Chapter 6

Prediction

A first category of applications, such as face recognition, sentiment analysis, object detection, or speech recognition, requires predicting an unknown value from an available signal.

6.1 Image denoising

A direct application of deep models to image processing is to recover from degradation by utilizing the redundancy in the statistical structure of images. The petals of a sunflower in a grayscale picture can be colored with high confidence, and the texture of a geometric shape such as a table on a low-light, grainy picture can be corrected by averaging it over a large area likely to be uniform.

A denoising autoencoder is a model that takes a degraded signal $\tilde{X}$ as input and computes an estimate of the original signal X. For images, it is a convolutional network that may integrate skip-connections, in particular to combine representations at the same resolution obtained early and late in the model, as well as attention layers to facilitate taking into account elements that are far away from each other.

Such a model is trained by collecting a large number of clean samples paired with their degraded inputs. The latter can be captured in degraded conditions, such as low-light or inadequate focus, or generated algorithmically, for instance, by converting the clean sample to grayscale, reducing its size, or aggressively compressing it with a lossy compression method.

The standard training procedure for denoising autoencoders uses the MSE loss summed across all pixels, in which case the model aims at computing the best average clean picture, given the degraded one, that is $\mathbb{E}[X \mid \tilde{X}]$. This quantity may be problematic when X is not completely determined by $\tilde{X}$, in which case some parts of the generated signal may be an unrealistic, blurry average.

6.2 Image classification

Image classification is the simplest strategy for extracting semantics from an image and consists of predicting a class from a finite, predefined number of classes, given an input image.

The standard models for this task are convolutional networks, such as ResNets (see § 5.2), and attention-based models such as ViT (see § 5.3). These models generate a vector of logits with as many dimensions as there are classes.

The training procedure simply minimizes the cross-entropy loss (see § 3.1). Usually, performance can be improved with data augmentation, which consists of modifying the training samples with hand-designed random transformations that do not change the semantic content of the image, such as cropping, scaling, mirroring, or color changes.

6.3 Object detection

A more complex task for image understanding is object detection, in which the objective is, given an input image, to predict the classes and positions of objects of interest.

An object position is formalized as the four coordinates (x_1, y_1, x_2, y_2) of a rectangular bounding box, and the ground truth associated with each training image is a list of such bounding boxes, each labeled with the class of the object contained therein.

The standard approach to solve this task, for instance, by the Single Shot Detector (SSD) [Liu et al., 2015]), is to use a convolutional neural network that produces a sequence of image representations Z_s of size $D_s \times H_s \times W_s$, $s = 1, \dots, S$, with decreasing spatial resolution $H_s \times W_s$ down to 1×1 for $s = S$ (see Figure 6.1). Each of these tensors covers the input image in full, so the h, w indices correspond to a partitioning of the image lattice into regular squares that gets coarser when s increases.

As seen in § 4.2, and illustrated in Figure 4.4, due to the succession of convolutional layers, a feature vector $(Z_s[0, h, w], \dots, Z_s[D_s - 1, h, w])$ is a descriptor of an area of the image, called its

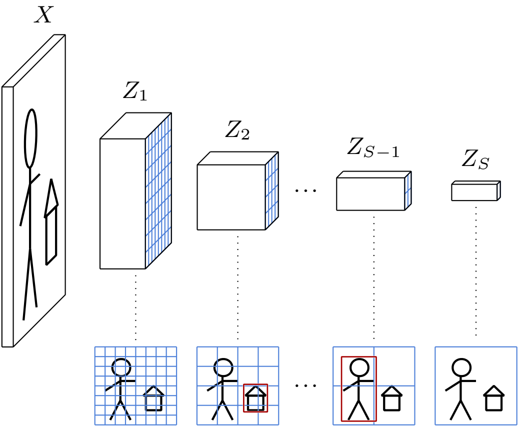

Figure 6.1: *A convolutional object detector processes the input image to generate a sequence of representations of decreasing resolutions. It computes for every* h, w*, at every scale* s*, a pre-defined number of bounding boxes whose centers are in the image area corresponding to that cell, and whose sizes are such that they fit in its receptive field. Each prediction takes the form of the estimates* $(\hat{x}_1, \hat{x}_2, \hat{y}_1, \hat{y}_2)$*, represented by the red boxes above, and a vector of* $C+1$ *logits for the* C *classes of interest, and an additional "no object" class.*

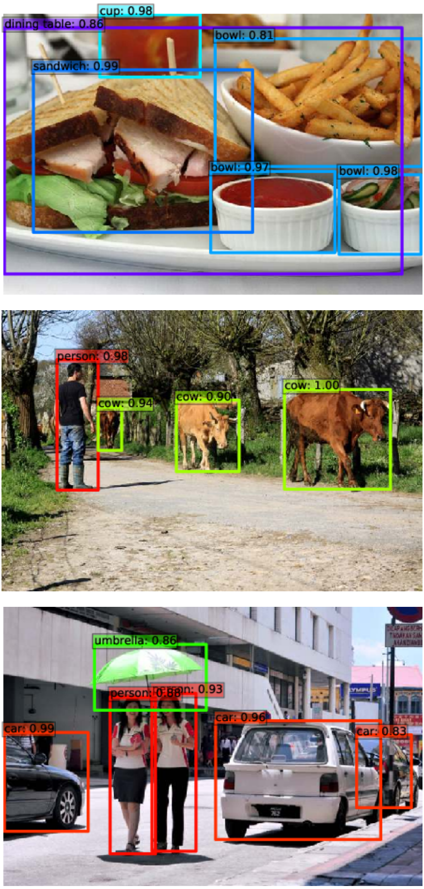

Figure 6.2: *Examples of object detection with the Single-Shot Detector [Liu et al., 2015].*

receptive field, that is larger than this square but centered on it. This results in a non-ambiguous matching of any bounding box (x_1, x_2, y_1, y_2) to a s, h, w, determined respectively by $\max(x_2 - x_1, y_2 - y_1)$, $\frac{y_1+y_2}{2}$, and $\frac{x_1+x_2}{2}$.

Detection is achieved by adding S convolutional layers, each processing a Z_s and computing, for every tensor indices h, w, the coordinates of a bounding box and the associated logits. If there are C object classes, there are $C+1$ logits, the additional one standing for "no object." Hence, each additional convolution layer has $4+C+1$ output channels. The SSD algorithm in particular generates several bounding boxes per s, h, w, each dedicated to a hard-coded range of aspect ratios.

Training sets for object detection are costly to create, since the labeling with bounding boxes requires a slow human intervention. To mitigate this issue, the standard approach is to fine-tune a convolutional model that has been pre-trained on a large classification dataset such as VGG-16 for the original SSD, and to replace its final fully-connected layers with additional convolutional ones. Surprisingly, models trained for classification only learn feature representations that can be repurposed for object detection, even though that task involves the regression of geometric quantities.

During training, every ground-truth bounding box is associated with its s, h, w, and induces a loss term composed of a cross-entropy loss for the logits, and a regression loss such as MSE for the bounding box coordinates. Every other s, h, w free of bounding-box match induces a cross-entropy only penalty to predict the class "no object".

6.4 *Semantic segmentation*

The finest-grain prediction task for image understanding is semantic segmentation, which consists of predicting, for each pixel, the class of the object to which it belongs. This can be achieved with a standard convolutional neural network that outputs a convolutional map with as many channels as classes, carrying the estimated logits for every pixel.

While a standard residual network, for instance, can generate a dense output of the same resolution as its input, as for object detection, this task requires operating at multiple scales. This is necessary so that any object, or sufficiently informative sub-part, regardless of its size, is captured somewhere in the model by the feature representation at a single tensor position. Hence, standard architectures for this task downscale the image with a series of convolutional layers to increase the receptive field of the activations, and re-upscale it with a series of transposed convolutional layers, or other upscaling methods such as bilinear interpolation, to make the prediction at high resolution.

However, a strict downscaling-upscaling architecture does not allow for operating at a fine

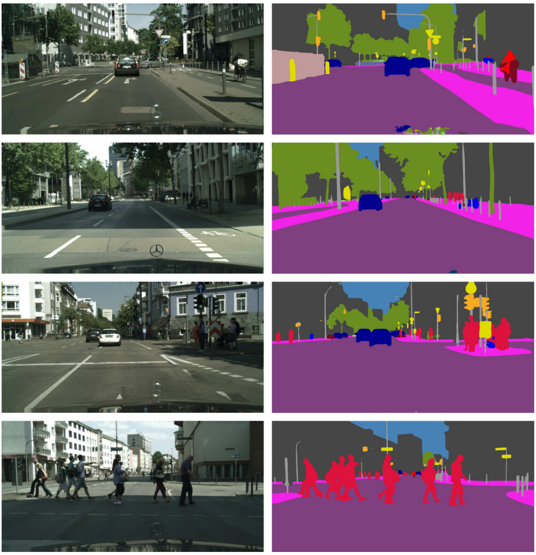

Figure 6.3: *Semantic segmentation results with the Pyramid Scene Parsing Network [Zhao et al., 2016].*

grain when making the final prediction, since all the signal has been transmitted through a low-resolution representation at some point. Models that apply such downscaling-upscaling serially mitigate these issues with skip connections from layers at a certain resolution, before downscaling, to layers at the same resolution, after upscaling [Long et al., 2014; Ronneberger et al., 2015]. Models that do it in parallel, after a convolutional backbone, concatenate the resulting multi-scale representation after upscaling, before making the final per-pixel prediction [Zhao et al., 2016].

Training is achieved with a standard cross-entropy summed over all the pixels. As for object detection, training can start from a network pre-trained on a large-scale image classification dataset to compensate for the limited availability of segmentation ground truth.

6.5 Speech recognition

Speech recognition consists of converting a sound sample into a sequence of words. There have been plenty of approaches to this problem historically, but a conceptually simple and recent one proposed by Radford et al. [2022] consists of casting it as a sequence-to-sequence translation and then solving it with a standard attention-based Transformer, as described in § 5.3.

Their model first converts the sound signal into a spectrogram, which is a one-dimensional series $T \times D$, that encodes at every time step a vector of energies in D frequency bands. The associated text is encoded with the BPE tokenizer (see § 3.2).

The spectrogram is processed through a few 1D convolutional layers, and the resulting representation is fed into the encoder of the Transformer. The decoder directly generates a discrete sequence of tokens, that correspond to one of the possible tasks considered during training. Multiple objectives are considered: transcription of English or non-English text, translation from any language to English, or detection of non-speech sequences, such as background music or ambient noise.

This approach allows leveraging extremely large datasets that combine multiple types of sound sources with diverse ground truths.

It is noteworthy that even though the ultimate goal of this approach is to produce a translation as deterministic as possible given the input signal, it is formally the sampling of a text distribution conditioned on a sound sample, hence a synthesis process. The decoder is, in fact, extremely similar to the generative model of § 7.1.

6.6 *Text-image representations*

A powerful approach to image understanding consists of learning consistent image and text representations, such that an image, or a textual description of it, would be mapped to the same feature vector.

The Contrastive Language-Image Pre-training (CLIP) proposed by Radford et al. [2021] combines an image encoder f, which is a ViT, and a text encoder g, which is a GPT. See § 5.3 for both.

To repurpose a GPT as a text encoder, instead of a standard autoregressive model, they add an "end of sentence" token to the input sequence, and use the representation of this token in the last layer as the embedding. Its dimension is between 512 and 1024, depending on the configuration.

Those two models are trained from scratch using a dataset of 400 million image-text pairs (i_k, t_k) collected from the internet. The training procedure follows the standard mini-batch stochastic gradient descent approach but relies on a contrastive loss. The embeddings are computed for every image and every text of the N pairs in the mini-batch, and a cosine similarity measure is computed not only between text and image embeddings from each pair, but also across pairs, resulting in an $N \times N$ matrix of similarity scores:

$$l_{m,n} = f(i_m) \cdot g(t_n), \, m = 1, \dots, N, n = 1, \dots, N.$$

The model is trained with cross-entropy so that, $\forall n$ the values $l_{1,n}, \dots, l_{N,n}$ interpreted as logit scores predict n, and similarly for $l_{n,1}, \dots, l_{n,N}$. This means that $\forall n, m,$ s.t. $n \neq m$ the similarity $l_{n,n}$ is unambiguously greater than both $l_{n,m}$ and $l_{m,n}$.

When it has been trained, this model can be used to do zero-shot prediction, that is, classifying a signal in the absence of training examples by defining a series of candidate classes with text descriptions, and computing the similarity of the embedding of an image with the embedding of each of those descriptions (see Figure 6.4).

Additionally, since the textual descriptions are often detailed, such a model has to capture a richer representation of images and pick up cues beyond what is necessary for instance for classification. This translates to excellent performance on challenging datasets such as ImageNet Adversarial [Hendrycks et al., 2019] which was specifically designed to degrade or erase cues on which standard predictors rely.

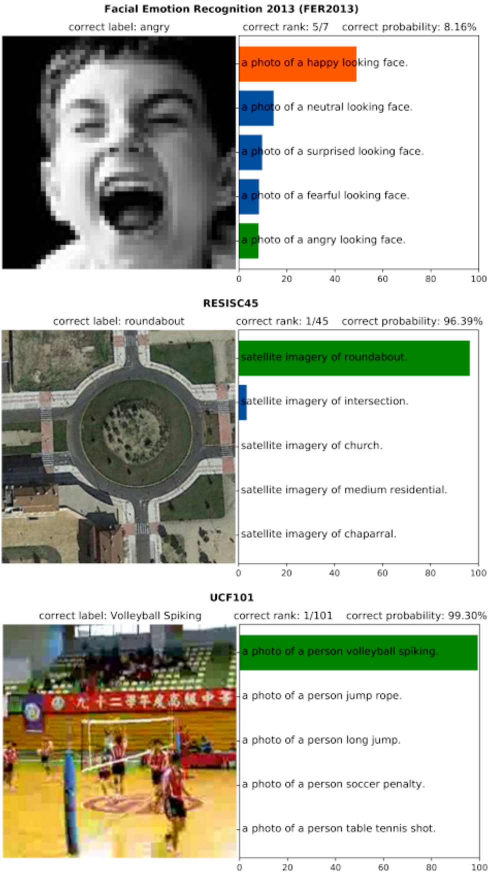

Figure 6.4: *The CLIP text-image embedding [Radford et al., 2021] allows for zero-shot prediction by predicting which class description embedding is the most consistent with the image embedding.*

6.7 *Reinforcement learning*

Many problems, such as strategy games or robotic control, can be formalized with a discrete-time state process S_t and reward process R_t that can be modulated by choosing actions A_t. If S_t is Markovian, meaning that it carries alone as much information about the future as all the past states until that instant, such an object is a Markovian Decision Process (MDP).

Given an MDP, the objective is classically to find a policy π such that $A_t = \pi(S_t)$ maximizes the expectation of the return, which is an accumulated discounted reward:

$$\mathbb{E}\left[\sum_{t \geq 0} \gamma^t R_t\right],$$

for a discount factor $0 < \gamma < 1$.

This is the standard setup of Reinforcement Learning (RL), and it can be worked out by introducing the optimal state-action value function $Q(s, a)$ which is the expected return if we execute action a in state s, and then follow the optimal policy. It provides a means to compute the optimal policy as $\pi(s) = \operatorname{argmax}_a Q(s, a)$, and, thanks to the Markovian assumption, it verifies the Bellman equation:

$$Q(s,a) = \mathbb{E}\left[R_t + \gamma \max_{a'} Q(S_{t+1}, a') \middle| S_t = s, A_t = a \right], \tag{6.1}$$

from which we can design a procedure to train a parametric model $Q(\cdot, \cdot; w)$.

To apply this framework to play classical Atari video games, Mnih et al. [2015] use for S_t the concatenation of the frame at time t and the three that precede, so that the Markovian assumption is reasonable, and use for Q a model dubbed the Deep Q-Network (DQN), composed of two convolutional layers and one fully connected layer with one output value per action, following the classical structure of a LeNet (see § 5.2).

Training is achieved by alternatively playing and recording episodes, and building mini-batches of tuples $(s_n, a_n, r_n, s'_n) \sim (S_t, A_t, R_t, S_{t+1})$ taken across stored episodes and time steps, and minimizing

$$\mathscr{L}(w) = \frac{1}{N} \sum_{n=1}^{N} \left(Q(s_n, a_n; w) - y_n \right)^2 \tag{6.2}$$

with one iteration of SGD, where $y_n = r_n$ if this tuple is the end of the episode, and $y_n = r_n + \gamma \max_a Q(s'_n, a; \bar{w})$ otherwise.

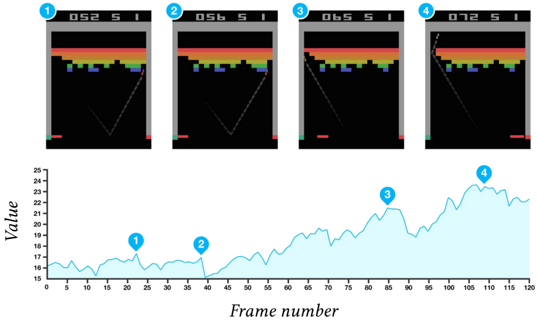

Figure 6.5: *This graph shows the evolution of the state value $V(S_t) = \max_a Q(S_t, a)$ during a game of Breakout. The spikes at time points (1) and (2) correspond to clearing a brick, at time point (3) it is about to break through to the top line, and at (4) it does, which ensures a high future reward [Mnih et al., 2015].*

Here $\bar{w}$ is a constant copy of w, i.e. the gradient does not propagate through it to w. This is necessary since the target value in Equation 6.1 is the expectation of y_n, while it is y_n itself which is used in Equation 6.2. Fixing w in y_n results in a better approximation of the desirable gradient.

A key issue is the policy used to collect episodes. Mnih et al. [2015] simply use the ϵ-greedy strategy, which consists of taking an action completely at random with probability ϵ, and the optimal action $\text{argmax}_a Q(s, a)$ otherwise. Injecting a bit of randomness is necessary to favor exploration.

Training is done with ten million frames corresponding to a bit less than eight days of gameplay. The trained network computes accurate estimates of the state values (see Figure 6.5), and reaches human performance on a majority of the 49 games used in the experimental validation.