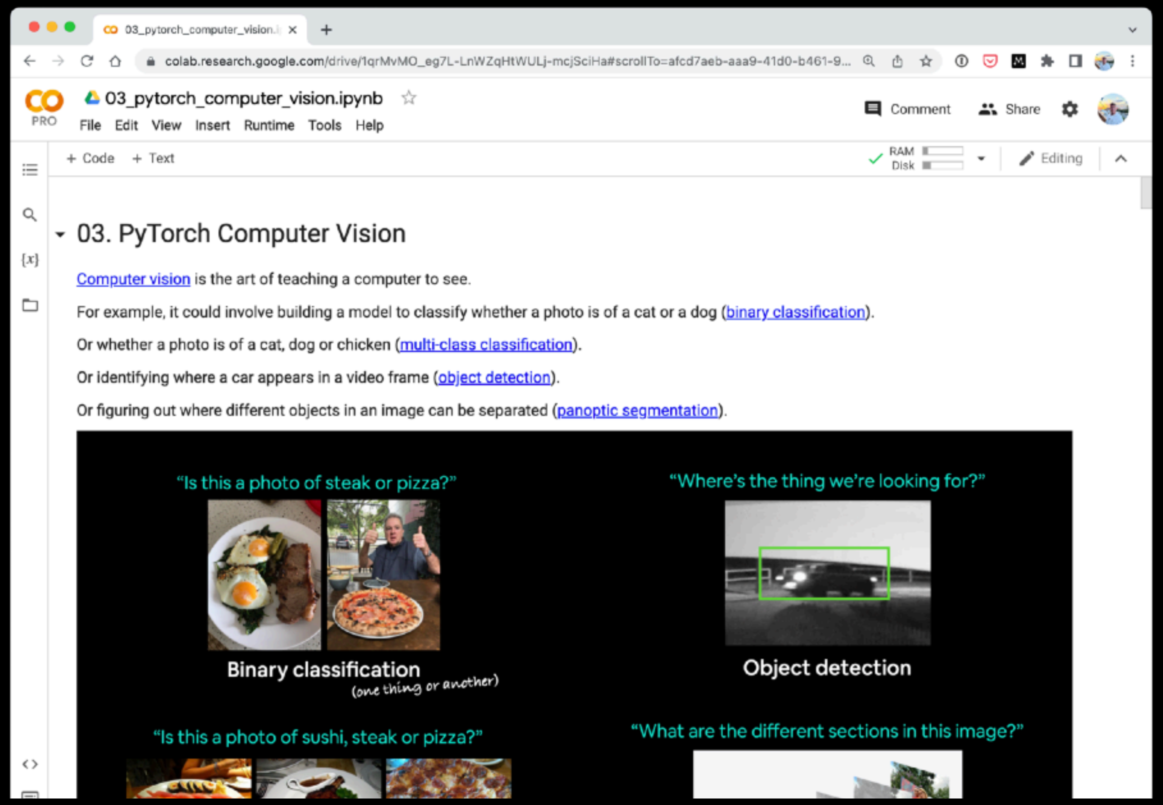
Computer Vision & Convolutional Neural Networks with

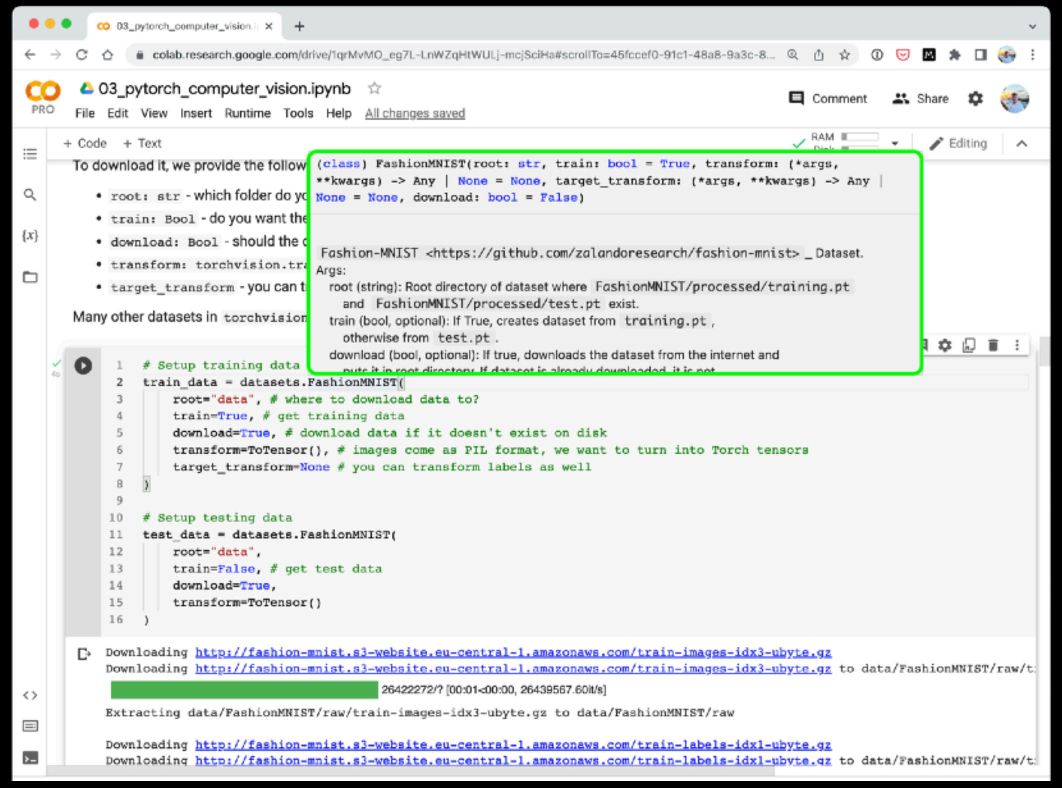
PyTorch

Where can you get help?

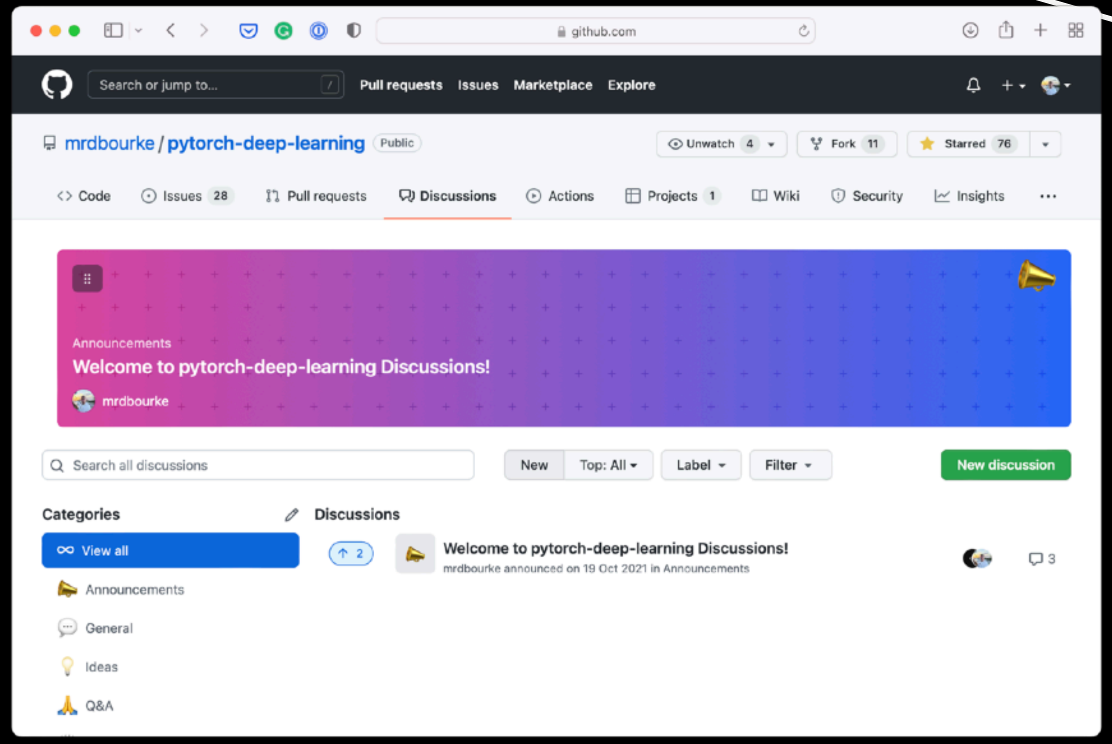
"If in doubt, run the code"

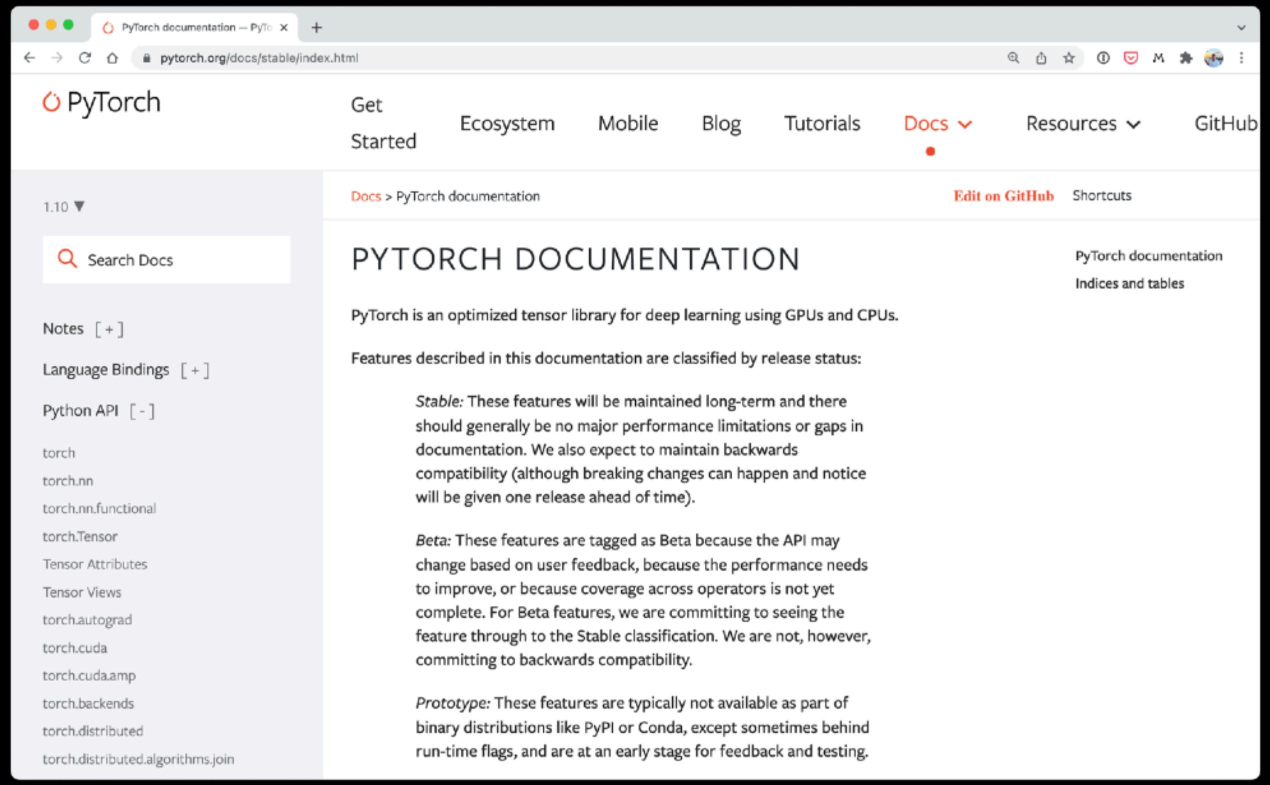
- Follow along with the code
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring
- Search for it
- Try again
- Ask





stackoverflow





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

“What is a computer vision problem?”

Example computer vision problems

"Is this a photo of steak or pizza?"

Binary classification

(one thing or another)

"Where's the thing we're looking for?"

Object detection

"Is this a photo of sushi, steak or pizza?"

Multiclass classification

(more than one thing or another)

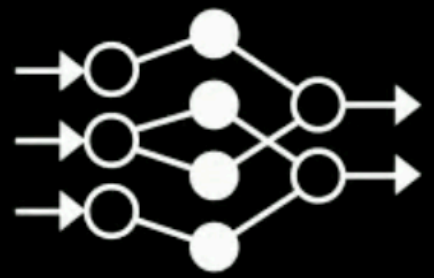
"What are the different sections in this image?"

Segmentation

Source: On-device Panoptic Segmentation for Camera Using Transformers.

Tesla Computer Vision

8 Cameras



3-Dimensional “Vector Space”

Source: Tesla AI Day Video (49:49). PS see 2:01:31 of the same video for surprise ;)

Tesla Computer Vision

Source: AI Drivr YouTube channel.

What we're going to cover

(broadly)

- Getting a vision dataset to work with using `torchvision.datasets`
- Architecture of a convolutional neural network (CNN) with PyTorch
- An end-to-end multi-class image classification problem
- Steps in modelling with CNNs in PyTorch
  - Creating a CNN model with PyTorch
  - Picking a loss and optimizer
  - Training a PyTorch computer vision model
  - Evaluating a model

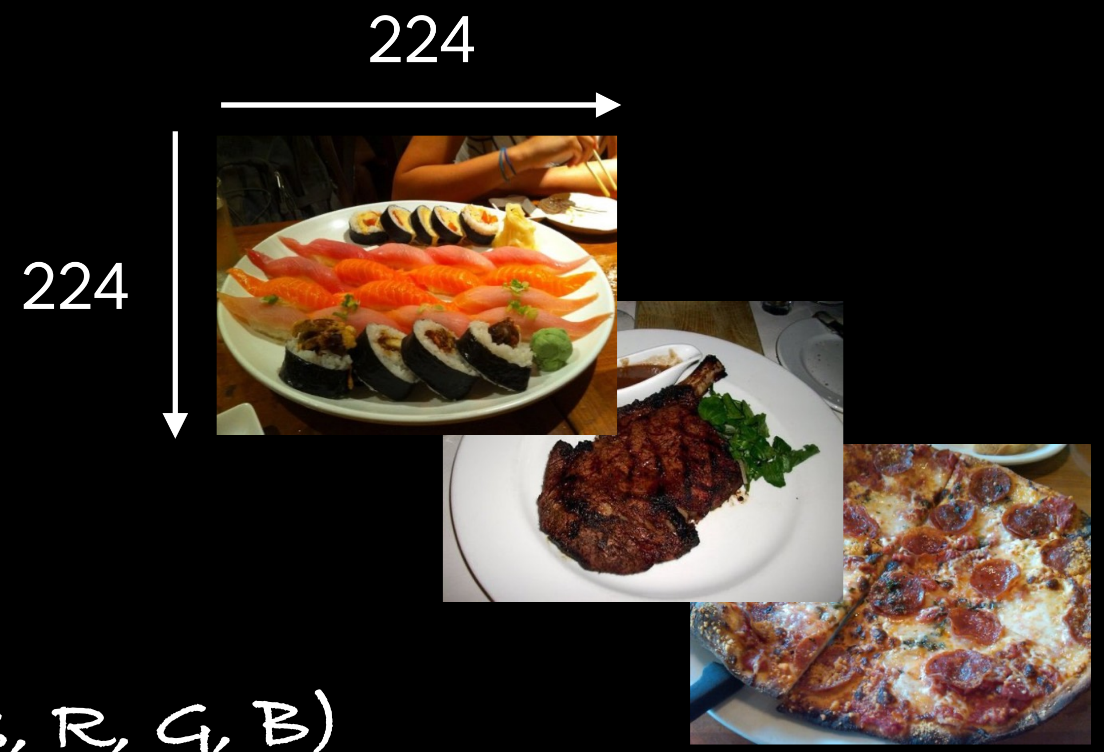
(we'll be cooking up lots of code!)

How:

Computer vision inputs and outputs

W = 224
H = 224
C = 3

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



→

→

Sushi 🍣
Steak 🥩
Pizza 🍕

Actual output

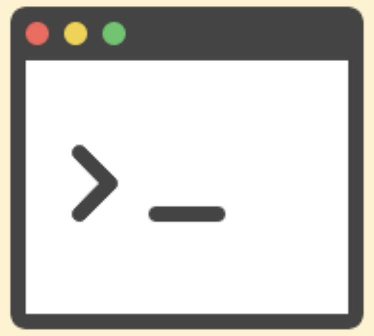
This is often a
convolutional neural network (CNN)!

[[0.31, 0.62, 0.44…],
[0.92, 0.03, 0.27…], →
[0.25, 0.78, 0.07…],
…,

(normalized pixel values)

Numerical encoding

Inputs →



Machine Learning Algorithm

→ Outputs →

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

🍣 🥩 🍕

[[0.97, 0.00, 0.03], ✓
[0.81, 0.14, 0.05], ✗
[0.03, 0.07, 0.90], ✓
…,

Predicted output

(comes from looking at lots of these)

Input and output shapes

Input and output shapes

[batch_size, height, width, colour_channels] (NHWC) (colour channels last)

or

[batch_size, colour_channels, height, width] (NCHW) (colour channels first)

Shape = [10]

Shape = [None, 28, 28, 1] (NHWC)
Shape = [None, 1, 28, 28] (NCHW)
or
Shape = [32, 28, 28, 1]

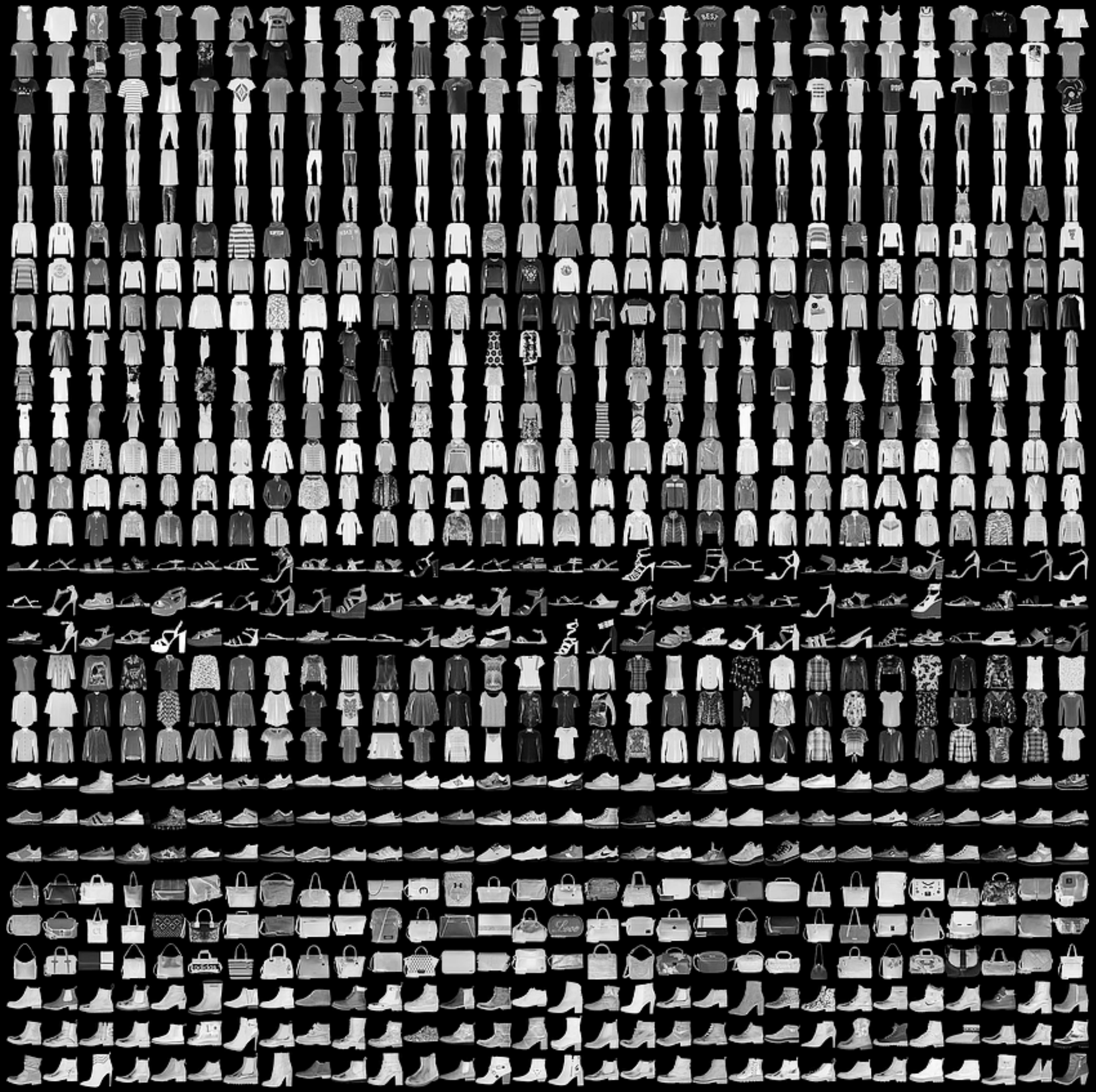
(32 is a very common batch size)

These will vary depending on the problem you're working on.

“What is a convolutional neural network (CNN)?”

Let’s code!

FashionMNIST



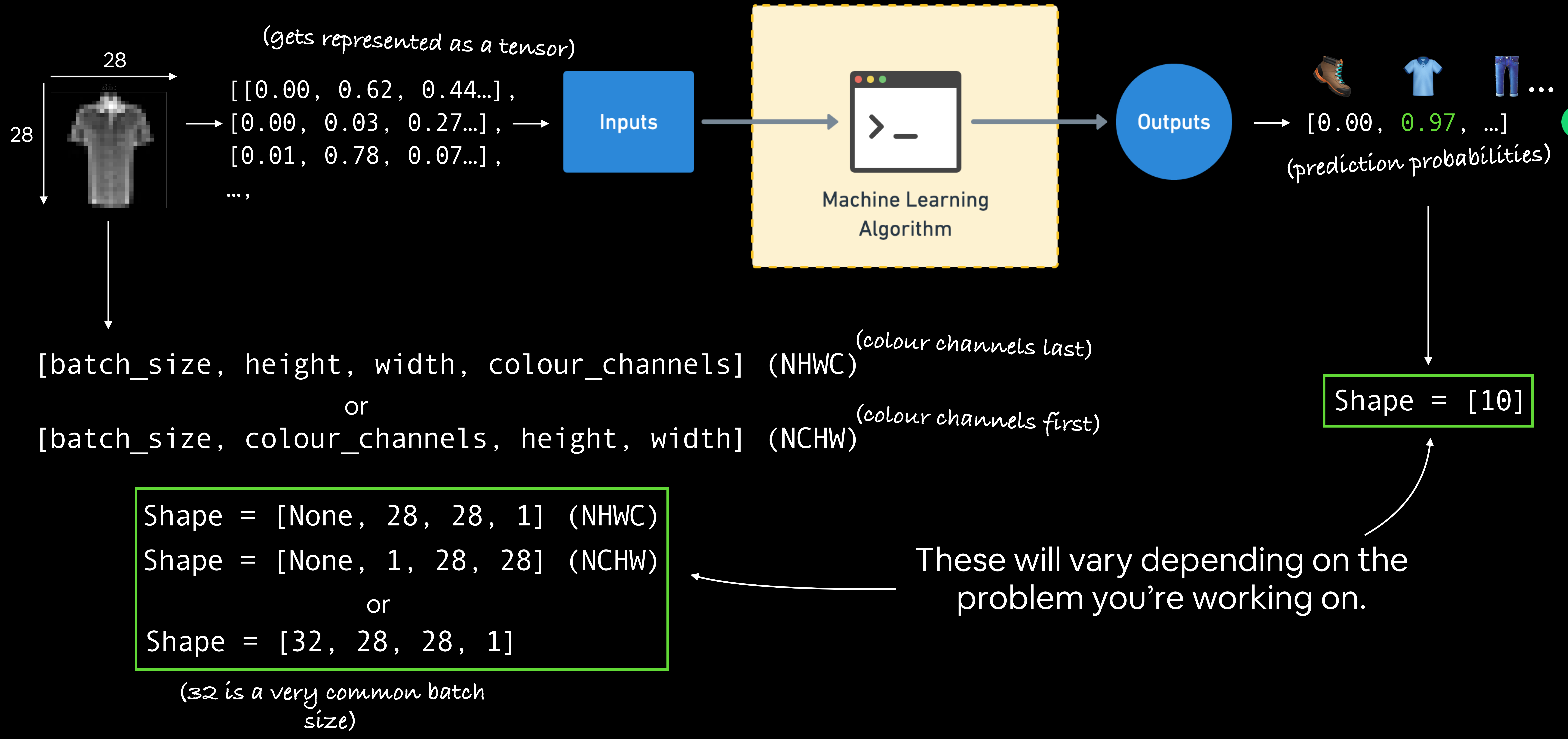
torchvision.datasets.FashionMNIST

"What type of clothing is in this image?"

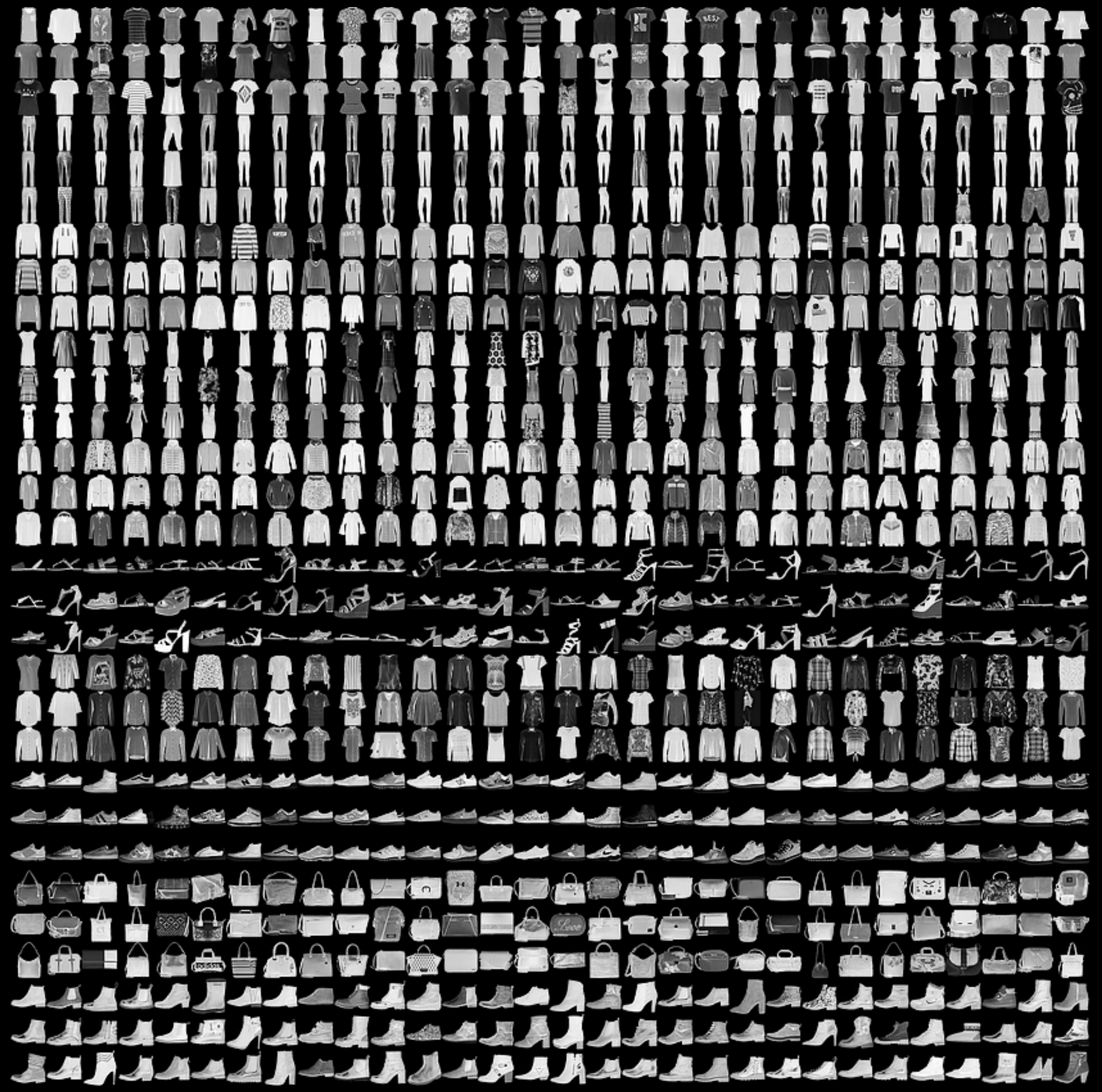
Multiclass classification

(more than one thing or another)

Input and output shapes



FashionMNIST: Batched



torchvision.datasets.FashionMNIST

```python
# Turn train dataset into DataLoader
from torch.utils.data import DataLoader
train_dataloader = DataLoader(dataset=train_data,
                              batch_size=32,
                              shuffle=True)

```

torch.utils.data.DataLoader

batch_size=32
(32 samples per batch)

Sample 0 1 2 3 4 5 32

Batch 0 1 2 3 4 ⋮

shuffle=True
(samples all mixed up)

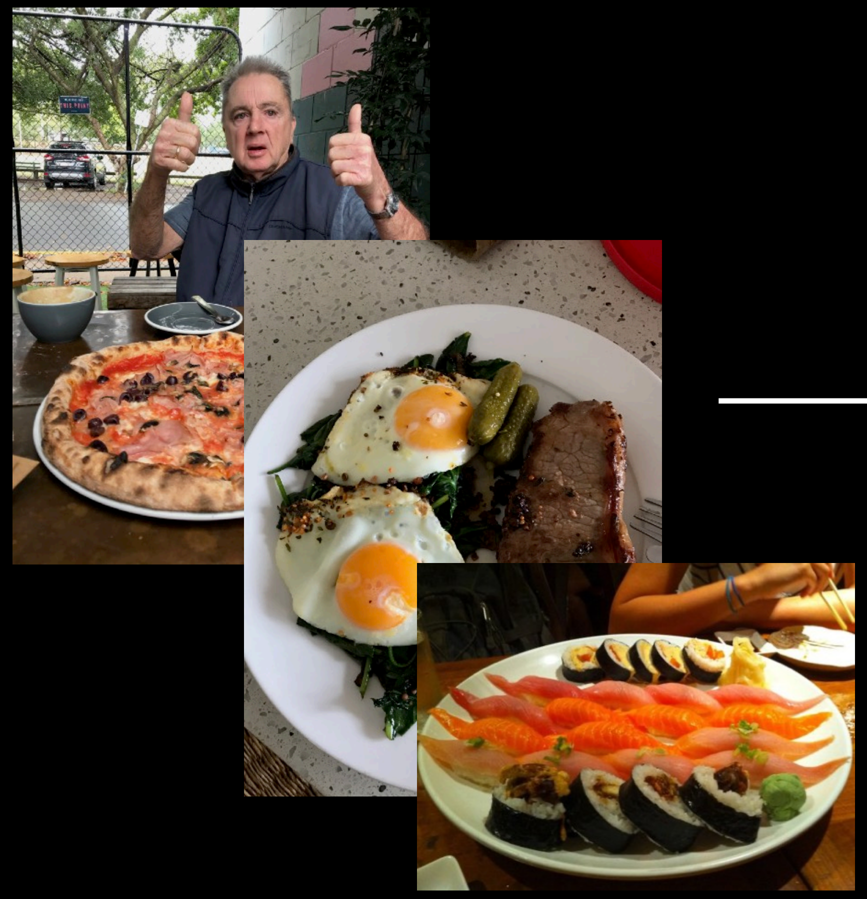
Num samples/
batch_size

(typical)*

Architecture of a CNN

Hyperparameter/Layer type	What does it do?	Typical values
Input image(s)	Target images you'd like to discover patterns in	Whatever you can take a photo (or video) of
Input layer	Takes in target images and preprocesses them for further layers	`input_shape = [batch_size, image_height, image_width, color_channels]` (channels last) or `input_shape = [batch_size, color_channels, image_height, image_width]` (channels first)
Convolution layer	Extracts/learns the most important features from target images	Multiple, can create with `torch.nn.ConvXd()` (X can be multiple values)
Hidden activation/non-linear activation	Adds non-linearity to learned features (non-straight lines)	Usually ReLU (`torch.nn.ReLU()`), though can be many more
Pooling layer	Reduces the dimensionality of learned image features	Max (`torch.nn.MaxPool2d()`) or Average (`torch.nn.AvgPool2d()`)
Output layer/linear layer	Takes learned features and outputs them in shape of target labels	`torch.nn.Linear(out_features=[number_of_classes])` (e.g. 3 for pizza, steak or sushi)
Output activation	Converts output logits to prediction probabilities	`torch.sigmoid()` (binary classification) or `torch.softmax()` (multi-class classification)

(what we're working towards building)



```python
# Create a Convolutional Neural Network
import torch
from torch import nn
class CNN_model(nn.Module):
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int):
        super().__init__()
        self.cnn_layers = nn.Sequential(
            nn.Conv2d(in_channels=input_shape,
                      out_channels=hidden_units,
                      kernel_size=3, # how big is the square that's going over the image?
                      stride=1, # take a step one pixel at a time
                      padding=1), # add an extra pixel around the input image
            nn.ReLU(), # non-linear activation
            nn.MaxPool2d(kernel_size=2,
                         stride=2) # default stride value is same as kernel_size
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(in_features=hidden_units * 32 * 32, # same shape as output of self.cnn_layers
                      out_features=output_shape)
        )

    def forward(self, x: torch.Tensor):
        x = self.cnn_layers(x)
        x = self.classifier(x)
        return x

cnn_model = CNN_model(input_shape=3, # same as number of input color channels
                      hidden_units=10,
                      output_shape=3) # same as number of classes
```

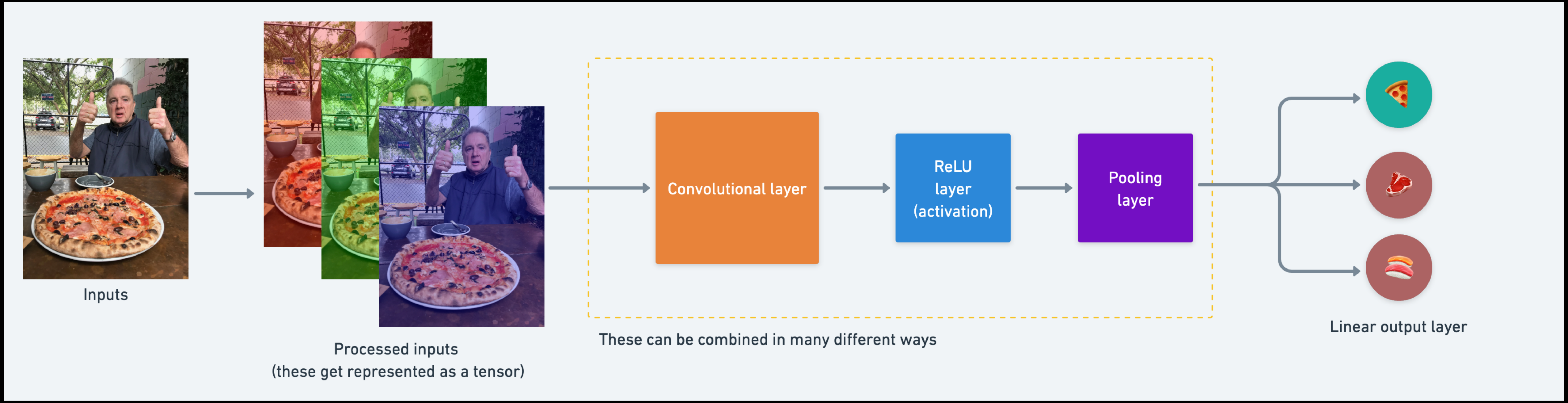
Steak 🥩
Pizza 🍕
Sushi 🍣

***Note:** there are almost an unlimited amount of ways you could stack together a convolutional neural network, this slide demonstrates only one.

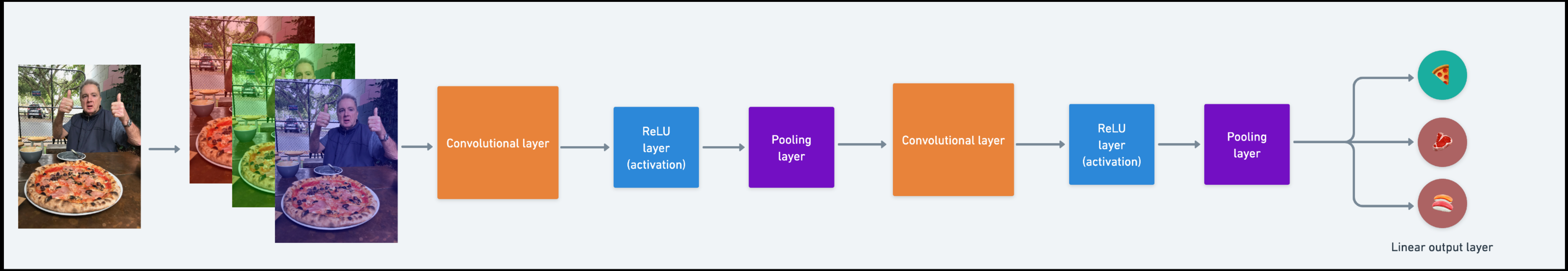
Typical architecture of a CNN

(coloured block edition)

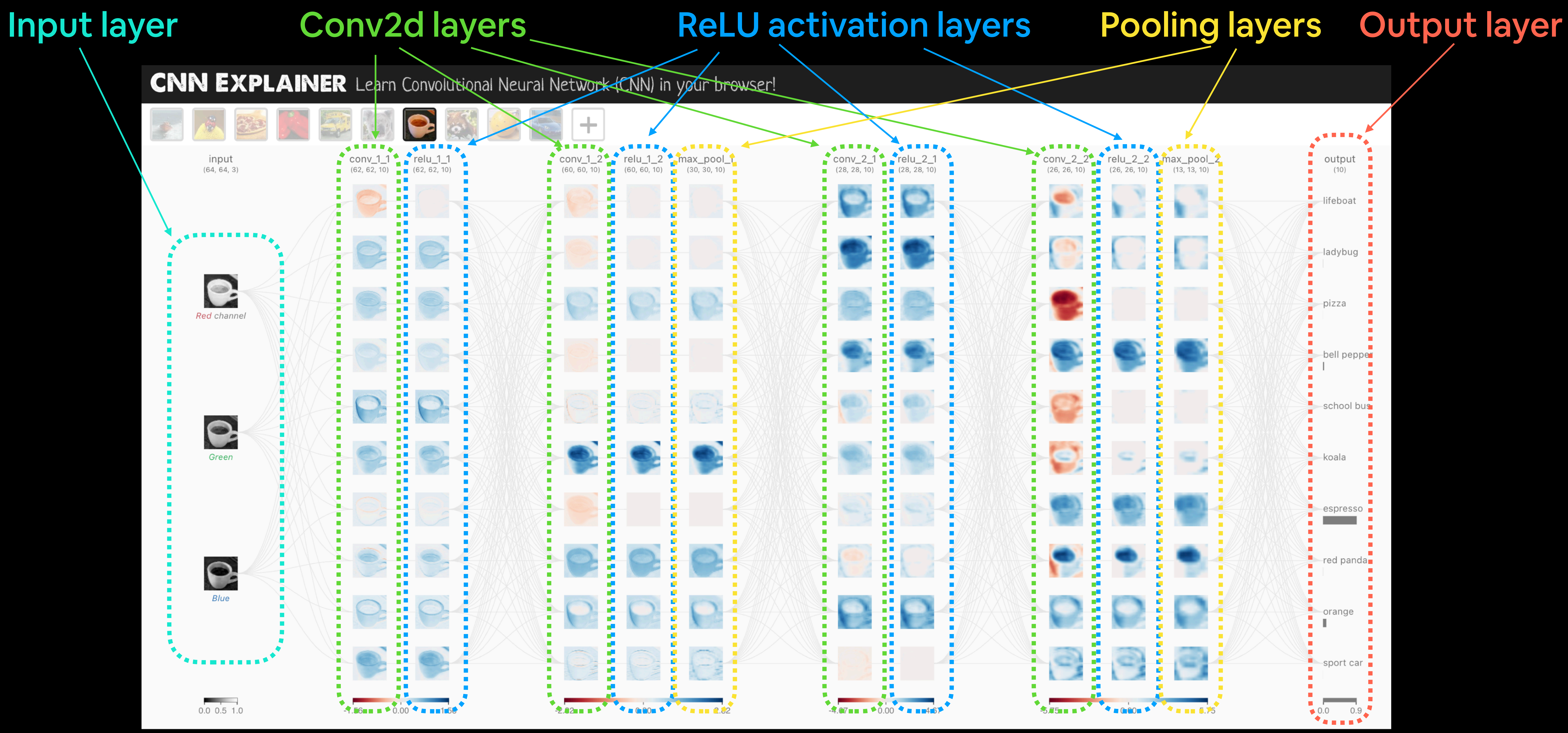
Simple CNN



Deeper CNN



CNN Explainer model



Source: CNN Explainer website, architecture is known as TinyVGG.

Breakdown of torch.nn.Conv2d layer

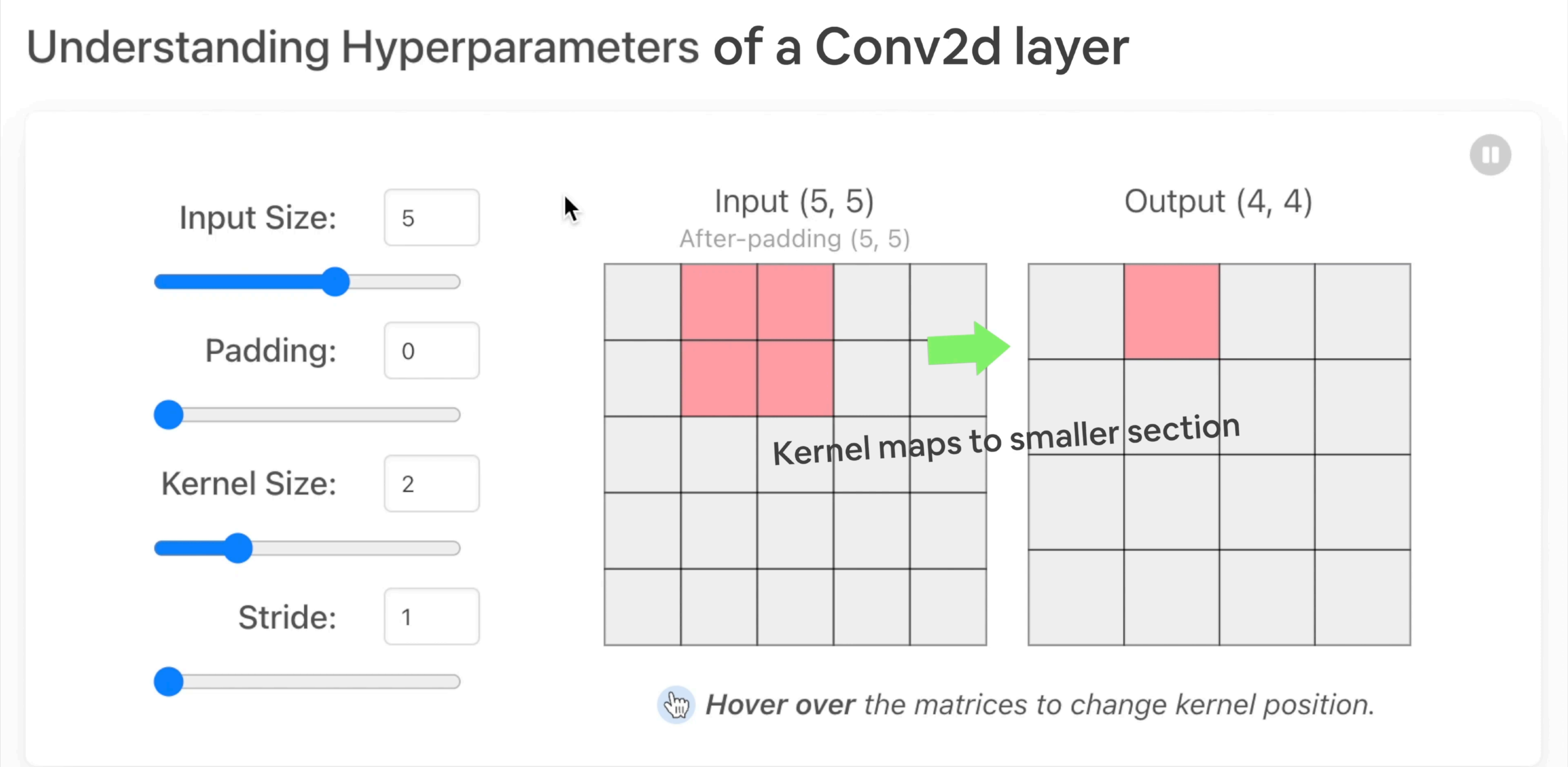
Example code: `torch.nn.Conv2d(in_channels=3, out_channels=10, kernel_size=(3, 3), stride=(1, 1), padding=0)`
Example 2 (same as above): `torch.nnConv2d(in_channels=3, out_channels=10, kernel_size=3, stride=1, padding=0)`

Hyperparameter name	What does it do?	Typical values
`in_channels`	Defines the number of input channels of the input data.	`1 (grayscale), 3 (RGB color images)`
`out_channels`	Defines the number output channels of the layer (could also be called hidden units).	`10, 128, 256, 512`
`kernel_size` (also referred to as filter size)	Determines the shape of the kernel (sliding windows) over the input.	`3, 5, 7 (lowers values learn smaller features, higher values learn larger features)`
`stride`	The number of steps a filter takes across an image at a time (e.g. if `strides=1`, a filter moves across an image 1 pixel at a time).	`1 (default), 2`
`padding`	Pads the target tensor with zeroes (if "`same`") to preserve input shape. Or leaves in the target tensor as is (if "`valid`"), lowering output shape.	`0, 1, "same", "valid"`

📖 **Resource:** For an interactive demonstration of the above hyperparameters, see the CNN Explainer website.

Breakdown of torch.nn.Conv2d layer

(visually)

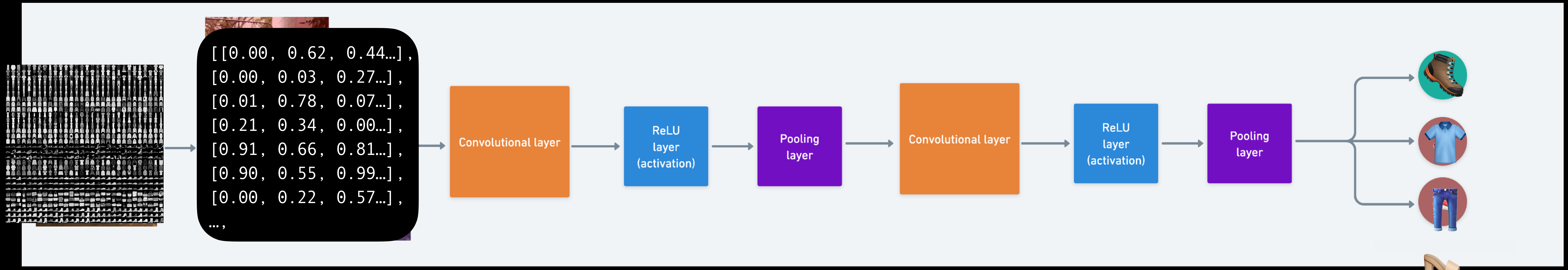


Source: CNN Explainer website

📖 **Resource:** For an interactive demonstration of the above hyperparameters, see the CNN Explainer website.

FashionMNIST -> CNN

Output layer outputs predictions

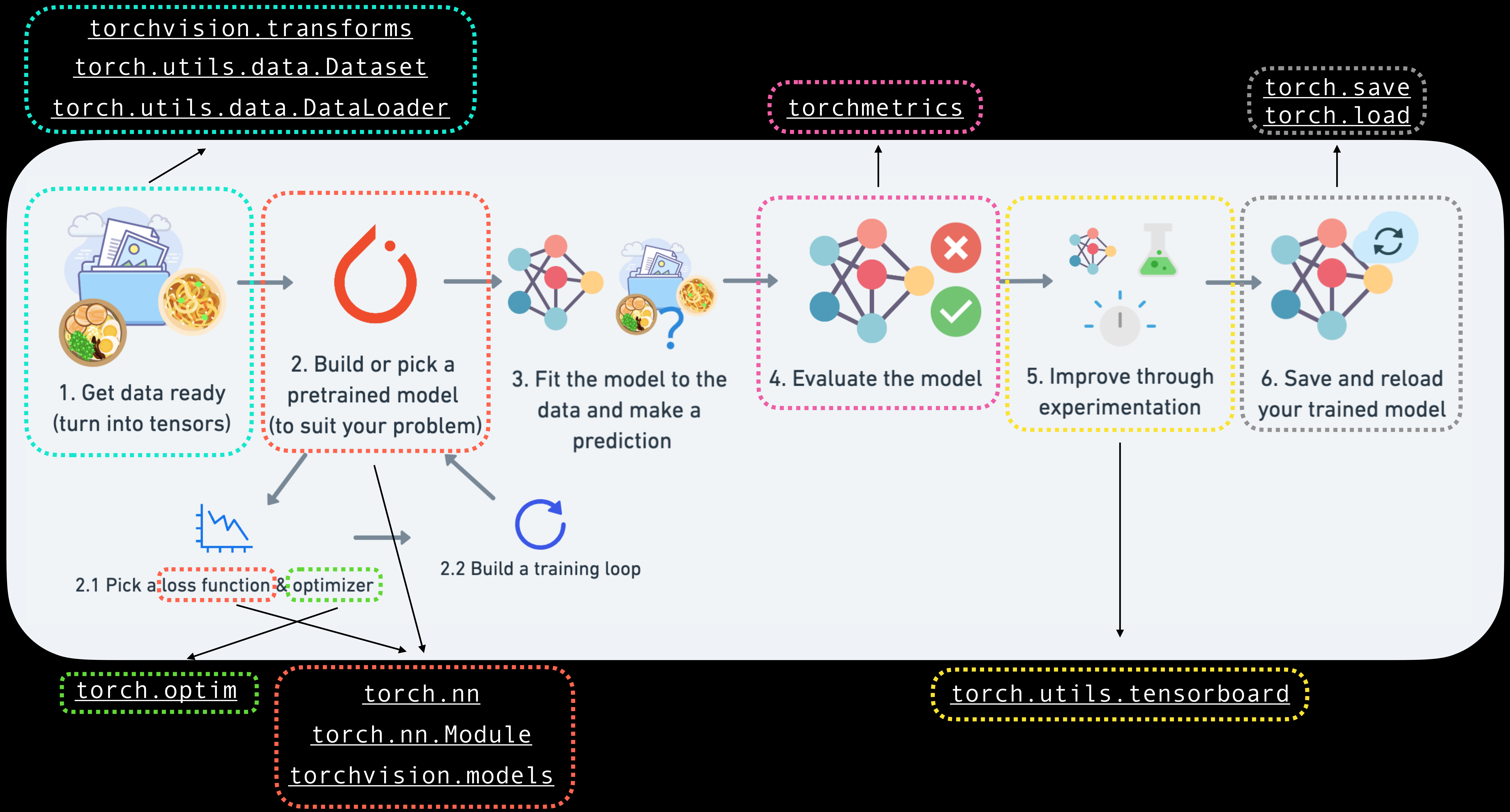


Inputs

Numerical encoding

Layers learn numerical representation

Keep going until number of classes is fulfilled

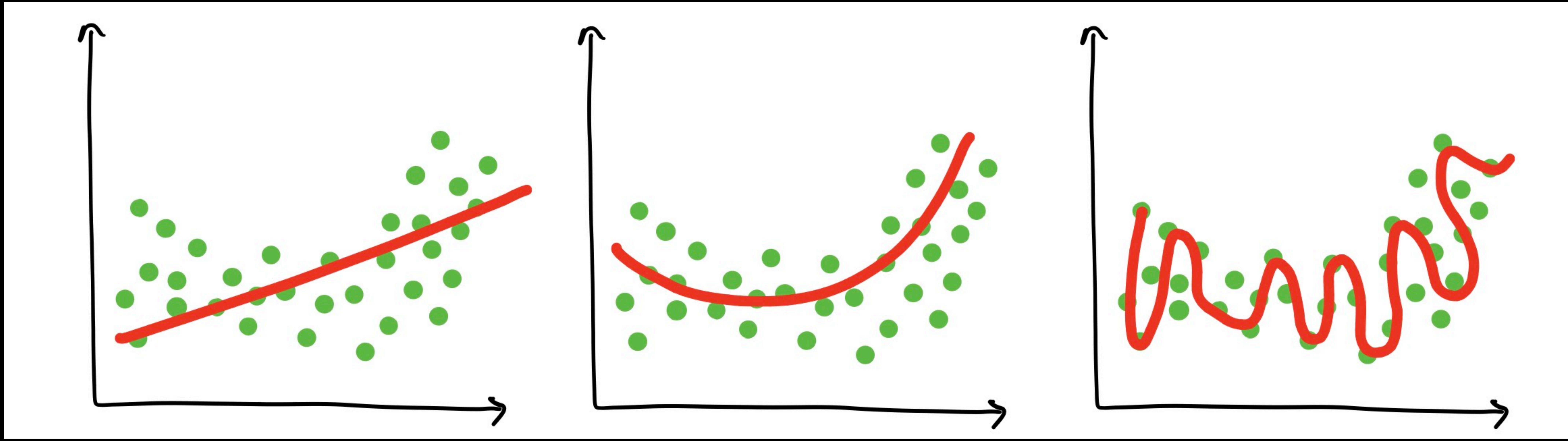


See more: https://pytorch.org/tutorials/beginner/ptcheat.html

What is overfitting?

Overfitting — when a model over learns patterns in a particular dataset and isn't able to generalise to unseen data.

For example, a student who studies the course materials too hard and then isn't able to perform well on the final exam. Or tries to put their knowledge into practice at the workplace and finds what they learned has nothing to do with the real world.



Underfitting | **Balanced** (goldilocks zone) | **Overfitting**

Improving a model

(from a model's perspective)

```python
# Create a model
model = nn.Sequential(
    nn.Linear(in_features=3, out_features=100),
    nn.Linear(in_features=100, out_features=100),
    nn.ReLU(),
    nn.Linear(in_features=100, out_features=3)
)

# Setup a loss function and optimizer
loss_fn = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(params=model.parameters(),
                            lr=0.001)

# Training code...
epochs = 10

# Testing code...
```

Smaller model

```python
# Create a larger model
model = nn.Sequential(
    nn.Linear(in_features=3, out_features=128),
    nn.ReLU(),
    nn.Linear(in_features=128, out_features=256),
    nn.ReLU(),
    nn.Linear(in_features=256, out_features=128),
    nn.ReLU(),
    nn.Linear(in_features=128, out_features=3)
)

# Setup a loss function and optimizer
loss_fn = nn.BCEWithLogitsLoss()
optimizer = torch.optim.Adam(params=model.parameters(),
                             lr=0.0001)

# Training code...
epochs = 100

# 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
- Fitting for longer

(because you can alter each of these, they're hyperparameters)

Improving a model

(from a data perspective)

Method to improve a model (reduce overfitting)	What does it do?
More data	Gives a model more of a chance to learn patterns between samples (e.g. if a model is performing poorly on images of pizza, show it more images of pizza).
Data augmentation	Increase the diversity of your training dataset without collecting more data (e.g. take your photos of pizza and randomly rotate them 30°). Increased diversity forces a model to learn more generalisation patterns.
Better data	Not all data samples are created equally. Removing poor samples from or adding better samples to your dataset can improve your model's performance.
Use transfer learning	Take a model's pre-learned patterns from one problem and tweak them to suit your own problem. For example, take a model trained on pictures of cars to recognise pictures of trucks.

What is data augmentation?

Looking at the same image but from different perspective(s)*.

Original

Rotate

Shift

Zoom

***Note:** There are many more different kinds of data augmentation such as, cropping, replacing, shearing. This slide only demonstrates a few.

Popular & useful computer vision architectures: see `torchvision.models`

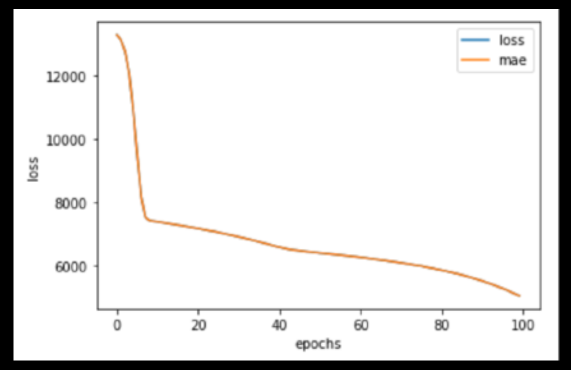
Architecture	Release Date	Paper	Use in PyTorch	When to use
ResNet (residual networks)	2015	https://arxiv.org/abs/1512.03385	torchvision.models.resnet...	A good backbone for many computer vision problems
EfficientNet(s)	2019	https://arxiv.org/abs/1905.11946	torchvision.models.efficientnet...	Typically now better than ResNets for computer vision
Vision Transformer (ViT)	2020	https://arxiv.org/abs/2010.11929	torchvision.models.vit_...	Transformer architecture applied to vision
MobileNet(s)	2017	https://arxiv.org/abs/1704.04861	torchvision.models.mobilenet...	Lightweight architecture suitable for devices with less computing power

The machine learning explorer's motto

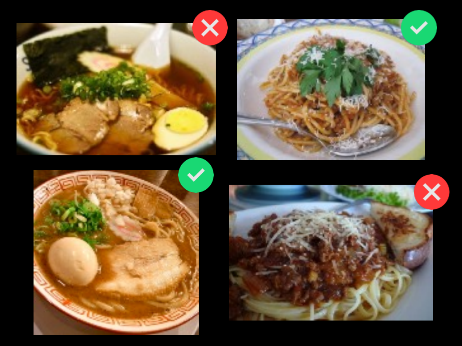
"Visualize, visualize, visualize"

Data

Model



Training



Predictions

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)