CPSC 330 Lecture 19: Introduction to deep learning and computer vision

Firas Moosvi

Clicker 18.1 (Recap)

Select all of the following statements which are TRUE.

- a. It's possible to use word2vec embedding representations for text classification instead of bag-of-words representation.
- b. The topic model approach we used in the last lecture, Latent Dirichlet Allocation (LDA), is an unsupervised approach.
- c. In an LDA topic model, the same word can be associated with two different topics with high probability.
- d. In an LDA topic model, a document is a mixture of multiple topics.
- e. If I train a topic model on a large collection of news articles with K = 10, I would get 10 topic labels (e.g., sports, culture, politics, finance) as output.

Multiclass classification

What is multiclass classification?

- So far, we've focused on binary classification (two classes).
- But many real-world problems have more than two classes:
  - Classifying types of emotions in text
  - Identifying species of flowers
  - Recognizing objects in an image

Goal: assign each example to exactly one of K possible classes.

How to handle multiple classes?

Consider this dataset with multiple classes.

	original_hm	target
0	Meeting up with my nephew and niece at my brot...	affection
1	The prospect of weekend coming as it is Friday...	leisure
2	We bought a new rug for our family room.	achievement
3	We got a big federal and state tax refund. I w...	achievement
4	I'm leaving work an hour early today!	enjoy_the_moment

```
1  hdb_df["target"].value_counts(normalize=True)
```

```
target
affection           0.3415
achievement         0.3043
bonding             0.1240
enjoy_the_moment    0.1036
leisure             0.0933
nature              0.0182
exercise            0.0151
Name: proportion, dtype: float64
```

- ❓ Can we use Decision Trees or KNNs for multiclass classification without any significant modifications?

Extending logistic regression

- What about logistic regression? The binary version works only for two classes.
- For multiclass problems, we use the multinomial (softmax) logistic regression model.
- It learns a separate weight vector and bias for each class:
  - Parameters per class: d weights + 1 bias
    - Total parameters: $(d + 1) \times K$

Example:

If we have 10,000 features (e.g., vocabulary size) and 7 moment classes, the model learns $(10{,}000 + 1) \times 7$ parameters.

Softmax function

- In binary logistic regression, we use the sigmoid function to map scores to probabilities.
- In multiclass, we need:
  - Probabilities that are positive, and
  - Sum to 1 across all classes.

The softmax function does exactly that.

(Optional) Softmax function

Given an input, the probability that it belongs to class $j \in \{1, 2, \ldots, K\}$ is calculated using the **softmax function**:

$$P(y = j \mid x_i) = \frac{e^{w_j^\top x_i + b_j}}{\sum_{k=1}^{K} e^{w_k^\top x_i + b_k}}$$

- x_i is the i^{th} example
- w_j is the weight vector for class j.
- b_j is the bias term for class j.
- K is the total number of classes.

Making predictions

1. **Compute Probabilities**:
   For each class j, compute the probability $P(y = j \mid x_i)$ using the softmax function.
2. **Select the Class with the Highest Probability**:
   The predicted class $\hat{y}$ is:
   $\hat{y} = \arg\max_{j \in \{1,\ldots,K\}} P(y = j \mid x_i)$

Example: Softmax output

Query: “I love my students!”

Class	Raw Score (Logit)	Exponentiated	Normalized Probability
❤️ affection	2.1	8.166	0.4684
🏆 achievement	1.8	6.050	0.3472
🤝 bonding	1.0	2.718	0.1559
🌅 enjoy_the_moment	0.3	1.350	0.0775
🎮 leisure	-0.2	0.819	0.0470
🌿 nature	-1.0	0.368	0.0211
🏃 exercise	-1.5	0.223	0.0128
Total	—	—	**1.000**

Binary vs multinomial logistic regression

Aspect	Binary Logistic Regression	Multinomial Logistic Regression
Target variable	2 classes (binary)	More than 2 classes (multi-class)
Getting probabilities	Sigmoid	Softmax
parameters	d weights, one per feature and the bias term	d weights and a bias term per class
Output	Single probability	Probability distribution over classes
Use case	Binary classification (e.g., spam detection)	Multi-class classification (e.g., flower species)

Deep Learning

Remember this picture we saw earlier?

- **Deep Learning (DL)** is a subset of machine learning

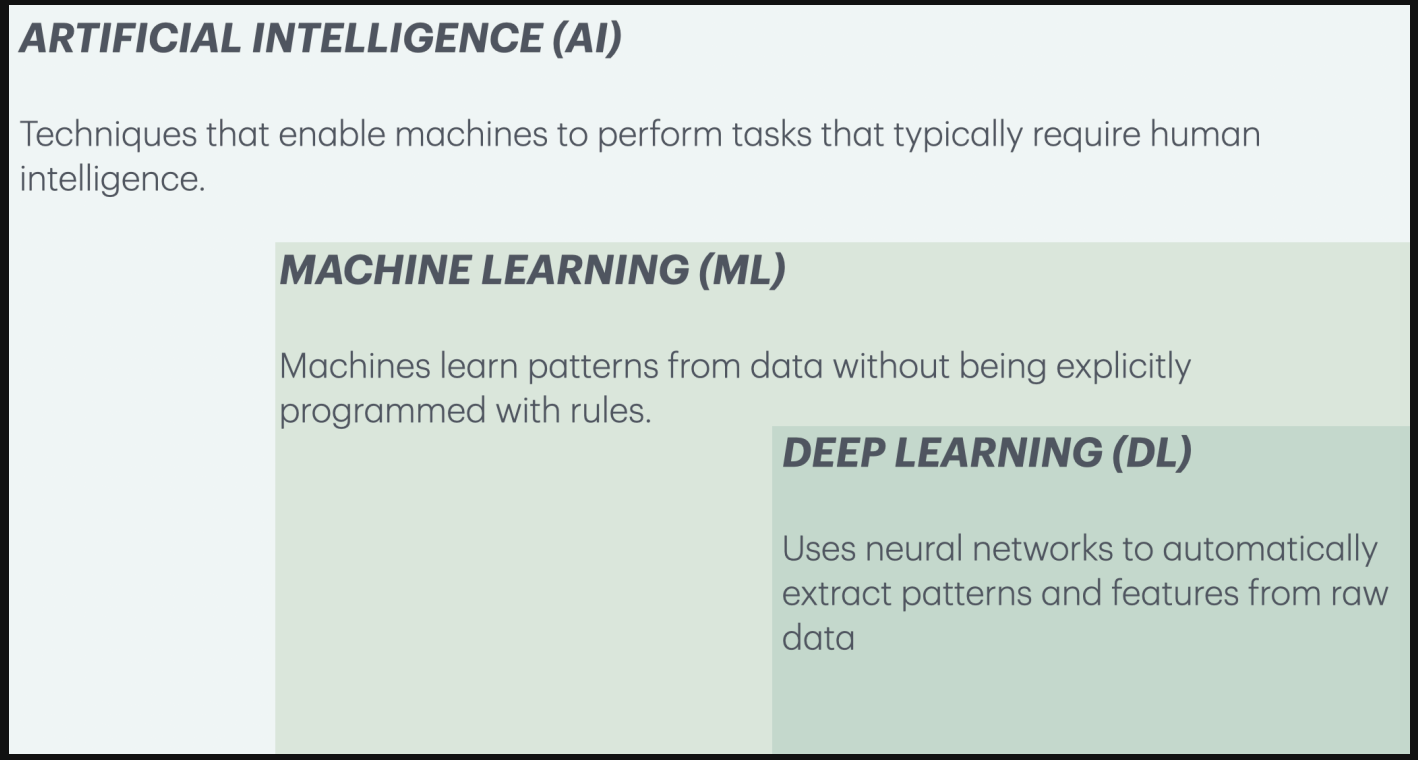
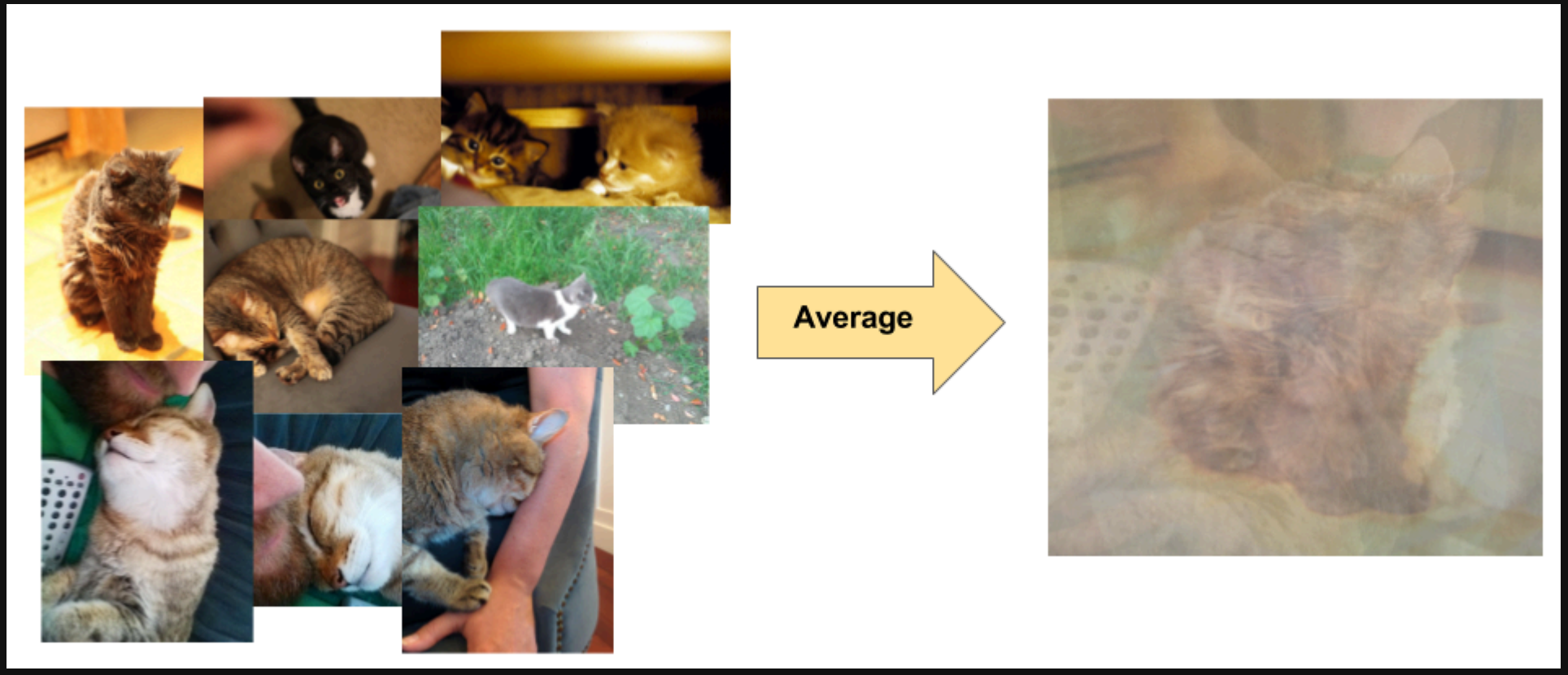


Image classification

Have you used search in Google Photos? You can search for “my photos of cat” and it will retrieve photos from your libraries containing cats. This can be done using **image classification**, which is treated as a supervised learning problem, where we define a set of target classes (objects to identify in images), and train a model to recognize them using labeled example photos.

Image classification

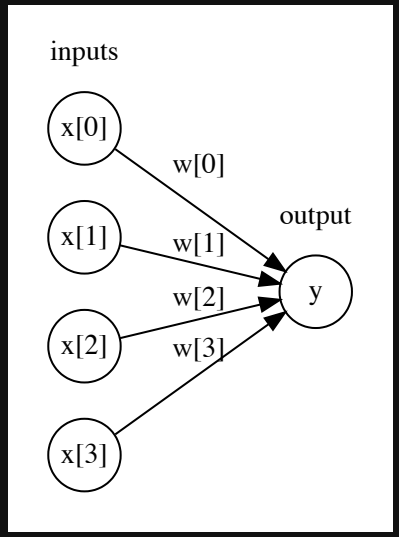
Image classification is not an easy problem because of the variations in the location of the object, lighting, background, camera angle, camera focus etc.



Neural networks

- Neural networks are perfect for these types of problems where local structures are important.
- A significant advancement in image classification was the application of **convolutional neural networks** (ConvNets or CNNs) to this problem.
  - ImageNet Classification with Deep Convolutional Neural Networks
  - Achieved a winning test error rate of 15.3%, compared to 26.2% achieved by the second-best entry in the ILSVRC-2012 competition.
- Let's go over the basics of a neural network.

A graphical view of a linear model



- Remember this graphical view of linear models?
- We have 4 features: x[0], x[1], x[2], x[3]
- The output is calculated as
  $y = x[0]w[0] + x[1]w[1] + x[2]w[2] + x[3]w[3]$
- For simplicity, we are ignoring the bias term.

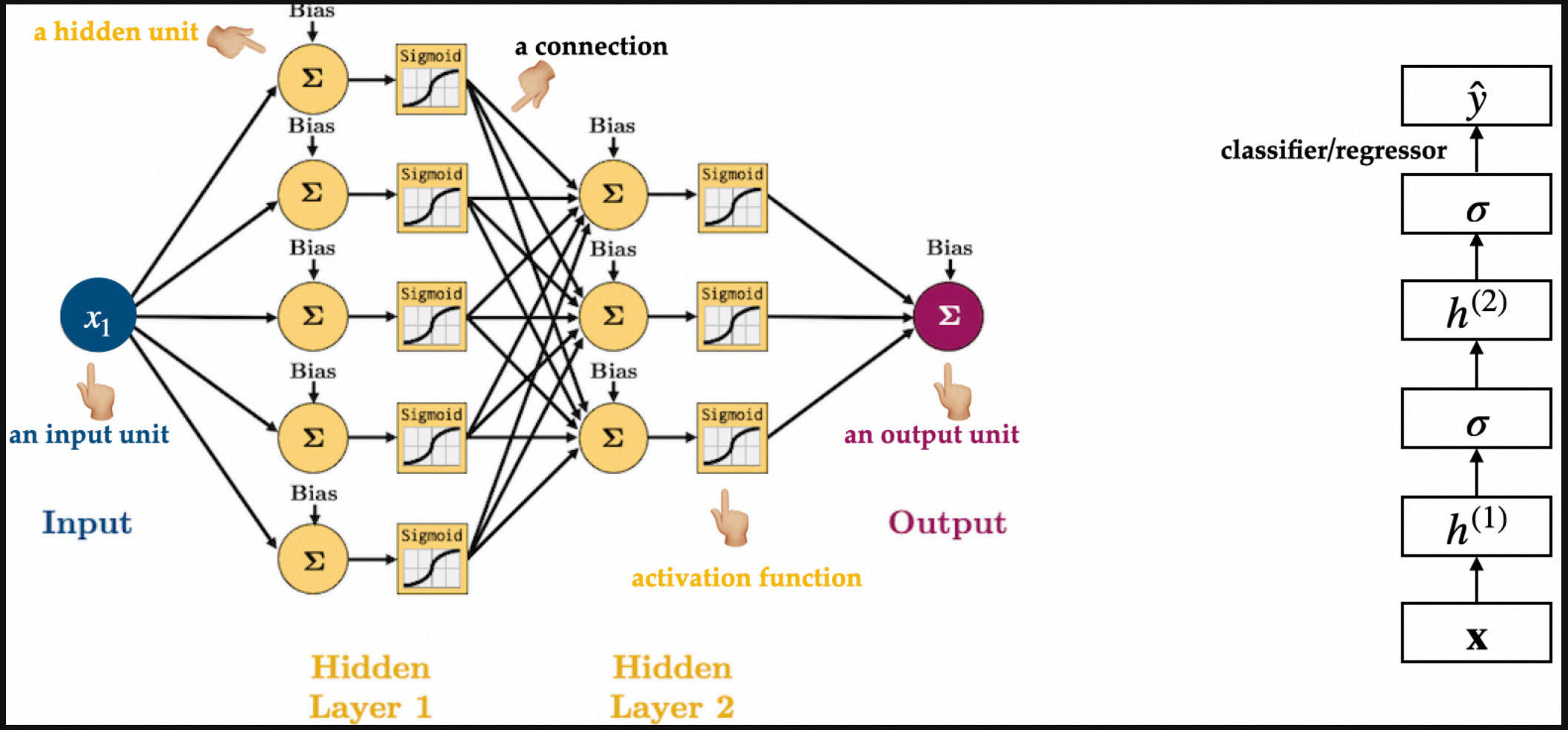
Introduction to neural networks

One more layer of transformations

Neural networks

- With a neural net, you specify the number of features after each transformation.
  - In the above, it goes from 4 to 3 to 3 to 1.
- To make them really powerful compared to the linear models, we apply a non-linear function to the weighted sum for each hidden node.
- Neural network = neural net
- Deep learning ~ using neural networks

Neural networks example and terminology



(Optional) How does training work in neural networks? (High-Level)

Training a neural network has **two main steps:**

Why neural networks?

- They can learn very complex functions.
  - The fundamental tradeoff is primarily controlled by the **number of layers** and **layer sizes**.
  - More layers / bigger layers –> more complex model.
  - You can generally get a model that will not underfit.
- They work really well for structured data:
  - 1D sequence, e.g. timeseries, language
  - 2D image
  - 3D image or video
- They've had some incredible successes in the last 12 years.
- Transfer learning (coming later today) is really useful.

Why not neural networks?

- Often they require a lot of data.
- They require a lot of compute time, and, to be faster, specialized hardware called GPUs.
- They have huge numbers of hyperparameters
  - Think of each layer having hyperparameters, plus some overall hyperparameters.
  - Being slow compounds this problem.
- They are not interpretable.
- I don't recommend training them on your own without further training
- Good news
  - You don't have to train your models from scratch in order to use them.
  - I'll show you some ways to use neural networks without training them yourselves.

Deep learning software

The current big players are:

1. PyTorch
2. TensorFlow

Both are heavily used in industry. If interested, see comparison of deep learning software.

Introduction to computer vision

- Computer vision refers to understanding images/videos, usually using ML/AI.
- In the last decade this field has been dominated by deep learning. We will explore **image classification** and **object detection**.

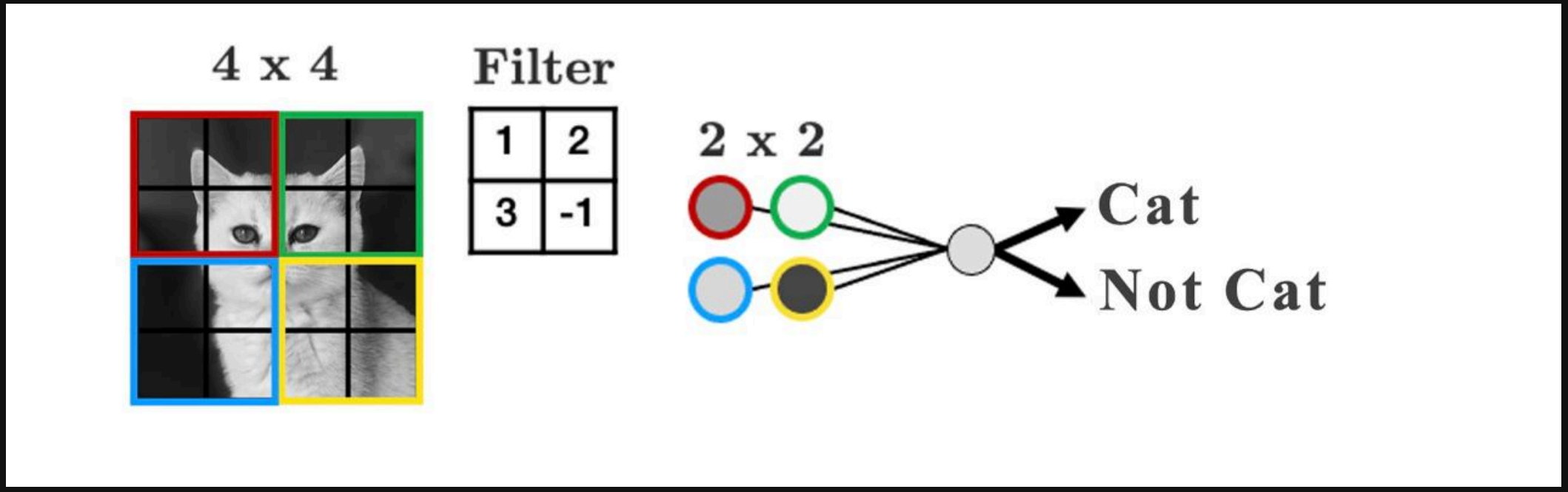
Introduction to computer vision

Convolutional Neural Networks (CNNs)

- Neural networks come in different shapes depending on the type of data and the task.
- CNNs are commonly used in **image classification, object detection, and medical imaging.**
- What's the problem if we use the above feedforward architecture to learn patterns from images?

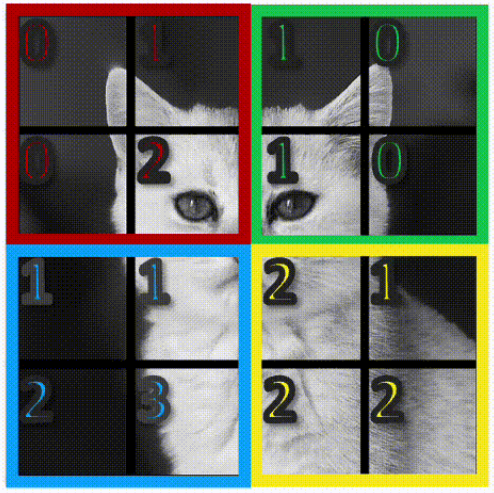
Filters

- CNNs use filters that “slide” over the input to detect local patterns.



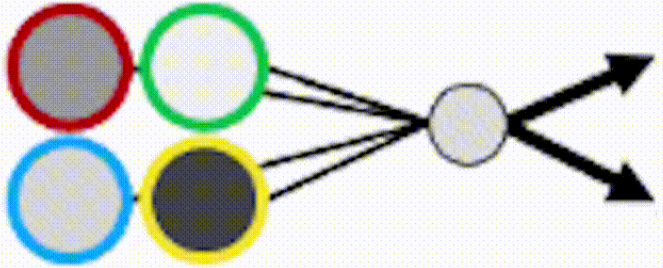
- Play around with filters: https://setosa.io/ev/image-kernels/

CNNs



Filter

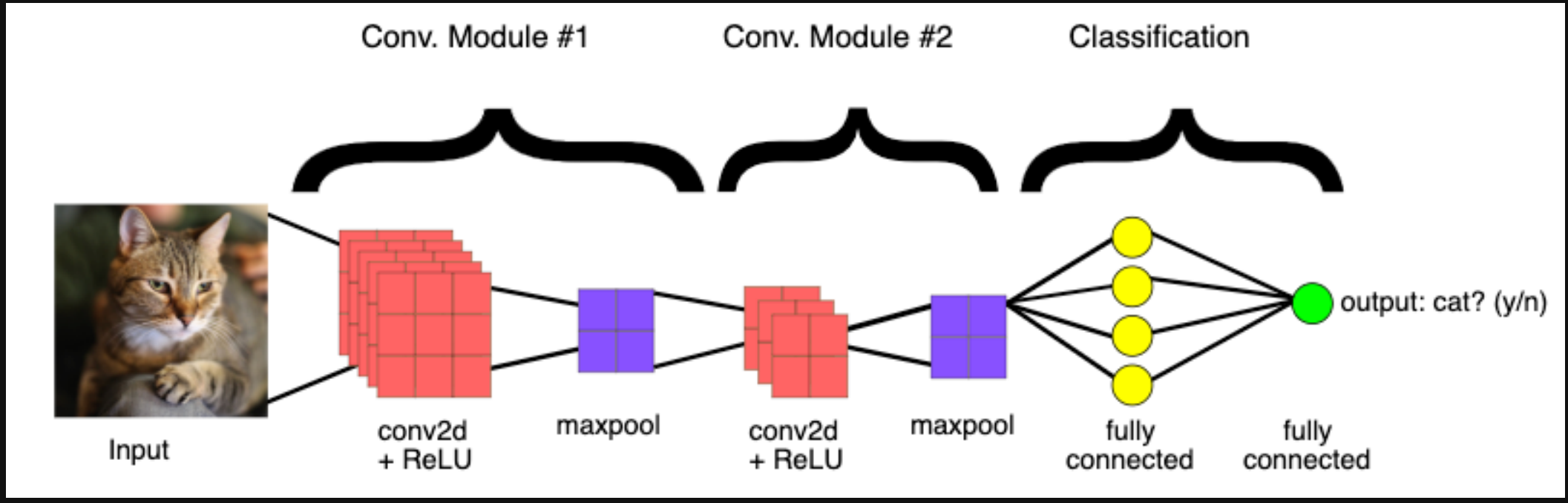
1	2
3	-1



Cat

Not Cat

CNNs big picture



source

Pre-trained models

- In practice, very few people train an entire CNN from scratch because it requires a large dataset, powerful computers, and a huge amount of human effort to train the model.
- Instead, a common practice is to download a pre-trained model and fine tune it for your task. This is called **transfer learning**.
- Transfer learning is one of the most common techniques used in the context of computer vision and natural language processing.
- It refers to using a model already trained on one task as a starting point for learning to perform another task.

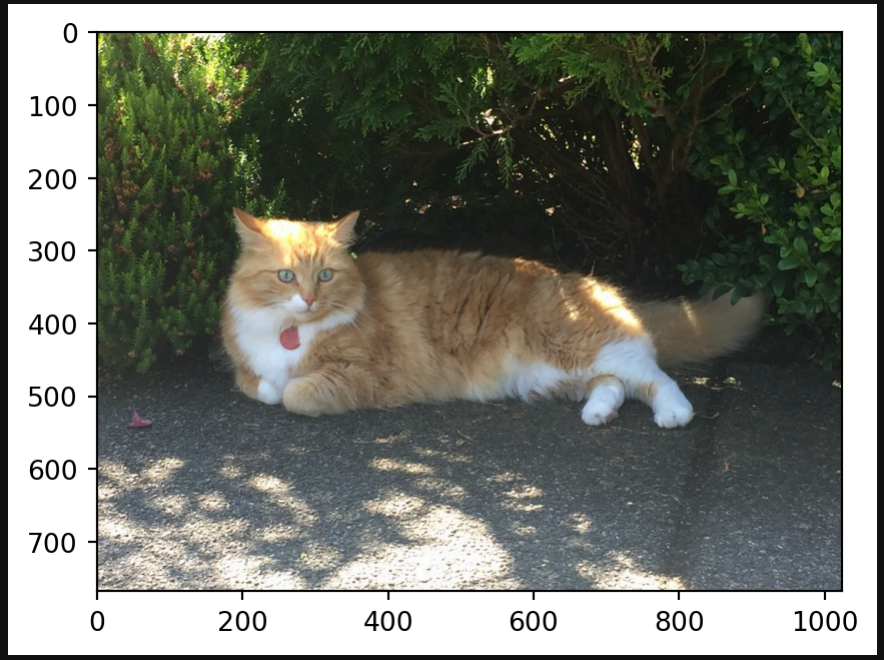
Pre-trained models out-of-the-box

Pre-trained models out-of-the-box

- We can easily download famous models using the `torchvision.models` module. All models are available with pre-trained weights (based on ImageNet's 224 x 224 images)
- We used a pre-trained model vgg16 which is trained on the ImageNet data.
- We preprocess the given image.
- We get prediction from this pre-trained model on a given image along with prediction probabilities.
- For a given image, this model will spit out one of the 1000 classes from ImageNet.

Pre-trained models out-of-the-box

- Let’s predict labels with associated probabilities for unseen images




```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /home/runner/.cache/torch/hub/checkpoint
```

Pre-trained models out-of-the-box

- We got these predictions without “doing the ML ourselves”.
- We are using **pre-trained** `vgg16` model which is available in `torchvision`.
  - `torchvision` has many such pre-trained models available that have been very successful across a wide range of tasks: AlexNet, VGG, ResNet, Inception, MobileNet, etc.
- Many of these models have been pre-trained on famous datasets like **ImageNet**.
- So if we use them out-of-the-box, they will give us one of the ImageNet classes as classification.

Pre-trained models out-of-the-box

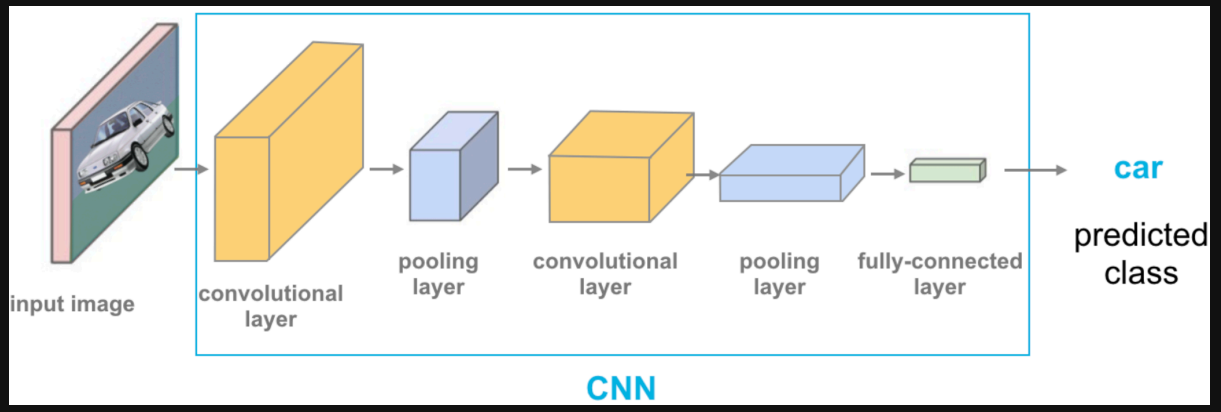
- Let’s try some images which are unlikely to be there in ImageNet.
- It’s not doing very well here because ImageNet doesn’t have proper classes for these images.

Pre-trained models out-of-the-box

- Here we used pre-trained models out-of-the-box.
- Can we use pre-trained models for our own classification problem with our classes?
- Yes!! We have two options here:
  1. Add some extra layers to the pre-trained network to suit our particular task
  2. Pass training data through the network and save the output to use as features for training some other model

Pre-trained models to extract features

- Let's use pre-trained models to extract features.
- We will pass our specific data through a pre-trained network to get a feature vector for each example in the data.
- The feature vector is usually extracted from the last layer, before the classification layer from the pre-trained network.
- You can think of each layer a transformer applying some transformations on the input received to that later.



Pre-trained models to extract features

- Once we extract these feature vectors for all images in our training data, we can train a machine learning classifier such as logistic regression or random forest.
- This classifier will be trained on our classes using feature representations extracted from the pre-trained models.
- Let's try this out.
- It's better to train such models with GPU. Since our dataset is quite small, we won't have problems running it on a CPU.

Pre-trained models to extract features

Let’s look at some sample images in the dataset.

Dataset statistics

Here is the stat of our toy dataset.

```
Classes: ['beet_salad', 'chocolate_cake', 'edamame', 'french_fries', 'pizza', 'spring_rolls', 'sushi']
Class count: 40, 38, 40
Samples: 283
First sample: ('data/food/train/beet_salad/104294.jpg', 0)
```

Pre-trained models to extract features

- Now for each image in our dataset, we'll extract a feature vector from a pre-trained model called densenet121, which is trained on the ImageNet dataset.

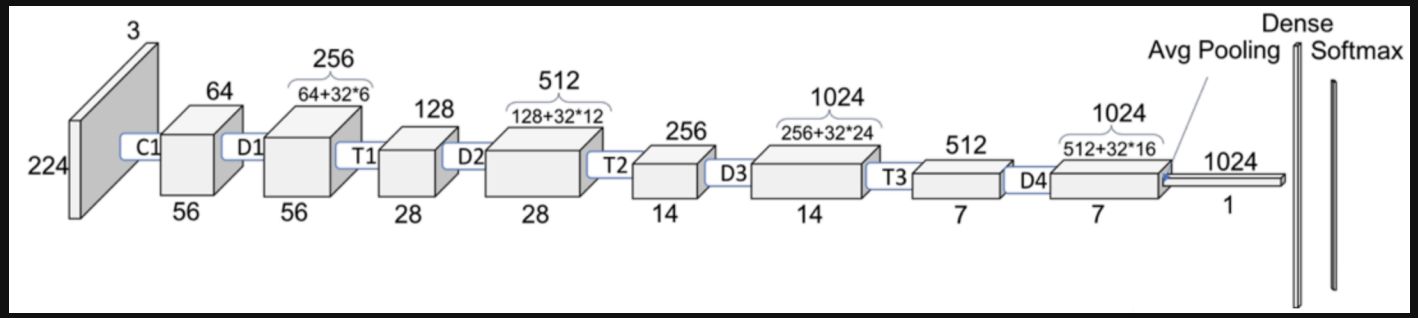
```
Downloading: "https://download.pytorch.org/models/densenet121-a639ec97.pth" to /home/runner/.cache/torch/hub/chec
```

Shape of the feature vector

- Now we have extracted feature vectors for all examples. What's the shape of these features?

```
torch.Size([283, 1024])
```

- The size of each feature vector is 1024 because the size of the last layer in densenet architecture is 1024.



Source

A feature vector given by densenet

- Let’s examine the feature vectors.

	0	1	2	3	4	5	6
0	0.000290	0.003821	0.005015	0.001307	0.052690	0.063403	0.000626
1	0.000407	0.005973	0.003206	0.001932	0.090702	0.438522	0.001513
2	0.000626	0.005090	0.002887	0.001299	0.091715	0.548535	0.000491
3	0.000169	0.006087	0.002489	0.002167	0.087537	0.623212	0.000427
4	0.000286	0.005520	0.001906	0.001599	0.186035	0.850148	0.000835

5 rows × 1024 columns

Logistic regression with the extracted features

- Let’s try out logistic regression on these extracted features.

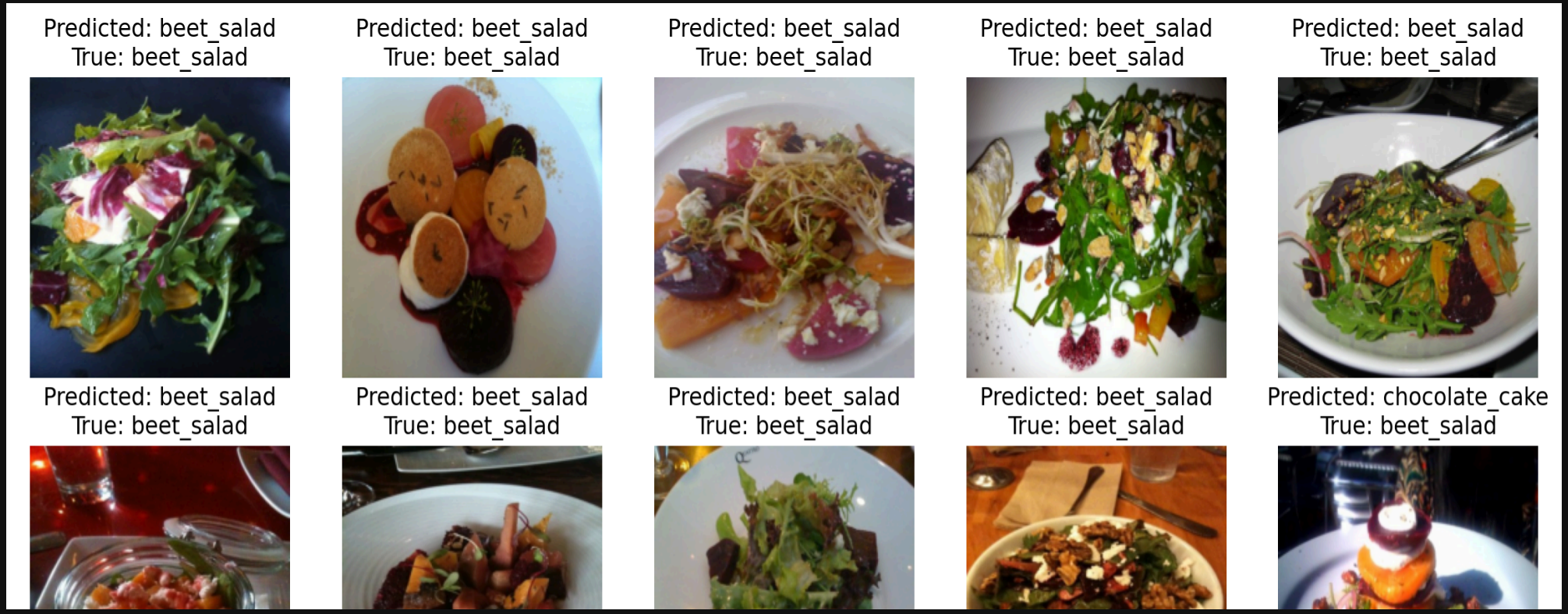
```
Training score:  1.0

Validation score:  0.835820895522388
```

- This is great accuracy for so little data and little effort!!!

Sample predictions

Let’s examine some sample predictions on the validation set.



Object detection

- Another useful task and tool to know is object detection using YOLO model.
- Let’s identify objects in a sample image using a pretrained model called YOLO8.
- List the objects present in this image.

Object detection using YOLO

Let's try this out using a pre-trained model.

```python
from ultralytics import YOLO
model = YOLO("yolov8n.pt")  # pretrained YOLOv8n model

yolo_input = "data/yolo_test/3356700488_183566145b.jpg"
yolo_result = "data/yolo_result.jpg"
# Run batched inference on a list of images
result = model(yolo_input)  # return a list of Results objects
result[0].save(filename=yolo_result)
```

```
Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/home/runner/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help see https://d

image 1/1 /home/runner/work/cpsc330-slides/cpsc330-slides/website/data/yolo_test/3356700488_183566145b.jpg: 512x6
Speed: 1.4ms preprocess, 86.0ms inference, 0.9ms postprocess per image at shape (1, 3, 512, 640)

'data/yolo_result.jpg'
```

Object detection output

Original Image

Result Image

Summary

- Neural networks are a flexible class of models.
  - They are particular powerful for structured input like images, videos, audio, etc.
  - They can be challenging to train and often require significant computational resources.
- The good news is we can use pre-trained neural networks.
  - This saves us a huge amount of time/cost/effort/resources.
  - We can use these pre-trained networks directly or use them as feature transformers.