

# Neural Network Classification with



# Where can you get help?

- Follow along with the code

What is a classification problem?  
A **classification problem** involves predicting whether something is one thing or another.

Problem type	What is it?	Example
Binary classification	Target can be one of two options, e.g. yes or no	Predict whether or not someone has heart disease based on their parameters.
Multi-class classification	Target can be one of more than two options	Decide whether a photo is of food, a person or a dog.
Multi-label classification	Target can be assigned more than one option	Predict what categories should be assigned to a Wikipedia article (e.g. mathematics, science & philosophy).

"Is this email spam or not spam?"  
To: daniel@mrbourke.com  
Hey Daniel.  
This deep learning course is incredible!  
I can't wait to use what I've learned!

"Is this a photo of sushi, steak or pizza?"  
To: daniel@mrbourke.com  
Hey Daniel.  
Congratulations! U win \$139239230

Not spam      Spam

Binary classification  
(one thing or another)

Multiclass classification  
(more than one thing or another)

"What tags should this article have?"

"If in doubt, run the code"

- Try it for yourself

```
n_samples=100, *, shuffle=True, noise=None, random_state=None, factor=0.8) : tuple[Any | list[NDArray[floating_NBit16_iadd_]] | ndarray]
```

Let's begin by making some data. We'll use the `make_circles()` function.

Make a large circle containing a smaller circle in 2d.

Read more in the User Guide <sample\_generators>.

Parameters

```
1 from sklearn.datasets import make_circles
2 # Make 1000 samples
3 n_samples = 1000
4
5 # Create circles
6 X, y = make_circles(n_samples,
7 noise=0.03, # a little bit of noise to the dots
8 random_state=42) # keep random state so we get the same values
```

Alright, now let's view the first 5 x and y values.

```
[ ] 1 print(f"First 5 X features:\n{X[:5]}")
2 print(f"\nFirst 5 y labels:\n{y[:5]}")
```

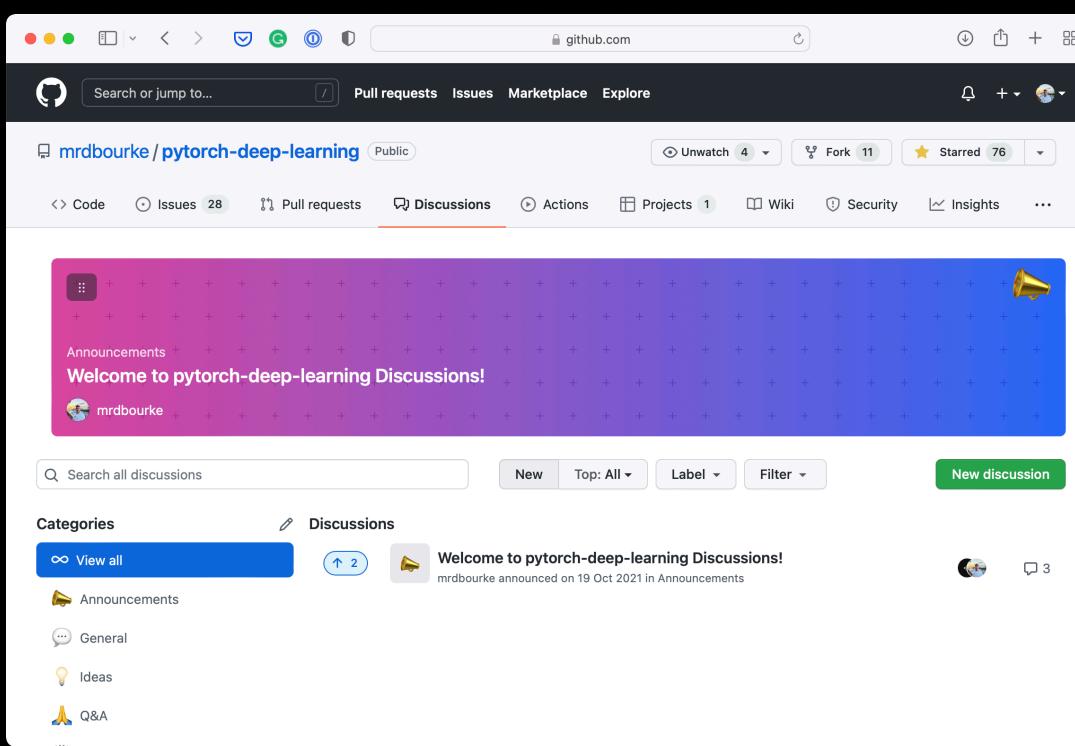
- Press SHIFT + CMD + SPACE to read the docstring

- Search for it

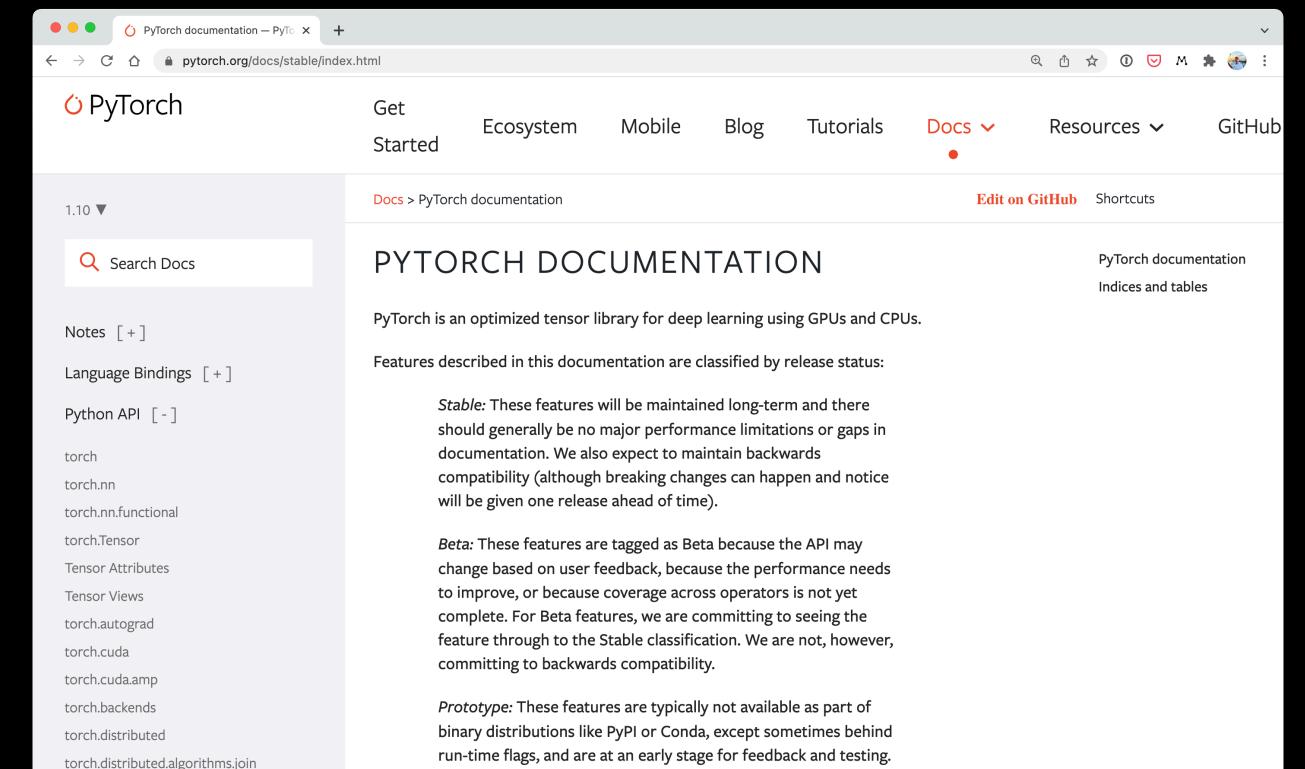


- Try again

- Ask



<https://www.github.com/mrdbourke/pytorch-deep-learning/discussions>



“What is a classification  
problem?”

# Example classification problems

“Is this email spam or not spam?”

To: [daniel@mrdourke.com](mailto:daniel@mrdourke.com)

Hey Daniel,

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

Not spam

To: [daniel@mrdourke.com](mailto:daniel@mrdourke.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 sidebar on the right contains several blue-highlighted tags: "Machine learning", "Representation learning", and "Artificial intelligence". These tags represent the classification categories for the article.

**Multiclass classification**  
*(more than one thing or another)*

*(multiple label options per sample)*

**Multilabel classification**

# Binary vs. Multi-class Classification



**Binary classification**  
(one thing or another)

**Multiclass classification**  
(more than one thing or  
another)

# 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, fit on and predict on
- Steps in modelling
  - Creating a model, setting a loss function and optimiser, creating a training loop, evaluating a model
- Saving and loading models
- Harnessing the power of non-linearity
- Different **classification** evaluation methods

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

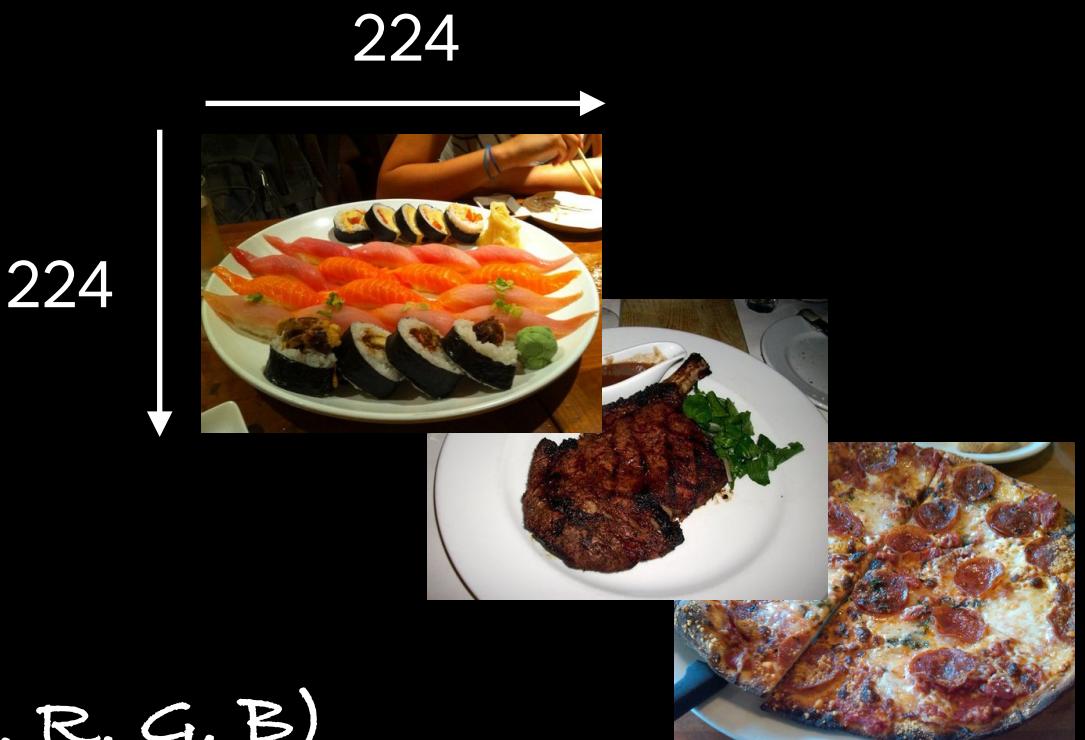
**How:**



# Classification inputs and outputs

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

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

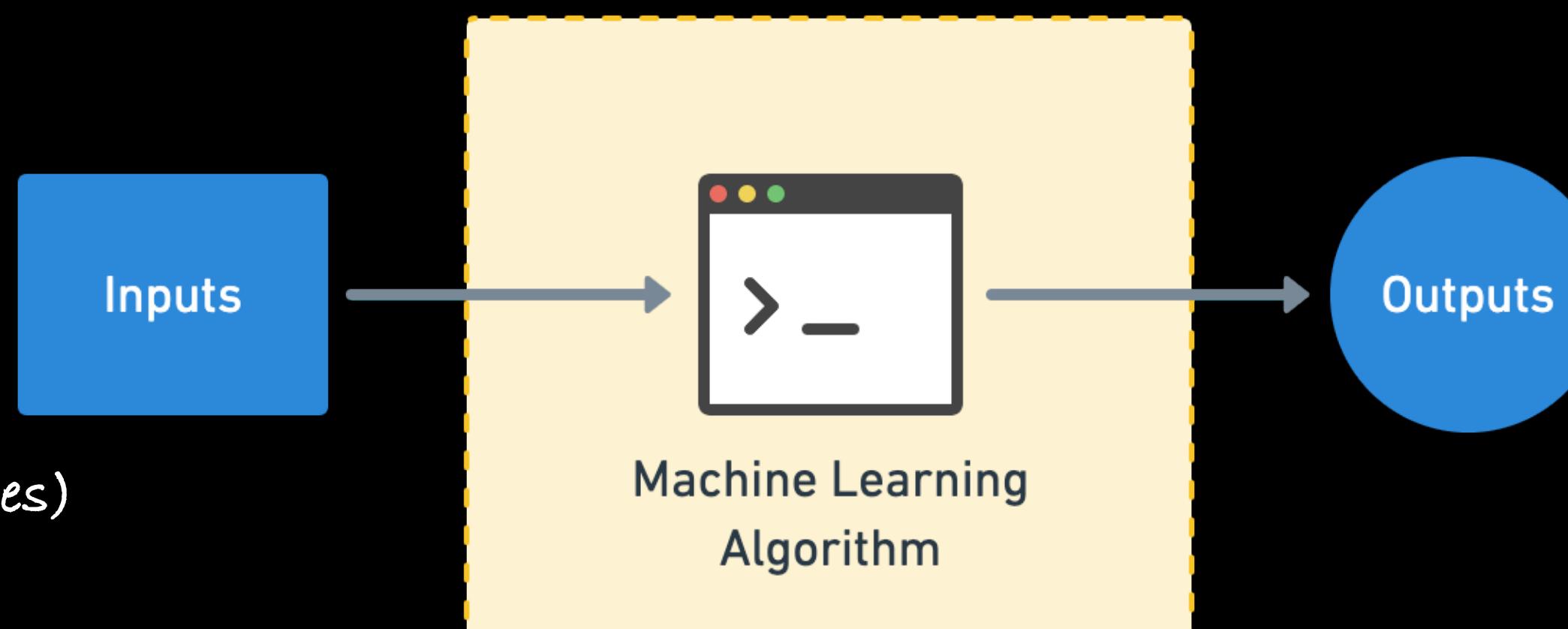


Sushi   
Steak   
Pizza

**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)

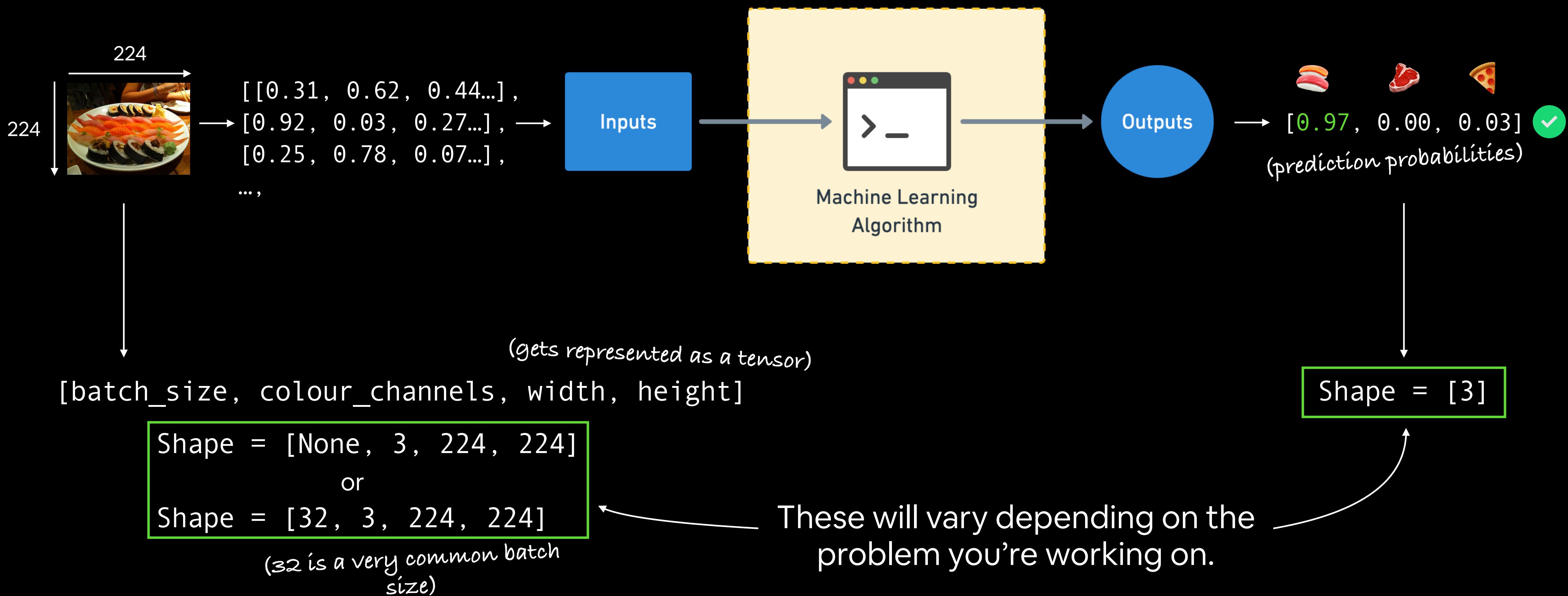
$\begin{bmatrix} [0.97, 0.00, 0.03], \\ [0.81, 0.14, 0.05], \\ [0.03, 0.07, 0.90], \\ \dots, \end{bmatrix}$

**Predicted output**

(comes from looking at lots  
of these)

# Input and output shapes

(for an image classification example)

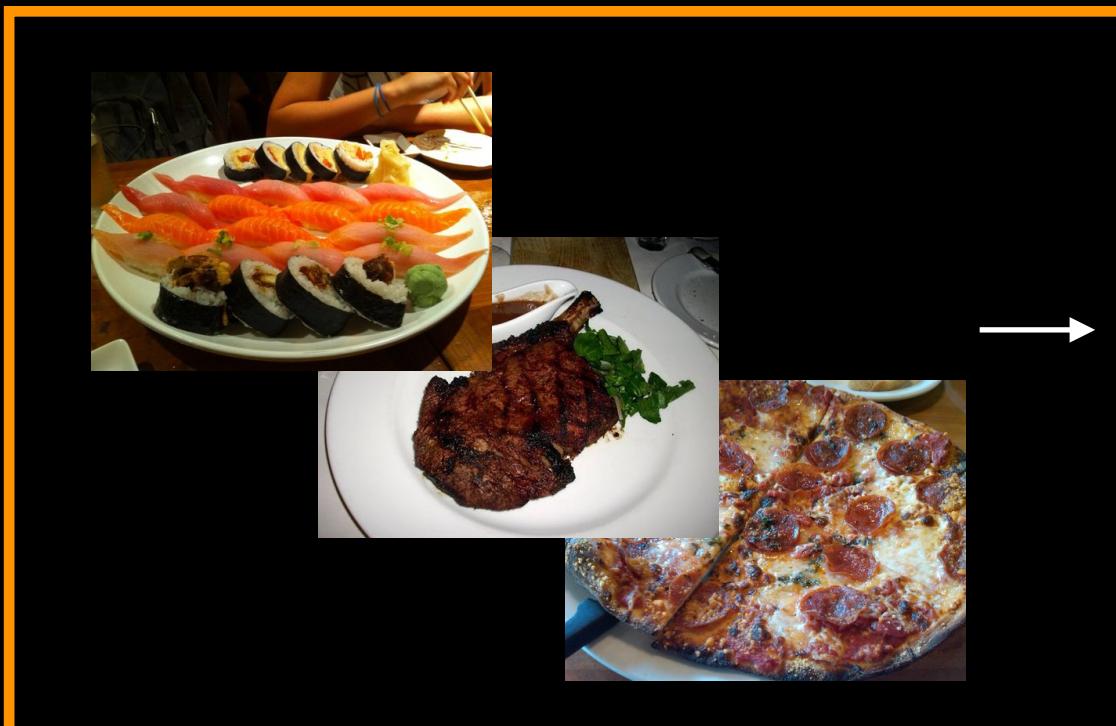


(typical)

# Architecture of a classification model

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

Hyperparameter	Binary Classification	Multiclass classification
Input layer shape ( <code>in_features</code> )	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 512	Same as binary classification
Output layer shape ( <code>out_features</code> )	1 (one class or the other)	1 per class (e.g. 3 for food, person or dog photo)
Hidden layer activation	Usually <code>ReLU</code> (rectified linear unit) but <a href="#">can be many others</a>	Same as binary classification
Output activation	<code>Sigmoid</code> ( <code>torch.sigmoid</code> in PyTorch)	<code>Softmax</code> ( <code>torch.softmax</code> in PyTorch)
Loss function	<code>Binary crossentropy</code> ( <code>torch.nn.BCELoss</code> in PyTorch)	Cross entropy ( <code>torch.nn.CrossEntropyLoss</code> in PyTorch)
Optimizer	<code>SGD</code> (stochastic gradient descent), <a href="#">Adam</a> (see <code>torch.optim</code> for more options)	Same as binary classification



```
1 # Create a model
2 model = nn.Sequential(
3     nn.Linear(in_features=3, out_features=100),
4     nn.Linear(in_features=100, out_features=100),
5     nn.ReLU(),
6     nn.Linear(in_features=100, out_features=3)
7 )
8
9 # Setup a loss function and optimizer
10 loss_fn = nn.BCEWithLogitsLoss()
11 optimizer = torch.optim.SGD(params=model.parameters(),
12                             lr=0.001)
13
14 # Training code...
15
16 # Testing code...
```



Sushi → Steak Pizza

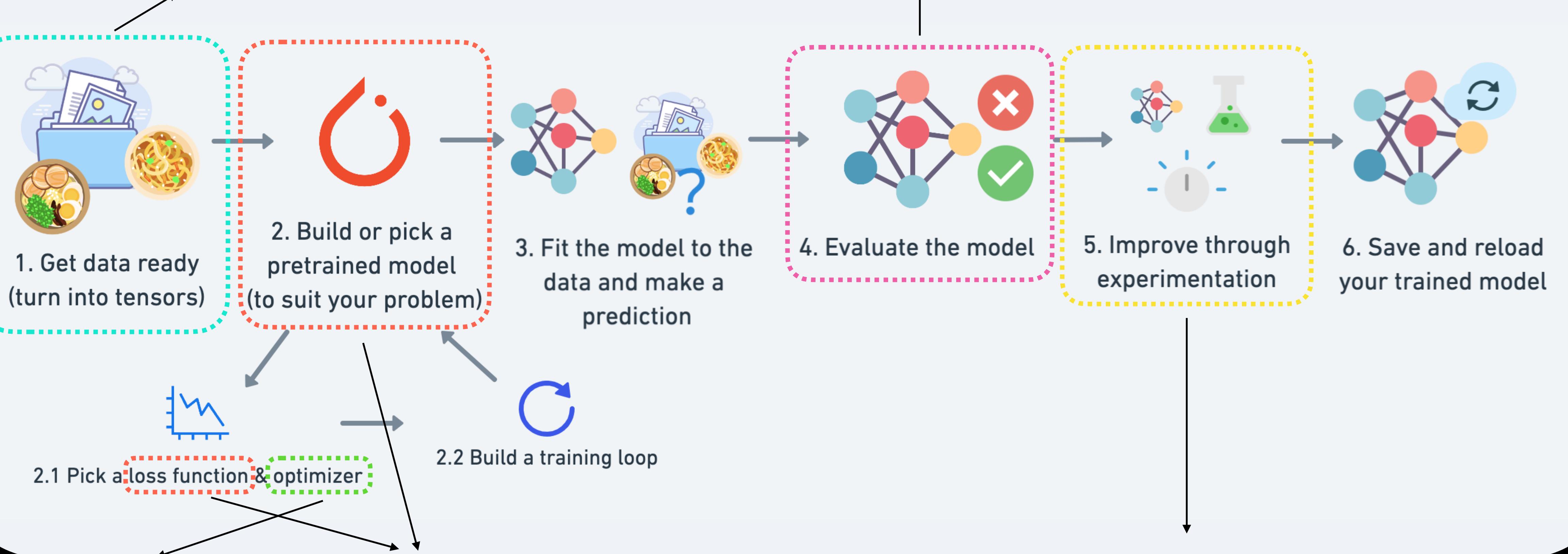
Let's code!

`torchvision.transforms`

`torch.utils.data.Dataset`

`torch.utils.data.DataLoader`

`torchmetrics`



# Improving a model

(from a model's perspective)

```
1 # Create a model
2 model = nn.Sequential(
3     nn.Linear(in_features=3, out_features=100),
4     nn.Linear(in_features=100, out_features=100),
5     nn.ReLU(),
6     nn.Linear(in_features=100, out_features=3)
7 )
8
9 # Setup a loss function and optimizer
10 loss_fn = nn.BCEWithLogitsLoss()
11 optimizer = torch.optim.SGD(params=model.parameters(),
12                             lr=0.001)
13
14 # Training code...
15 epochs = 10
16
17 # Testing code...
```

Smaller model



```
1 # Create a larger model
2 model = nn.Sequential(
3     nn.Linear(in_features=3, out_features=128),
4     nn.ReLU(),
5     nn.Linear(in_features=128, out_features=256),
6     nn.ReLU(),
7     nn.Linear(in_features=256, out_features=128),
8     nn.ReLU(),
9     nn.Linear(in_features=128, out_features=3)
10 )
11
12 # Setup a loss function and optimizer
13 loss_fn = nn.BCEWithLogitsLoss()
14 optimizer = torch.optim.Adam(params=model.parameters(),
15                             lr=0.0001)
16
17 # Training code...
18 epochs = 100
19
20 # Testing code...
```

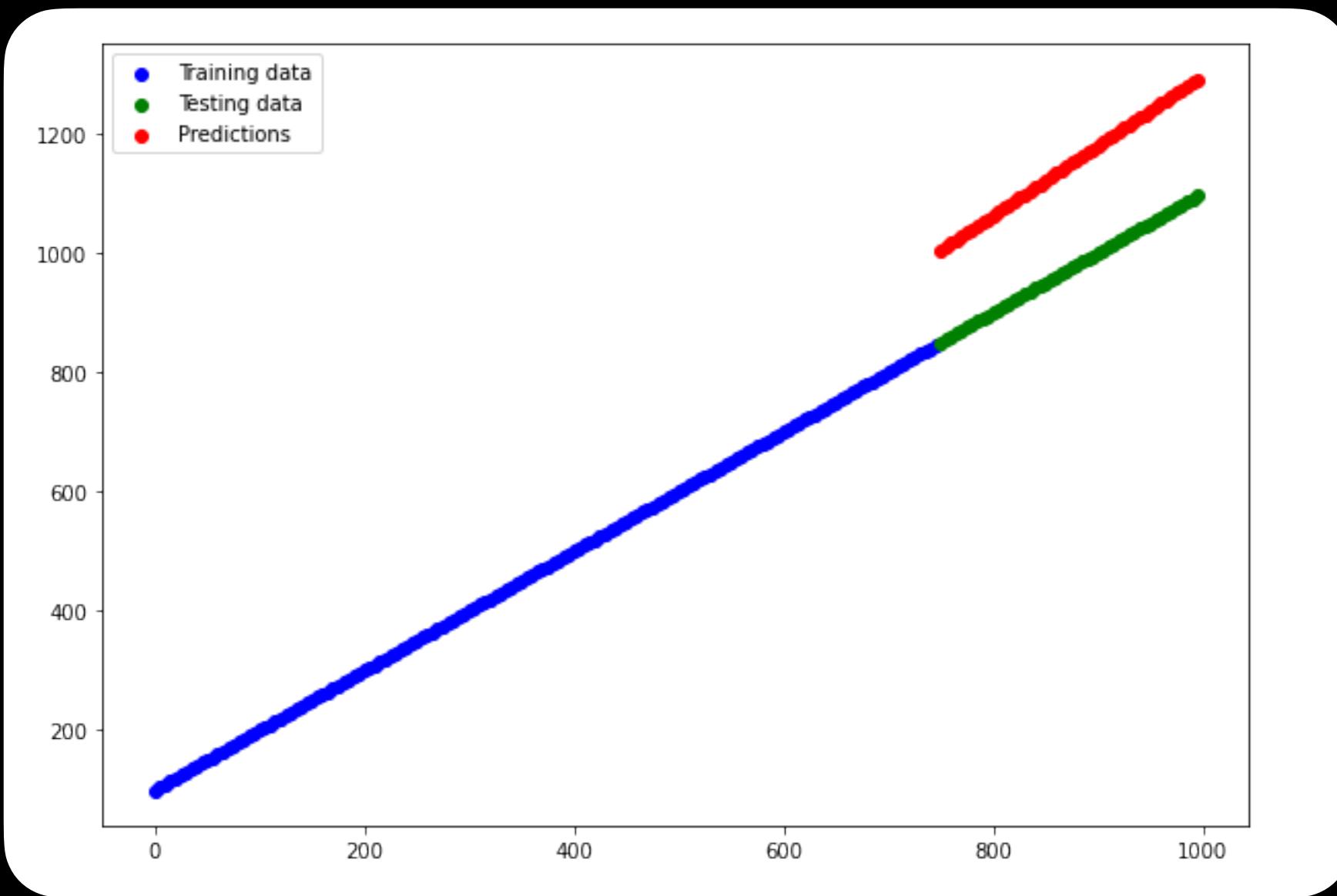
Larger model

## Common ways to improve a deep model:

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

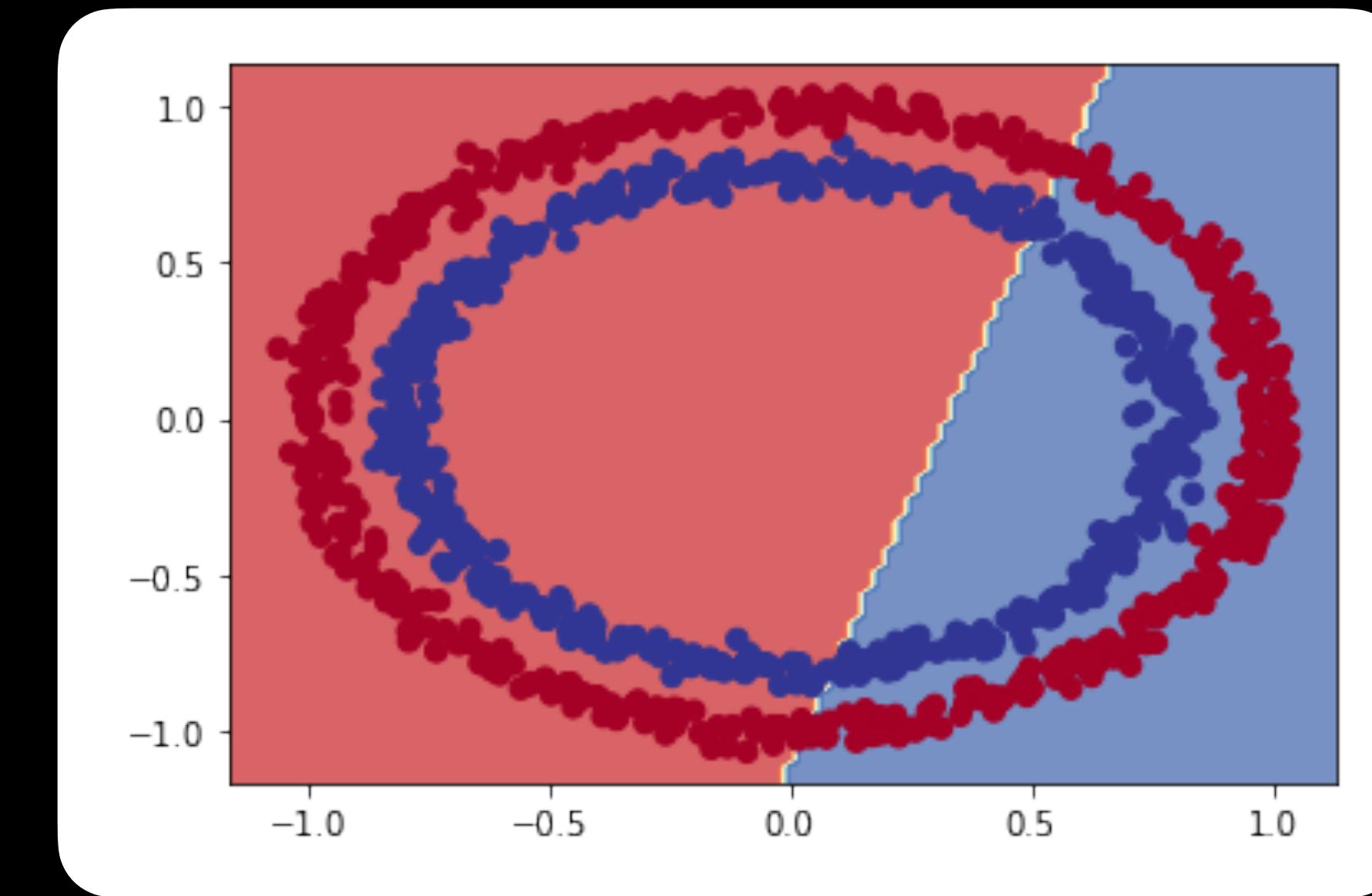
# The missing piece: Non-linearity

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



**Linear data**

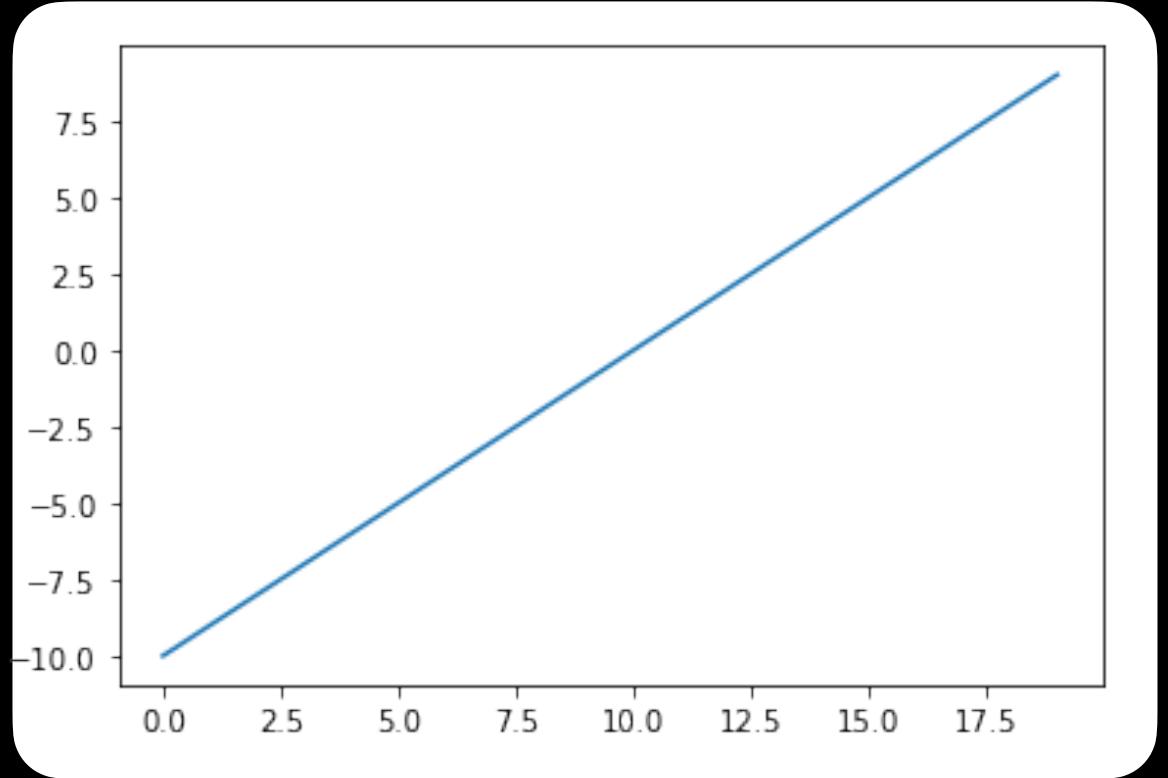
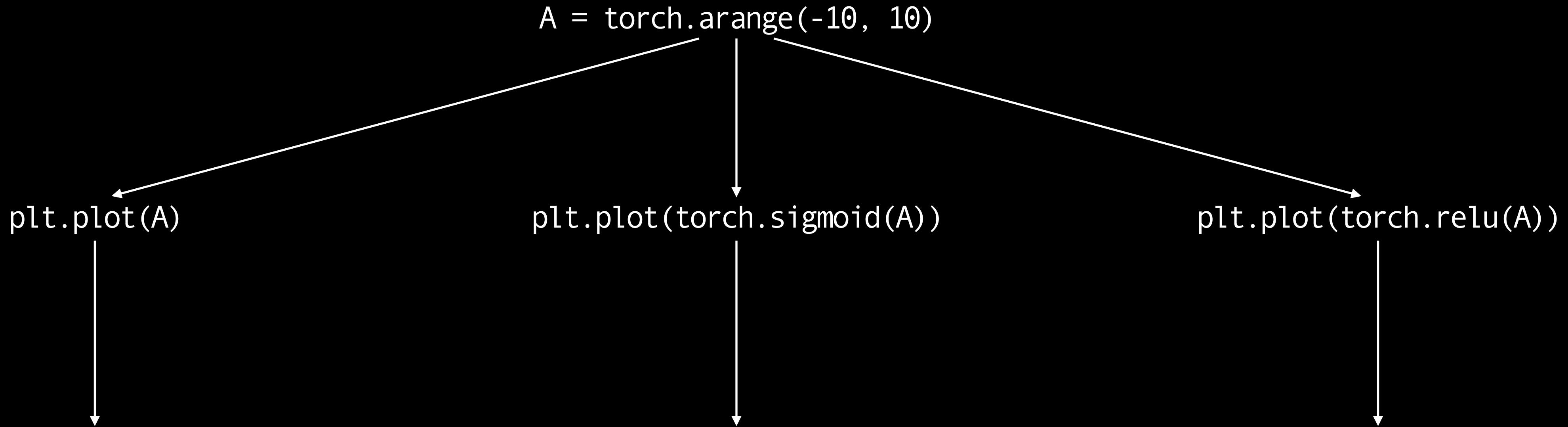
(possible to model with straight lines)



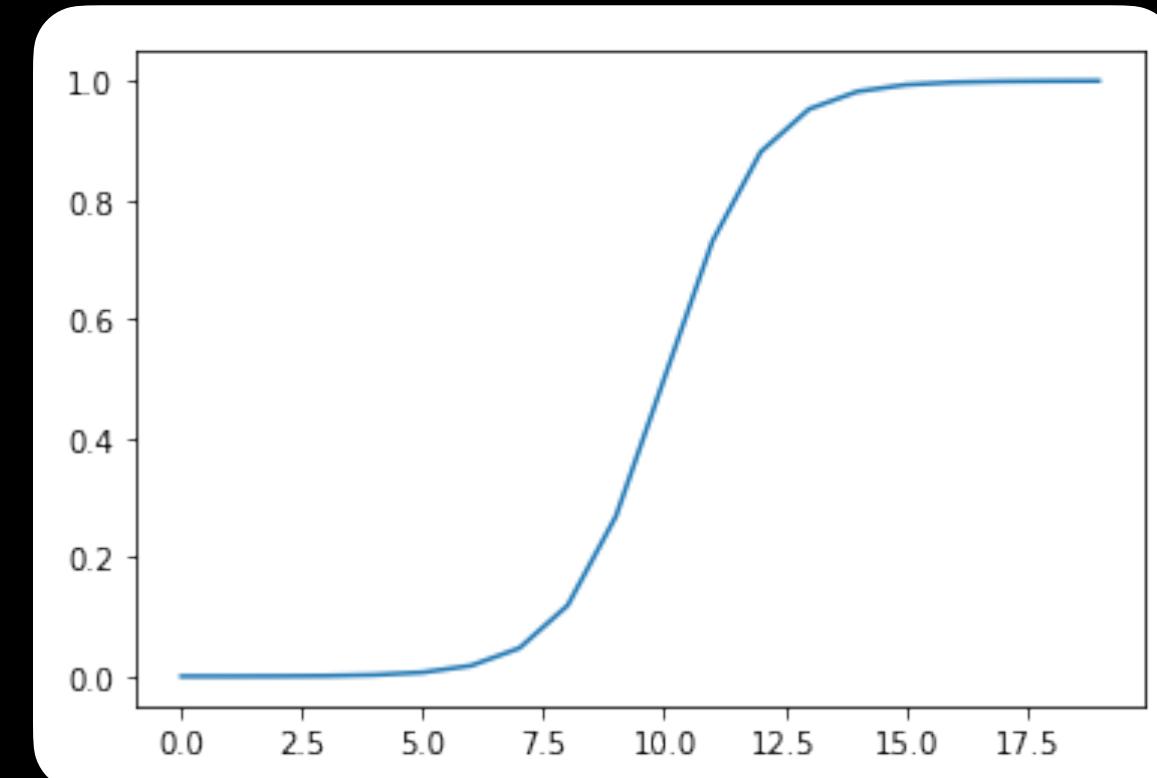
**Non-linear data**

(not possible to model with straight lines)

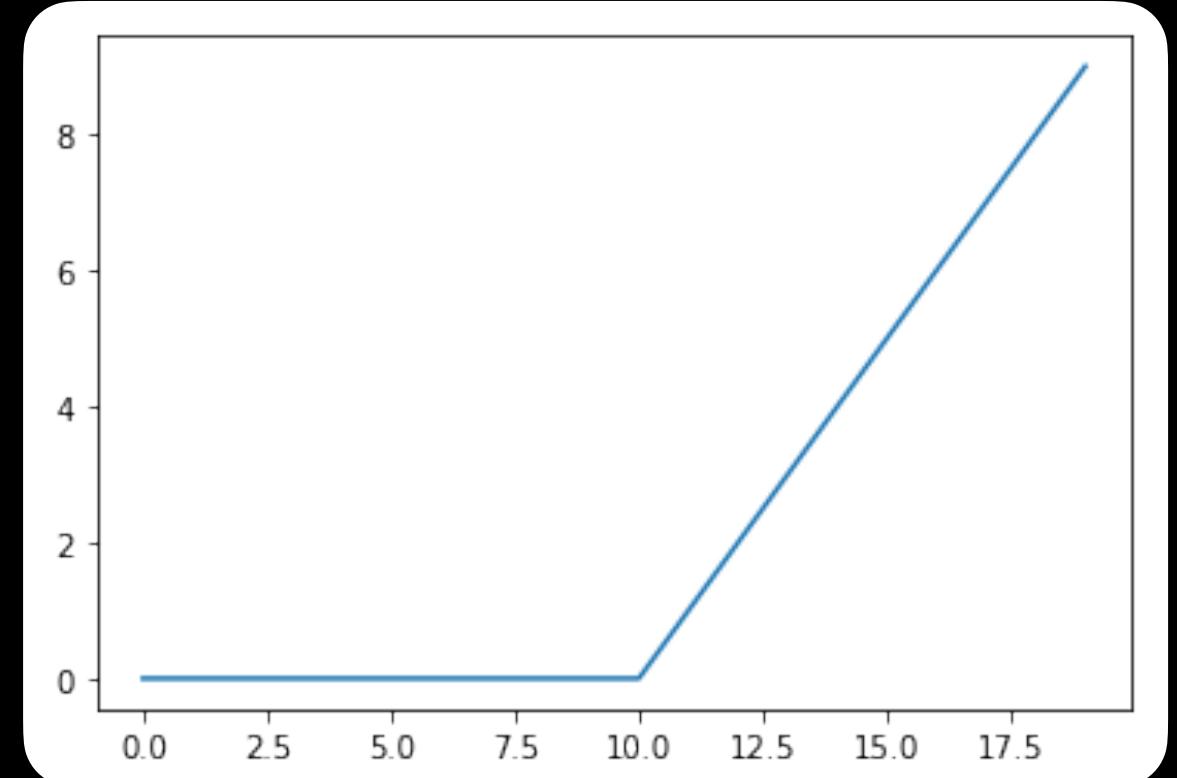
# The missing piece: Non-linearity



**Linear activation**  
(same as original values)



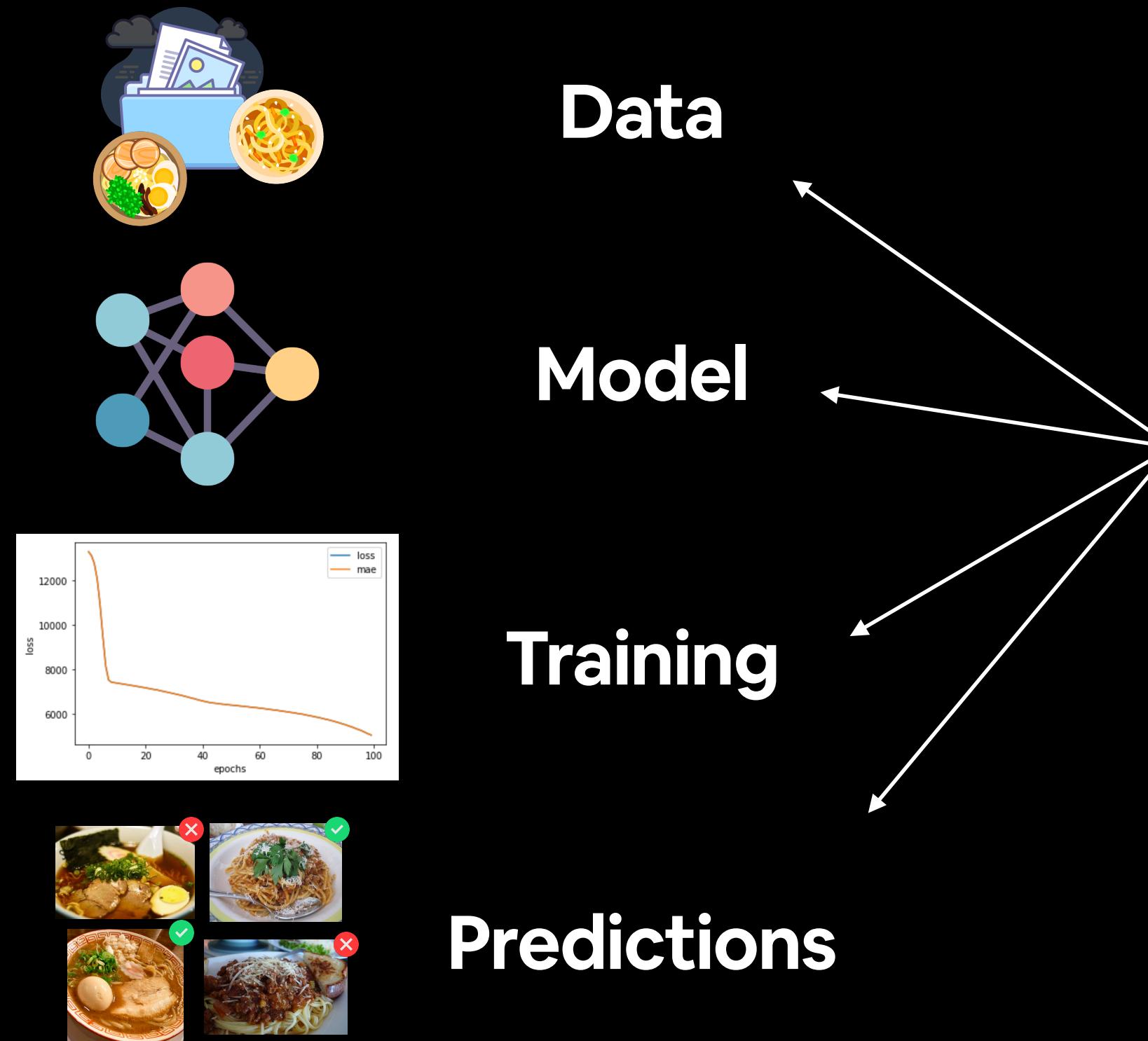
**Sigmoid activation**  
(non-linear)



**ReLU activation**  
(non-linear)

# The machine learning explorer's motto

“Visualize, visualize, visualize”



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

# The machine learning practitioner's motto

“Experiment, experiment, experiment”



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

# Steps in modelling with PyTorch

```
1 # Create a model
2 model = nn.Sequential(
3     nn.Linear(in_features=3, out_features=100),
4     nn.Linear(in_features=100, out_features=100),
5     nn.ReLU(),
6     nn.Linear(in_features=100, out_features=3)
7 )
8
9 # Setup a loss function and optimizer
10 loss_fn = nn.BCEWithLogitsLoss()
11 optimizer = torch.optim.SGD(params=model.parameters(),
12                             lr=0.001)
13
14 # Training code...
15
16 # Testing code...
```

1. Construct or import a pretrained model relevant to your problem
2. Prepare the loss function, optimizer and training loop
  - **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.
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?)

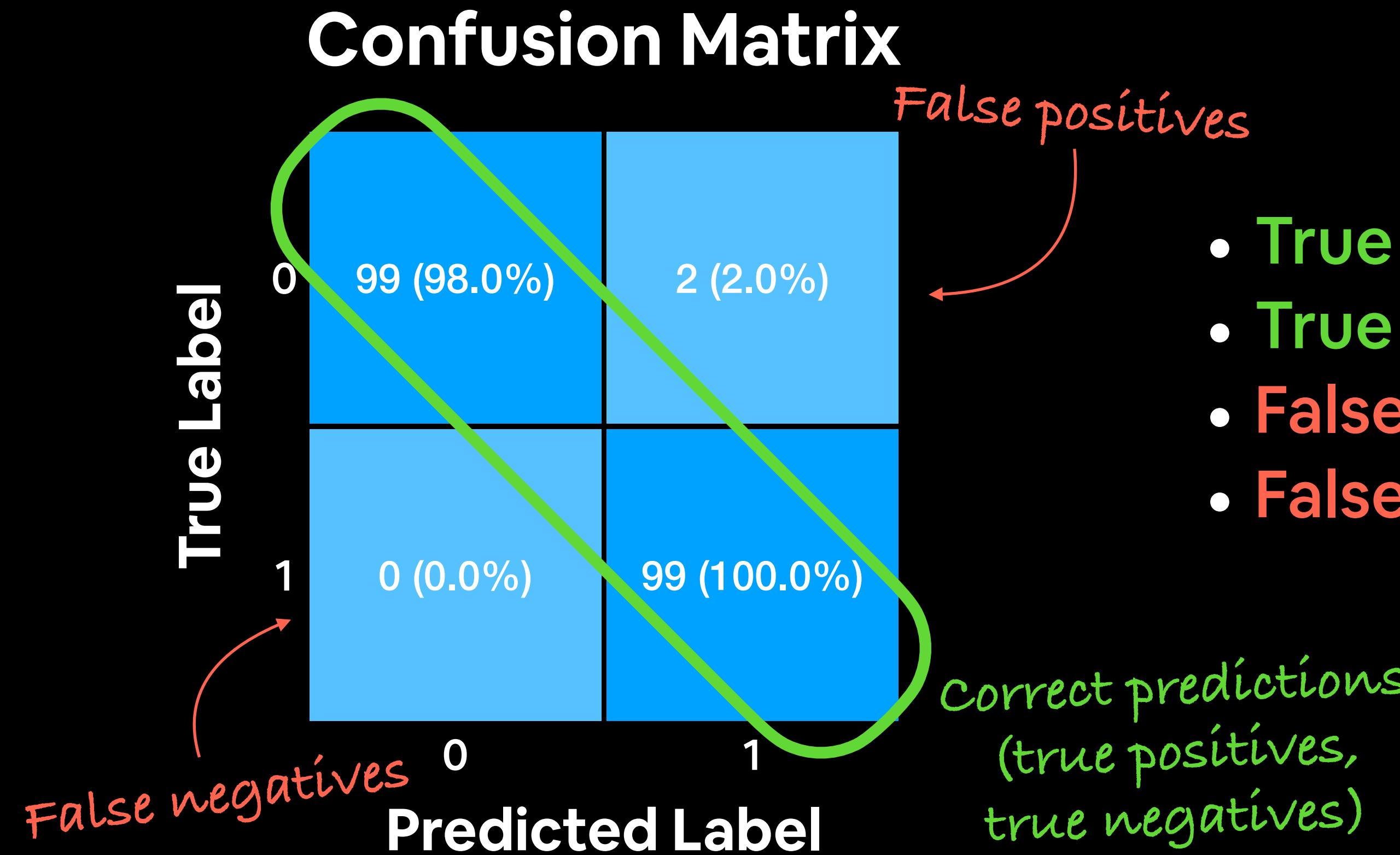
(some common)

# 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	<b>Accuracy</b> = $\frac{tp + tn}{tp + tn + fp + fn}$	<code>torchmetrics.Accuracy()</code> or <code>sklearn.metrics.accuracy_score()</code>	Default metric for classification problems. Not the best for imbalanced classes.
Precision	<b>Precision</b> = $\frac{tp}{tp + fp}$	<code>torchmetrics.Precision()</code> or <code>sklearn.metrics.precision_score()</code>	Higher precision leads to less false positives.
Recall	<b>Recall</b> = $\frac{tp}{tp + fn}$	<code>torchmetrics.Recall()</code> or <code>sklearn.metrics.recall_score()</code>	Higher recall leads to less false negatives.
F1-score	<b>F1-score</b> = $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$	<code>torchmetrics.F1Score()</code> or <code>sklearn.metrics.f1_score()</code>	Combination of precision and recall, usually a good overall metric for a classification model.
Confusion matrix	NA	<code>torchmetrics.ConfusionMatrix()</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

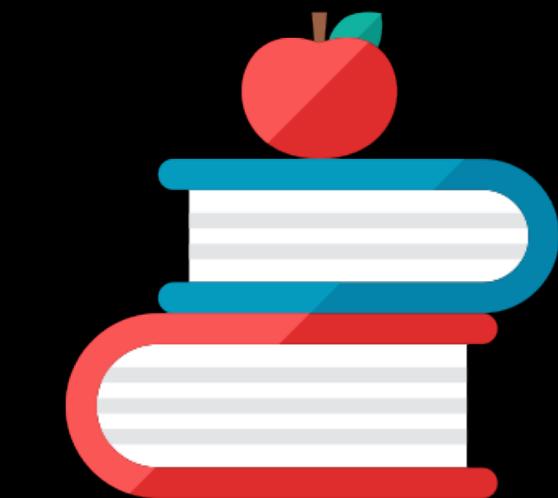
# Three datasets

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

Model learns patterns from here



**Course materials  
(training set)**



**Practice exam  
(validation set)**

Tune model patterns



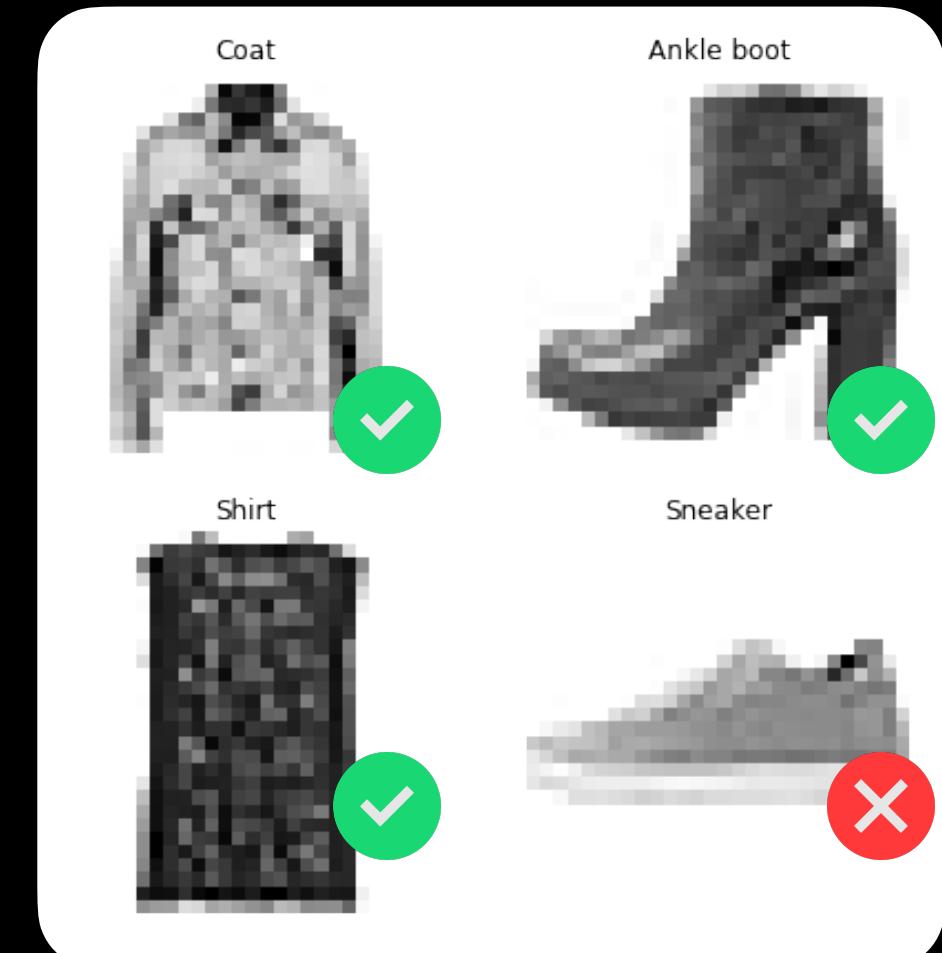
**Final exam  
(test set)**

See if the model is ready for the wild

## Generalization

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

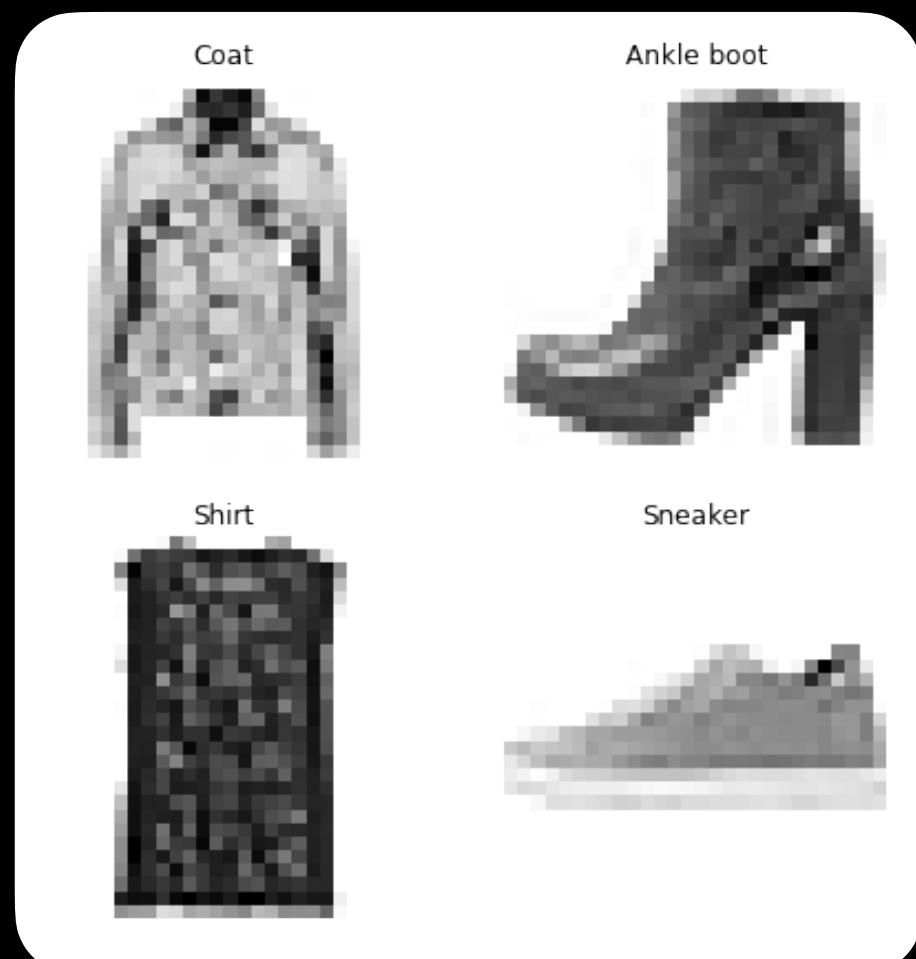
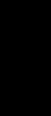
## 2. Show examples



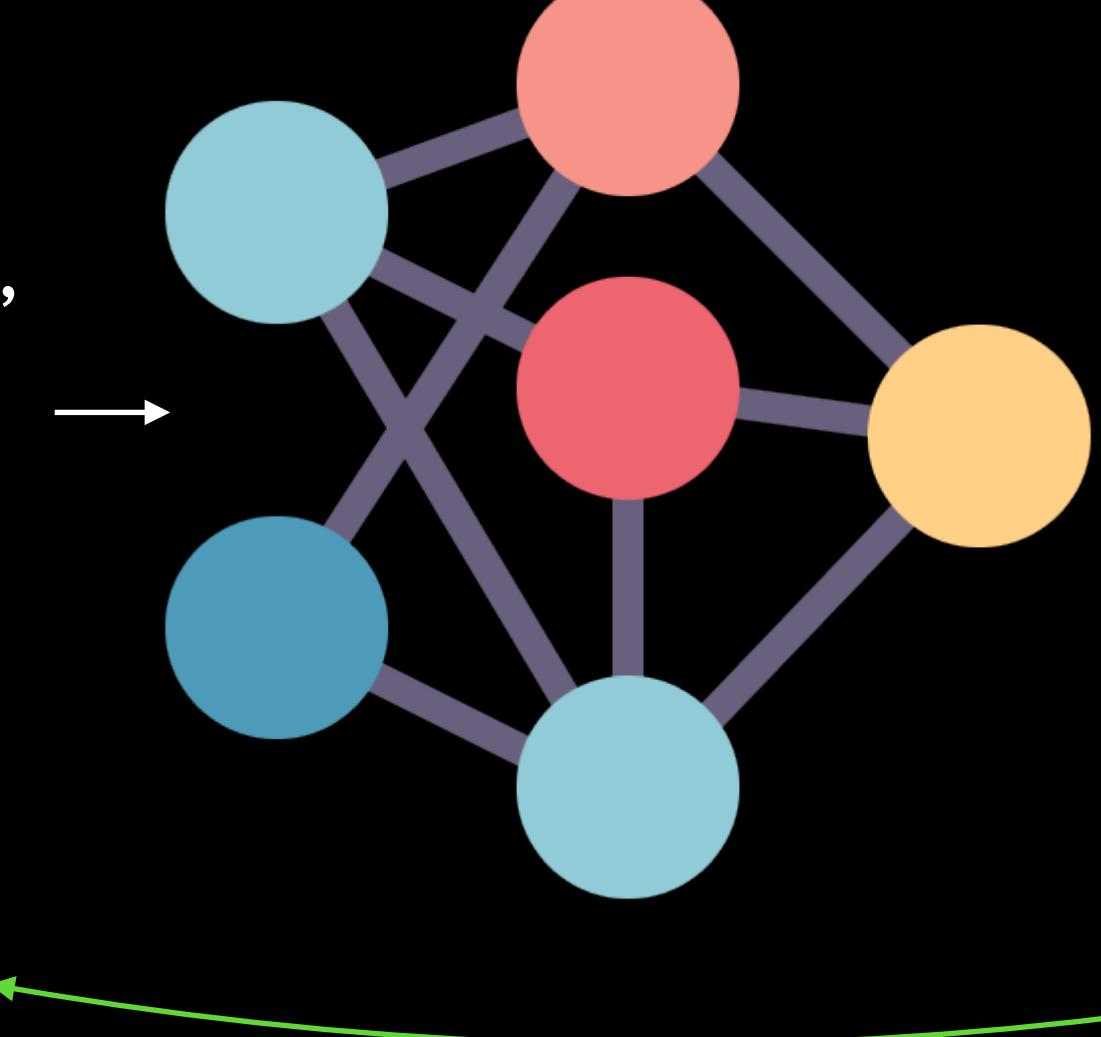
1. Initialise with random weights (only at beginning)

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

$\dots,$



4. Repeat with more examples



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

3. Update representation outputs (weights & biases)

Coat,  
Ankle boot,  
Shirt,  
Sandal

Inputs

Numerical encoding

Learns representation patterns/features/weights)

Representation outputs

Outputs

# Resources

This course

## Course materials

A screenshot of a GitHub repository page for 'mrdbourke/pytorch-deep-learning'. The repository has 28 issues, 11 forks, and 76 stars. The 'Code' tab is selected, showing a list of files and commits. Notable commits include 'update exercises', 'Update make\_docs.yml', and 'add exercises and solutions for 01'. The repository is described as 'Materials for upcoming beginner-friendly PyTorch course (work in progress)'.

<https://www.github.com/mrdbourke/pytorch-deep-learning>

## Course Q&A

A screenshot of the GitHub Discussions page for the same repository. The 'Discussions' tab is selected. It features a prominent purple banner with the text 'Welcome to pytorch-deep-learning Discussions!' and a megaphone icon. Below the banner, there's a search bar and a 'New discussion' button. Categories listed include 'Announcements', 'General', 'Ideas', and 'Q&A'.

<https://www.github.com/mrdbourke/pytorch-deep-learning/discussions>

## Course online book

A screenshot of the 'Zero to Mastery Learn PyTorch for Deep Learning' online book homepage. It welcomes visitors to the 'Learn PyTorch for Deep Learning book (work in progress)'. It explains that the course will teach foundations of deep learning and PyTorch, and that the videos are based on the contents of the online book. It includes sections for 'Expected release date' (Early 2022), 'Get updates', and a link to the course GitHub.

<https://learnpytorch.io>

## PyTorch website & forums

The official PyTorch website homepage. It features a large purple background with the text 'FROM RESEARCH TO PRODUCTION'. Below it, it says 'An open source machine learning framework that accelerates the path from research prototyping to production deployment.' A 'Install >' button is present. At the bottom, it mentions 'PyTorch 1.10 Release, including CUDA Graphs APIs, TorchScript improvements'.

All things PyTorch

The PyTorch forum interface. It shows a list of categories: 'vision', 'nlp', 'Uncategorized', 'autograd', 'mixed-precision', 'C++', and 'distributed'. Each category has a count of topics per month. Below the categories, individual posts are listed with details like author, topic, and time since posted.