CHAPTER 11

Future Directions

Throughout this book we've explored the powerful capabilities of transformers across a wide range of NLP tasks. In this final chapter, we'll shift our perspective and look at some of the current challenges with these models and the research trends that are trying to overcome them. In the first part we explore the topic of scaling up transformers, both in terms of model and corpus size. Then we turn our attention toward various techniques that have been proposed to make the self-attention mechanism more efficient. Finally, we explore the emerging and exciting field of *multimodal transformers*, which can model inputs across multiple domains like text, images, and audio.

Scaling Transformers

In 2019, the researcher Richard Sutton wrote a provocative essay entitled "The Bitter Lesson" in which he argued that:

> The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin…. Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation. These two need not run counter to each other, but in practice they tend to…. And the human-knowledge approach tends to complicate methods in ways that make them less suited to taking advantage of general methods leveraging computation.

The essay provides several historical examples, such as playing chess or Go, where the approach of encoding human knowledge within AI systems was ultimately outdone by increased computation. Sutton calls this the "bitter lesson" for the AI research field:

> We have to learn the bitter lesson that building in how we think we think does not work in the long run…. One thing that should be learned from the bitter lesson is the great power of general purpose methods, of methods that continue to scale with increased computation even as the available computation becomes very great. The two methods that seem to scale arbitrarily in this way are *search* and *learning*.

There are now signs that a similar lesson is at play with transformers; while many of the early BERT and GPT descendants focused on tweaking the architecture or pretraining objectives, the best-performing models in mid-2021, like GPT-3, are essentially basic scaled-up versions of the original models without many architectural modifications. In Figure 11-1 you can see a timeline of the development of the largest models since the release of the original Transformer architecture in 2017, which shows that model size has increased by over four orders of magnitude in just a few years!

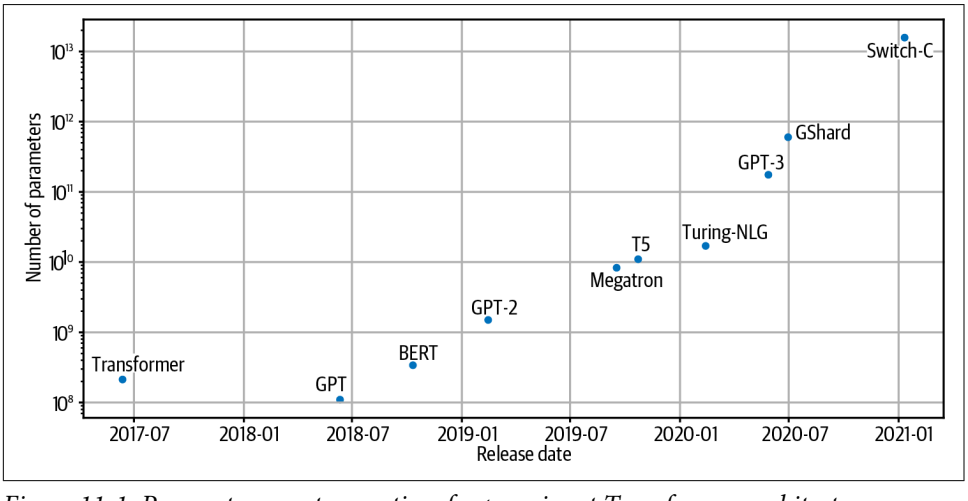


Figure 11-1. Parameter counts over time for prominent Transformer architectures

This dramatic growth is motivated by empirical evidence that large language models perform better on downstream tasks and that interesting capabilities such as zero-shot and few-shot learning emerge in the 10- to 100-billion parameter range. However, the number of parameters is not the only factor that affects model performance; the amount of compute and training data must also be scaled in tandem to train these monsters. Given that large language models like GPT-3 are estimated to cost $4.6 million to train, it is clearly desirable to be able to estimate the model's performance in advance. Somewhat surprisingly, the performance of language models appears to obey a *power law relationship* with model size and other factors that is codified in a set of scaling laws.[1] Let's take a look at this exciting area of research.

1 J. Kaplan et al., "Scaling Laws for Neural Language Models", (2020).

Scaling Laws

Scaling laws allow one to empirically quantify the "bigger is better" paradigm for language models by studying their behavior with varying compute budget C, dataset size D, and model size N.[2] The basic idea is to chart the dependence of the cross-entropy loss L on these three factors and determine if a relationship emerges. For autoregressive models like those in the GPT family, the resulting loss curves are shown in Figure 11-2, where each blue curve represents the training run of a single model.

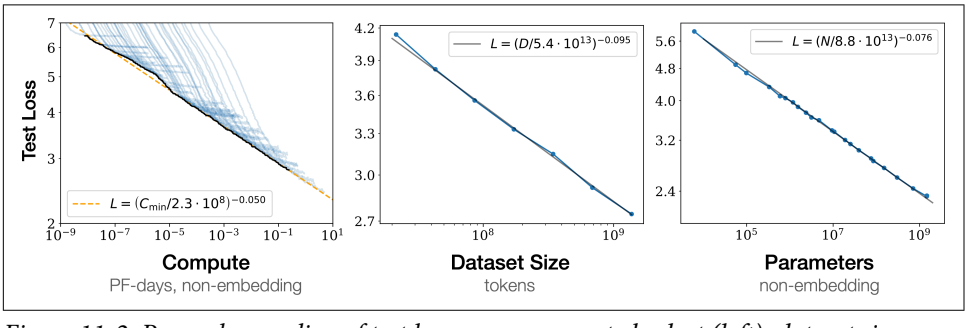


Figure 11-2. Power-law scaling of test loss versus compute budget (left), dataset size (middle), and model size (right) (courtesy of Jared Kaplan)

From these loss curves we can draw a few conclusions about:

The relationship of performance and scale
: Although many NLP researchers focus on architectural tweaks or hyperparameter optimization (like tuning the number of layers or attention heads) to improve performance on a fixed set of datasets, the implication of scaling laws is that a more productive path toward better models is to focus on increasing N, C, and D in tandem.

Smooth power laws
: The test loss L has a power law relationship with each of N, C, and D across several orders of magnitude (power law relationships are linear on a log-log scale). For $X = N, C, D$ we can express these power law relationships as $L(X) \sim 1/X^{\alpha}$, where α is a scaling exponent that is determined by a fit to the loss curves shown in Figure 11-2.[3] Typical values for α_X lie in the 0.05–0.095 range, and one attractive feature of these power laws is that the early part of a loss curve can be extrapolated to predict what the approximate loss would be if training was conducted for much longer.

2 The dataset size is measured in the number of tokens, while the model size excludes parameters from the embedding layers.

3 T. Henighan et al., "Scaling Laws for Autoregressive Generative Modeling", (2020).

Sample efficiency
: Large models are able to reach the same performance as smaller models with a smaller number of training steps. This can be seen by comparing the regions where a loss curve plateaus over some number of training steps, which indicates one gets diminishing returns in performance compared to simply scaling up the model.

Somewhat surprisingly, scaling laws have also been observed for other modalities, like images, videos, and mathematical problem solving, as illustrated in Figure 11-3.

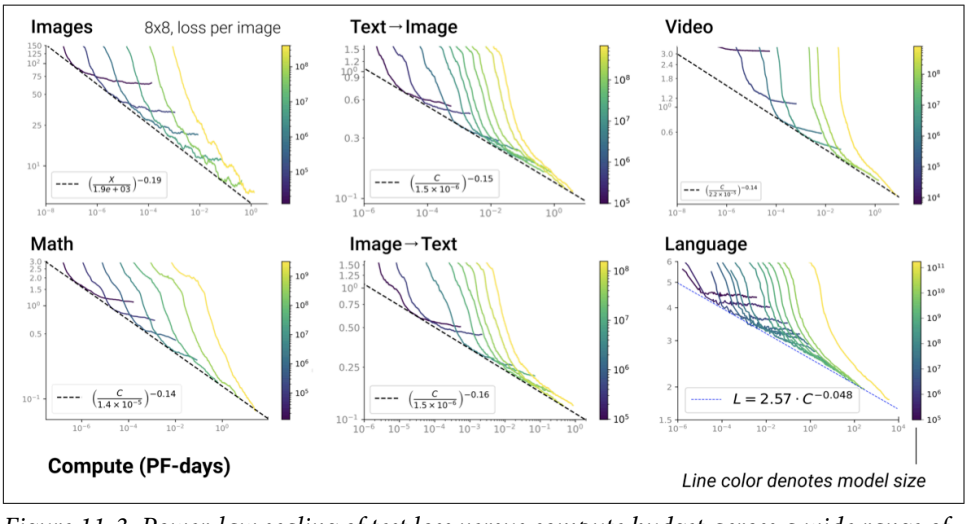


Figure 11-3. Power-law scaling of test loss versus compute budget across a wide range of modalities (courtesy of Tom Henighan)

Whether power-law scaling is a universal property of transformer language models is currently unknown. For now, we can use scaling laws as a tool to extrapolate large, expensive models without having to explicitly train them. However, scaling isn't quite as easy as it sounds. Let's now look at a few challenges that crop up when charting this frontier.

Challenges with Scaling

While scaling up sounds simple in theory ("just add more layers!"), in practice there are many difficulties. Here are a few of the biggest challenges you're likely to encounter when scaling language models:

Infrastructure
: Provisioning and managing infrastructure that potentially spans hundreds or thousands of nodes with as many GPUs is not for the faint-hearted. Are the required number of nodes available? Is communication between nodes a bottle-

neck? Tackling these issues requires a very different skill set than that found in most data science teams, and typically involves specialized engineers familiar with running large-scale, distributed experiments.

Cost

Most ML practitioners have experienced the feeling of waking up in the middle of the night in a cold sweat, remembering they forgot to shut down that fancy GPU on the cloud. This feeling intensifies when running large-scale experiments, and most companies cannot afford the teams and resources necessary to train models at the largest scales. Training a single GPT-3-sized model can cost several million dollars, which is not the kind of pocket change that many companies have lying around.[4]

Dataset curation

A model is only as good as the data it is trained on. Training large models requires large, high-quality datasets. When using terabytes of text data it becomes harder to make sure the dataset contains high-quality text, and even preprocessing becomes challenging. Furthermore, one needs to ensure that there is a way to control biases like sexism and racism that these language models can acquire when trained on large-scale webtext corpora. Another type of consideration revolves around licensing issues with the training data and personal information that can be embedded in large text datasets.

Model evaluation

Once the model is trained, the challenges don't stop. Evaluating the model on downstream tasks again requires time and resources. In addition, you'll want to probe the model for biased and toxic generations, even if you are confident that you created a clean dataset. These steps take time and need to be carried out thoroughly to minimize the risks of adverse effects later on.

Deployment

Finally, serving large language models also poses a significant challenge. In Chapter 8 we looked at a few approaches, such as distillation, pruning, and quantization, to help with these issues. However, this may not be enough if you are starting with a model that is hundreds of gigabytes in size. Hosted services such as the OpenAI API or Hugging Face's Accelerated Inference API are designed to help companies that cannot or do not want to deal with these deployment challenges.

4 However, recently a distributed deep learning framework has been proposed that enables smaller groups to pool their computational resources and pretrain models in a collaborative fashion. See M. Diskin et al., "Distributed Deep Learning in Open Collaborations", (2021).

This is by no means an exhaustive list, but it should give you an idea of the kinds of considerations and challenges that go hand in hand with scaling language models to ever larger sizes. While most of these efforts are centralized around a few institutions that have the resources and know-how to push the boundaries, there are currently two community-led projects that aim to produce and probe large language models in the open:

BigScience
: This is a one-year-long research workshop that runs from 2021 to 2022 and is focused on large language models. The workshop aims to foster discussions and reflections around the research questions surrounding these models (capabilities, limitations, potential improvements, bias, ethics, environmental impact, role in the general AI/cognitive research landscape) as well as the challenges around creating and sharing such models and datasets for research purposes and among the research community. The collaborative tasks involve creating, sharing, and evaluating a large multilingual dataset and a large language model. An unusually large compute budget was allocated for these collaborative tasks (several million GPU hours on several thousands GPUs). If successful, this workshop will run again in the future, focusing on involving an updated or different set of collaborative tasks. If you want to join the effort, you can find more information at the project's website.

EleutherAI
: This is a decentralized collective of volunteer researchers, engineers, and developers focused on AI alignment, scaling, and open source AI research. One of its aims is to train and open-source a GPT-3-sized model, and the group has already released some impressive models like GPT-Neo and GPT-J, which is a 6-billion-parameter model and currently the best-performing publicly available transformer in terms of zero-shot performance. You can find more information at EleutherAI's website.

Now that we've explored how to scale transformers across compute, model size, and dataset size, let's examine another active area of research: making self-attention more efficient.

Attention Please!

We've seen throughout this book that the self-attention mechanism plays a central role in the architecture of transformers; after all, the original Transformer paper is called "Attention Is All You Need"! However, there is a key challenge associated with self-attention: since the weights are generated from pairwise comparisons of all the tokens in a sequence, this layer becomes a computational bottleneck when trying to process long documents or apply transformers to domains like speech processing or computer vision. In terms of time and memory complexity, the self-attention layer of

the Transformer architecture naively scales like $\mathcal{O}\left(n^2\right)$, where n is the length of the sequence.[5]

As a result, much of the recent research on transformers has focused on making self-attention more efficient. The research directions are broadly clustered in Figure 11-4.

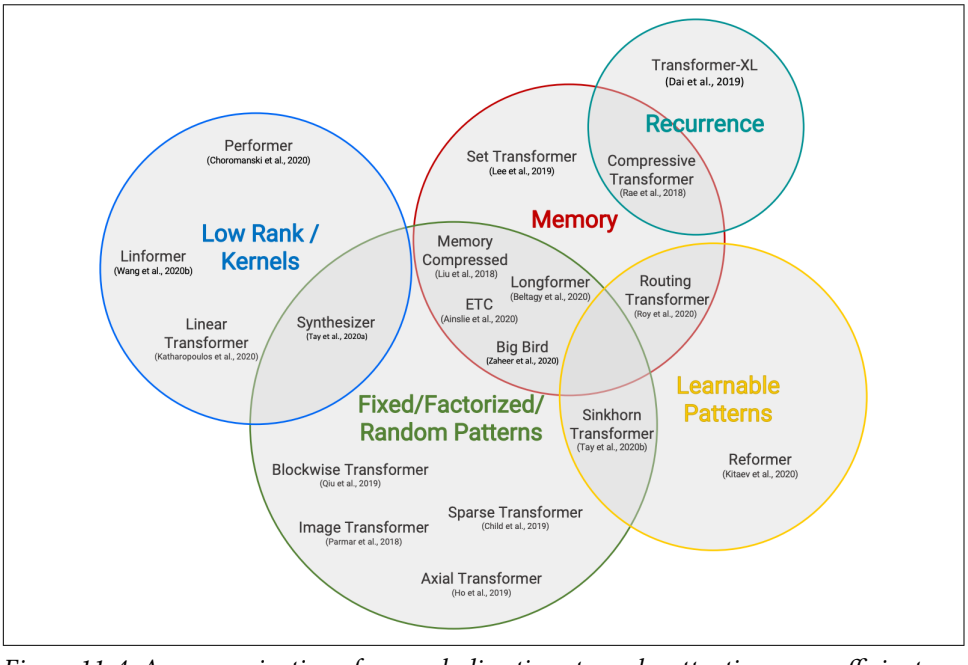


Figure 11-4. A summarization of research directions to make attention more efficient (courtesy of Yi Tay et al.)[6]

A common pattern is to make attention more efficient by introducing sparsity into the attention mechanism or by applying kernels to the attention matrix. Let's take a quick look at some of the most popular approaches to make self-attention more efficient, starting with sparsity.

Sparse Attention

One way to reduce the number of computations that are performed in the self-attention layer is to simply limit the number of query-key pairs that are generated

5 Although standard implementations of self-attention have $O\left(n^2\right)$ time and memory complexity, a recent paper by Google researchers shows that the memory complexity can be reduced to $O(\log n)$ via a simple reordering of the operations.

6 Yi Tay et al., "Efficient Transformers: A Survey", (2020).

according to some predefined pattern. There have been many sparsity patterns explored in the literature, but most of them can be decomposed into a handful of "atomic" patterns illustrated in Figure 11-5.

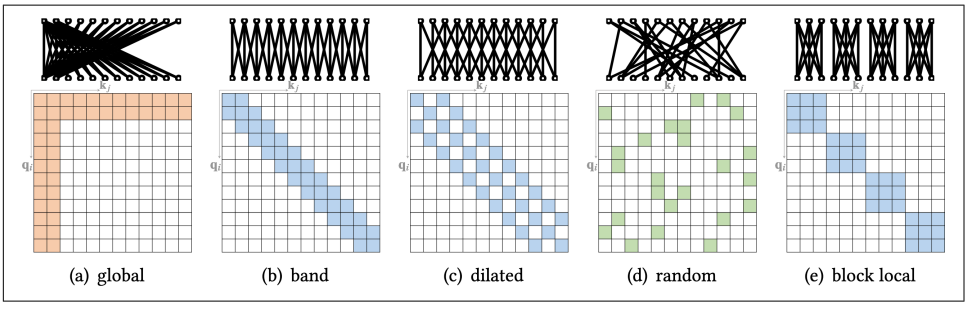


Figure 11-5. Common atomic sparse attention patterns for self-attention: a colored square means the attention score is calculated, while a blank square means the score is discarded (courtesy of Tianyang Lin)

We can describe these patterns as follows:[7]

Global attention
: Defines a few special tokens in the sequence that are allowed to attend to all other tokens

Band attention
: Computes attention over a diagonal band

Dilated attention
: Skips some query-key pairs by using a dilated window with gaps

Random attention
: Randomly samples a few keys for each query to compute attention scores

Block local attention
: Divides the sequence into blocks and restricts attention within these blocks

In practice, most transformer models with sparse attention use a mix of the atomic sparsity patterns shown in Figure 11-5 to generate the final attention matrix. As illustrated in Figure 11-6, models like Longformer use a mix of global and band attention, while BigBird adds random attention to the mix. Introducing sparsity into the attention matrix enables these models to process much longer sequences; in the case of Longformer and BigBird the maximum sequence length is 4,096 tokens, which is 8 times larger than BERT!

7 T. Lin et al., "A Survey of Transformers", (2021).

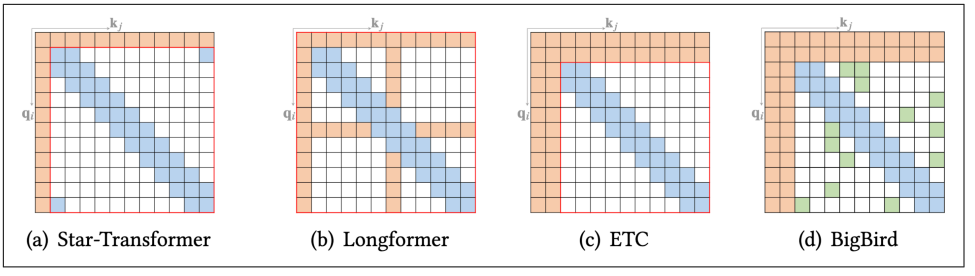


Figure 11-6. Sparse attention patterns for recent transformer models (courtesy of Tianyang Lin)

It is also possible to *learn* the sparsity pattern in a data-driven manner. The basic idea behind such approaches is to cluster the tokens into chunks. For example, Reformer uses a hash function to cluster similar tokens together.

Now that we've seen how sparsity can reduce the complexity of self-attention, let's take a look at another popular approach based on changing the operations directly.

Linearized Attention

An alternative way to make self-attention more efficient is to change the order of operations that are involved in computing the attention scores. Recall that to compute the self-attention scores of the queries and keys we need a similarity function, which for the transformer is just a simple dot product. However, for a general similarity function $\text{sim}(q_i, k_j)$ we can express the attention outputs as the following equation:

$$y_i = \sum_j \frac{\text{sim}(Q_i, K_j)}{\Sigma_k \text{sim}(Q_i, K_k)} V_j$$

The trick behind linearized attention mechanisms is to express the similarity function as a *kernel function* that decomposes the operation into two pieces:

$$\text{sim}(Q_j, K_j) = \phi(Q_i)^T \phi(K_j)$$

where ϕ is typically a high-dimensional feature map. Since $\phi(Q_i)$ is independent of j and k, we can pull it under the sums to write the attention outputs as follows:

$$y_i = \frac{\phi(Q_i)^T \Sigma_j \phi(K_j) V_j^T}{\phi(Q_i)^T \Sigma_k \phi(K_k)}$$

By first computing $\Sigma_j \phi(K_j) V_j^T$ and $\Sigma_k \phi(K_k)$, we can effectively linearize the space and time complexity of self-attention! The comparison between the two approaches is illustrated in Figure 11-7. Popular models that implement linearized self-attention include Linear Transformer and Performer.[8]

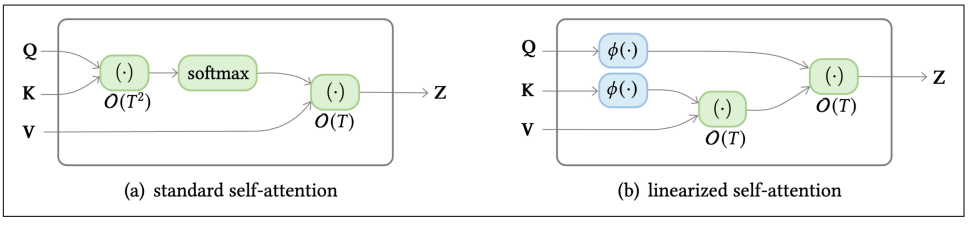


Figure 11-7. Complexity difference between standard self-attention and linearized self-attention (courtesy of Tianyang Lin)

In this section we've seen how Transformer architectures in general and attention in particular can be scaled up to achieve even better performance on a wide range of tasks. In the next section we'll have a look at how transformers are branching out of NLP into other domains such as audio and computer vision.

Going Beyond Text

Using text to train language models has been the driving force behind the success of transformer language models, in combination with transfer learning. On the one hand, text is abundant and enables self-supervised training of large models. On the other hand, textual tasks such as classification and question answering are common, and developing effective strategies for them allows us to address a wide range of real-world problems.

However, there are limits to this approach, including:

Human reporting bias
: The frequencies of events in text may not represent their true frequencies.[9] A model solely trained on text from the internet might have a very distorted image of the world.

8 A. Katharopoulos et al., "Transformers Are RNNs: Fast Autoregressive Transformers with Linear Attention", (2020); K. Choromanski et al., "Rethinking Attention with Performers", (2020).

9 J. Gordon and B. Van Durme, "Reporting Bias and Knowledge Extraction", (2013).

Common sense
: Common sense is a fundamental quality of human reasoning, but is rarely written down. As such, language models trained on text might know many facts about the world, but lack basic common-sense reasoning.

Facts
: A probabilistic language model cannot store facts in a reliable way and can produce text that is factually wrong. Similarly, such models can detect named entities, but have no direct way to access information about them.

Modality
: Language models have no way to connect to other modalities that could address the previous points, such as audio or visual signals or tabular data.

So, if we could solve the modality limitations we could potentially address some of the others as well. Recently there has been a lot of progress in pushing transformers to new modalities, and even building multimodal models. In this section we'll highlight a few of these advances.

Vision

Vision has been the stronghold of convolutional neural networks (CNNs) since they kickstarted the deep learning revolution. More recently, transformers have begun to be applied to this domain and to achieve efficiency similar to or better than CNNs. Let's have a look at a few examples.

iGPT

Inspired by the success of the GPT family of models with text, iGPT (short for image GPT) applies the same methods to images.[10] By viewing images as sequences of pixels, iGPT uses the GPT architecture and autoregressive pretraining objective to predict the next pixel values. Pretraining on large image datasets enables iGPT to "autocomplete" partial images, as displayed in Figure 11-8. It also achieves performant results on classification tasks when a classification head is added to the model.

10 M. Chen et al., "Generative Pretraining from Pixels," *Proceedings of the 37th International Conference on Machine Learning* 119 (2020):1691–1703, *https://proceedings.mlr.press/v119/chen20s.html*.

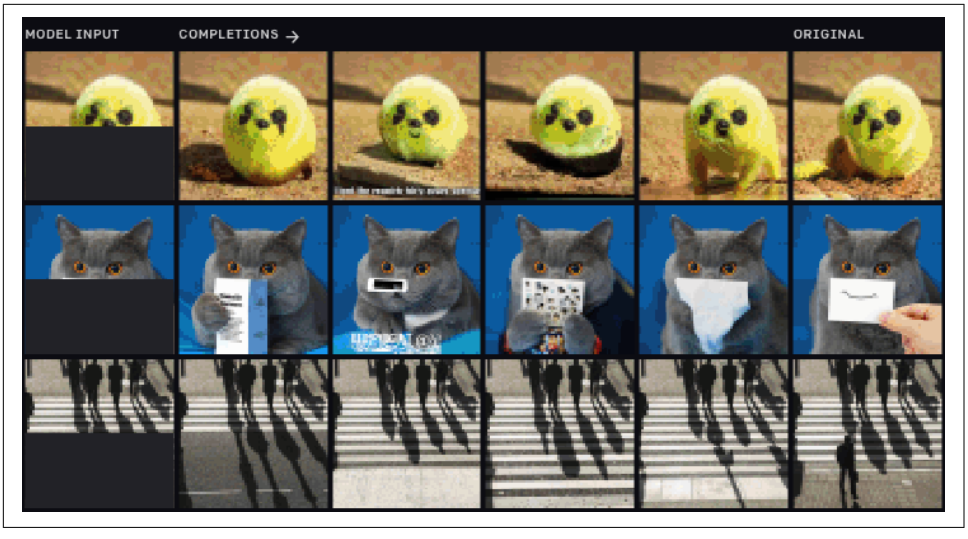


Figure 11-8. Examples of image completions with iGPT (courtesy of Mark Chen)

ViT

We saw that iGPT follows closely the GPT-style architecture and pretraining procedure. Vision Transformer (ViT)[11] is a BERT-style take on transformers for vision, as illustrated in Figure 11-9. First the image is split into smaller patches, and each of these patches is embedded with a linear projection. The results strongly resemble the token embeddings in BERT, and what follows is virtually identical. The patch embeddings are combined with position embeddings and then fed through an ordinary transformer encoder. During pretraining some of the patches are masked or distorted, and the objective is to predict the average color of the masked patch.

11 A. Dosovitskiy et al., "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale", (2020).

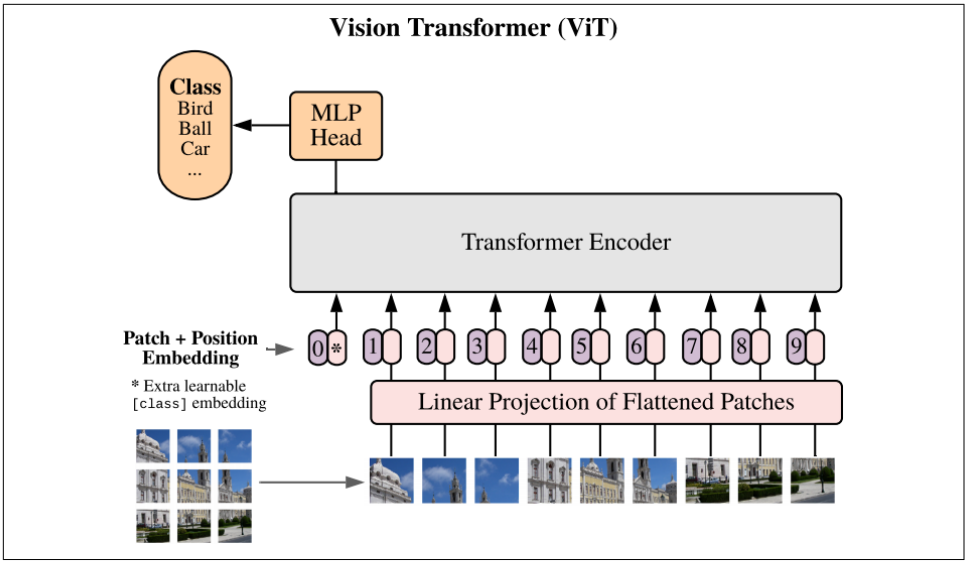


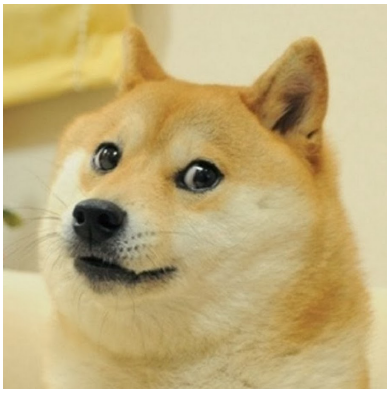
Figure 11-9. The ViT architecture (courtesy of Alexey Dosovitskiy et al.)

Although this approach did not produce better results when pretrained on the standard ImageNet dataset, it scaled significantly better than CNNs on larger datasets.

ViT is integrated in 🤗 Transformers, and using it is very similar to the NLP pipelines that we've used throughout this book. Let's start by loading the image of a rather famous dog:

```
from PIL import Image
import matplotlib.pyplot as plt

image = Image.open("images/doge.jpg")
plt.imshow(image)
plt.axis("off")
plt.show()
```



To load a ViT model, we just need to specify the `image-classification` pipeline, and then we feed in the image to extract the predicted classes:

```
import pandas as pd
from transformers import pipeline

image_classifier = pipeline("image-classification")
preds = image_classifier(image)
preds_df = pd.DataFrame(preds)
preds_df
```

	score	label
0	0.643599	Eskimo dog, husky
1	0.207407	Siberian husky
2	0.060160	dingo, warrigal, warragal, Canis dingo
3	0.035359	Norwegian elkhound, elkhound
4	0.012927	malamute, malemute, Alaskan malamute

Great, the predicted class seems to match the image!

A natural extension of image models is video models. In addition to the spatial dimensions, videos come with a temporal dimension. This makes the task more challenging as the volume of data gets much bigger and one needs to deal with the extra dimension. Models such as TimeSformer introduce a spatial and temporal attention mechanism to account for both.[12] In the future, such models can help build tools for a wide range of tasks such as classification or annotation of video sequences.

12 G. Bertasius, H. Wang, and L. Torresani, "Is Space-Time Attention All You Need for Video Understanding?", (2021).

Tables

A lot of data, such as customer data within a company, is stored in structured databases instead of as raw text. We saw in Chapter 7 that with question answering models we can query text with a question in natural text. Wouldn't it be nice if we could do the same with tables, as shown in Figure 11-10?

Table

Rank	Name	No. of reigns	Combined days
1	Lou Thesz	3	3,749
2	Ric Flair	8	3,103
3	Harley Race	7	1,799
4	Dory Funk Jr.	1	1,563
5	Dan Severn	2	1,559
6	Gene Kiniski	1	1,131

Example questions

#	Question	Answer	Example Type
1	*Which wrestler had the most number of reigns?*	Ric Flair	Cell selection
2	*Average time as champion for top 2 wrestlers?*	AVG(3749,3103)=3426	Scalar answer
3	*How many world champions are there with only one reign?*	COUNT(Dory Funk Jr., Gene Kiniski)=2	Ambiguous answer
4	*What is the number of reigns for* Harley Race?	7	Ambiguous answer
5	*Which of the following wrestlers were ranked in the bottom 3?*	{Dory Funk Jr., Dan Severn, Gene Kiniski}	Cell selection
	Out of these, who had more than one reign?	Dan Severn	Cell selection

Figure 11-10. Question answering over a table (courtesy of Jonathan Herzig)

TAPAS (short for Table Parser)[13] to the rescue! This model applies the Transformer architecture to tables by combining the tabular information with the query, as illustrated in Figure 11-11.

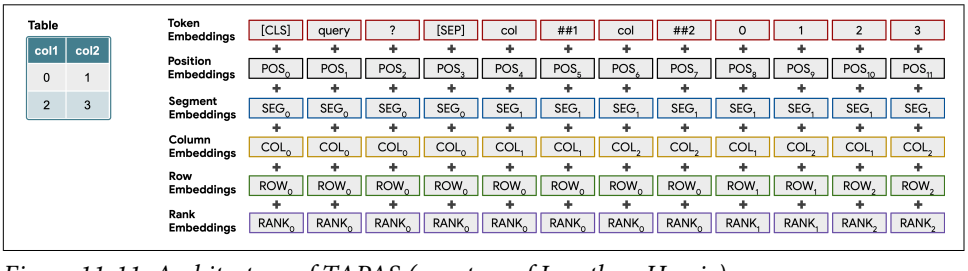


Figure 11-11. Architecture of TAPAS (courtesy of Jonathan Herzig)

Let's look at an example of how TAPAS works in practice. We have created a fictitious version of this book's table of contents. It contains the chapter number, the name of the chapter, as well as the starting and ending pages of the chapters:

```
book_data = [
    {"chapter": 0, "name": "Introduction", "start_page": 1, "end_page": 11},
    {"chapter": 1, "name": "Text classification", "start_page": 12,
     "end_page": 48},
    {"chapter": 2, "name": "Named Entity Recognition", "start_page": 49,
     "end_page": 73},
    {"chapter": 3, "name": "Question Answering", "start_page": 74,
```

13 J. Herzig et al., "TAPAS: Weakly Supervised Table Parsing via Pre-Training", (2020).

```
     "end_page": 120},
    {"chapter": 4, "name": "Summarization", "start_page": 121,
     "end_page": 140},
    {"chapter": 5, "name": "Conclusion", "start_page": 141,
     "end_page": 144}
]
```

We can also easily add the number of pages each chapter has with the existing fields. In order to play nicely with the TAPAS model, we need to make sure that all columns are of type `str`:

```
table = pd.DataFrame(book_data)
table['number_of_pages'] = table['end_page']-table['start_page']
table = table.astype(str)
table
```

	chapter	name	start_page	end_page	number_of_pages
0	0	Introduction	1	11	10
1	1	Text classification	12	48	36
2	2	Named Entity Recognition	49	73	24
3	3	Question Answering	74	120	46
4	4	Summarization	121	140	19
5	5	Conclusion	141	144	3

By now you should know the drill. We first load the `table-question-answering` pipeline:

```
table_qa = pipeline("table-question-answering")
```

and then pass some queries to extract the answers:

```
table_qa = pipeline("table-question-answering")
queries = ["What's the topic in chapter 4?",
           "What is the total number of pages?",
           "On which page does the chapter about question-answering start?",
           "How many chapters have more than 20 pages?"]
preds = table_qa(table, queries)
```

These predictions store the type of table operation in an `aggregator` field, along with the answer. Let's see how well TAPAS fared on our questions:

```
for query, pred in zip(queries, preds):
    print(query)
    if pred["aggregator"] == "NONE":
        print("Predicted answer: " + pred["answer"])
    else:
        print("Predicted answer: " + pred["answer"])
    print('='*50)
```

```
What's the topic in chapter 4?
Predicted answer: Summarization
==================================================
What is the total number of pages?
Predicted answer: SUM > 10, 36, 24, 46, 19, 3
==================================================
On which page does the chapter about question-answering start?
Predicted answer: AVERAGE > 74
==================================================
How many chapters have more than 20 pages?
Predicted answer: COUNT > 1, 2, 3
==================================================
```

For the first chapter, the model predicted exactly one cell with no aggregation. If we look at the table, we see that the answer is in fact correct. In the next example the model predicted all the cells containing the number of pages in combination with the sum aggregator, which again is the correct way of calculating the total number of pages. The answer to question three is also correct; the average aggregation is not necessary in that case, but it doesn't make a difference. Finally, we have a question that is a little bit more complex. To determine how many chapters have more than 20 pages we first need to find out which chapters satisfy that criterion and then count them. It seem that TAPAS again got it right and correctly determined that chapters 1, 2, and 3 have more than 20 pages, and added a count aggregator to the cells.

The kinds of questions we asked can also be solved with a few simple Pandas commands; however, the ability to ask questions in natural language instead of Python code allows a much wider audience to query the data to answer specific questions. Imagine such tools in the hands of business analysts or managers who are able verify their own hypotheses about the data!

Multimodal Transformers

So far we've looked at extending transformers to a single new modality. TAPAS is arguably multimodal since it combines text and tables, but the table is also treated as text. In this section we examine transformers that combine two modalities at once: audio plus text and vision plus text.

Speech-to-Text

Although being able to use text to interface with a computer is a huge step forward, using spoken language is an even more natural way for us to communicate. You can see this trend in industry, where applications such as Siri and Alexa are on the rise and becoming progressively more useful. Also, for a large fraction of the population, writing and reading are more challenging than speaking. So, being able to process and understand audio is not only convenient, but can help many people access more information. A common task in this domain is *automatic speech recognition* (ASR),

which converts spoken words to text and enables voice technologies like Siri to answer questions like "What is the weather like today?"

The wav2vec 2.0 family of models are one of the most recent developments in ASR: they use a transformer layer in combination with a CNN, as illustrated in Figure 11-12.[14] By leveraging unlabeled data during pretraining, these models achieve competitive results with only a few minutes of labeled data.

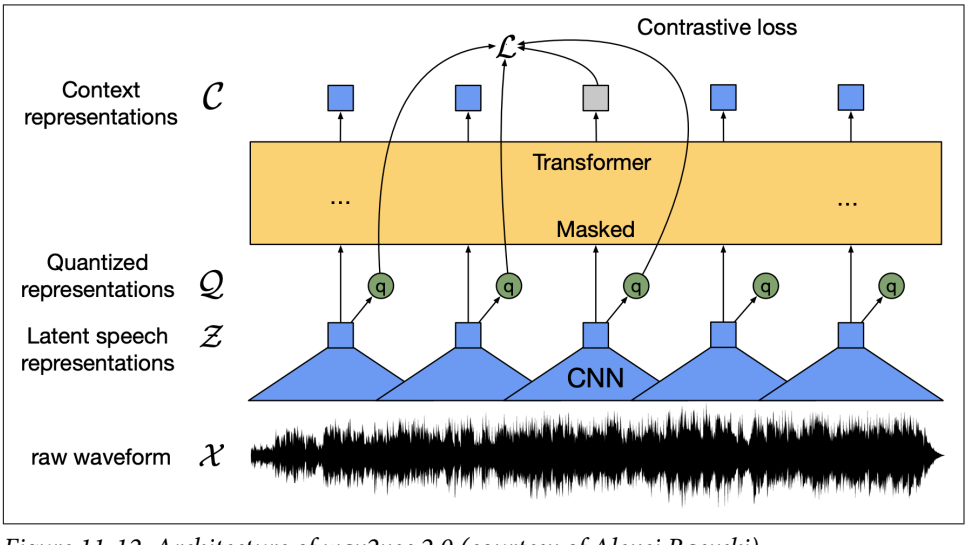


Figure 11-12. Architecture of wav2vec 2.0 (courtesy of Alexei Baevski)

The wav2vec 2.0 models are integrated in 🤗 Transformers, and you won't be surprised to learn that loading and using them follows the familiar steps that we have seen throughout this book. Let's load a pretrained model that was trained on 960 hours of speech audio:

```
asr = pipeline("automatic-speech-recognition")
```

To apply this model to some audio files we'll use the ASR subset of the SUPERB dataset, which is the same dataset the model was pretrained on. Since the dataset is quite large, we'll just load one example for our demo purposes:

```
from datasets import load_dataset

ds = load_dataset("superb", "asr", split="validation[:1]")
print(ds[0])
```

```
{'chapter_id': 128104, 'speaker_id': 1272, 'file': '~/.cache/huggingf
ace/datasets/downloads/extracted/e4e70a454363bec1c1a8ce336139866a39442114d86a433
```

14 A. Baevski et al., "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations", (2020).

```
6014acd4b1ed55e55/LibriSpeech/dev-clean/1272/128104/1272-128104-0000.flac',
'id': '1272-128104-0000', 'text': 'MISTER QUILTER IS THE APOSTLE OF THE MIDDLE
CLASSES AND WE ARE GLAD TO WELCOME HIS GOSPEL'}
```

Here we can see that the audio in the `file` column is stored in the FLAC coding format, while the expected transcription is given by the `text` column. To convert the audio to an array of floats, we can use the *SoundFile* library to read each file in our dataset with `map()`:

```
import soundfile as sf

def map_to_array(batch):
    speech, _ = sf.read(batch["file"])
    batch["speech"] = speech
    return batch

ds = ds.map(map_to_array)
```

If you are using a Jupyter notebook you can easily play the sound files with the following `IPython` widgets:

```
from IPython.display import Audio

display(Audio(ds[0]['speech'], rate=16000))
```

Finally, we can pass the inputs to the pipeline and inspect the prediction:

```
pred = asr(ds[0]["speech"])
print(pred)
```

```
{'text': 'MISTER QUILTER IS THE APOSTLE OF THE MIDDLE CLASSES AND WE ARE GLAD TO
WELCOME HIS GOSPEL'}
```

This transcription seems to be correct. We can see that some punctuation is missing, but this is hard to get from audio alone and could be added in a postprocessing step. With only a handful of lines of code we can build ourselves a state-of-the-art speech-to-text application!

Building a model for a new language still requires a minimum amount of labeled data, which can be challenging to obtain, especially for low-resource languages. Soon after the release of wav2vec 2.0, a paper describing a method named wav2vec-U was published.[15] In this work, a combination of clever clustering and GAN training is used to build a speech-to-text model using only independent unlabeled speech and unlabeled text data. This process is visualized in detail in Figure 11-13. No aligned speech and text data is required at all, which enables the training of highly performant speech-to-text models for a much larger spectrum of languages.

15 A. Baevski et al., "Unsupervised Speech Recognition", (2021).

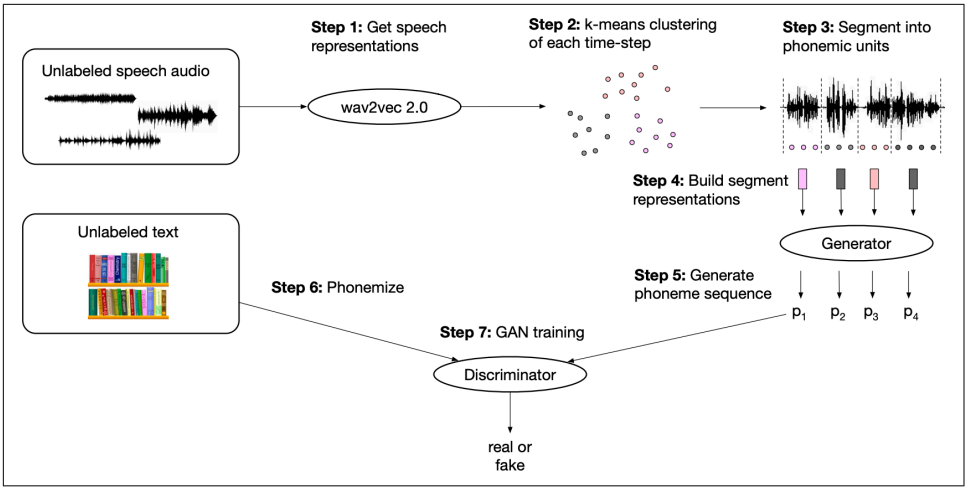


Figure 11-13. Training scheme for wav2vec-U (courtesy of Alexsei Baevski)

Great, so transformers can now "read" text and "hear" audio—can they also "see"? The answer is yes, and this is one of the current hot research frontiers in the field.

Vision and Text

Vision and text are another natural pair of modalities to combine since we frequently use language to communicate and reason about the contents of images and videos. In addition to the vision transformers, there have been several developments in the direction of combining visual and textual information. In this section we will look at four examples of models combining vision and text: VisualQA, LayoutLM, DALL·E, and CLIP.

VQA

In Chapter 7 we explored how we can use transformer models to extract answers to text-based questions. This can be done ad hoc to extract information from texts or offline, where the question answering model is used to extract structured information from a set of documents. There have been several efforts to expand this approach to vision with datasets such as VQA,[16] shown in Figure 11-14.

16 Y. Goyal et al., "Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering", (2016).

Figure 11-14. Example of a visual question answering task from the VQA dataset (courtesy of Yash Goyal)

Models such as LXMERT and VisualBERT use vision models like ResNets to extract features from the pictures and then use transformer encoders to combine them with the natural questions and predict an answer.[17]

LayoutLM

Analyzing scanned business documents like receipts, invoices, or reports is another area where extracting visual and layout information can be a useful way to recognize text fields of interest. Here the LayoutLM family of models are the current state of the art. They use an enhanced Transformer architecture that receives three modalities as input: text, image, and layout. Accordingly, as shown in Figure 11-15, there are embedding layers associated with each modality, a spatially aware self-attention mechanism, and a mix of image and text/image pretraining objectives to align the different modalities. By pretraining on millions of scanned documents, LayoutLM models are able to transfer to various downstream tasks in a manner similar to BERT for NLP.

17 H. Tan and M. Bansal, "LXMERT: Learning Cross-Modality Encoder Representations from Transformers", (2019); L.H. Li et al., "VisualBERT: A Simple and Performant Baseline for Vision and Language", (2019).

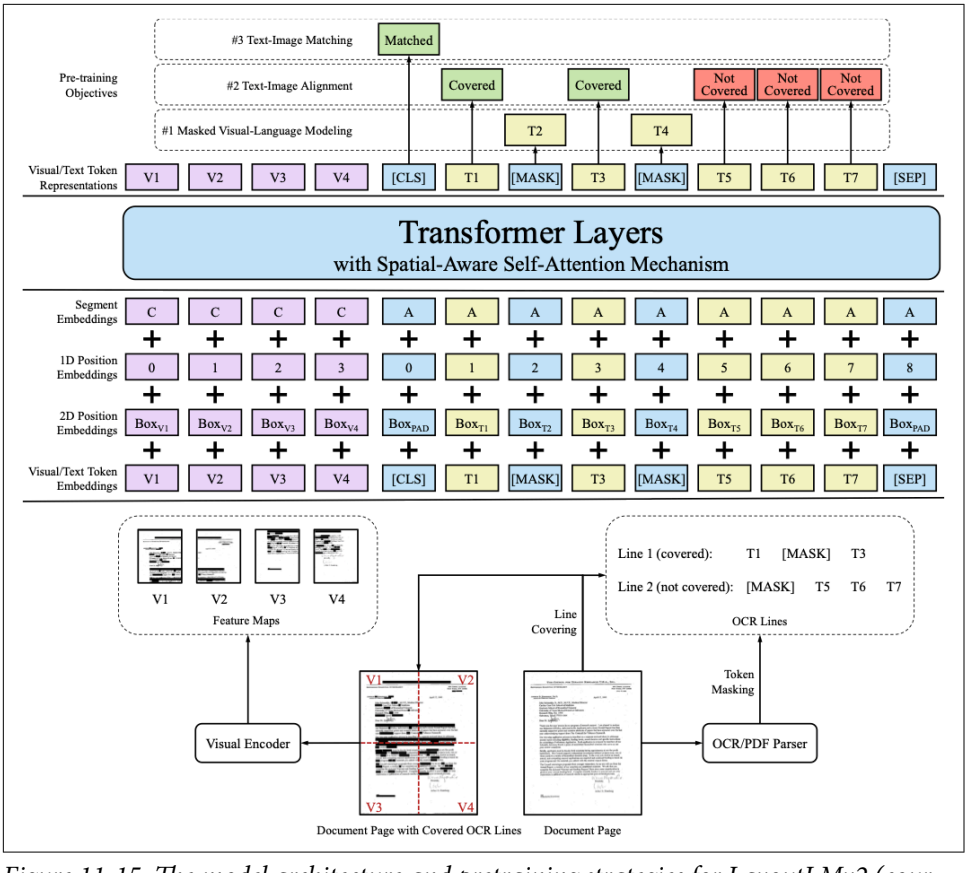


Figure 11-15. The model architecture and pretraining strategies for LayoutLMv2 (courtesy of Yang Xu)

DALL·E

A model that combines vision and text for *generative* tasks is DALL·E.[18] It uses the GPT architecture and autoregressive modeling to generate images from text. Inspired by iGPT, it regards the words and pixels as one sequence of tokens and is thus able to continue generating an image from a text prompt, as shown in Figure 11-16.

18 A. Ramesh et al., "Zero-Shot Text-to-Image Generation", (2021).

Figure 11-16. Generation examples with DALL·E (courtesy of Aditya Ramesh)

CLIP

Finally, let's have a look at CLIP,[19] which also combines text and vision but is designed for supervised tasks. Its creators constructed a dataset with 400 million image/caption pairs and used contrastive learning to pretrain the model. The CLIP architecture consists of a text and an image encoder (both transformers) that create embeddings of the captions and images. A batch of images with captions is sampled, and the contrastive objective is to maximize the similarity of the embeddings (as measured by the dot product) of the corresponding pair while minimizing the similarity of the rest, as illustrated in Figure 11-17.

In order to use the pretrained model for classification the possible classes are embedded with the text encoder, similar to how we used the zero-shot pipeline. Then the embeddings of all the classes are compared to the image embedding that we want to classify, and the class with the highest similarity is chosen.

19 A. Radford et al., "Learning Transferable Visual Models from Natural Language Supervision", (2021).

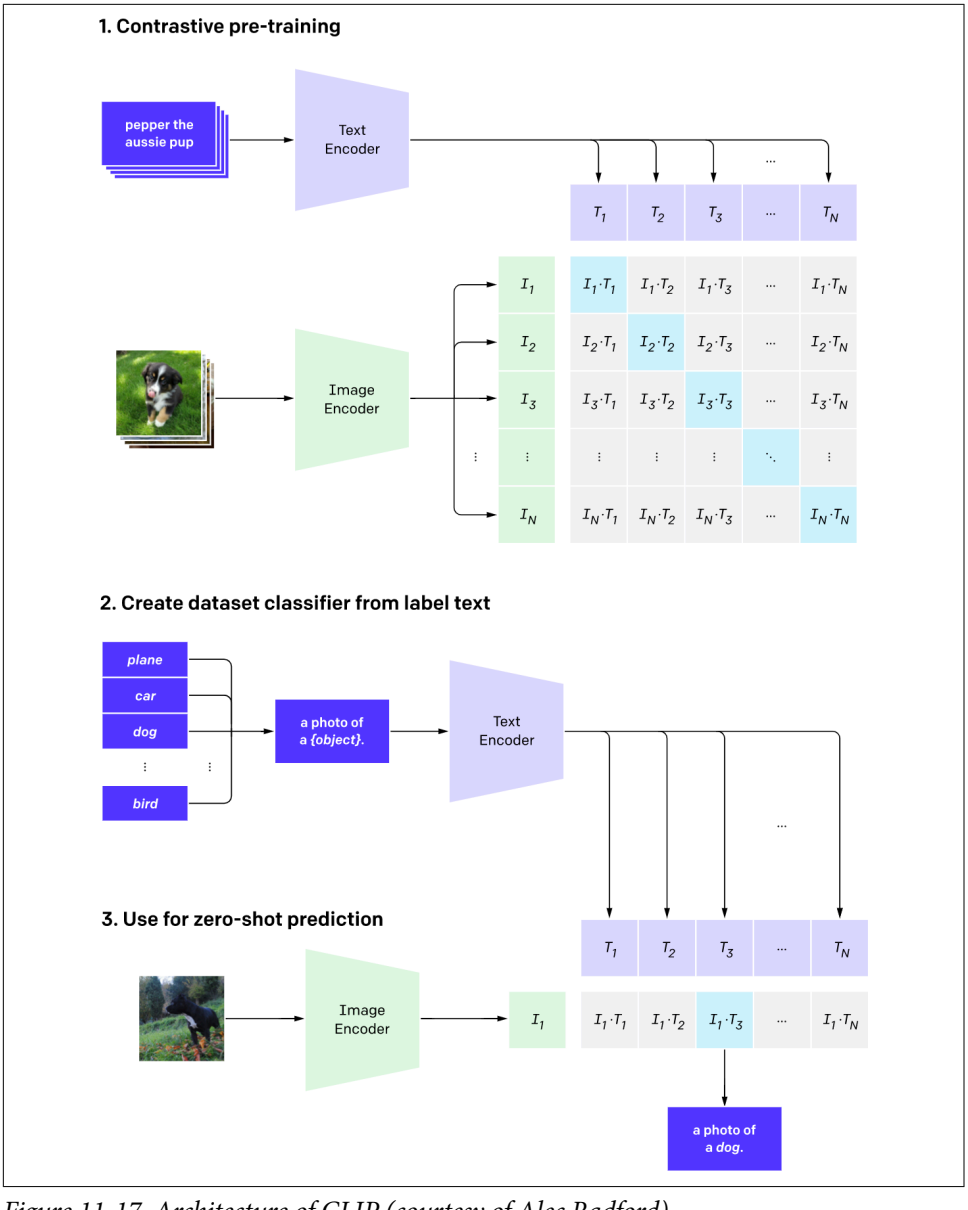


Figure 11-17. Architecture of CLIP (courtesy of Alec Radford)

The zero-shot image classification performance of CLIP is remarkable and competitive with fully supervised trained vision models, while being more flexible with regard to new classes. CLIP is also fully integrated in 🤗 Transformers, so we can try it out. For image-to-text tasks, we instantiate a *processor* that consists of a *feature extractor* and a tokenizer. The role of the feature extractor is to convert the image into a

form suitable for the model, while the tokenizer is responsible for decoding the model's predictions into text:

```
from transformers import CLIPProcessor, CLIPModel

clip_ckpt = "openai/clip-vit-base-patch32"
model = CLIPModel.from_pretrained(clip_ckpt)
processor = CLIPProcessor.from_pretrained(clip_ckpt)
```

Then we need a fitting image to try it out. What would be better suited than a picture of Optimus Prime?

```
image = Image.open("images/optimusprime.jpg")
plt.imshow(image)
plt.axis("off")
plt.show()
```

Next, we set up the texts to compare the image against and pass it through the model:

```
import torch

texts = ["a photo of a transformer", "a photo of a robot", "a photo of agi"]
inputs = processor(text=texts, images=image, return_tensors="pt", padding=True)
with torch.no_grad():
    outputs = model(**inputs)
logits_per_image = outputs.logits_per_image
probs = logits_per_image.softmax(dim=1)
probs
```

```
tensor([[0.9557, 0.0413, 0.0031]])
```

Well, it almost got the right answer (a photo of AGI of course). Jokes aside, CLIP makes image classification very flexible by allowing us to define classes through text instead of having the classes hardcoded in the model architecture. This concludes our tour of multimodal transformer models, but we hope we've whetted your appetite.

Where to from Here?

Well that's the end of the ride; thanks for joining us on this journey through the transformers landscape! Throughout this book we've explored how transformers can address a wide range of tasks and achieve state-of-the-art results. In this chapter we've seen how the current generation of models are being pushed to their limits with scaling and how they are also branching out into new domains and modalities.

If you want to reinforce the concepts and skills that you've learned in this book, here are a few ideas for where to go from here:

Join a Hugging Face community event
: Hugging Face hosts short sprints focused on improving the libraries in the ecosystem, and these events are a great way to meet the community and get a taste for open source software development. So far there have been sprints on adding 600+ datasets to 🤗 Datasets, fine-tuning 300+ ASR models in various languages, and implementing hundreds of projects in JAX/Flax.

Build your own project
: One very effective way to test your knowledge in machine learning is to build a project to solve a problem that interests you. You could reimplement a transformer paper, or apply transformers to a novel domain.

Contribute a model to 🤗 Transformers
: If you're looking for something more advanced, then contributing a newly published architecture to 🤗 Transformers is a great way to dive into the nuts and bolts of the library. There is a detailed guide to help you get started in the 🤗 Transformers documentation.

Blog about what you've learned
: Teaching others what you've learned is a powerful test of your own knowledge, and in a sense this was one of the driving motivations behind us writing this book! There are great tools to help you get started with technical blogging; we recommend *fastpages* as you can easily use Jupyter notebooks for everything.