

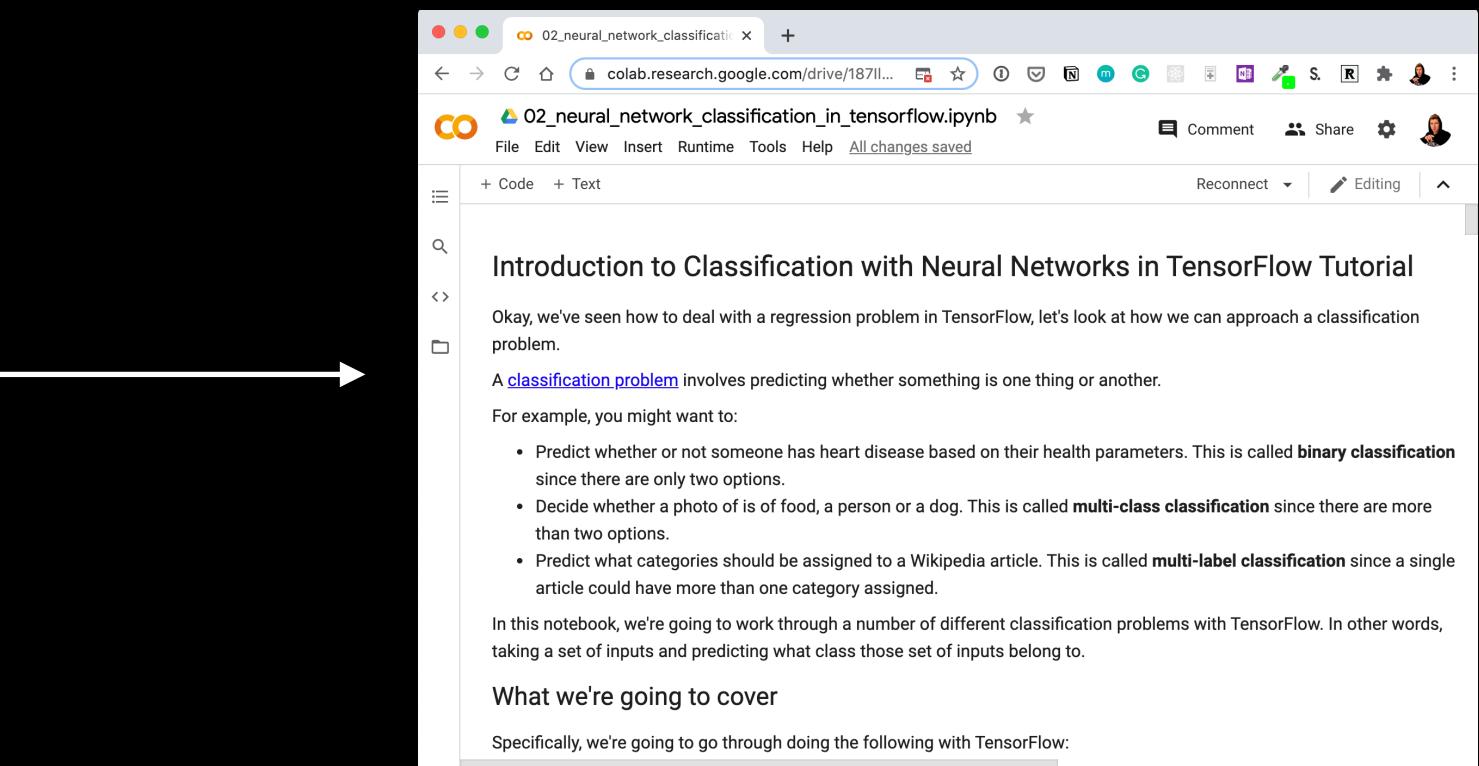
Neural Network Classification with



TensorFlow

Where can you get help?

- Follow along with the code

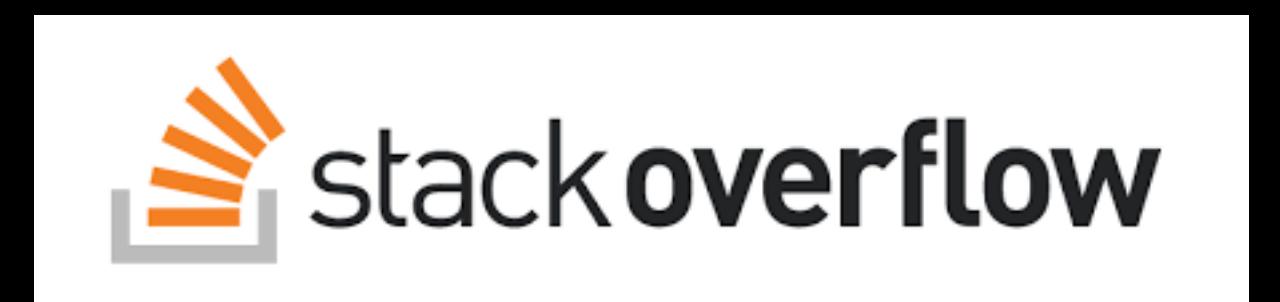


A screenshot of a Google Colab notebook titled "02_neural_network_classification_in_tensorflow.ipynb". The notebook contains text explaining classification problems and what we're going to cover, followed by a code cell starting with "# Set random seed".

"If in doubt, run the code"

- Try it for yourself

- Press SHIFT + CMD + SPACE to read the docstring

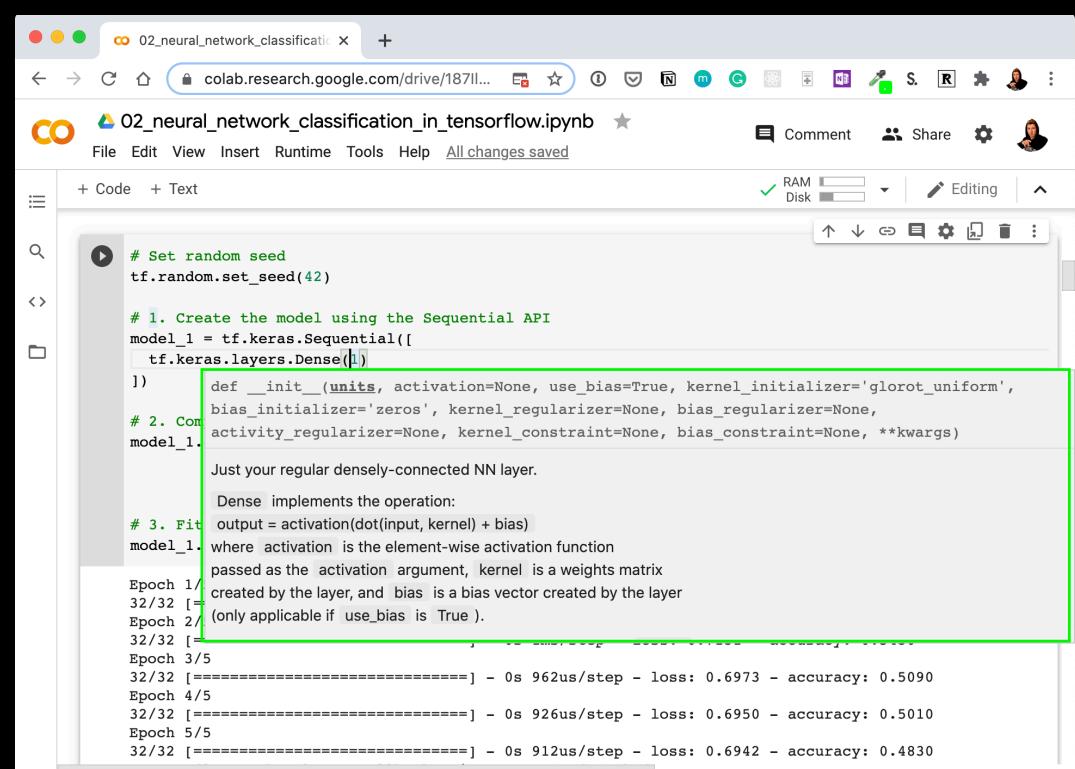


- Search for it

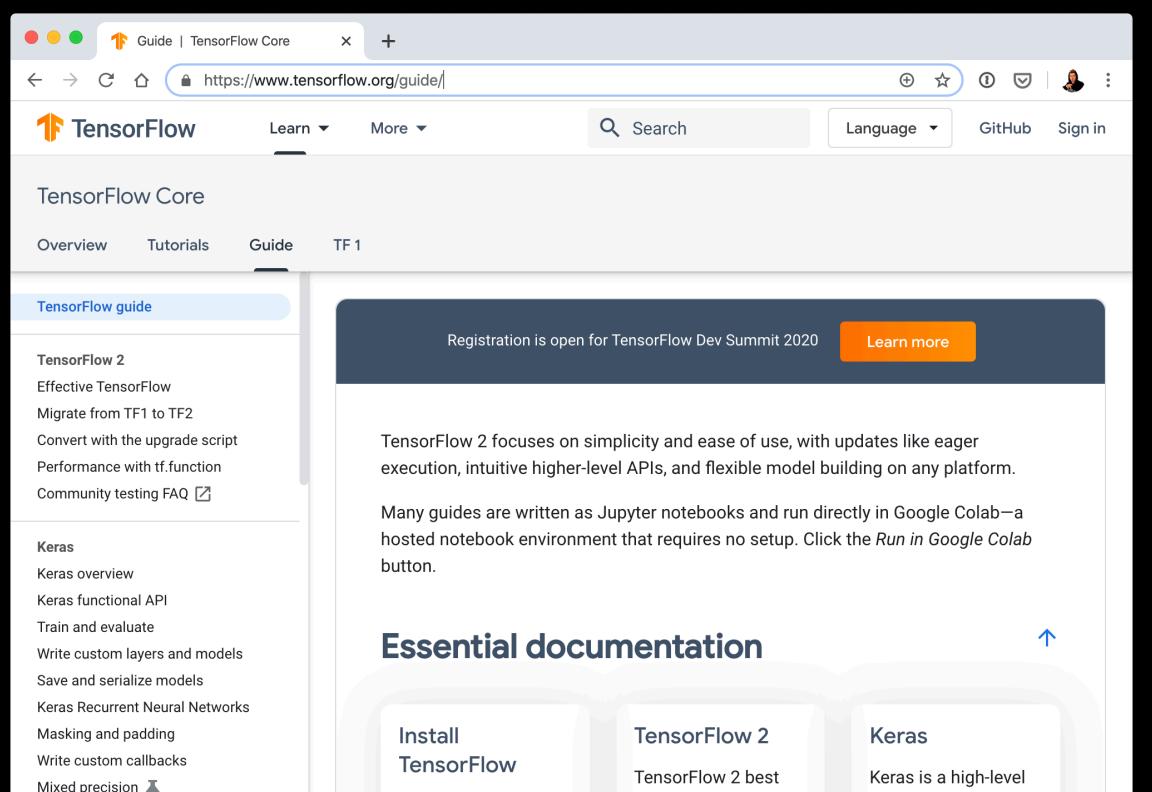
- Try again

- Ask (don't forget the Discord chat!)

(yes, including the "dumb" questions)



A screenshot of a Jupyter notebook cell containing Python code for a Dense layer. A green box highlights the docstring: "Dense implements the operation: output = activation(dot(input, kernel) + bias)".



A screenshot of the TensorFlow website's "TensorFlow Core" guide page. It features a sidebar with links like "TensorFlow 2", "Keras", and "Install TensorFlow". The main content area discusses TensorFlow 2 and provides links to "Install TensorFlow", "TensorFlow 2", and "Keras".

“What is a classification
problem?”

Example classification problems

“Is this email spam or not spam?”

To: daniel@mrdbourke.com

Hey Daniel,

This deep learning course is incredible!
I can't wait to use what I've learned!

Not spam

To: daniel@mrdbourke.com

Hay daniel...

C0ongratu1ations! U win \$1139239230

Spam

“Is this a photo of sushi, steak or pizza?”



Binary classification
(one thing or another)

“What tags should this article have?”

A screenshot of a Wikipedia article page for "Deep learning". The page content discusses deep learning as a subset of machine learning. To the right of the main text, there is a sidebar titled "Part of a series on Machine learning and data mining". This sidebar lists several topics: "Machine learning", "Supervised learning (classification - regression)", "Clustering", "Dimensionality reduction", "Structured prediction", "Anomaly detection", "Artificial neural network", and "Reinforcement learning". Each topic has a "[show]" link next to it.

Multiclass classification
(more than one thing or another)

(multiple label options per sample)

Multilabel classification

What we're going to cover

(broadly)

- Architecture of a neural network **classification** model
- Input shapes and output shapes of a **classification** model (features and labels)
- Creating custom data to view and fit
- Steps in modelling
 - Creating a model, compiling a model, fitting a model, evaluating a model
- Different **classification** evaluation methods
- Saving and loading models

(we'll be cooking up lots of code!)

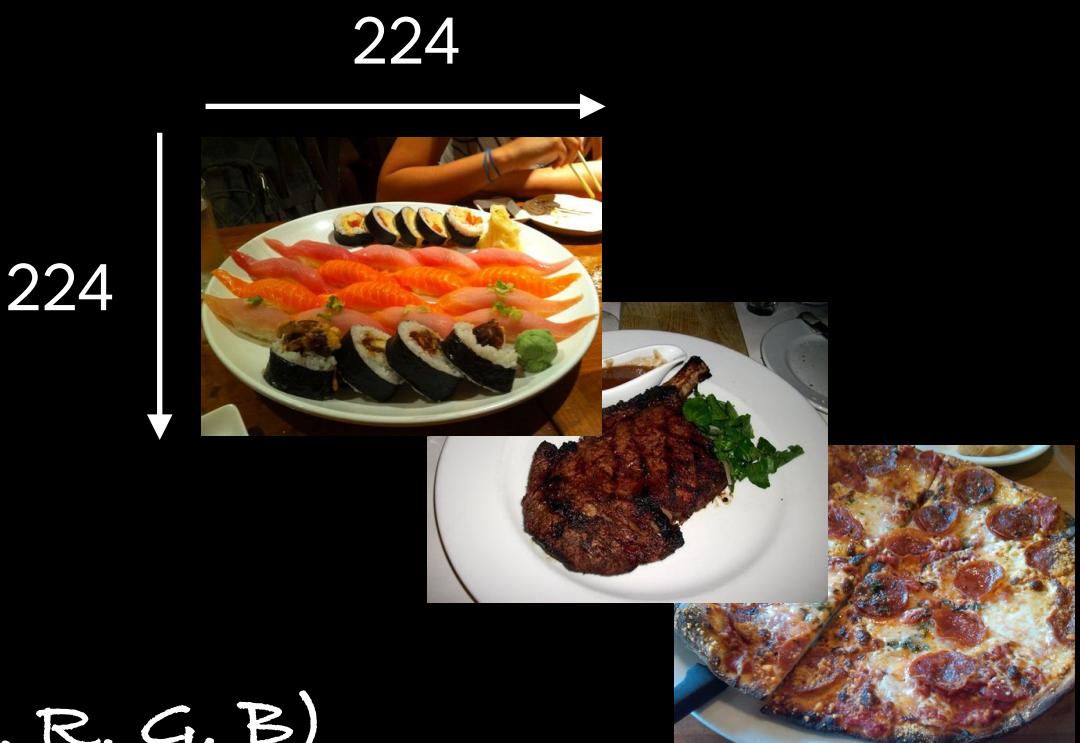
How:



Classification inputs and outputs

$$\begin{aligned} W &= 224 \\ H &= 224 \\ C &= 3 \end{aligned}$$

(c = colour channels, R, G, B)

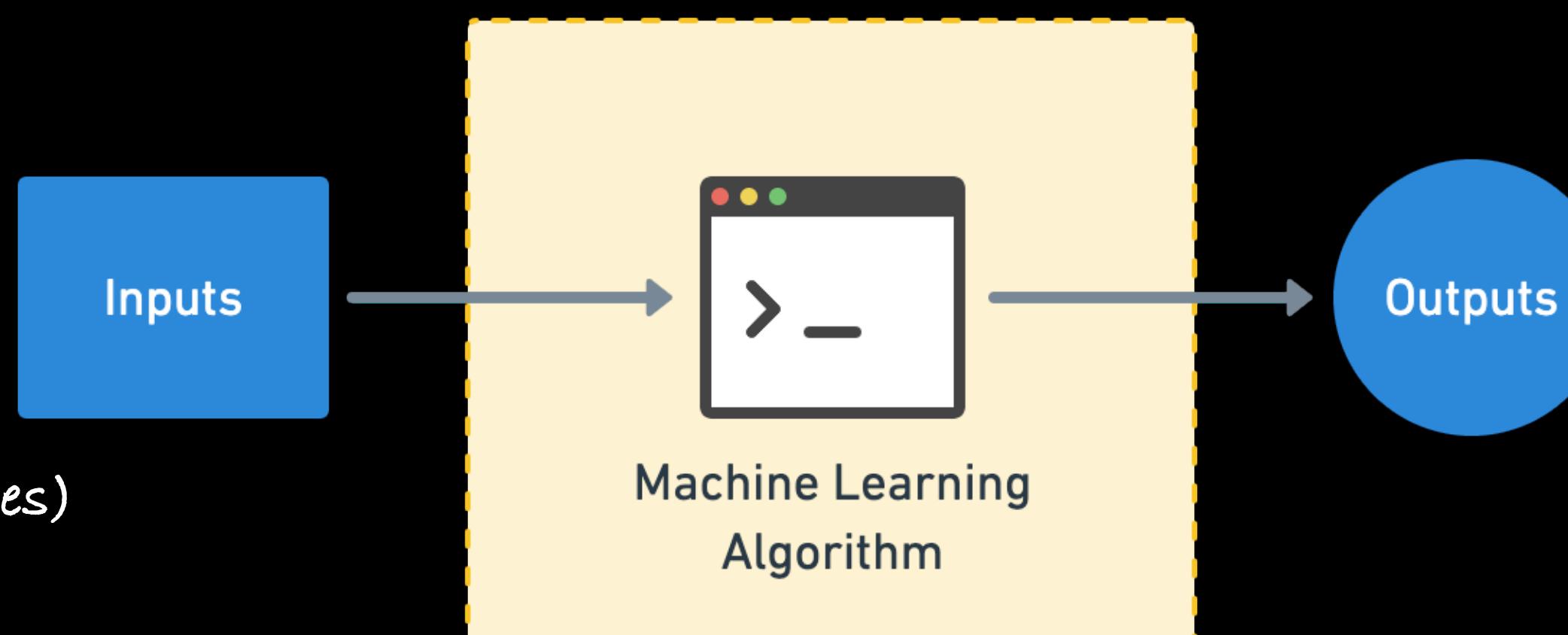


Sushi
Steak
Pizza

Actual output

$\begin{bmatrix} [0.31, 0.62, 0.44...], \\ [0.92, 0.03, 0.27...], \\ [0.25, 0.78, 0.07...], \\ \dots, \end{bmatrix}$ (normalized pixel values)

Numerical encoding



(often already exists, if not,
you can build one)

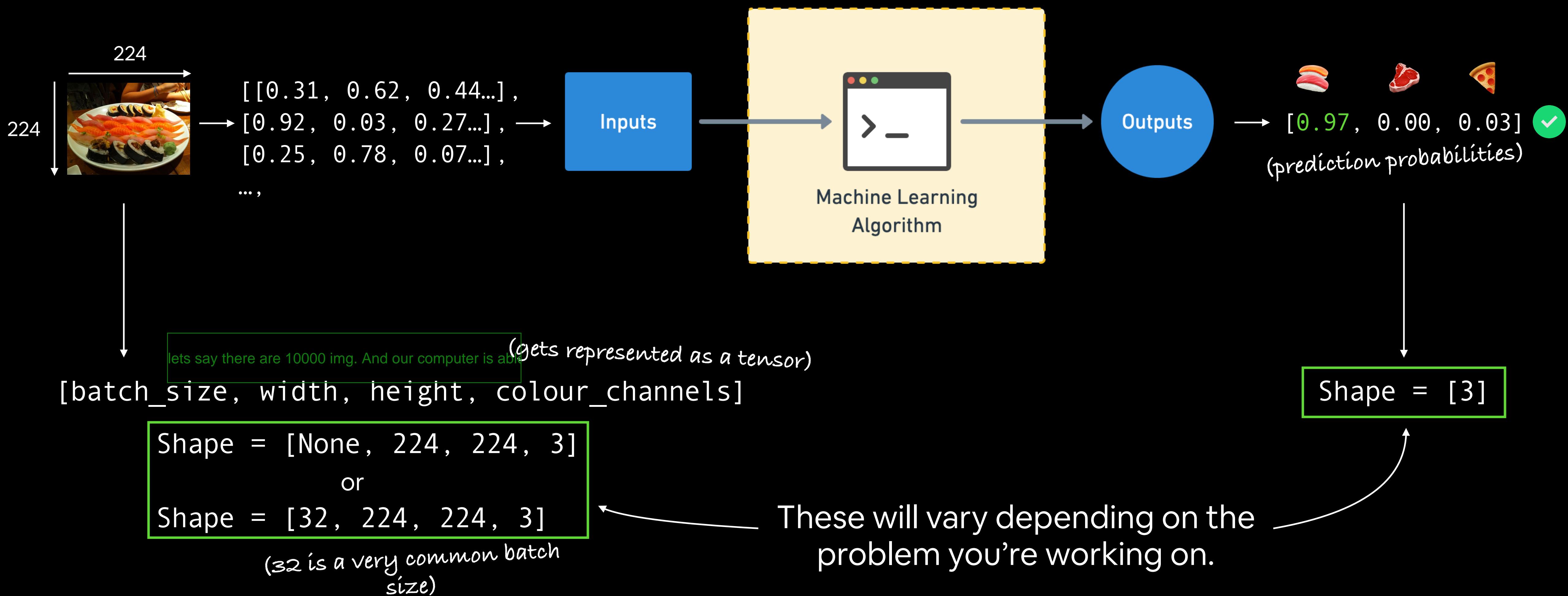
→ $\begin{bmatrix} [0.97, 0.00, 0.03], \\ [0.81, 0.14, 0.05], \\ [0.03, 0.07, 0.90], \\ \dots, \end{bmatrix}$

Predicted output

(comes from looking at lots
of these)

Input and output shapes

(for an image classification example)



(typical)

Architecture of a classification model

(we're going to be building lots of these)

Hyperparameter	Binary Classification	Multiclass classification
Input layer shape	Same as number of features (e.g. 5 for age, sex, height, weight, smoking status in heart disease prediction)	Same as binary classification
Hidden layer(s)	Problem specific, minimum = 1, maximum = unlimited	Same as binary classification
Neurons per hidden layer	Problem specific, generally 10 to 100	Same as binary classification
Output layer shape	1 (one class or the other)	1 per class (e.g. 3 for food, person or dog photo)
Hidden activation	Usually ReLU (rectified linear unit)	Same as binary classification
Output activation	Sigmoid	Softmax
Loss function	Cross entropy (tf.keras.losses.BinaryCrossentropy in TensorFlow)	Cross entropy (tf.keras.losses.CategoricalCrossentropy in TensorFlow)
Optimizer	SGD (stochastic gradient descent), Adam	Same as binary classification

Source: Adapted from page 295 of [Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow](#) Book by Aurélien Géron

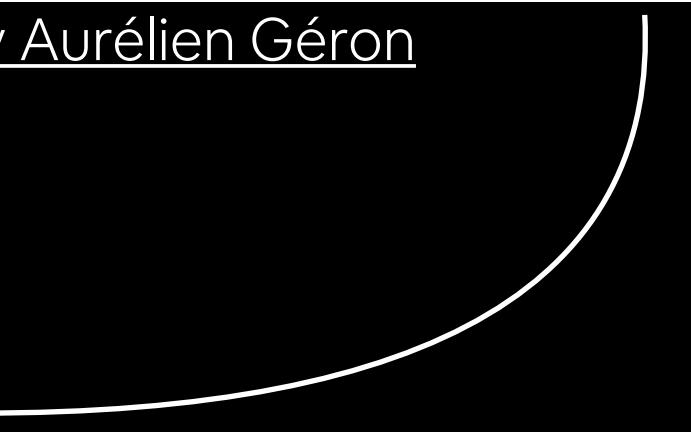


```
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.Input(shape=(224, 224, 3)),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(3, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)
```

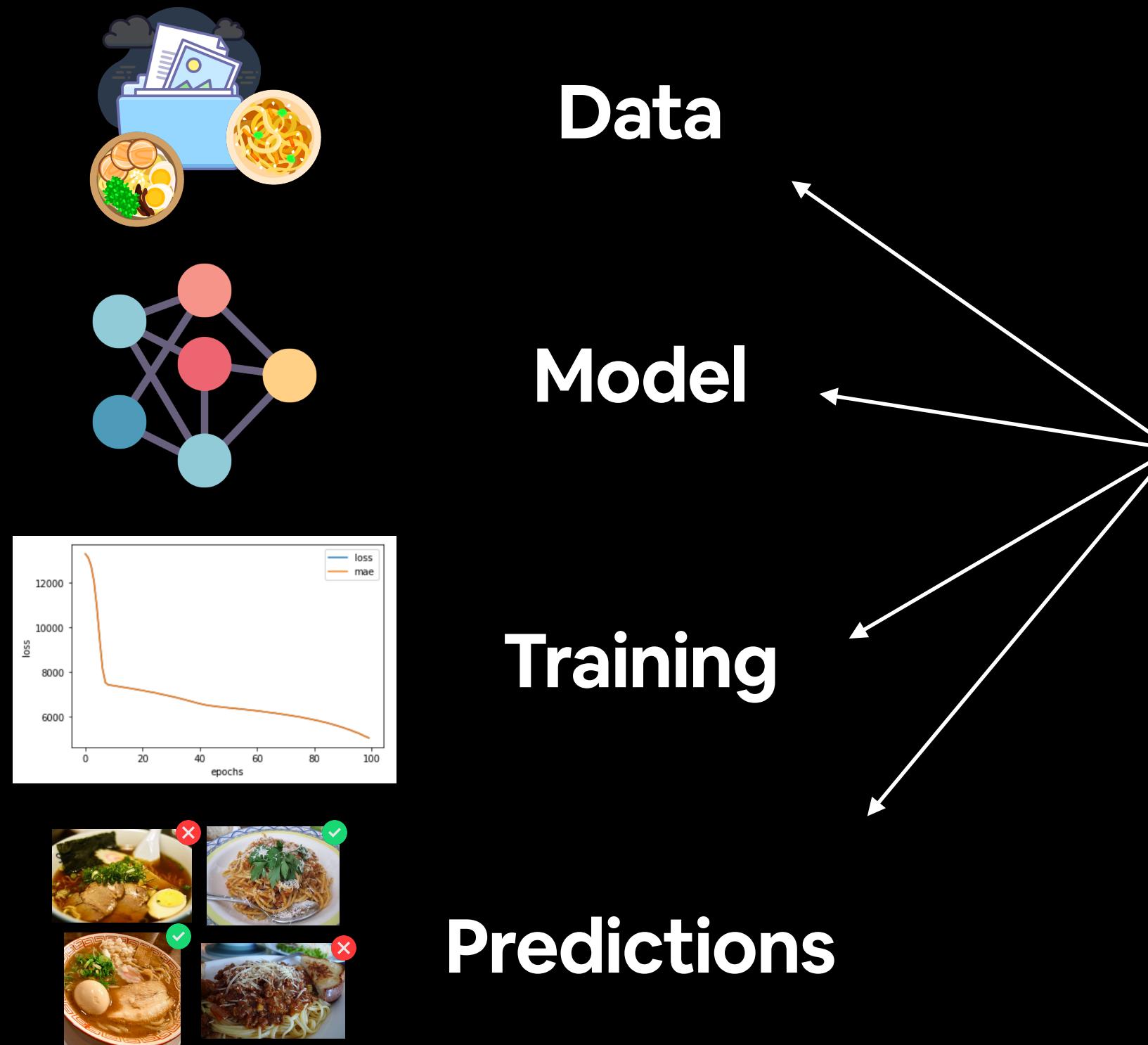


Sushi → Steak Pizza

Let's code!

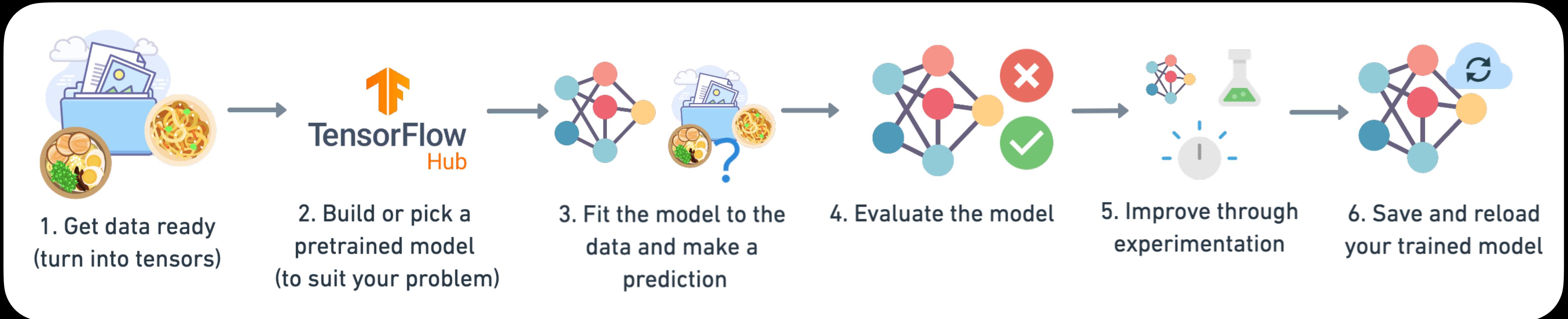
The machine learning explorer's motto

“Visualize, visualize, visualize”



It's a good idea to visualize these as often as possible.

Steps in modelling with TensorFlow



```
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.Input(shape=(224, 224, 3)),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(3, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=[ "accuracy" ])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)
```

(typical)

Architecture of a classification model

(we're going to be building lots of these)

Hyperparameter	Binary Classification	Multiclass classification
Input layer shape	Same as number of features (e.g. 5 for age, sex, height, weight, smoking status in heart disease prediction)	Same as binary classification
Hidden layer(s)	Problem specific, minimum = 1, maximum = unlimited	Same as binary classification
Neurons per hidden layer	Problem specific, generally 10 to 100	Same as binary classification
Output layer shape	1 (one class or the other)	1 per class (e.g. 3 for food, person or dog photo)
Hidden activation	Usually ReLU (rectified linear unit)	Same as binary classification
Output activation	Sigmoid	Softmax
Loss function	Cross entropy (tf.keras.losses.BinaryCrossentropy in TensorFlow)	Cross entropy (tf.keras.losses.CategoricalCrossentropy in TensorFlow)
Optimizer	SGD (stochastic gradient descent), Adam	Same as binary classification

Source: Adapted from page 295 of [Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow](#) Book by Aurélien Géron

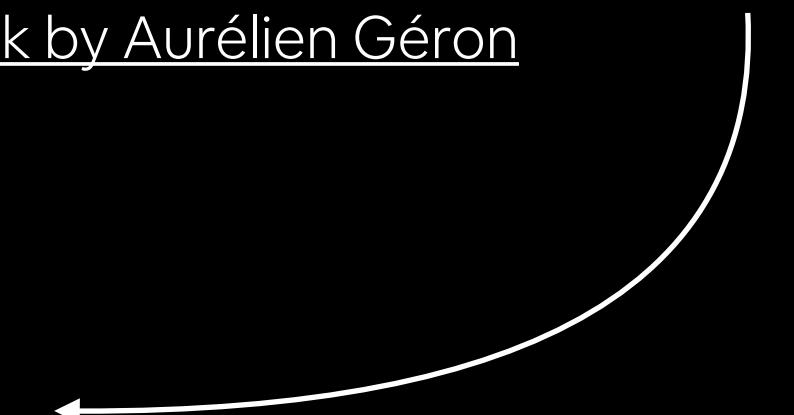


```
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.Input(shape=(224, 224, 3)),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(3, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)
```



Sushi

→ Steak

Pizza

Steps in modelling with TensorFlow

```
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.Input(shape=(224, 224, 3)),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(3, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
               optimizer=tf.keras.optimizers.Adam(),
               metrics=[ "accuracy" ])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)
```

1. Construct or import a pretrained model relevant to your problem
2. Compile the model (prepare it to be used with data)
 - **Loss** — how wrong your model's predictions are compared to the truth labels (you want to minimise this).
 - **Optimizer** — how your model should update its internal patterns to better its predictions.
 - **Metrics** — human interpretable values for how well your model is doing.
3. Fit the model to the training data so it can discover patterns
 - **Epochs** — how many times the model will go through all of the training examples.
4. Evaluate the model on the test data (how reliable are our model's predictions?)

Improving a model

(from a model's perspective)

```
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(4, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.001),
              metrics=[ "accuracy" ])

# 3. Fit the model
model.fit(x_train_subset, y_train_subset, epochs=5)
```

Smaller model

```
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              metrics=[ "accuracy" ])

# 3. Fit the model
model.fit(x_train_full, y_train_full, epochs=100)
```

Larger model

Common ways to improve a deep model:

- Adding layers
- Increase the number of hidden units
- Change the activation functions
- Change the optimization function
- Change the learning rate (because you can alter each of these, they're hyperparameters)
- Fitting on more data
- Fitting for longer

Improving a model

(from a model's perspective)

```
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(4, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.001),
              metrics=[ "accuracy" ])

# 3. Fit the model
model.fit(x_train_subset, y_train_subset, epochs=5)
```

Smaller model

```
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              metrics=[ "accuracy" ])

# 3. Fit the model
model.fit(x_train_full, y_train_full, epochs=100)
```

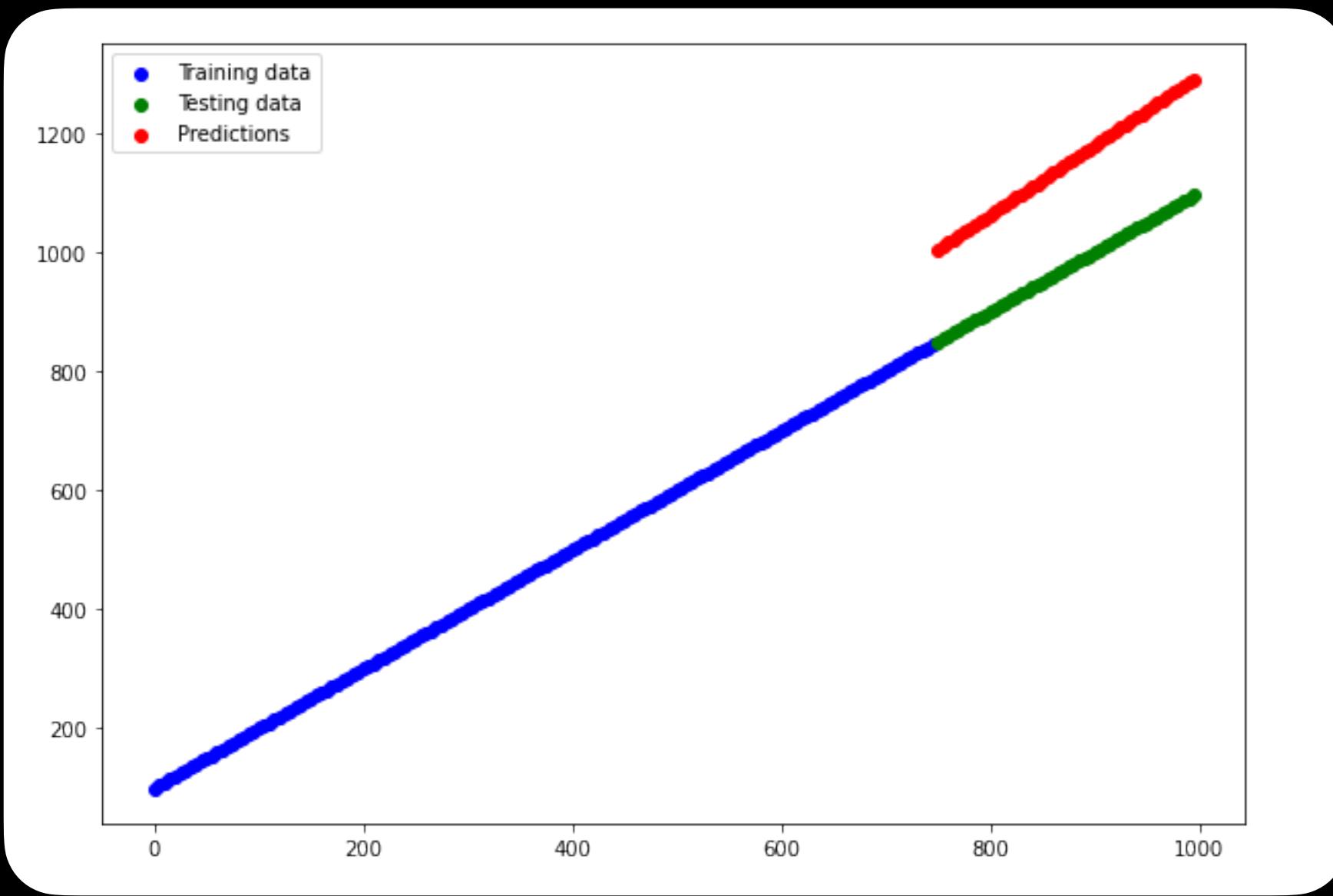
Larger model

Common ways to improve a deep model:

- Adding layers
- Increase the number of hidden units
- Change the activation functions
- Change the optimization function
- Change the learning rate (because you can alter each of these, they're hyperparameters)
- Fitting on more data
- Fitting for longer

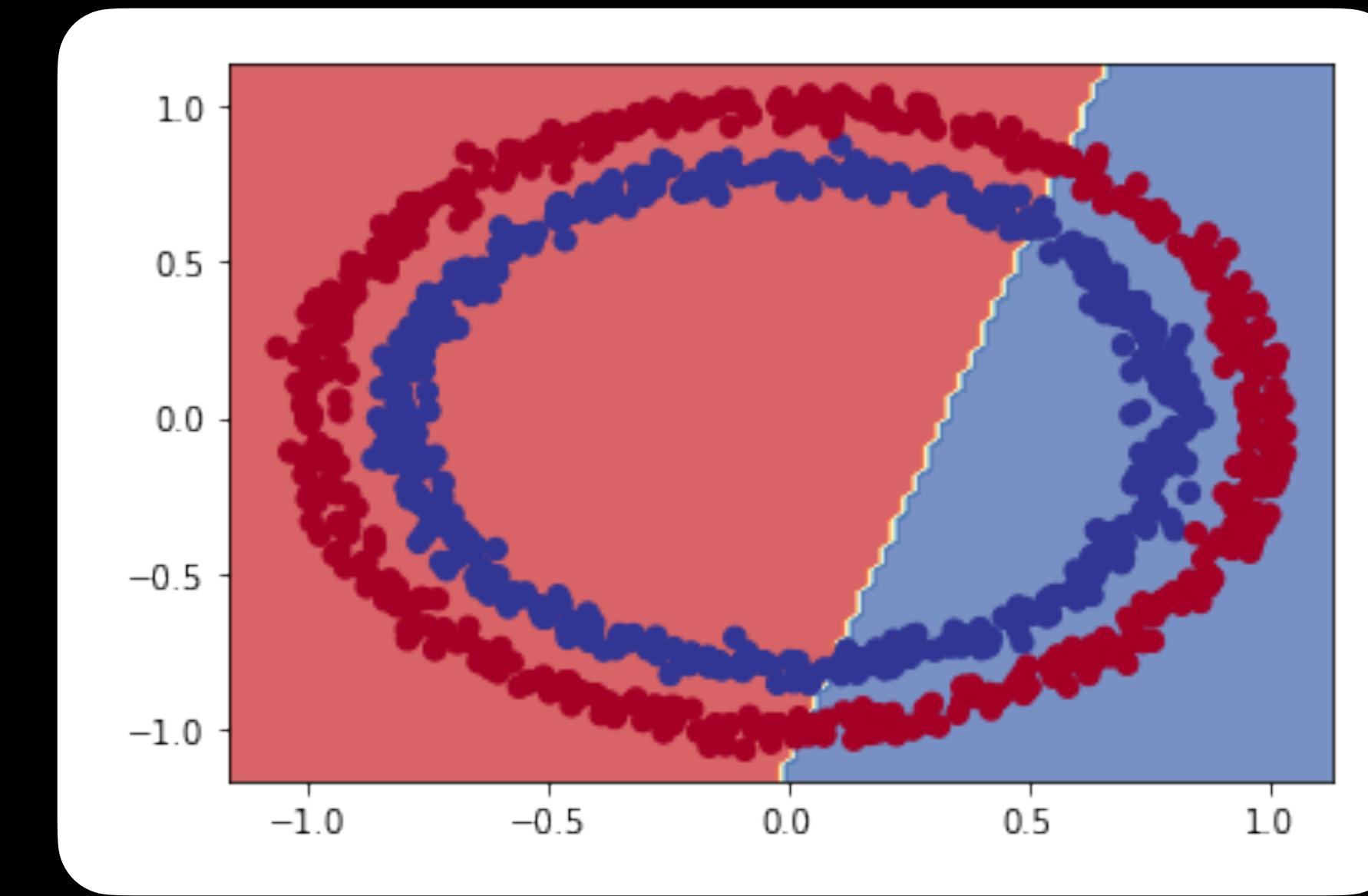
The missing piece: Non-linearity

🤔 “What could you draw if you had an unlimited amount of straight (linear) and non-straight (non-linear) lines?”



Linear data

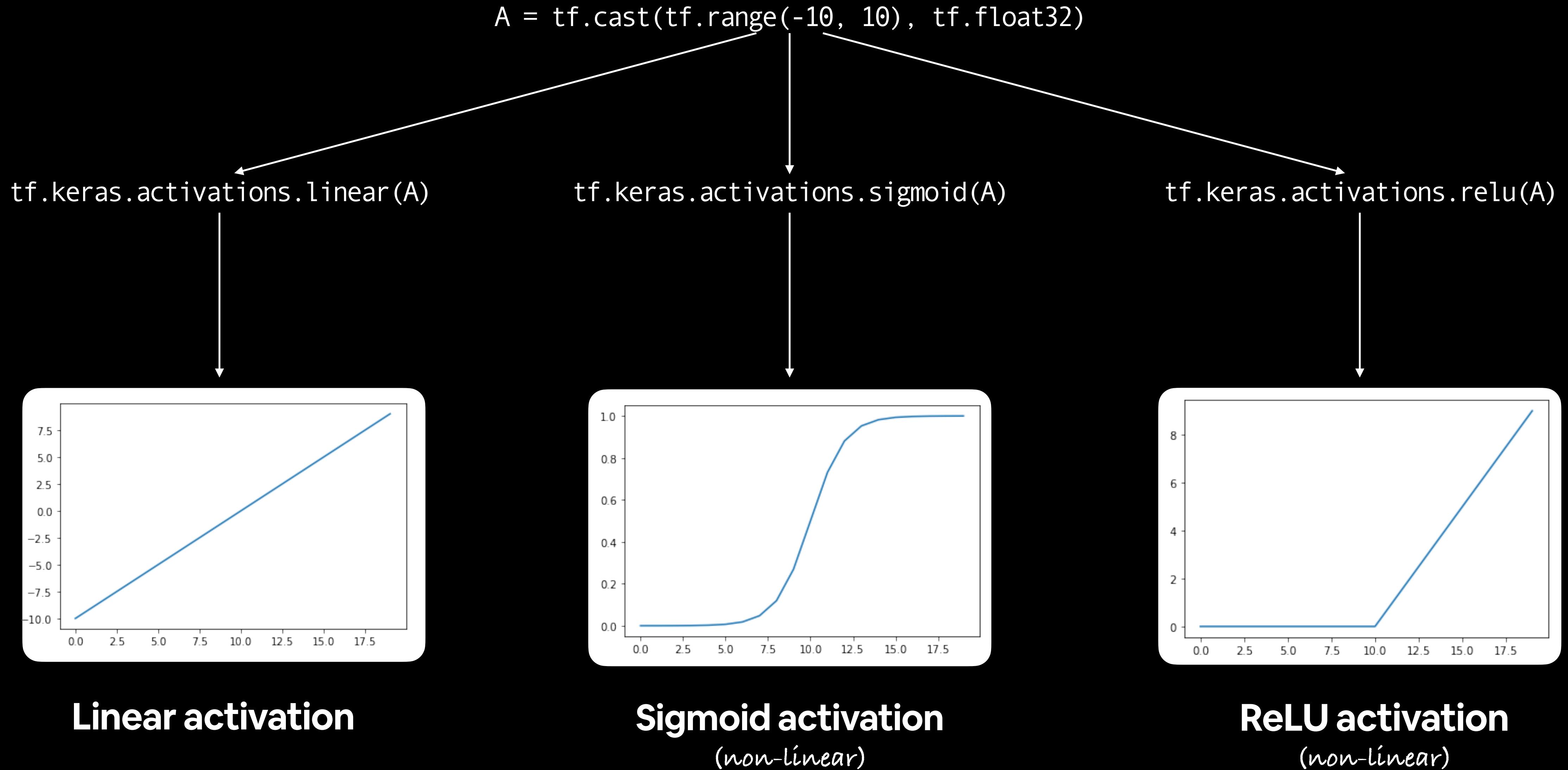
(possible to model with straight lines)



Non-linear data

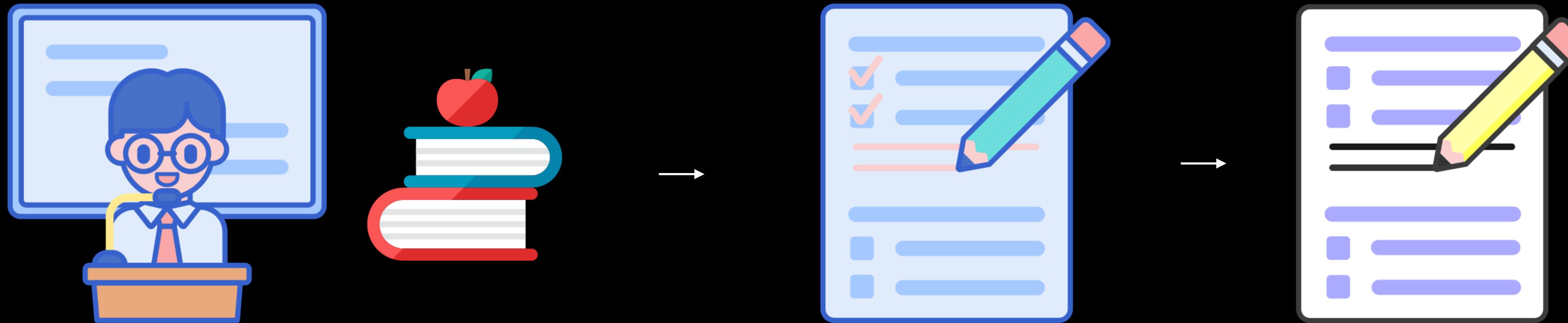
(not possible to model with straight lines)

The missing piece: Non-linearity



Three datasets

(possibly the most important concept
in machine learning...)



**Course materials
(training set)**

**Practice exam
(validation set)**

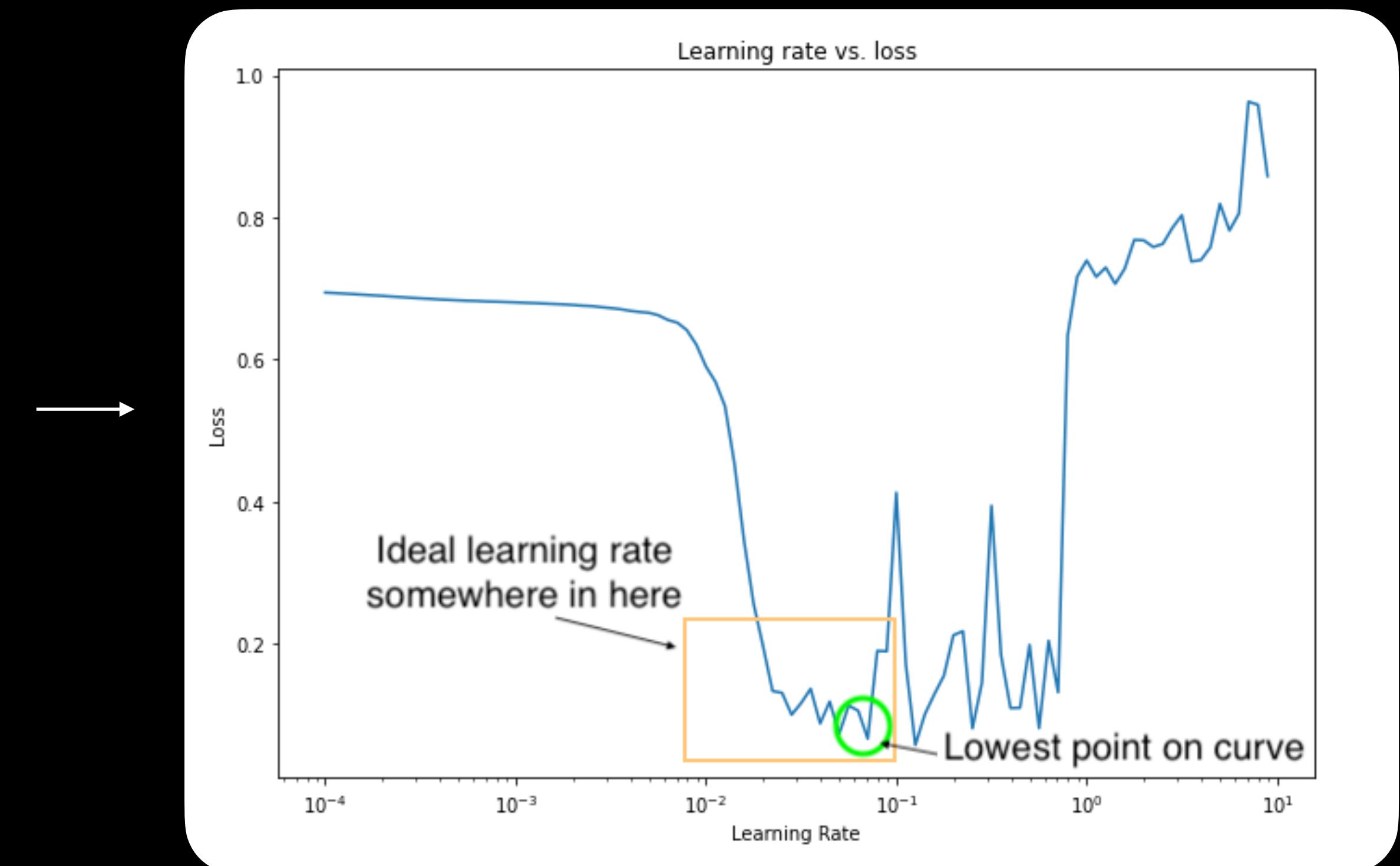
**Final exam
(test set)**

Generalization

The ability for a machine learning model to perform well on data it hasn't seen before.

Finding the ideal learning rate

```
# Model code  
...  
  
# Create a learning rate scheduler callback  
# (traverse set of learning rate values from 1e-4, increasing by 10**((epoch/20) every epoch)  
lr_scheduler = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-4 * 10**((epoch/20)))  
  
# Fit the model (passing the lr_scheduler callback)  
history = model.fit(X_train,  
                      y_train,  
                      epochs=100,  
                      callbacks=[lr_scheduler])  
  
# Plot the learning rate versus the loss  
lrs = 1e-4 * (10 ** (np.arange(100)/20))  
plt.figure(figsize=(10, 7))  
plt.semilogx(lrs, history.history["loss"])  
plt.xlabel("Learning Rate")  
plt.ylabel("Loss")  
plt.title("Learning rate vs. loss");
```



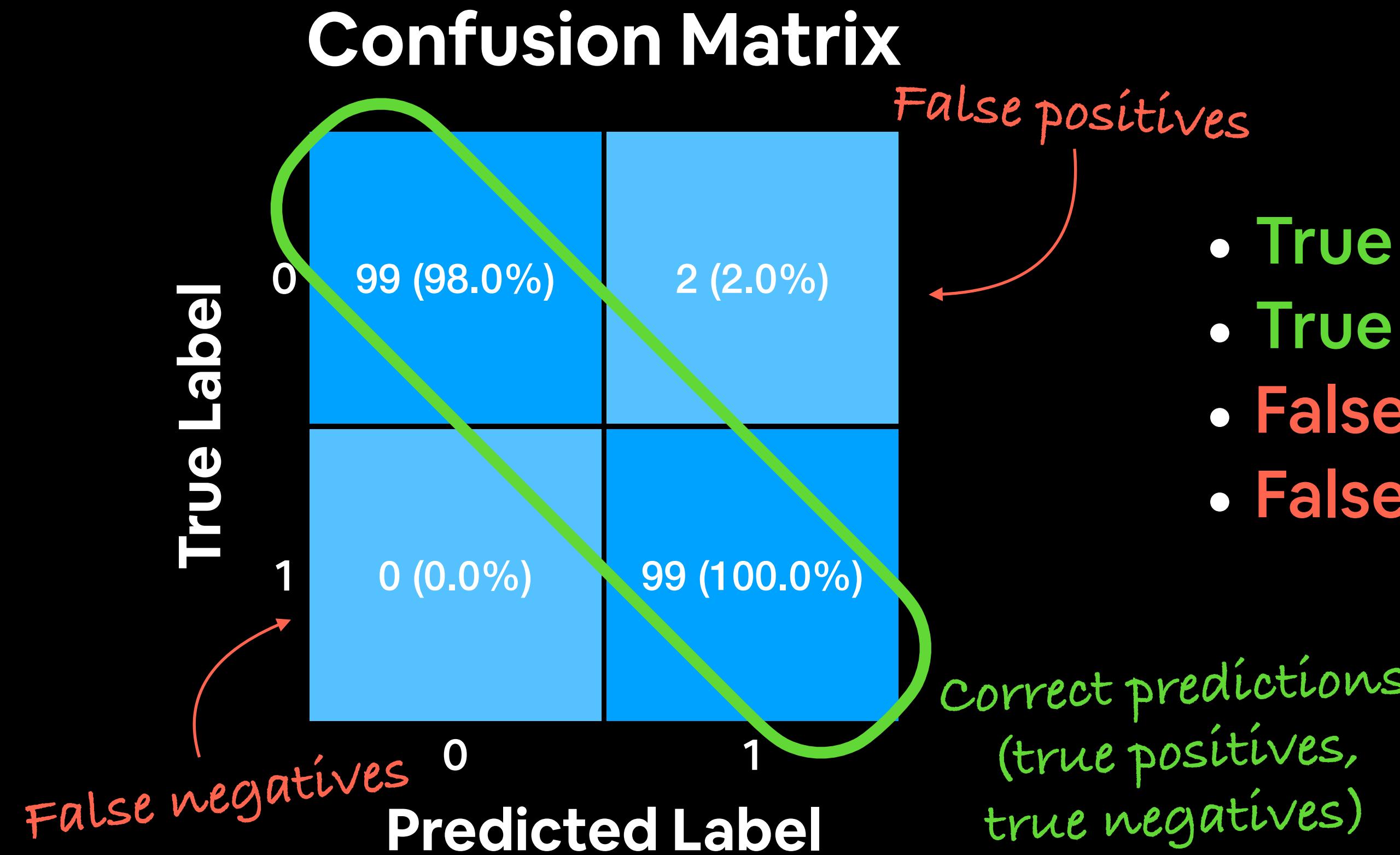
(some common)

Classification evaluation methods

Key: **tp** = True Positive, **tn** = True Negative, **fp** = False Positive, **fn** = False Negative

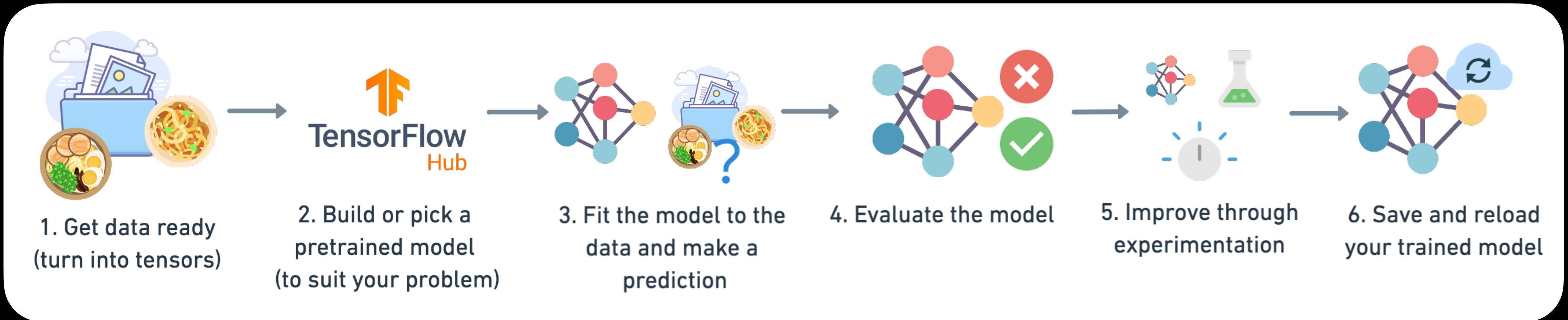
Metric Name	Metric Forumla	Code	When to use
Accuracy	Accuracy = $\frac{tp + tn}{tp + tn + fp + fn}$	<code>tf.keras.metrics.Accuracy()</code> or <code>sklearn.metrics.accuracy_score()</code>	Default metric for classification problems. Not the best for imbalanced classes.
Precision	Precision = $\frac{tp}{tp + fp}$	<code>tf.keras.metrics.Precision()</code> or <code>sklearn.metrics.precision_score()</code>	Higher precision leads to less false positives.
Recall	Recall = $\frac{tp}{tp + fn}$	<code>tf.keras.metrics.Recall()</code> or <code>sklearn.metrics.recall_score()</code>	Higher recall leads to less false negatives.
F1-score	F1-score = $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$	<code>sklearn.metrics.f1_score()</code>	Combination of precision and recall, usually a good overall metric for a classification model.
Confusion matrix	NA	Custom function or <code>sklearn.metrics.confusion_matrix()</code>	When comparing predictions to truth labels to see where model gets confused. Can be hard to use with large numbers of classes.

Anatomy of a confusion matrix



- **True positive** = model predicts 1 when truth is 1
- **True negative** = model predicts 0 when truth is 0
- **False positive** = model predicts 1 when truth is 0
- **False negative** = model predicts 0 when truth is 1

Steps in modelling with TensorFlow



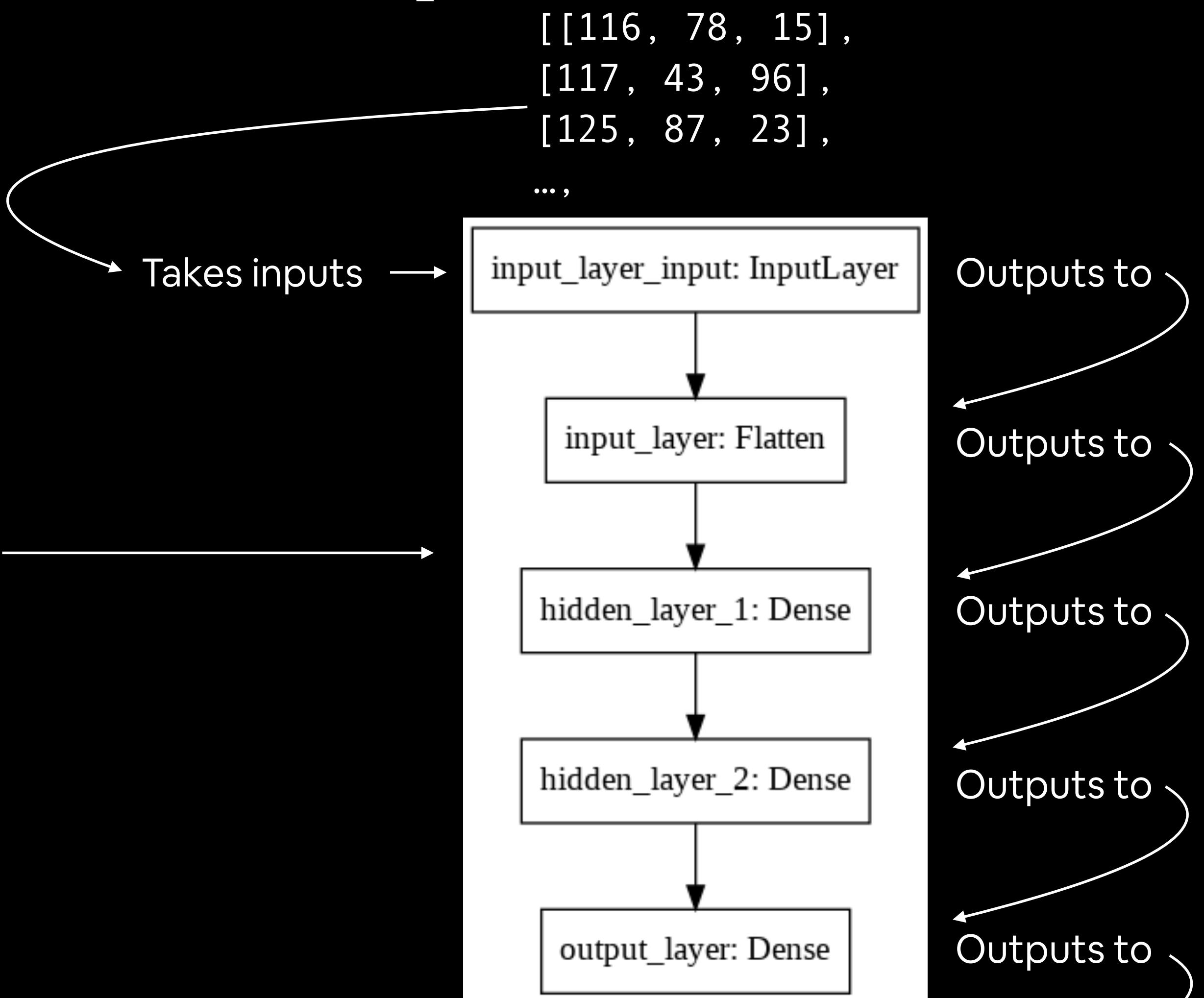
- ↓
1. Turn all data into numbers (neural networks can't handle strings)
 2. Make sure all of your tensors are the right shape (inputs and outputs)
 3. Scale features — normalize or standardize, neural networks tend to prefer normalization (values between 0 & 1)

Inputs and outputs

```
import tensorflow as tf

# Create the model
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28), name="input_layer"),
    tf.keras.layers.Dense(4, activation="relu", name="hidden_layer_1"),
    tf.keras.layers.Dense(4, activation="relu", name="hidden_layer_2"),
    tf.keras.layers.Dense(10, activation="softmax", name="output_layer")
], name="10_class_classification_model")

# Plot the model
tf.keras.utils.plot_model(model)
```



[[116, 78, 15],
 [117, 43, 96],
 [125, 87, 23],
 ...,

input_layer_input: InputLayer

input_layer: Flatten

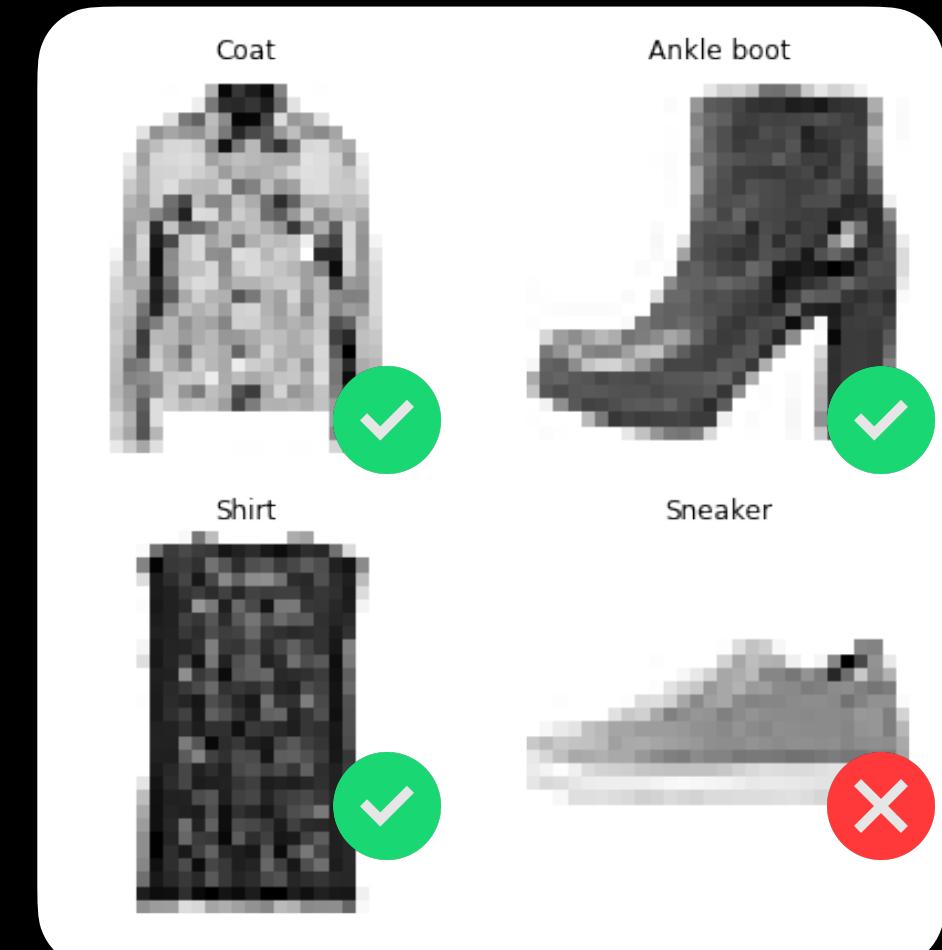
hidden_layer_1: Dense

hidden_layer_2: Dense

output_layer: Dense

[[0.983, 0.004, 0.013],
 [0.110, 0.889, 0.001],
 [0.023, 0.027, 0.985],
 ...,

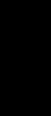
2. Show examples



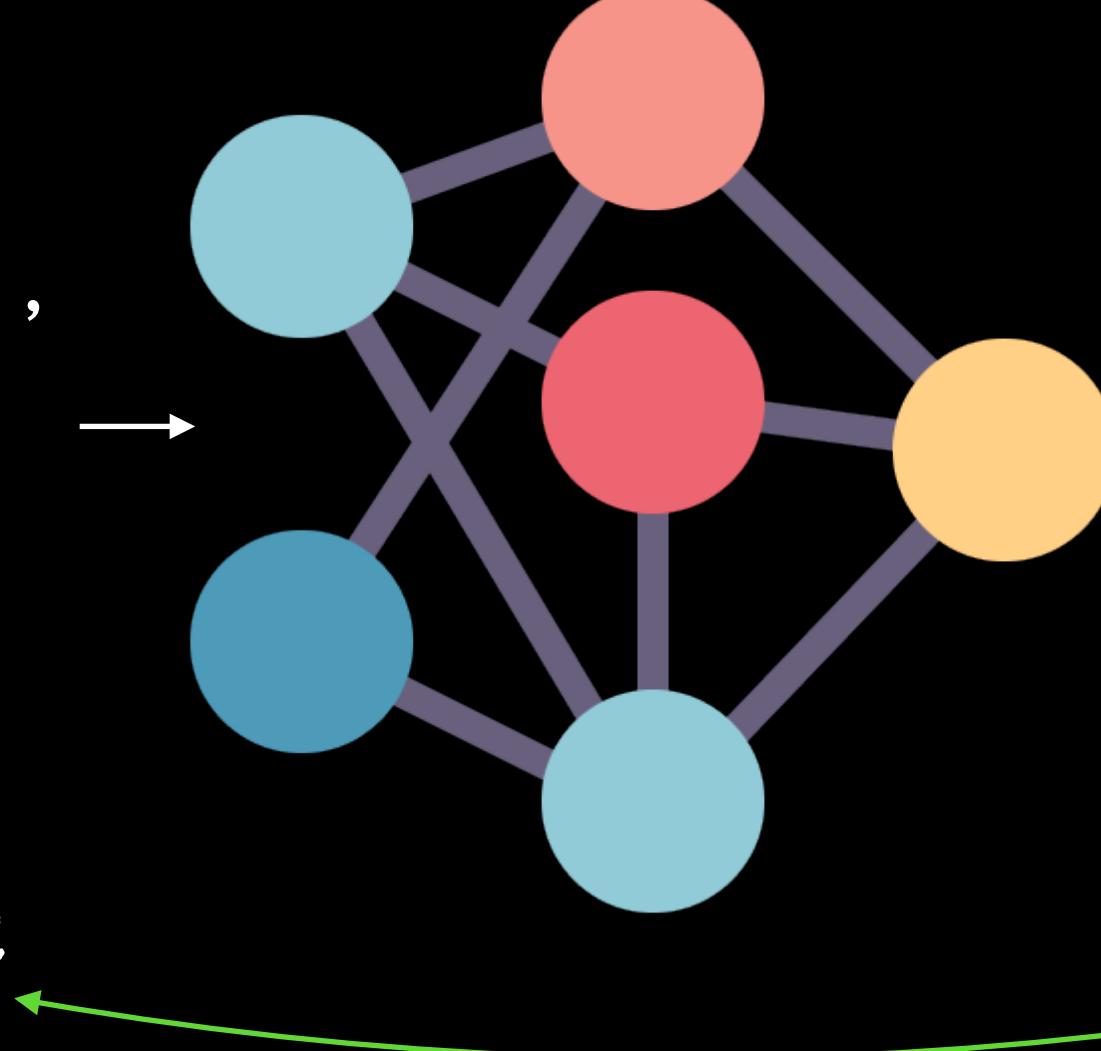
1. Initialise with random weights (only at beginning)

$[[0.092, 0.210, 0.415],$
 $[0.778, 0.929, 0.030], \rightarrow$
 $[0.019, 0.182, 0.555],$

$\dots,$



4. Repeat with more examples



$\rightarrow [[116, 78, 15],$
 $\rightarrow [117, 43, 96], \rightarrow$
 $[125, 87, 23],$
 $\dots,$

3. Update representation outputs (weights & biases)

Inputs

Numerical encoding

Learns representation patterns/features/weights)

Representation outputs

Outputs

Coat,
Ankle boot,
Shirt,
Sandal

The machine learning practitioner's motto

“Experiment, experiment, experiment”



*(try lots of things and see what
tastes good)*