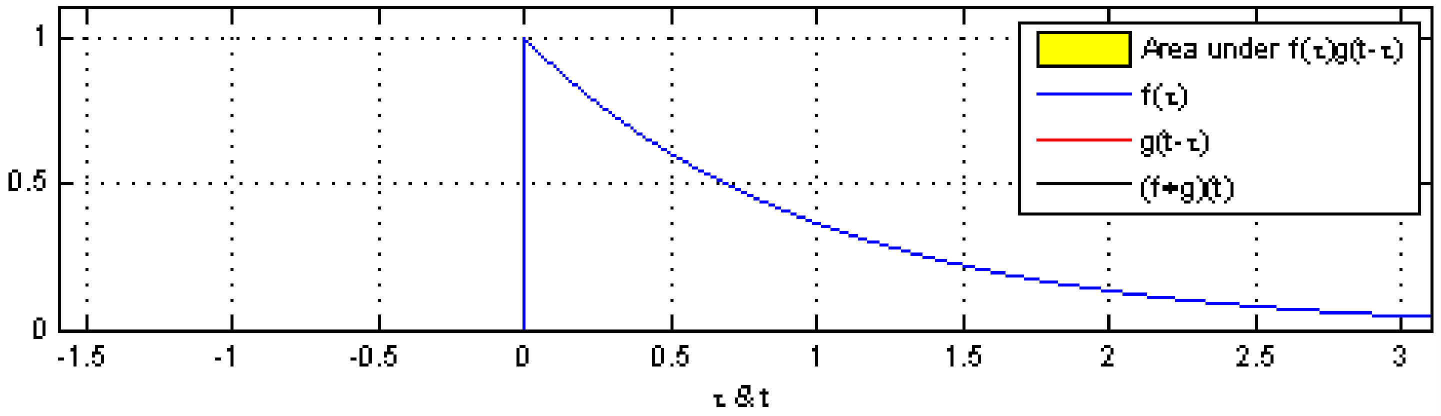


More advanced image techniques

How CNNs work

- CNNs use repeated convolution, usually looking at slightly bigger chunks of data each iteration
- But what is convolution? It is illustrated by the following graphs (from Wikipedia):



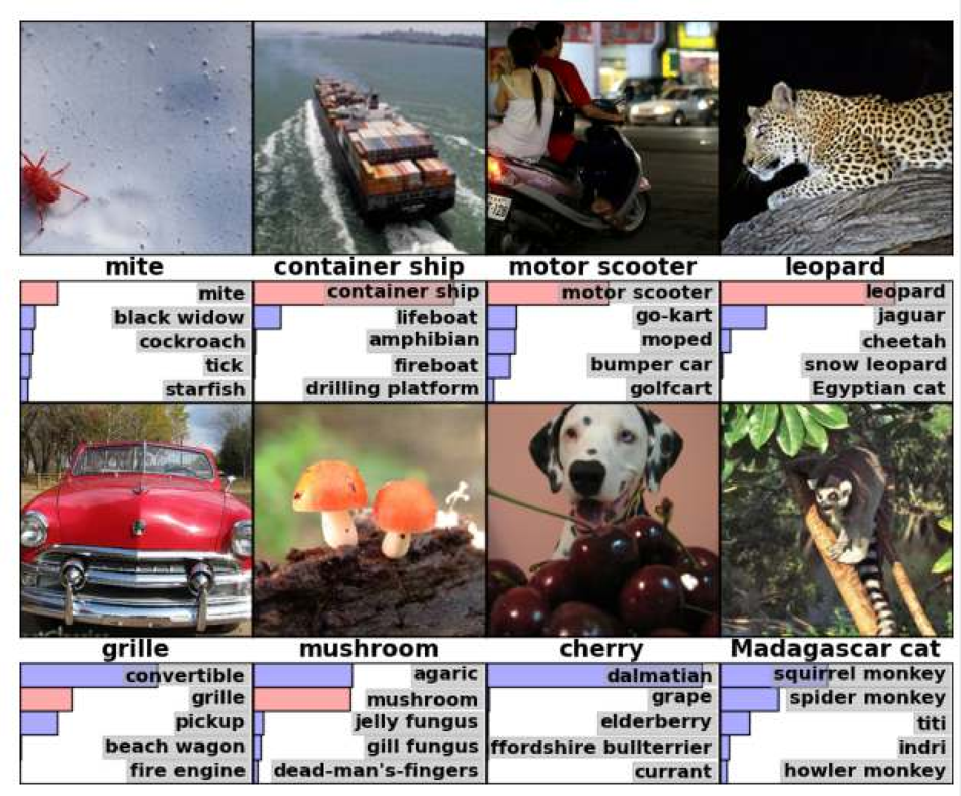


Further reading

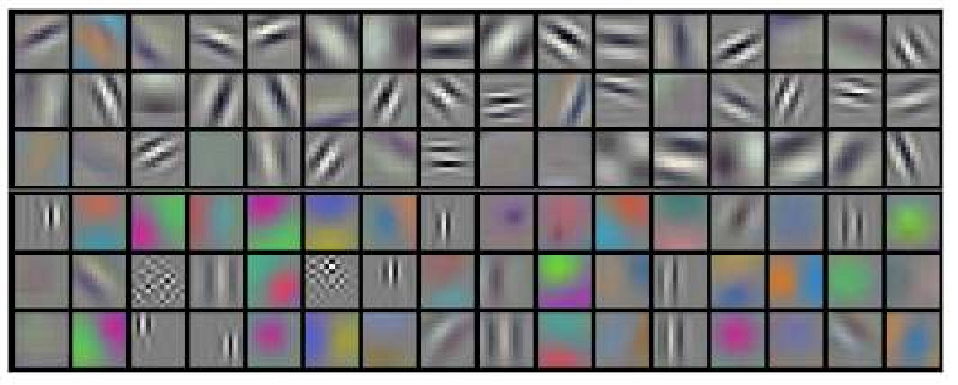
CNN

- AlexNet (paper)

Example output of AlexNet



The first (of 5) layers learned



- AlexNet ([illegible])

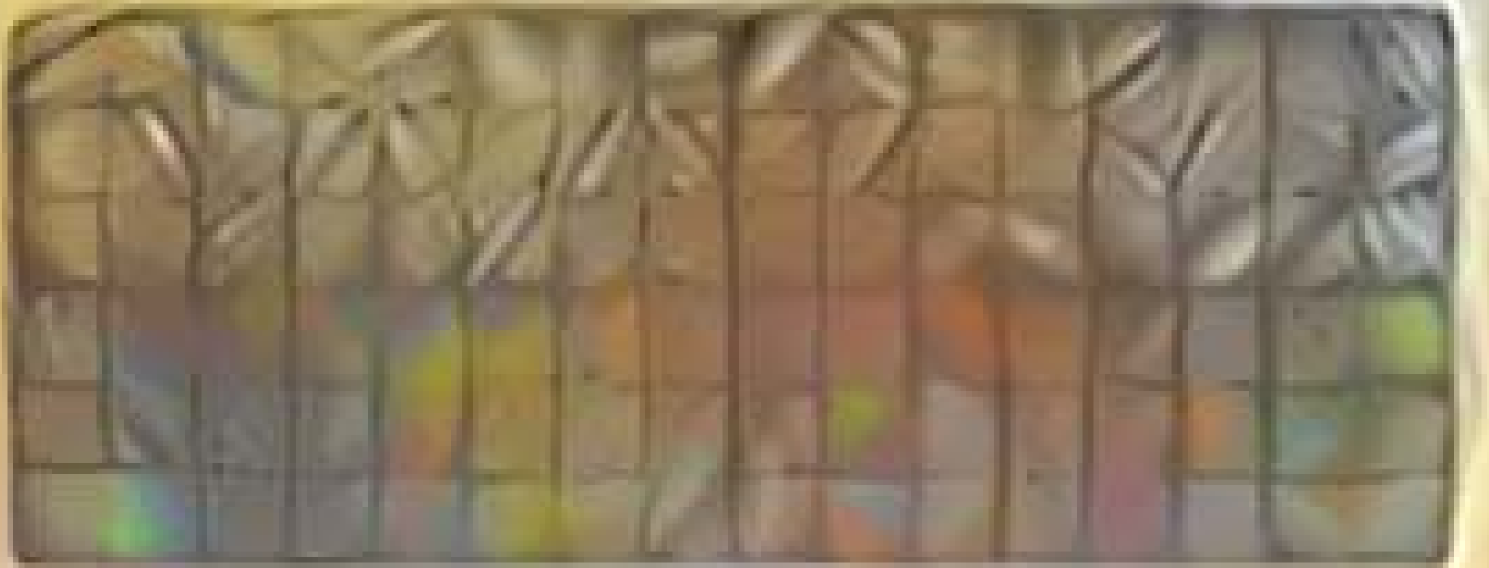
Example output of AlexNet

The first [illegible] layers learned

- AlexNet (paper)

Sample output of AlexNet

The first (of 5) layers learned



Transfer Learning

- The previous slide is an example of *style transfer*
- This is also done using CNNs
- More details here

What is transfer learning?

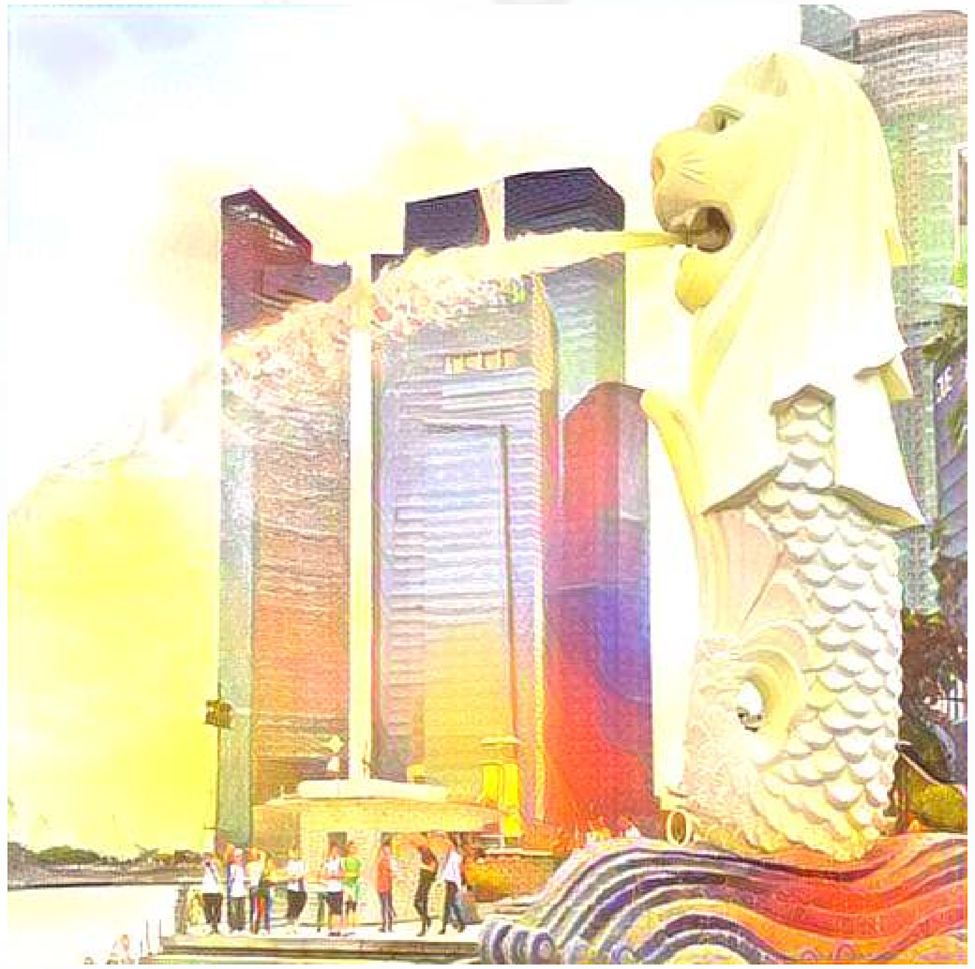
- It is a method of training an algorithm on one domain and then applying the algorithm on another domain
- It is useful when…
  - You don’t have enough data for your primary task
    - And you have enough for a related task
  - You want to augment a model with even more

> There are a couple papers using this for BERT models in accounting

If you want to try it out…

- Colab file available at this link
  - Largely based off of dsgiitr/Neural-Style-Transfer
  - It just took a few tweaks to get it working in a Google Colaboratory environment properly

Inputs:



Recent attempts at explaining CNNs

- Google & Stanford's "Automated Concept-based Explanation"

Figure 2: **The output of ACE for three ImageNet classes.** Here we depict three randomly selected examples of the top-4 important concepts of each class (each example is shown above the original image it was segmented from). Using this result, for instance, we could see that the network classifies police vans using the van's tire and the police logo.

Explaining a CNN with SHAP

SHAP and TensorFlow

- Recall that Wich, Bauer and Groh (2020 WOAH) used `shap.DeepExplainer()` to analyze a neural network
  - We can do the same!
- First, feed SHAP the model and some sample images

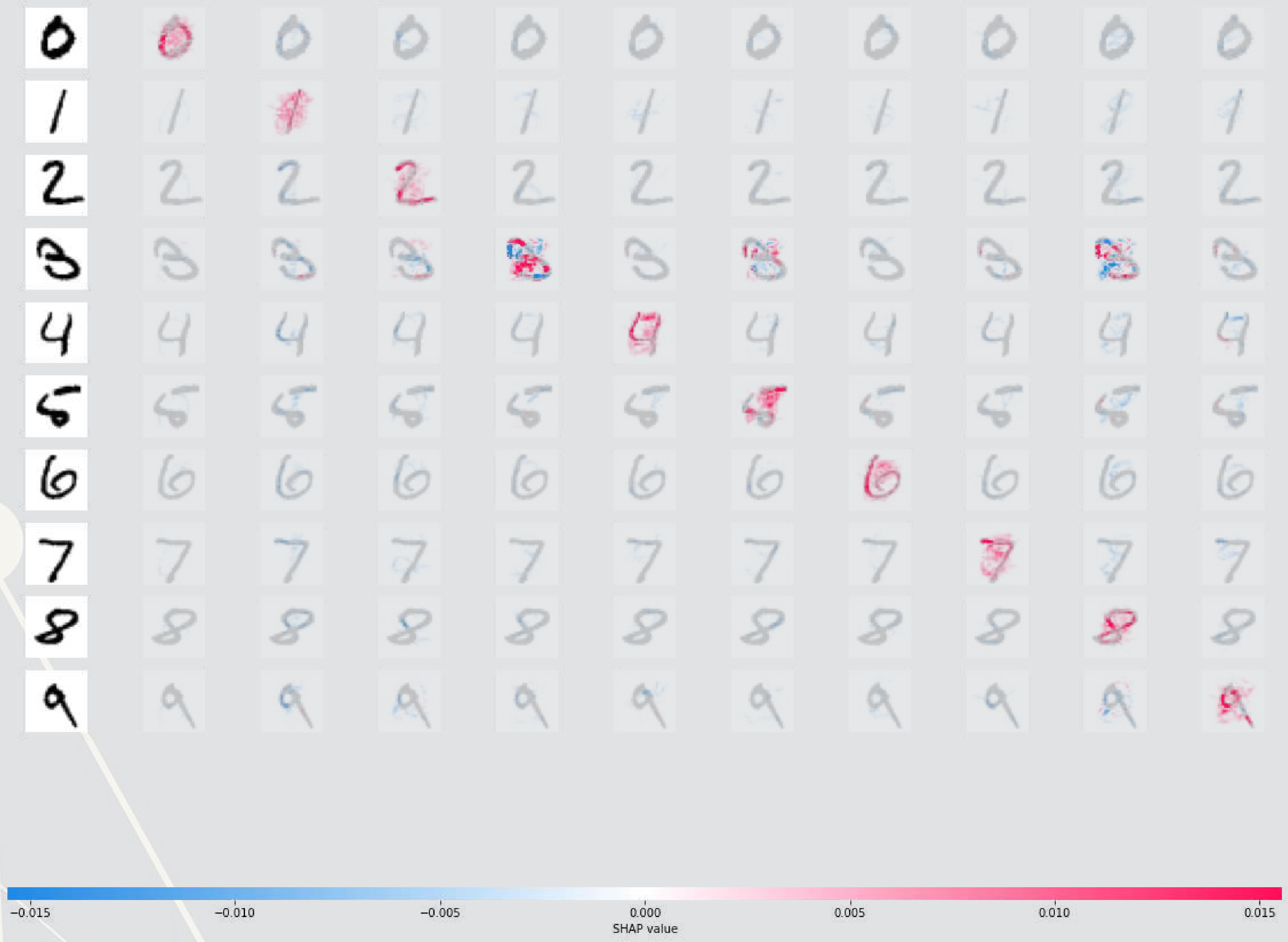
```python
images = np.random.randint(0, train_X.shape[0], size=25)
e = shap.DeepExplainer(model_cnn, train_X[images])
```

- Then we will select 1 of each digit that the CNN got correct and incorrect

```python
correct = [np.where((np.argmax(model_cnn.predict(test_X), axis=-1) == np.argmax(test_Y, axis=-1)) & \
                    (np.argmax(test_Y, axis=-1) == i))[0][0] for i in range(0, 10)]
incorrect = [np.where((np.argmax(model_cnn.predict(test_X), axis=-1) != np.argmax(test_Y, axis=-1)) & \
                    (np.argmax(test_Y, axis=-1) == i))[0][0] for i in range(0, 10)]
```

SHAP for correct images

```python
shap_values = e.shap_values(test_X[correct])
shap.image_plot(shap_values, -test_X[correct])
```



SHAP for incorrect images

```python
shap_values = e.shap_values(test_X[incorrect])
shap.image_plot(shap_values, -test_X[incorrect])
```

Working with pretrained models

Where can I find pretrained models?

- There are many pretrained models on TensorFlow Hub
- There are also models contained in the TensorFlow Github page:
  - Research models
  - Community models
- Google Brain also maintains a collection of models in trax

> Other platforms also maintain model collections

- PyTorch has PyTorch Hub
- Hugging Face maintains a large collection of text models
- ONNX maintains a collection of framework-agnostic models

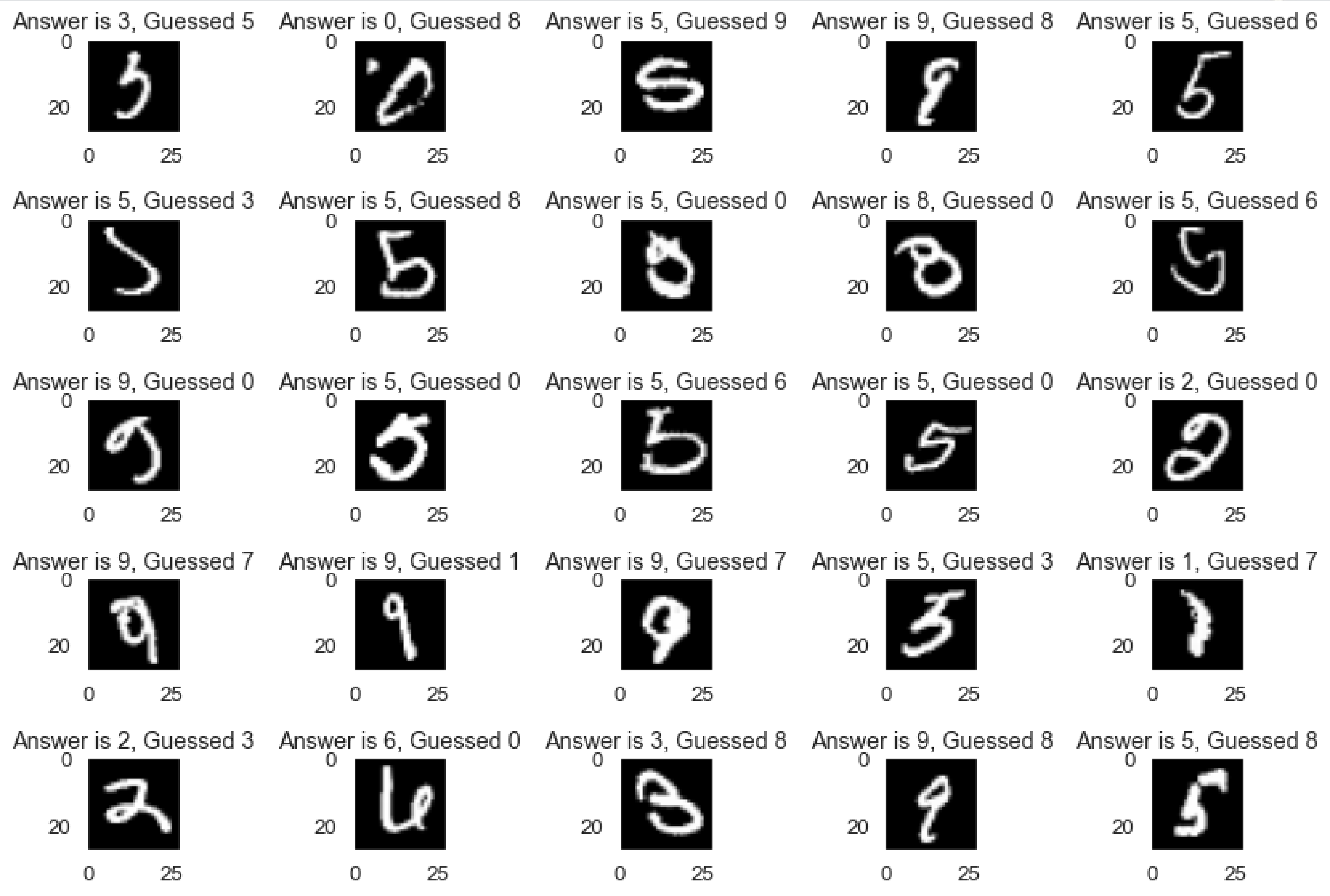
> We will look at TensorFlow Hub today

MNIST off-the-shelf

- The model we will be using is GAN-based MNIST classifier
  - tfgan/eval/mnist/logits
- Use `hub.load()` to load in a model
- Apply it to our testing data, same as before
  - Just apply the model to our data

```python
model_tfgan = hub.load("https://tfhub.dev/tensorflow/tfgan/eval/mnist/logits/1")
logits = model_tfgan(test_X).numpy()

# Check accuracy
sum(np.argmax(logits,-1) == np.argmax(test_Y, -1))
```

Examine incorrect answers



Sentence embeddings off-the-shelf

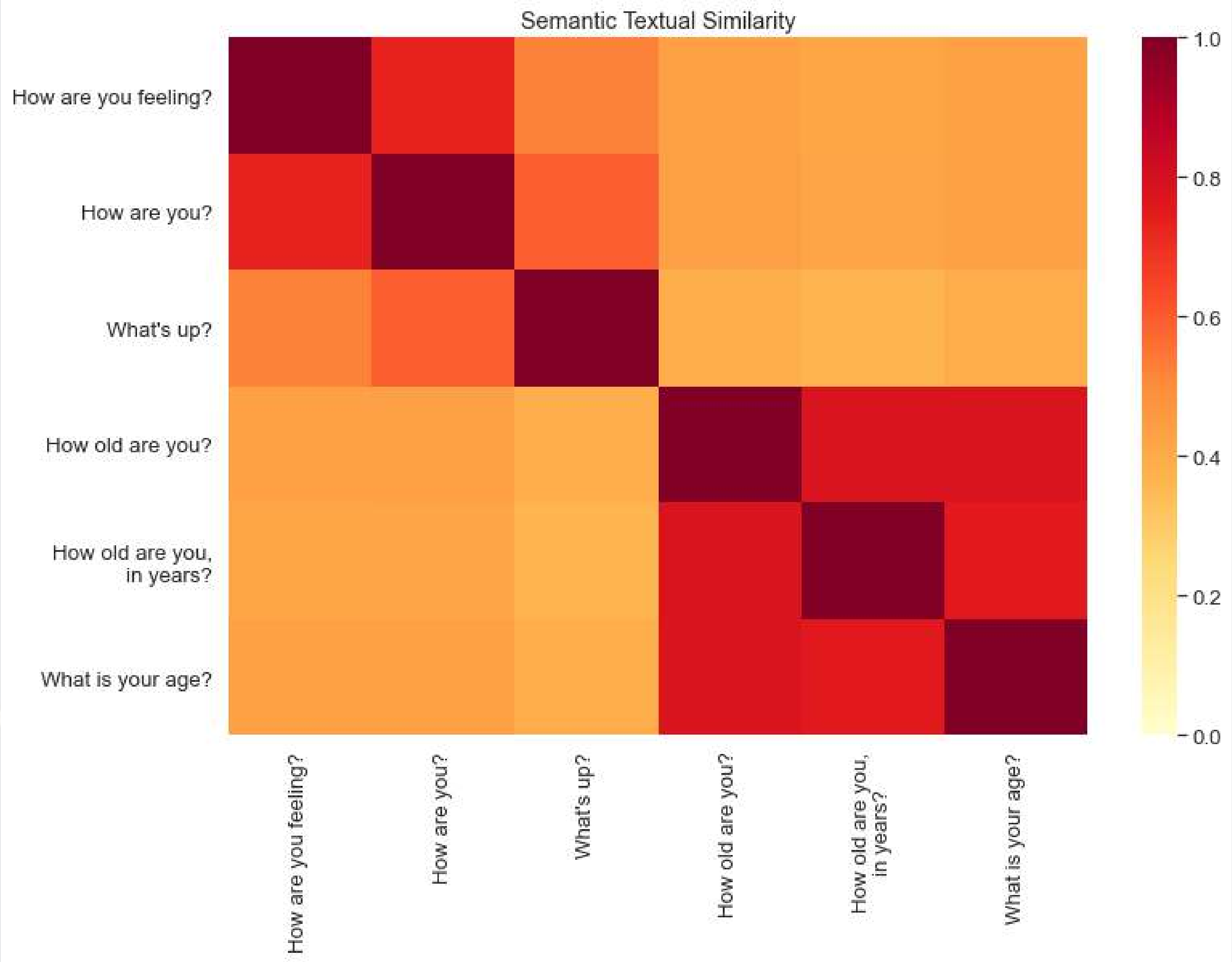
- The model we will be using is the Universal Sentence Encoder (USE) by Cer et al. (2018)
- Converts text that is between phrase and paragraph length into 512-dimensional vectors
- Used in a couple of my papers

```python
embed = hub.load("https://tfhub.dev/google/universal-sentence-encoder-large/5")

messages = ['Two words',
            'This is a sentence.',
            'This is a few sentences.  They are strung together.  They are in one string'
           ]

embeddings = embed(messages)
embeddings
```

```
## <tf.Tensor: shape=(3, 512), dtype=float32, numpy=
## array([[-1.0184747e-02, -3.1019164e-02, -4.2781506e-02, ...,
##          1.0805108e-01,  7.7099161e-05, -6.1001875e-03],
##        [-1.2058644e-02, -3.8627390e-02,  1.5427187e-03, ...,
##          3.3353332e-02, -7.0963770e-02, -1.7223844e-03],
##        [ 3.6280617e-02,  1.7835487e-03, -7.6090815e-03, ...,
##          5.9779502e-02, -1.0792013e-01, -6.0476218e-03]], dtype=float32)>
```

Compare sentences with USE

```python
messages = ["How are you feeling?","How are you?","What's up?",
    "How old are you?","How old are you, in years?","What is your age?"]
embeddings = embed(messages)
plot_similarity(messages, embeddings, 90)
```



Object detection off-the-shelf

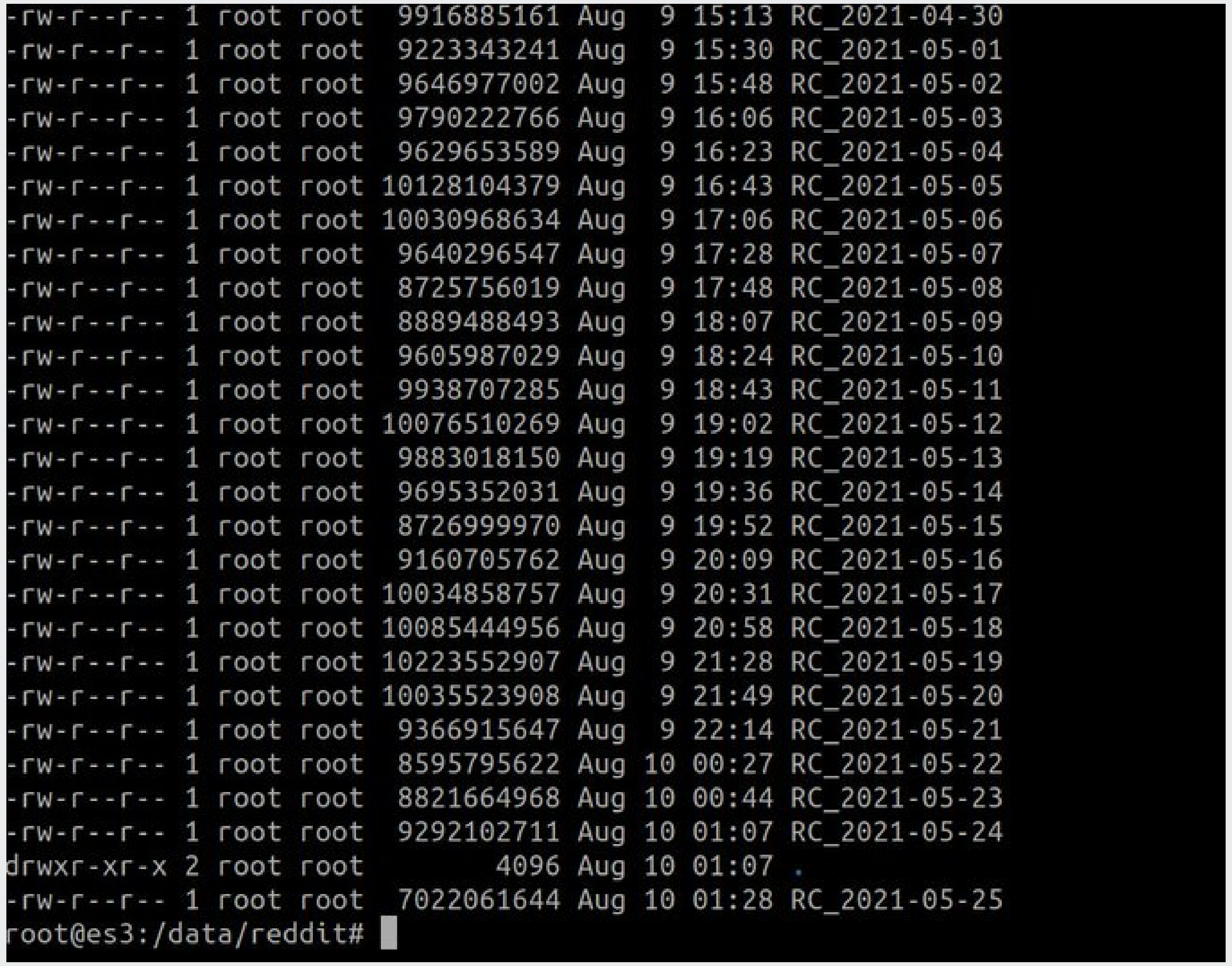
- There are a lot of options for this
- We will use a model trained on COCO from CenterNet
  - centernet/hourglass_512x512
- This can detect 80 different object types, including people

```python
# Full list of object types
labels = load_COCO_labelmap()
print(list(labels.values()))
```

```
## ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train',
## 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking meter',
## 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear',
## 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase',
## 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat',
## 'baseball glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle',
## 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
## 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake',
## 'chair', 'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv',
## 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven',
## 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors',
## 'teddy bear', 'hair drier', 'toothbrush']
```

Using the model

```python
centernet = hub.load('https://tfhub.dev/tensorflow/centernet/hourglass_512x512/1')

image1, image1_np = load_image('../Data/S6_1.jpeg')
image2, image2_np = load_image('https://pbs.twimg.com/media/E8ZIIKGXIAAipIh?format=jpg&name=small')
```



Applying the model

- We apply the model to the numpy matrix representation of the image
- `result` is just a numpy version of `results`
  - This contains four types of information

```python
results = centernet(image1_np)
result = {key:value.numpy() for key,value in results.items()}
print(result.keys())
```

```
## dict_keys(['detection_scores', 'num_detections', 'detection_boxes', 'detection_classes'])
```

Applying the model

- The below functions are defined out of convenience

```python
def top_k_objects(result, k=3):
    top_scores = result['detection_scores'][0][0:k]
    top_ids = [labels[str(int(i))] for i in result['detection_classes'][0]][0:k]
    for row in zip(top_scores, top_ids):
        print('Object: ' + row[1] + ', score: ' + str(row[0]))


def prob_person(result):
    id_person = 1
    top_person_loc = np.where(result['detection_classes'][0] == 1)[0][0]
    prob = result['detection_scores'][0][top_person_loc]
    print('Probability of a person in the photo: ' + str(prob))
```

- The first function reports the top k objects detected, based on weights assigned by the model
- The second function reports the highest probability that a person was included in the image

Analyzing the first image

```python
top_k_objects(result, 3)
```

```
## Object: tie, score: 0.56596684
## Object: person, score: 0.45707893
## Object: tv, score: 0.3345726
```

```python
prob_person(result)
```

```
## Probability of a person in the photo: 0.45707893
```

Applying to the second image

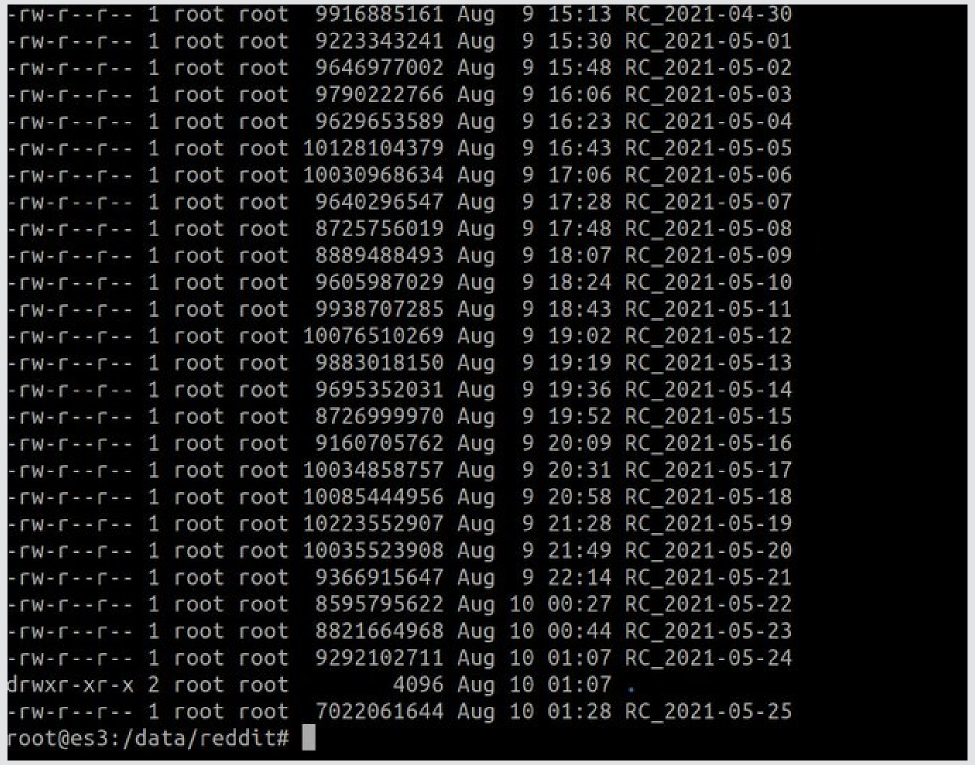
```python
results = centernet(image2_np)
result = {key:value.numpy() for key,value in results.items()}
```

```python
top_k_objects(result, 3)
```

```
## Object: book, score: 0.7087656
## Object: tv, score: 0.10406752
## Object: book, score: 0.07747121
```

```python
prob_person(result)
```

```
## No person found
```



```
-rw-r--r-- 1 root root  9916885161 Aug  9 15:13 RC_2021-04-30
-rw-r--r-- 1 root root  9223343241 Aug  9 15:30 RC_2021-05-01
-rw-r--r-- 1 root root  9646977002 Aug  9 15:48 RC_2021-05-02
-rw-r--r-- 1 root root  9790222766 Aug  9 16:06 RC_2021-05-03
-rw-r--r-- 1 root root  9629653589 Aug  9 16:23 RC_2021-05-04
-rw-r--r-- 1 root root 10128104379 Aug  9 16:43 RC_2021-05-05
-rw-r--r-- 1 root root 10030968634 Aug  9 17:06 RC_2021-05-06
-rw-r--r-- 1 root root  9640296547 Aug  9 17:28 RC_2021-05-07
-rw-r--r-- 1 root root  8725756019 Aug  9 17:48 RC_2021-05-08
-rw-r--r-- 1 root root  8889488493 Aug  9 18:07 RC_2021-05-09
-rw-r--r-- 1 root root  9605987029 Aug  9 18:24 RC_2021-05-10
-rw-r--r-- 1 root root  9938707285 Aug  9 18:43 RC_2021-05-11
-rw-r--r-- 1 root root 10076510269 Aug  9 19:02 RC_2021-05-12
-rw-r--r-- 1 root root  9883018150 Aug  9 19:19 RC_2021-05-13
-rw-r--r-- 1 root root  9695352031 Aug  9 19:36 RC_2021-05-14
-rw-r--r-- 1 root root  8726999970 Aug  9 19:52 RC_2021-05-15
-rw-r--r-- 1 root root  9160705762 Aug  9 20:09 RC_2021-05-16
-rw-r--r-- 1 root root 10034858757 Aug  9 20:31 RC_2021-05-17
-rw-r--r-- 1 root root 10085444956 Aug  9 20:58 RC_2021-05-18
-rw-r--r-- 1 root root 10223552907 Aug  9 21:28 RC_2021-05-19
-rw-r--r-- 1 root root 10035523908 Aug  9 21:49 RC_2021-05-20
-rw-r--r-- 1 root root  9366915647 Aug  9 22:14 RC_2021-05-21
-rw-r--r-- 1 root root  8595795622 Aug 10 00:27 RC_2021-05-22
-rw-r--r-- 1 root root  8821664968 Aug 10 00:44 RC_2021-05-23
-rw-r--r-- 1 root root  9292102711 Aug 10 01:07 RC_2021-05-24
drwxr-xr-x 2 root root        4096 Aug 10 01:07 .
-rw-r--r-- 1 root root  7022061644 Aug 10 01:28 RC_2021-05-25
root@es3:/data/reddit#
```

Video data

Working with video

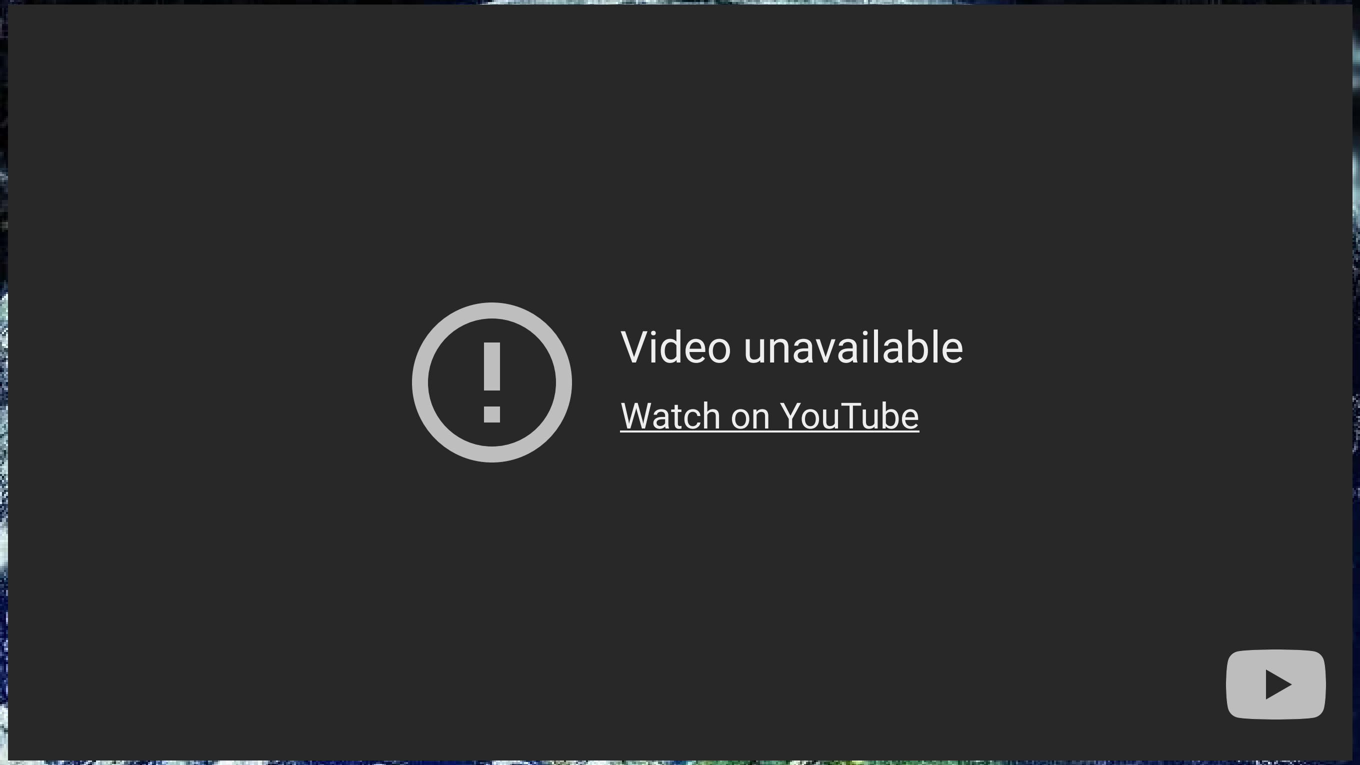
- Video data is challenging – very storage intensive
  - Ex.: Uber's self driving cars would generate >100GB of data *per hour* **per car**
- Video data is very promising
  - Think of how many task involve vision!
    - Driving
    - Photography
    - Warehouse auditing…
- At the end of the day though, video is just a sequence of images

One method for video

YOLOv3

- You
- Only
-

- Once

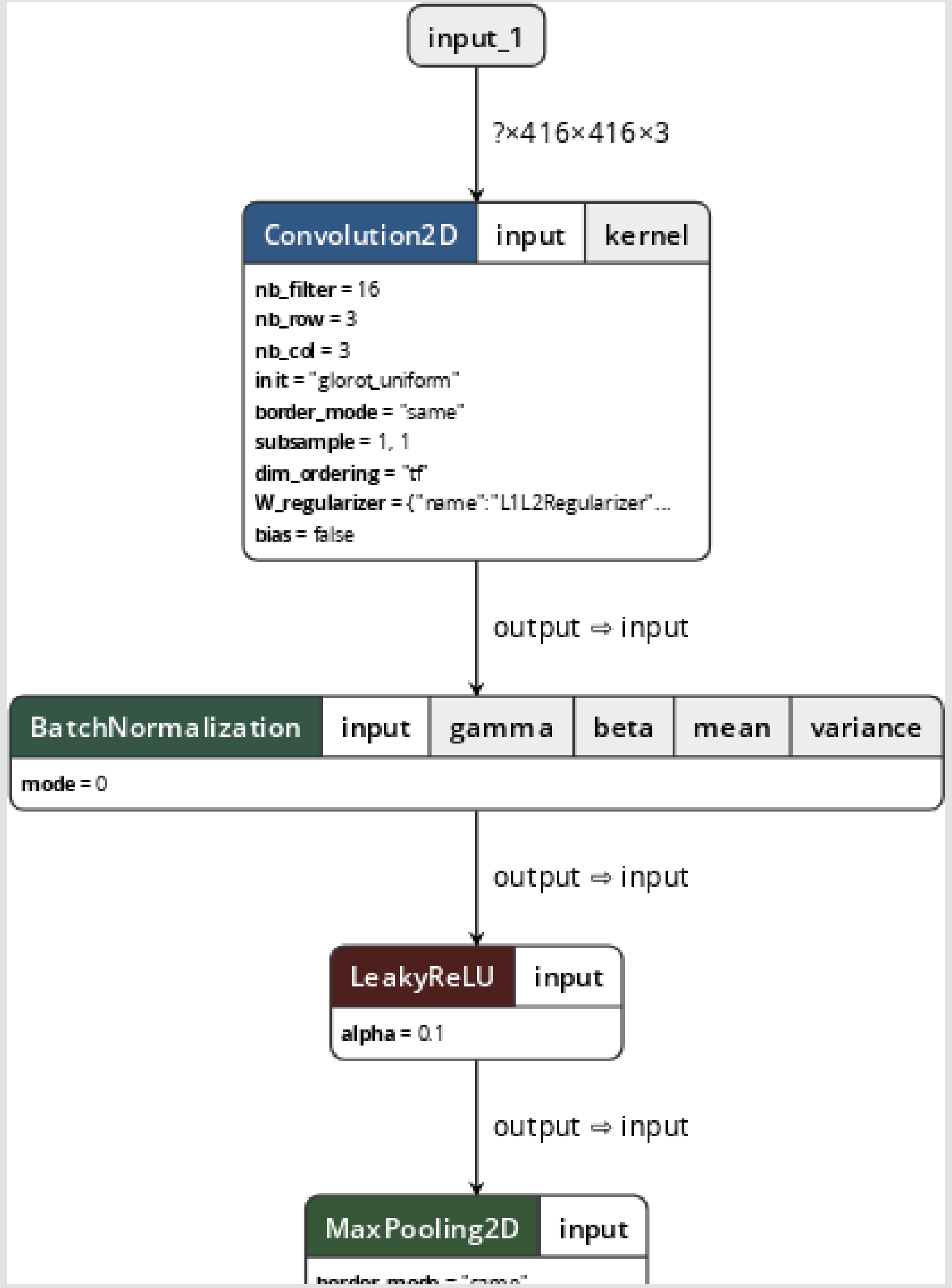


Video link

What does YOLO do?

- It spots objects in videos and labels them
  - It also figures out a *bounding box* – a box containing the object inside the video frame
- It can spot *overlapping* objects
- It can spot multiple of the same or different object types
- The baseline model (using the COCO dataset) can detect 80 different object types
  - There are other datasets with more objects

How does Yolo do it? Map of Tiny YOLO



Yolo model and graphing tool from [lutzroeder/netron](https://github.com/lutzroeder/netron)

How does Yolo do it?

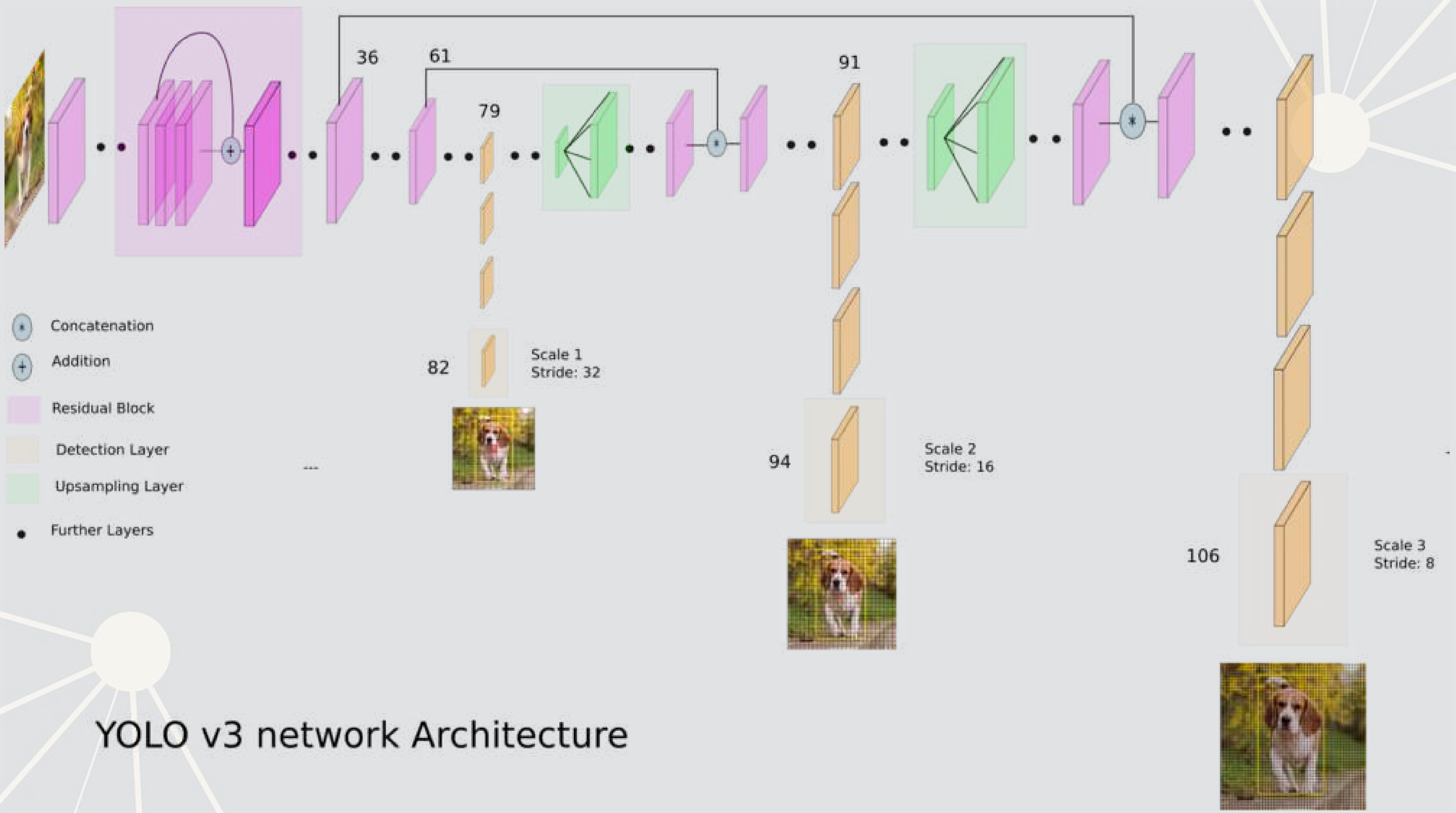


Diagram from *What's new in YOLO v3* by Ayoosh Kathuria

Final word on object detection

- An algorithm like YOLO v3 is somewhat tricky to run
- Preparing the algorithm takes a long time
  - The final output, though, can run on much cheaper hardware
- These algorithms just recently became feasible so their impact has yet to be felt so strongly

> Think about how facial recognition showed up everywhere for images over the past few years

Where to get video data

- One extensive source is Youtube-8M
  - 6.1M videos, 3-10 minutes each
  - Each video has >1,000 views
  - 350,000 hours of video
  - 237,000 labeled 5 second segments
  - 1.3B video features that are machine labeled
  - 1.3B audio features that are machine labeled

Conclusion

Wrap-up

Neural networks

- Highly flexible, nonparametric algorithms
- Good for multiclass classification
- Good for generating measures

Off-the-shelf models

- Many options available
- A lot of state of the art text models are freely available
- Decent image processing models are also available
- Many other model types are also available, such as translation algorithms

Course wrap-up

> Over the past 6 sessions, we have covered a wide variety of *practical* machine learning algorithms for accounting research

1. Simple econometric models like LASSO
2. More complex, nonlinear or nonparametric models like SVM/SVR and XGBoost
3. Working with text in python, including ML-based grammar and dependency parsing
4. Simpler text models including word embeddings (word2vec) and topic modeling (LDA)
5. Economics-oriented ML: bias detection with SHAP and causality with DoubleML
6. Neural networks

> Hopefully this course gave you a lot to think about and jogged some interesting research ideas!

Packages used for these slides

Python

- matplotlib
- numpy
- pandas
- PIL (pillow)
- requests
- seaborn
- shap
- tensorflow
- tensorflow_gan
- tensorflow_hub

R

- kableExtra
- knitr
- reticulate
- revealjs

References

- Cer, Daniel, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant et al. “Universal sentence encoder.” arXiv preprint arXiv:1803.11175 (2018).
- Wich, Maximilian, Jan Bauer, and Georg Groh. “Impact of politically biased data on hate speech classification.” In Proceedings of the Fourth Workshop on Online Abuse and Harms, pp. 54-64. 2020.

Custom code

```python
# USE helper code

# Efficient distance calculation
def distance_matrix_np(pts):
    """Returns matrix of pairwise Euclidean distances. Vectorized numpy version."""
    return np.sum((pts[None,:] - pts[:, None])**2, -1)**0.5

# Plot USE similarity
def plot_similarity(messages, embeddings, rotation):
    messages2 = []
    for message in messages:
        if len(message.split()) > 4:
            c = 0
            temp = ''
            for m in message.split():
                temp += m
                c += 1
                if c==4:
                    temp += '\n'
                    c = 0
                else:
                    temp += ' '
            temp = temp[:-1]
            messages2.append(temp)
        else:
            messages2.append(message)
    messages = messages2
    corr = distance_matrix_np(embeddings)
    corr = 1 - corr/2
    sns.set(font_scale=1.2)
    g = sns.heatmap(
        corr,
        xticklabels=messages,
        yticklabels=messages,
        vmin=0,
        vmax=1,
        cmap="YlOrRd")
    g.set_xticklabels(messages, rotation=rotation)
    g.set_yticklabels(messages, rotation=0)
    g.set_title("Semantic Textual Similarity")
    return g
```

Custom code

```python
# Object detection helper code

# Image loader -- works for local or internet files
def load_image(path):
    if path.startswith('http'):
        response = requests.get(path)
        image_data = io.BytesIO(response.content)
        image = Image.open(image_data)
    else:
        image_data = tf.io.gfile.GFile('../Data/S6_1.jpeg', 'rb').read()
        image = Image.open(io.BytesIO(image_data))
    (im_width, im_height) = image.size
    image_np = np.array(image.getdata()).reshape((1, im_height, im_width, 3)).astype(np.uint8)
    return image, image_np

# Load in the COCO labels from disk
def load_COCO_labelmap():
    with open('../Data/S6_COCO_labelmap.txt', 'rt') as f:
        text = f.readlines()
    output = {}
    for row in text:
        if 'id' in row:
            id = row[6:8]
            if '\n' in id:
                id = id[0]
        elif 'display_name' in row:
            output[id] = row.split('"')[1]
    return output
```