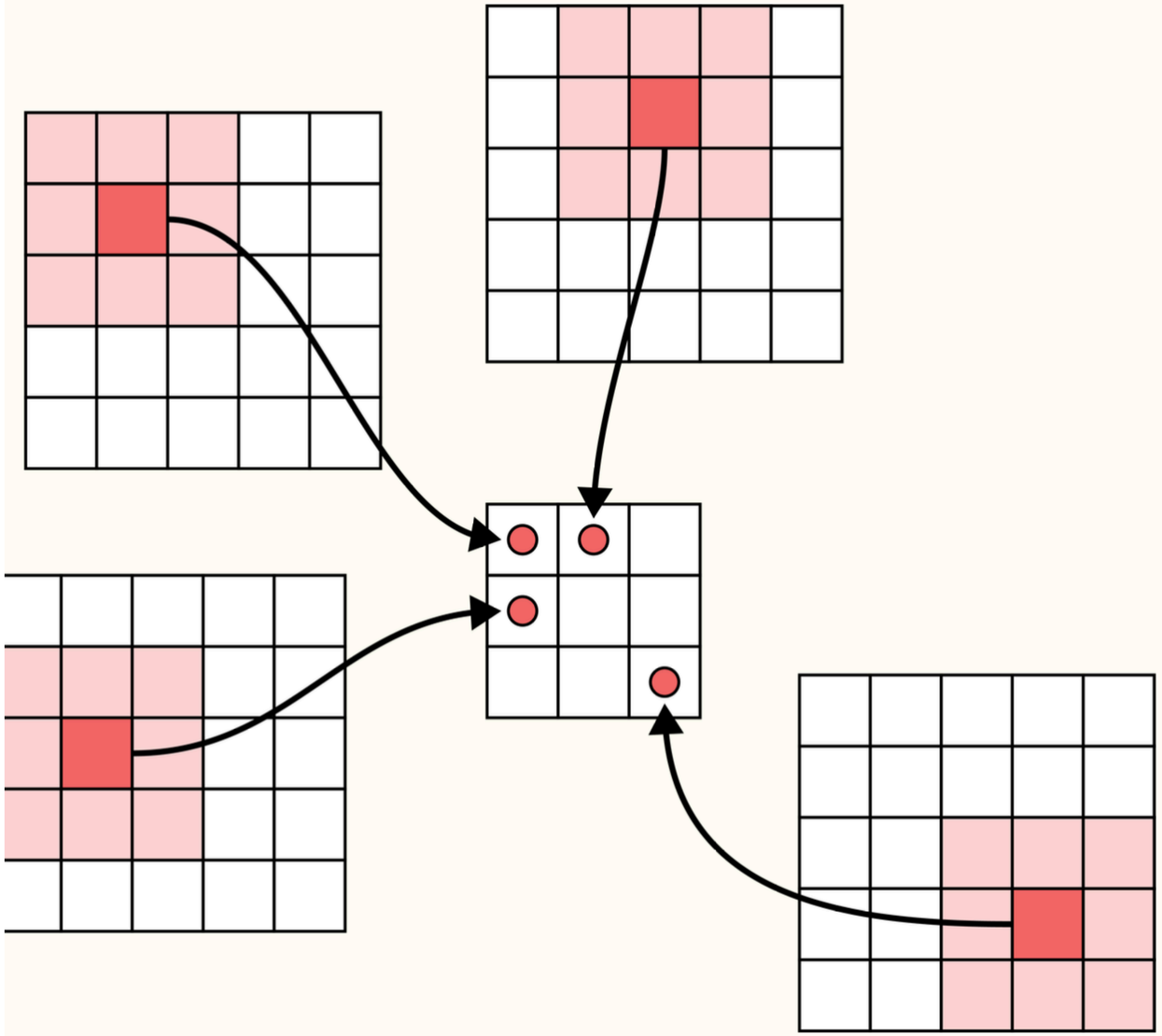
Images, Text, and Fine Tuning

The secret of why AI works for ordinary companies

CM3-Computer Science

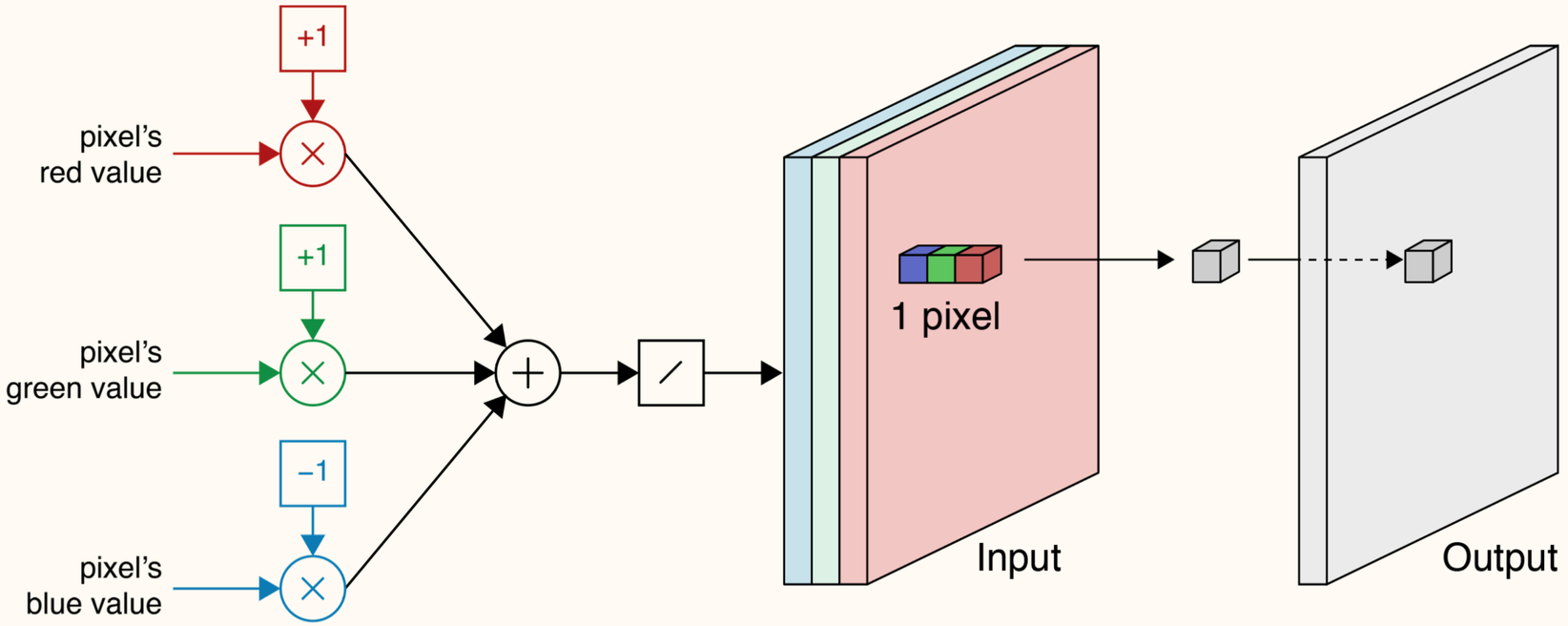
Convolutional Neural Networks (CNNs)



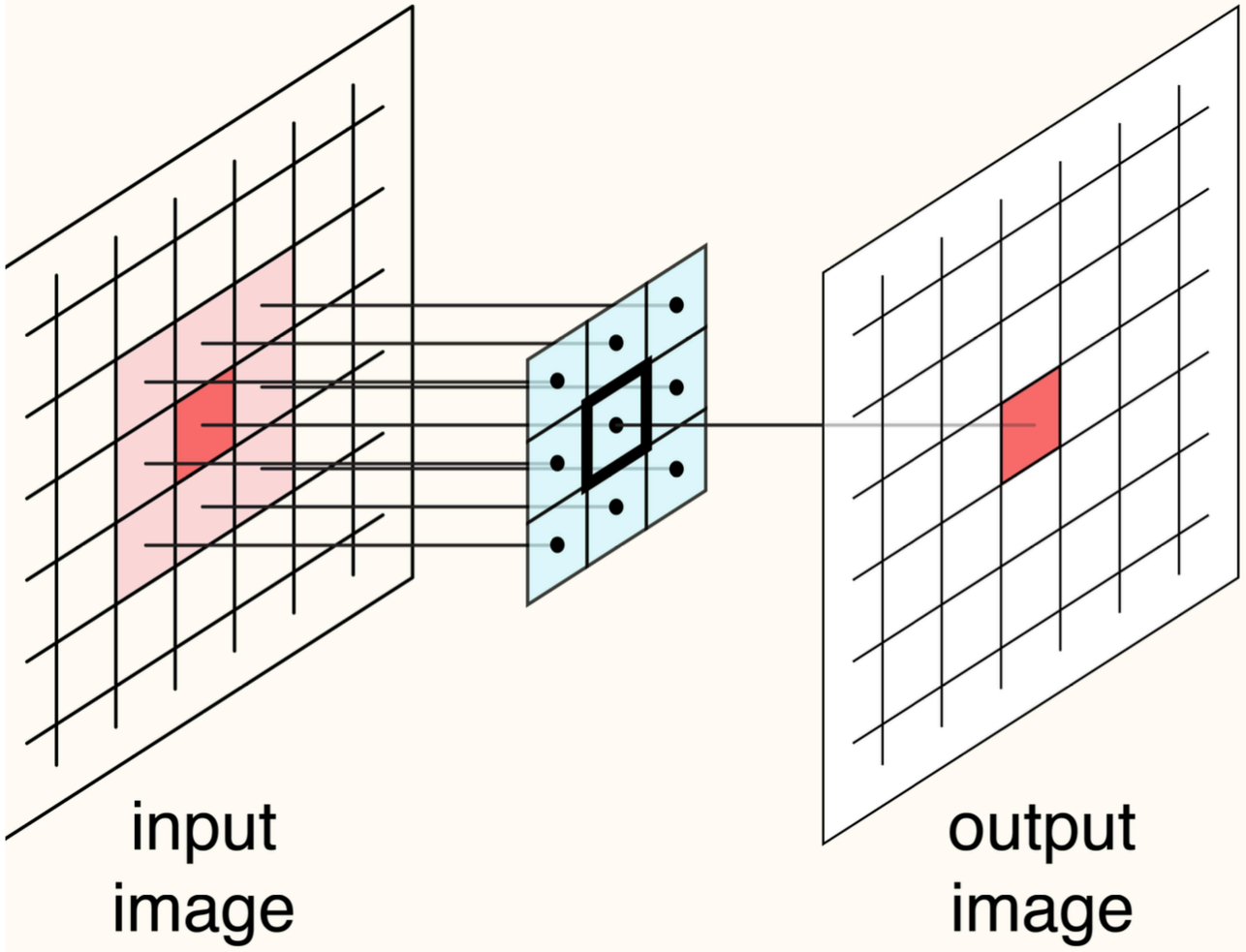
What is a convolution

- slide 3x3 filter across image of size 5x5
- filters must fit into image, thus output will be of smaller size, in this case 3x3
- multiply values in filter by values of pixel and add a bias: $(w, b) \cdot (x, 1)$

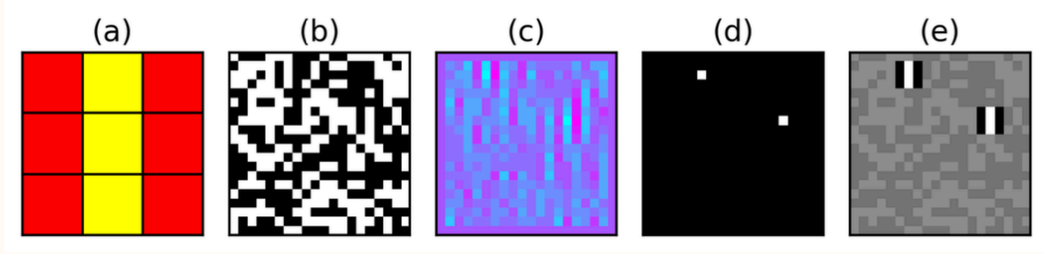
The idea of a filter: detecting yellow



Convolution looks for patterns



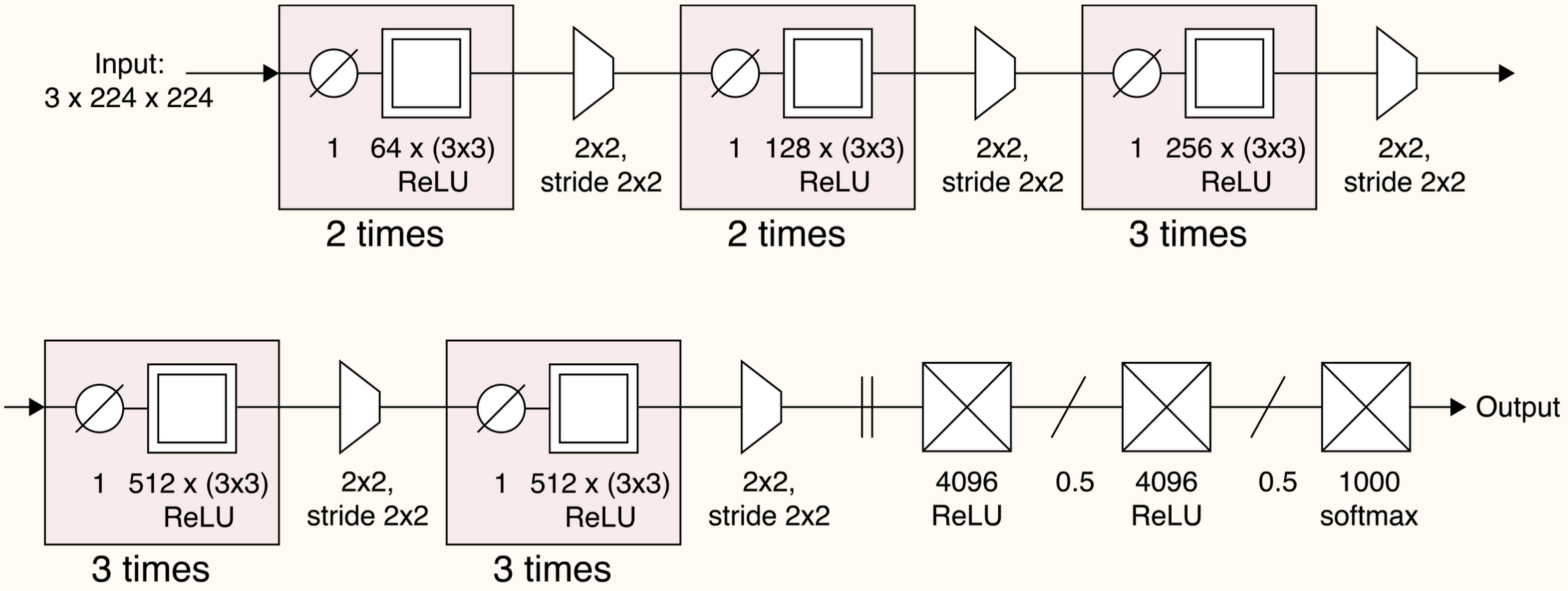
Movethe filter over the original image and produce a new one



$$\begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ 0 & 0 & 0 \\ -1 & 0 & -1 \end{bmatrix} \Rightarrow -3 + 1 = -2$$

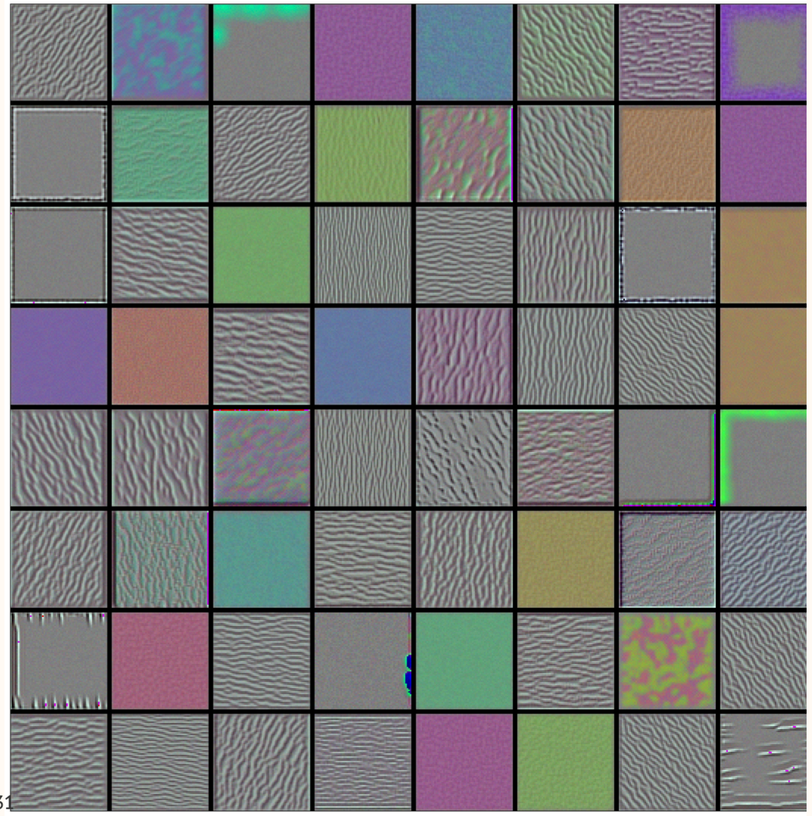
$$\begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \Rightarrow 3$$

VGG16

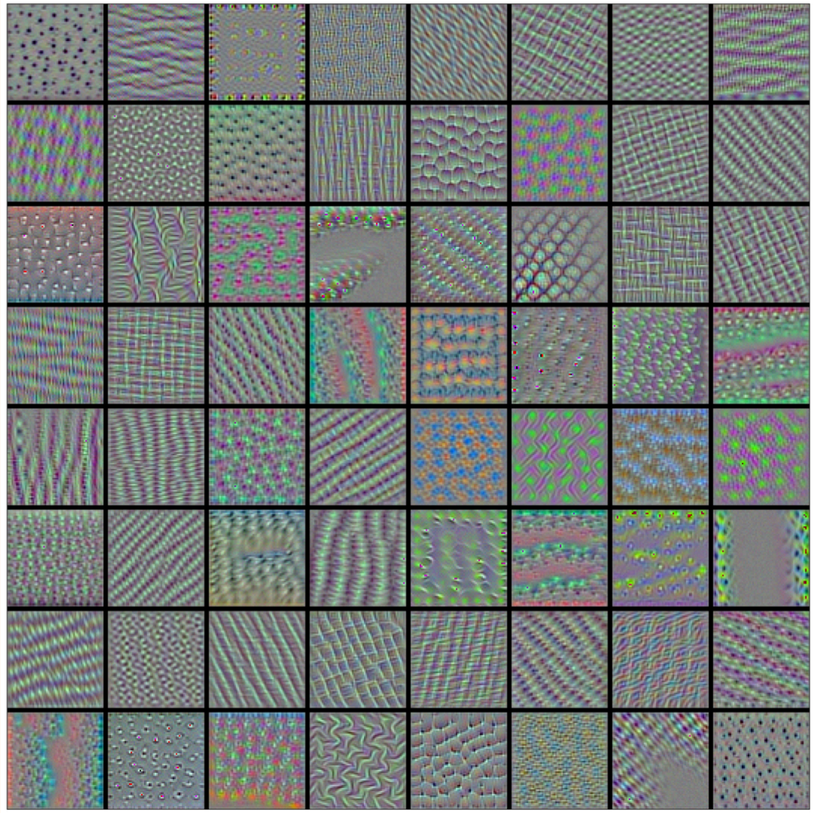

Input:
3 x 224 x 224
1 64 x (3x3)
ReLU
2 times
2x2,
stride 2x2
1 128 x (3x3)
ReLU
2 times
2x2,
stride 2x2
1 256 x (3x3)
ReLU
3 times
2x2,
stride 2x2
1 512 x (3x3)
ReLU
3 times
2x2,
stride 2x2
1 512 x (3x3)
ReLU
3 times
2x2,
stride 2x2
4096
ReLU
0.5
4096
ReLU
0.5
1000
softmax
Output

The closer you get to the output in a CNN, the more complex are the structures learned

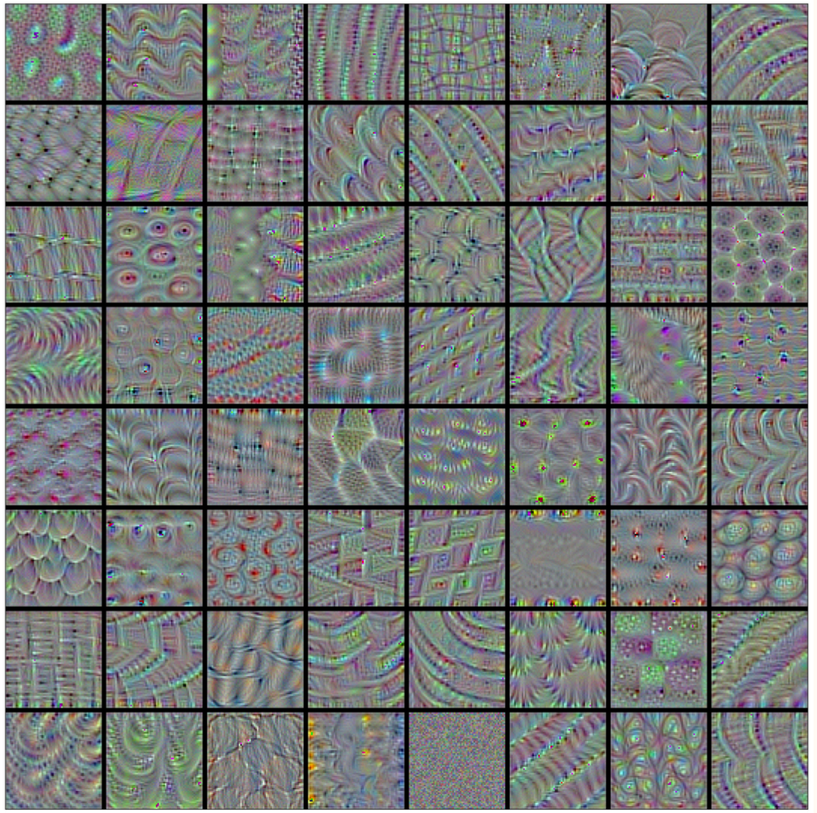
Filters from VGG16 layer block1_conv2

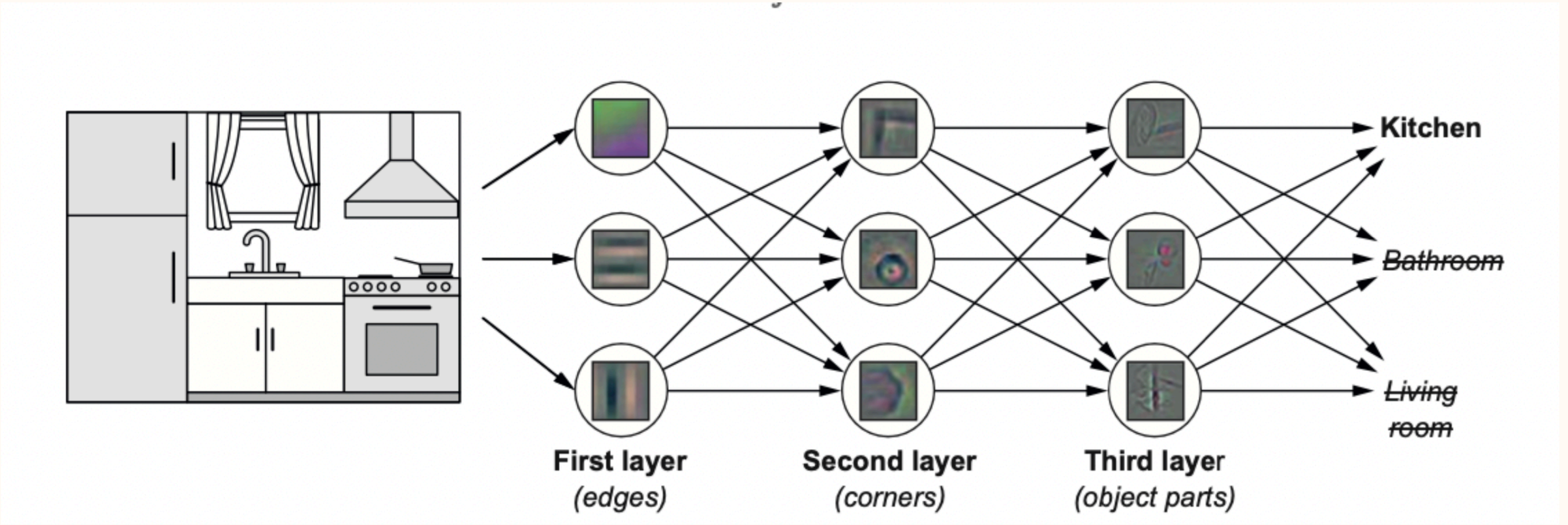


Filters from VGG16 layer block3_conv1



Filters from VGG16 layer block4_conv1

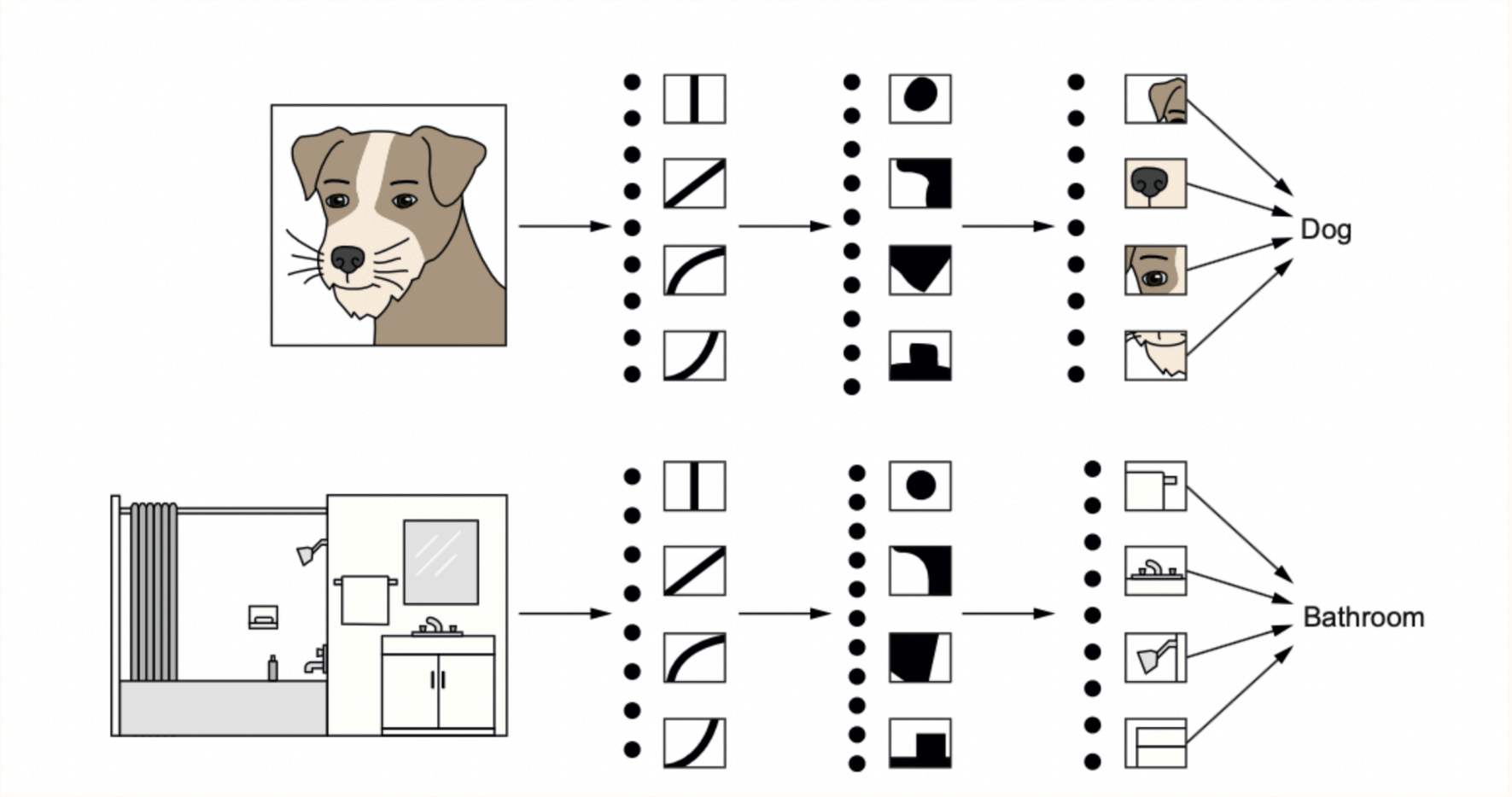




The process of transformation of pixel data from an image by a CNN. The warping of the data is identical at different parts of the image because of the convolutional filters. At each layer, multiple new images are learned. These correspond to combinations of pixels into lines, edges and corners. Further layers combine these shapes into richer ones, such as circles. Image from Valigi and Mauro (2020).

Fine Tuning or Transfer Learning

Using the representation hierarchy: transfer learning

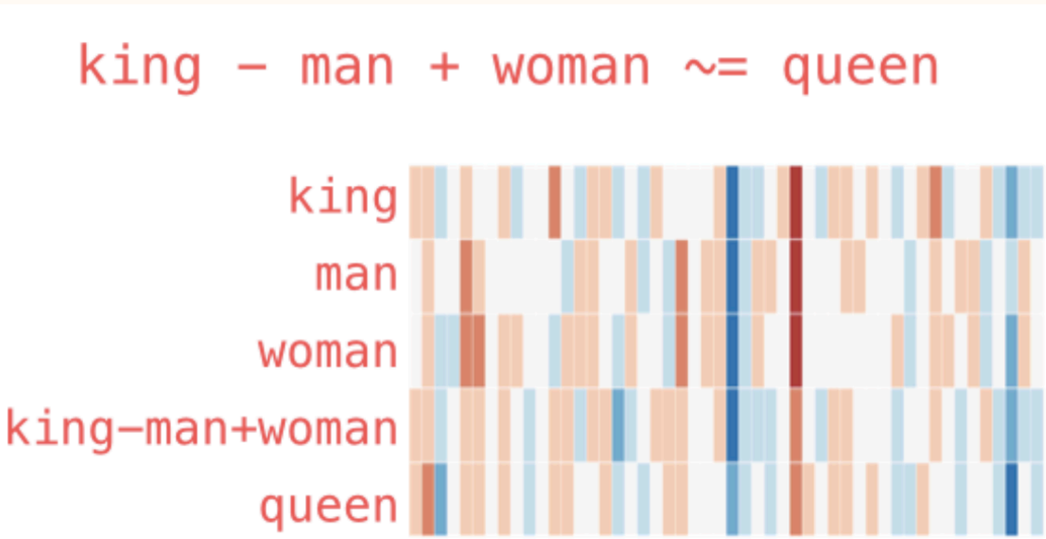


Basic Idea of Transfer Learning

Because the early later structures that make up a dog are the same as the ones that make up furniture, we can use a network trained to distinguish dogs from cats to classify rooms. To do this we take the former model and retrain the later layers of the neural network to combine shapes into furniture rather than animals. The benefit of this technique, called transfer learning, is that because the network has seen so many images of animals, the work of the earlier layers is done. It just needs to see some examples of furniture to make a furniture classifier.

Natural Language Processing

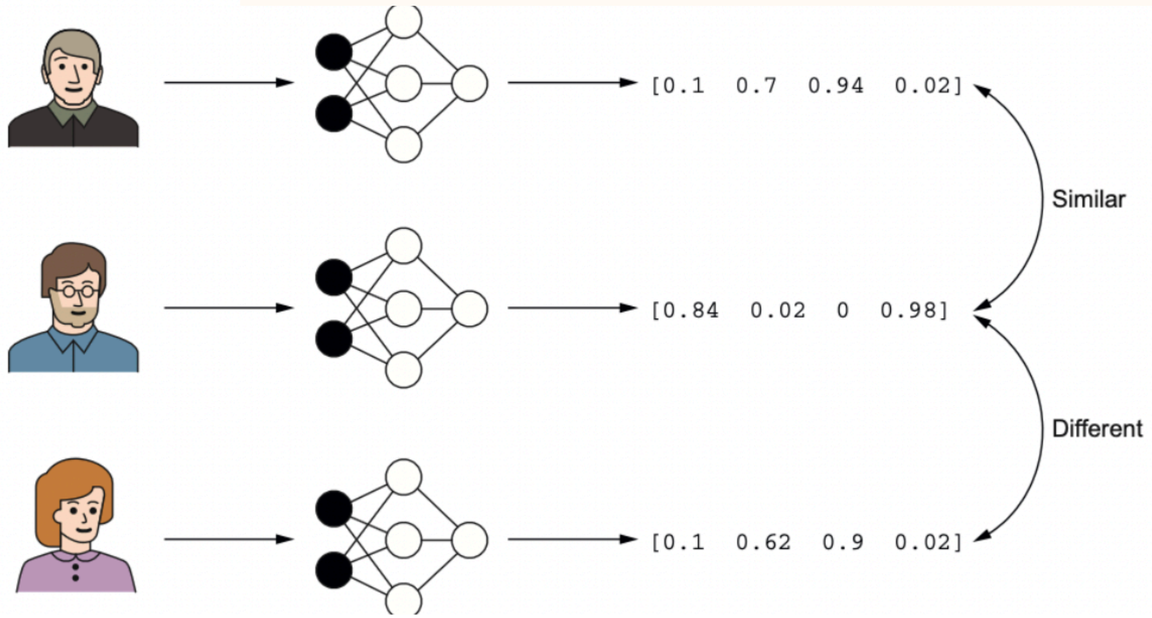
Language networks use similar convolution techniques but also add in notions of time..the current word depends on a past one..and of attention: the idea that a particular phrase in one sentence is “attentive” to a phrase/ concept in a past or future one, thus enabling us to hold a conceptual thread through a paragraph or longer.



Embeddings map similar concepts to similar vectors

These concepts can be from different spaces, like users and movies

Any model can be used to produce embeddings: Apple face recognition.



Pre-trained models and Embeddings

You can imagine that word and sentence embeddings will be useful in document search, both in the more general case, and in specific domains as well. So you can take some of these larger models like BERT or GPT-3 trained by Google and OpenAI respectively and transfer from them to your domain.

But, even if you dont transfer-them, these embeddings can be very useful.

For example, you can use them for prompting in 0-shot learning

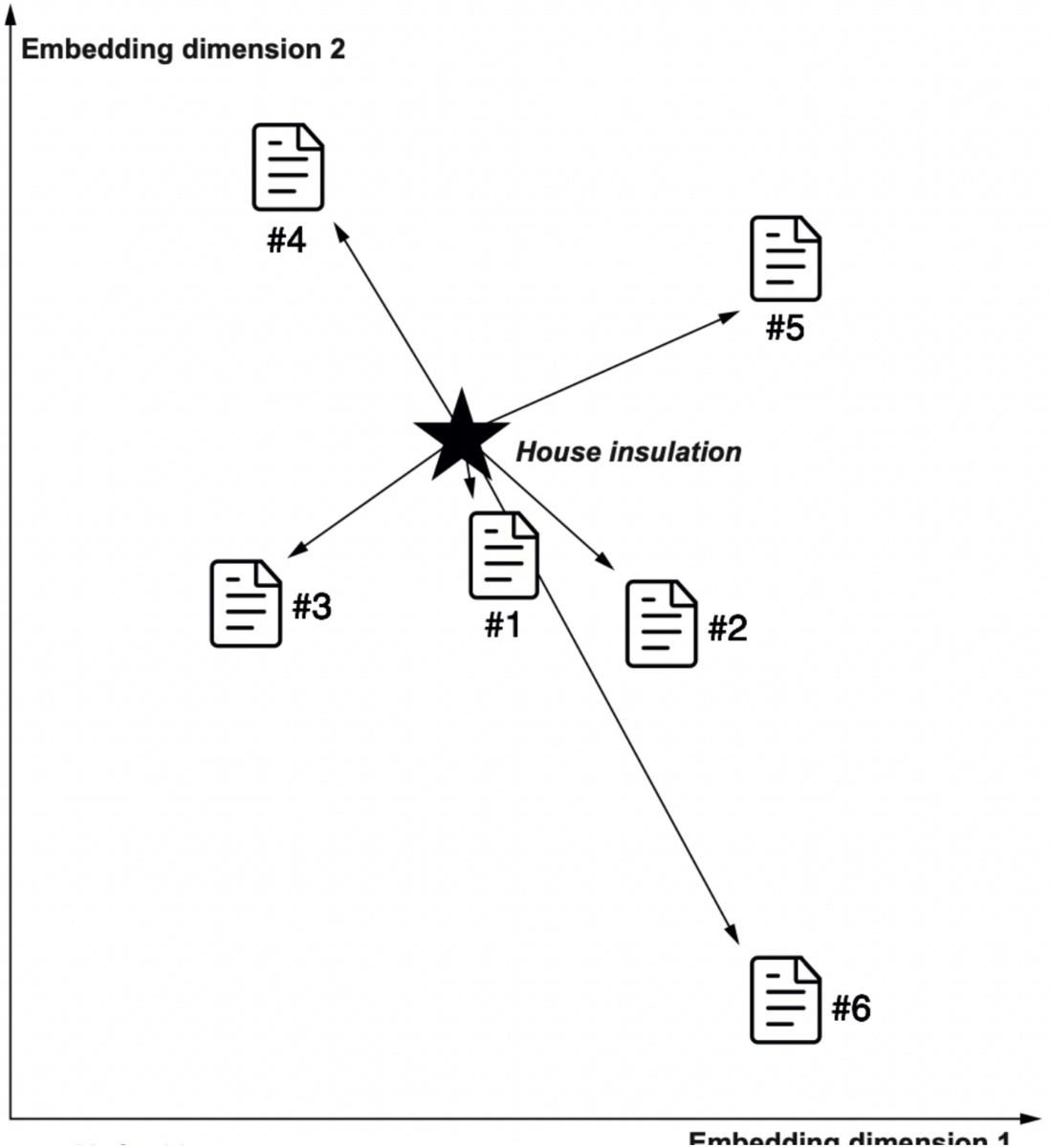
There are different kind of promptings: 0-shot, multi-shot, chain-of-thought, etc.
They are all created to mimic the structure of text.

Prompting for 0 shot-learning

```
movie_desc = "The main characters of the movie madacascar \
are a lion, a zebra, a giraffe, and a hippo. "
prompt = "The movie is about [MASK]."

output = pipe(movie_desc + prompt)
for element in output:
    print(f"Token {element['token_str']}:\t{element['score']:.3f}%")

Token animals:  0.103%
Token lions:    0.066%
Token birds:    0.025%
Token love: 0.015%
Token hunting:  0.013%
```



Semantic Search Engines with Embeddings

We can use domain specific embeddings to create search engines for domain specific documents. The input query is converted to an embedding. We then find the most similar embeddings in a corpus. Voila, you have a domain specific intranet search engine. Image from Valigi and Mauro (2020).

So if you are Home Depot and a user asks about “techniques for house insulation” we can transform these words into their embeddings, and if we’ve done the same with all the documents we have, we can return to the user the documents whose numerical representation or embedding is closest to the one we got for their query.

Stuffing the “context” from the retrieval from embedding searches into a prompt is called RAG, or Retrieval Augmented Generation

```
query =  np.array(embs_valid[i]["embedding"], dtype=np.float32)
scores, samples = embs_train.get_nearest_examples("embedding", query, k=k)


QUERY LABELS: ['new model']
QUERY TEXT:
Implementing efficient self attention in T5
Retrieved documents:
TEXT:
Add Linformer model
SCORE: 54.92
LABELS: ['new model']
==================================================
TEXT:
Add FAVOR+ / Performer attention
SCORE: 57.90
LABELS: ['new model']
==================================================
TEXT:
Implement DeLighT: Very Deep and Light-weight Transformers
SCORE: 60.12
LABELS: ['new model']
```

Gettting the text with the closest embeddings.

The idea behind fine tuning in language.

Imagine you are looking at legal documents, or contracts. The number of contracts you have access to are likely to be smaller than the size of reddit or wikipedia. But my training your initial model on the larger Reddit and wikipedia, and then transfering to your smaller contract data, you can particularize domain specific embeddings from the more general ones. The models trained on the bigger data pick up basic vocabulary and grammar, which are useful for pretty much everything and now you can make them specific to your domain: insurance claims or EMR notes or contracts.

Fine tuning a model to labeled data

If we have access to labeled data, we can also try to do the obvious thing: simply fine- tune a pretrained transformer model.

```
model_ckpt = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
model = AutoModelForSequenceClassification.from_pretrained(model_ckpt,
                                                           config=config)
trainer = Trainer(
    model=model, tokenizer=tokenizer,
    args=training_args_fine_tune,
    compute_metrics=compute_metrics,
    train_dataset=ds_enc["train"],
    eval_dataset=ds_enc["valid"],)

trainer.train()
pred = trainer.predict(ds_enc["test"])
metrics = compute_metrics(pred)
```

Domain Adaptation: fine tune to unlabeled data

We can leverage the pretrained weights for other tasks on a wide variety of texts. This is the core idea of transfer learning in NLP.

In this step we use the classic language model objective of predicting masked words, which means we don't need any labeled data. After that we can load the adapted model as a classifier and fine-tune it, thus leveraging the unlabeled data. The beauty of domain adaptation is that compared to labeled data, unlabeled data is often abundantly available.

Furthermore, the adapted model can be reused for many use cases. Imagine you want to build an email classifier and apply domain adaptation on all your historic emails. You can later use the same model for named entity recognition or another classification task like sentiment analysis, since the approach is agnostic to the downstream task.

Natural Language Processing is a different kind of data: you can always try and predict the next data

More fine tuning

- You can always combing fine-tuning on unlabeled data and put some effort into labeling a smaller fraction of the data. The former will capture the language of your domain, while the latter will bring the labels in. This strategy lets you get away with fewer labels

- These days (since about 2021) people instruction-tune their models. This is a different kind of fine tuning, you take the text, the label, AND an instruction, and train the model to learn these triplets. This then generalizes across tasks such as sentiment analysis, summarization, question-answering, etc. This is how most modern models like ChatGPT are fine-tuned

After Fine Tuning?

- A problem remains. Chatbots can give answwers to questions that are sexist/racist/irrelevant/etc. This is the “hallucination” or alignment problem
- As a company you dont want your users to get wrong answers.
- So you engage humans to choose between a few-possible answers, being specific on what you want your humans to do: no sexism/no racism/etc. This is called RLHF fine-tuning
- Recently (late 2023) a new technique has come out called direct preference optimization (DPO) which has users chose between 2 choices asking which one is better. It scales to more user labeling but is not as good for alignment as RLHF.