Deep neural networks

-Attention, transformers, and Homework C

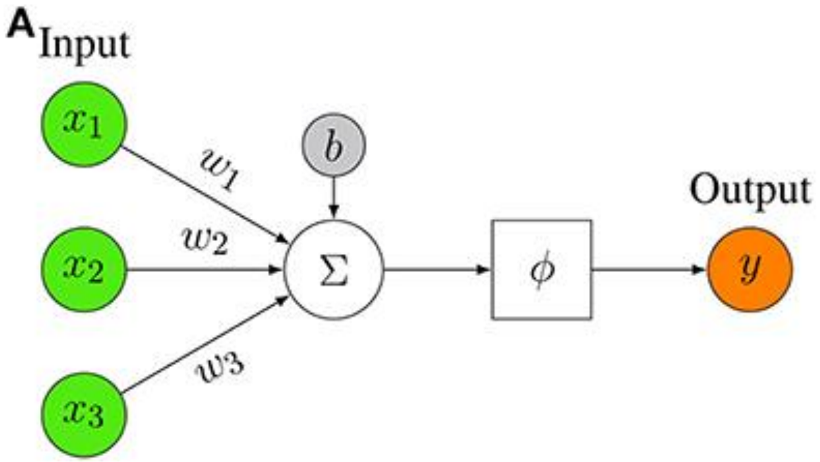
Daniel Midtvedt

Purpose

- To provide intuition about some common deep learning techniques:
  - Convolutional neural networks
  - Encoder-decoder structures
  - Attention and transformers
- Specifically, these are the tools needed for Homework C

What is a neural network

One neuron

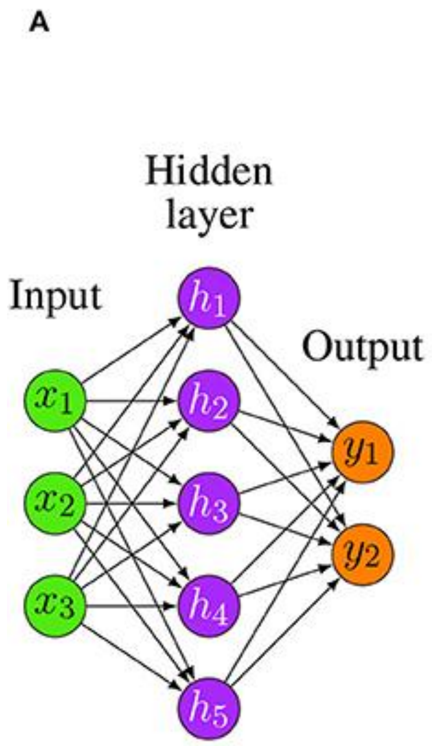


$$y = \phi(z) = \phi(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

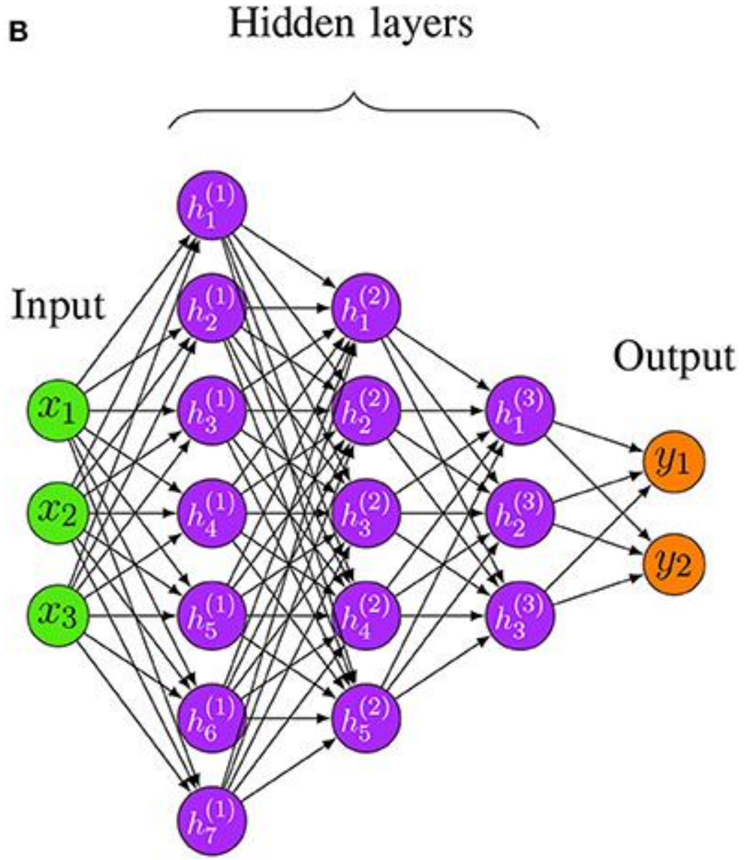
$\phi(z)$: Activation function

What is a neural network

Shallow network



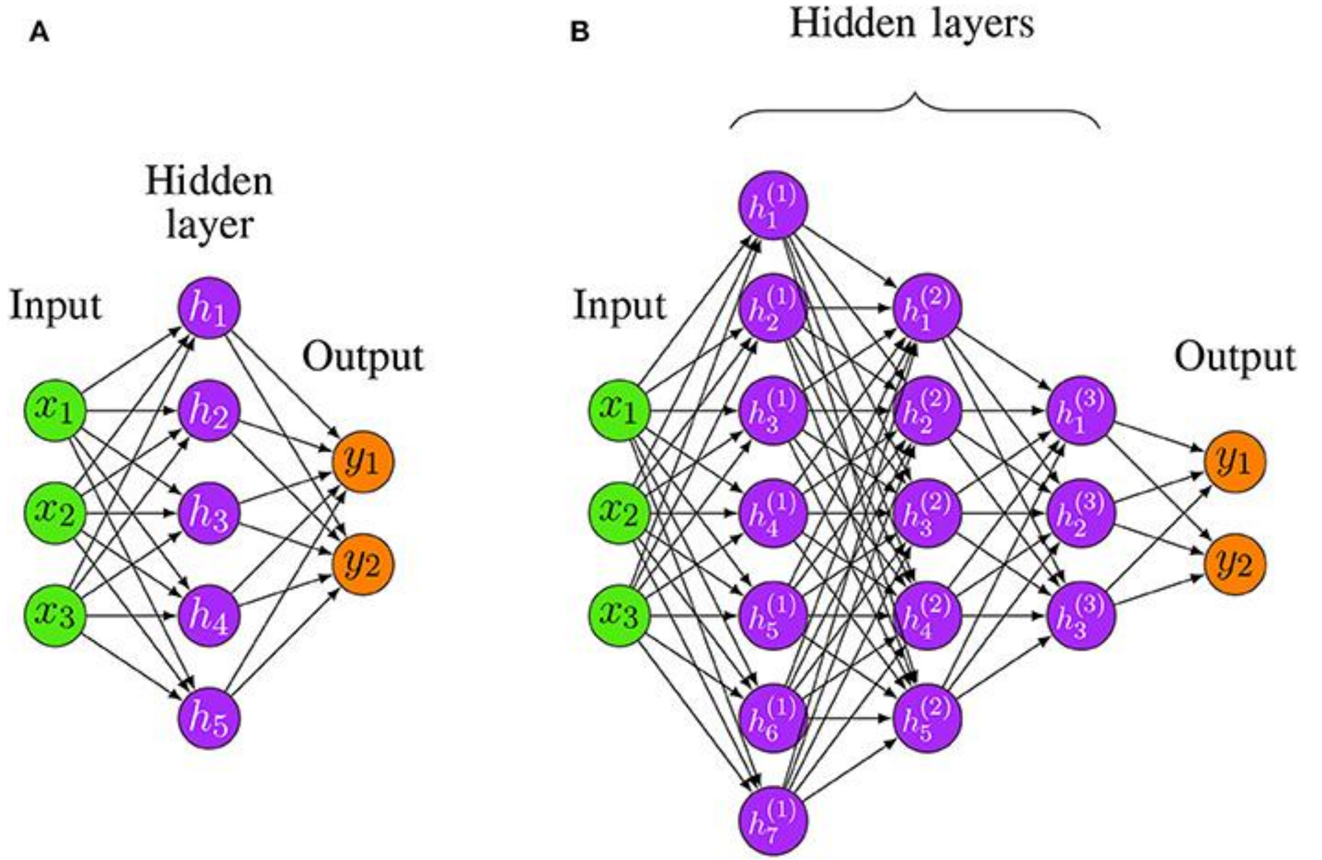
Deep network



Each arrow represents one weight (i.e. one parameter) that needs to be determined

Why deep networks?

Shallow network | Deep network



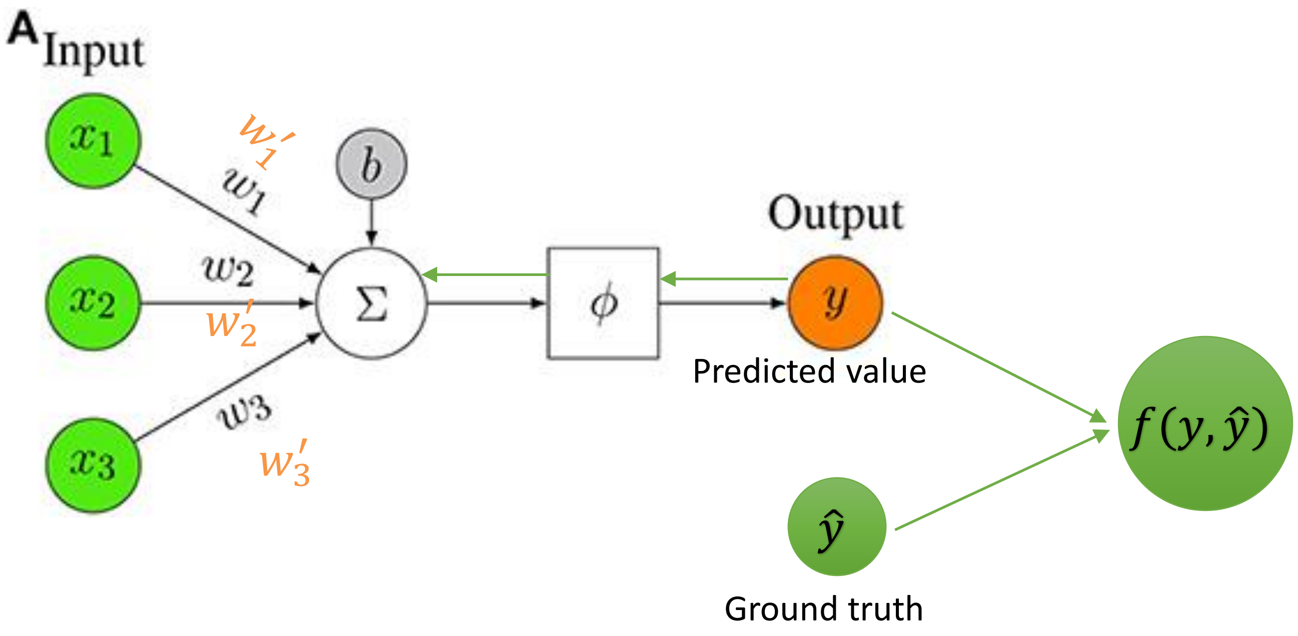
Each arrow represents one weight (i.e. one parameter) that needs to be determined

Think:

- Why deep networks instead of shallow?
- Can networks be too deep?

How are the weights optimized?

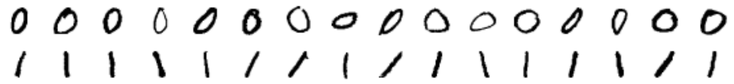
- The model is trained on data with known, ground truth, output
  - The model is supplemented with a loss function
  - Depends on difference between predicted values and ground truth
  - Weights are optimized using back propagation to minimize this loss function

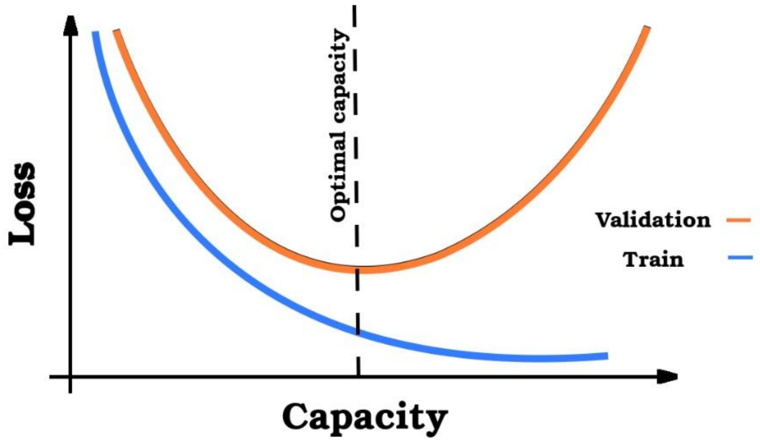


https://www.frontiersin.org/articles/10.3389/frai.2020.00004/full

How does it generalize?

- The network is optimized for predicting data in the training set
  - Does not necessarily predict outside the training set (overfitting)
  - Overcome by use of validation data

Training  Validation



https://mljar.com/blog/validation-learning-not-memorizing/

Fundamental problem

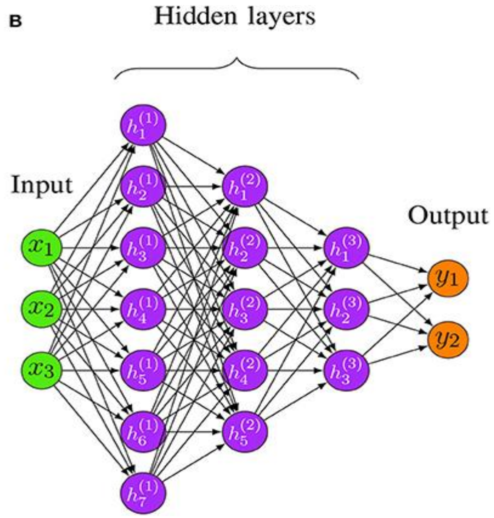
- Machine learning algorithms are often benchmarked against human capacity
- Humans are generating the training data!
- How to extend neural networks *beyond* human capacity?

Think:

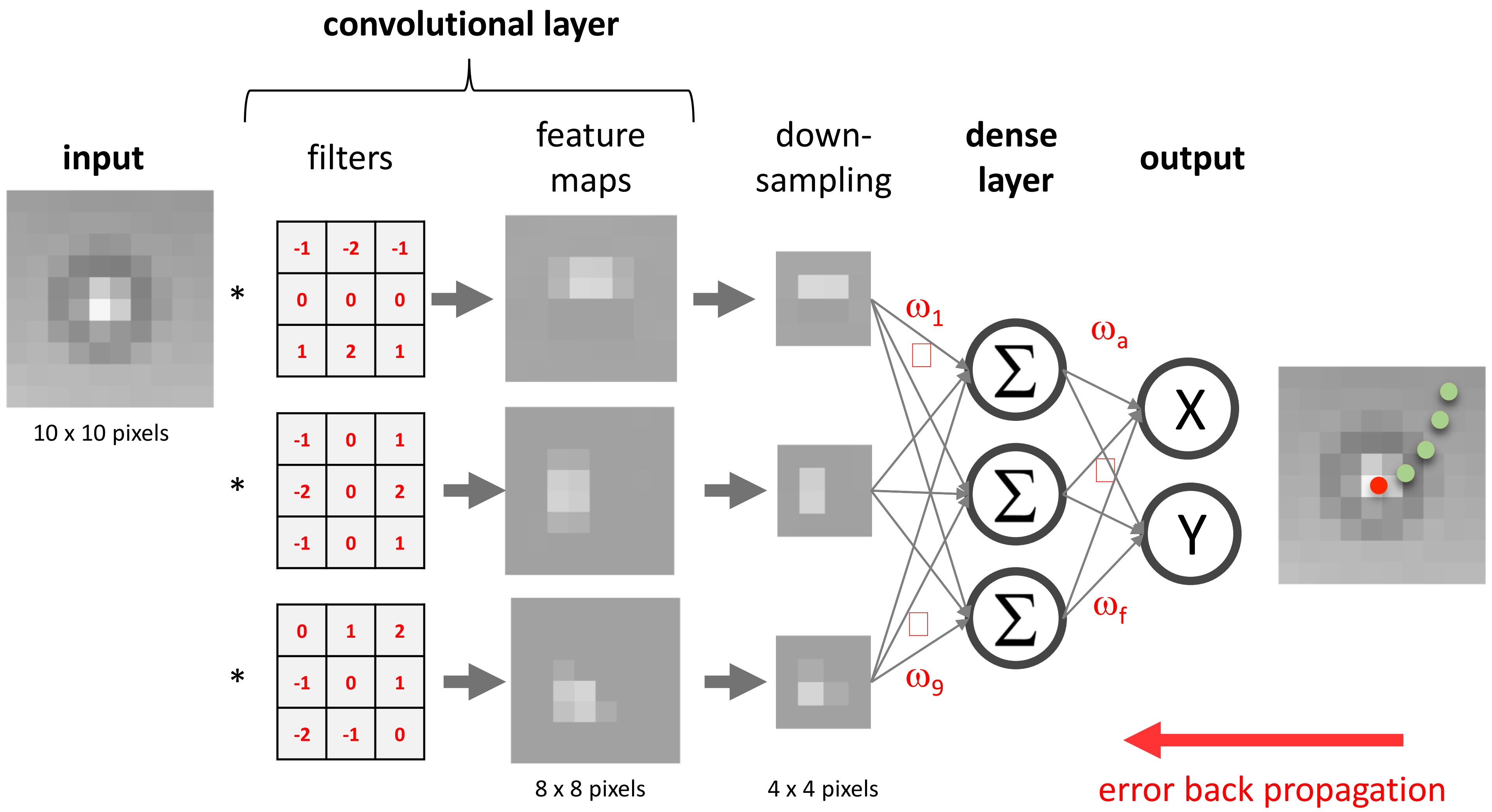
- Can neural network performance be enhanced beyond human capability?
- If so, how can their results be validated?

Problems with densely connected networks

- Scales poorly with input size
  - Each pixel in an image corresponds to one input
- Not adapted to learn sequences of data

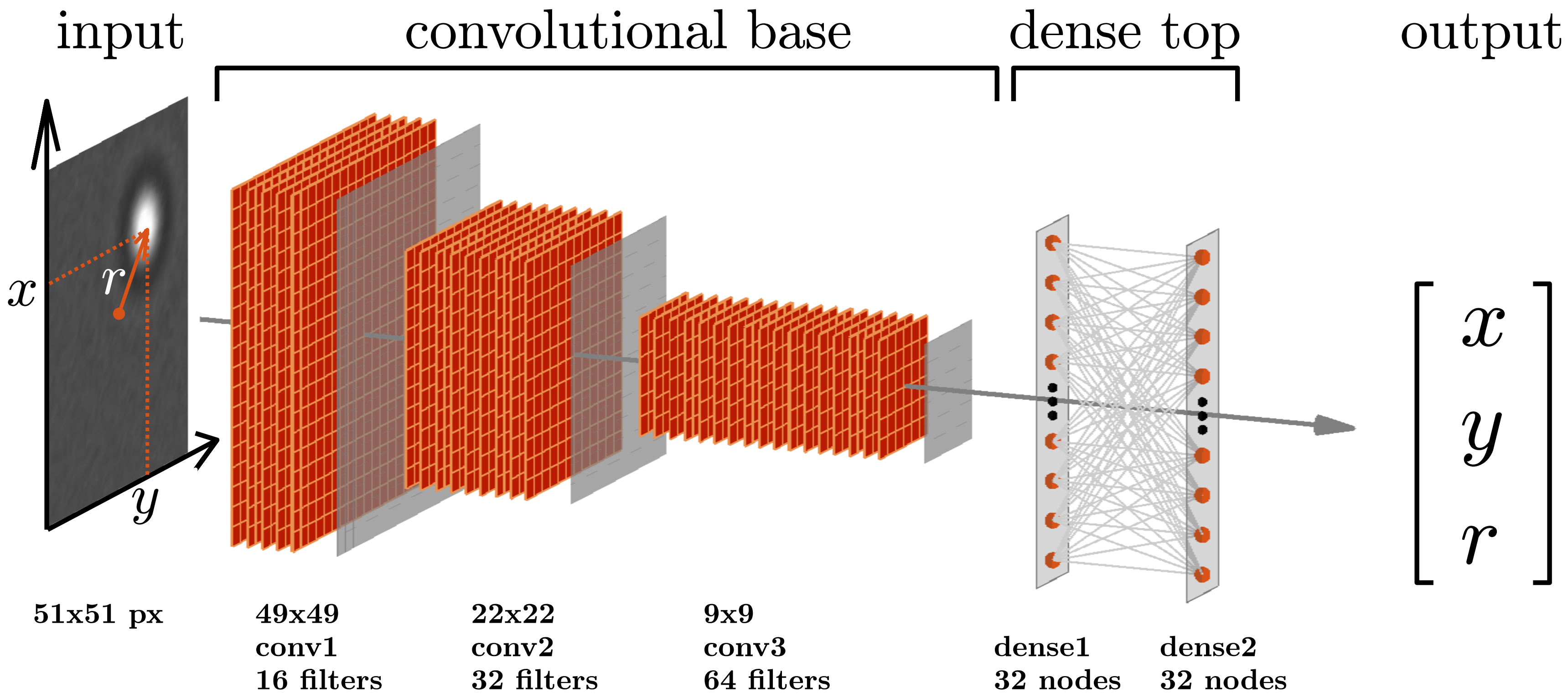


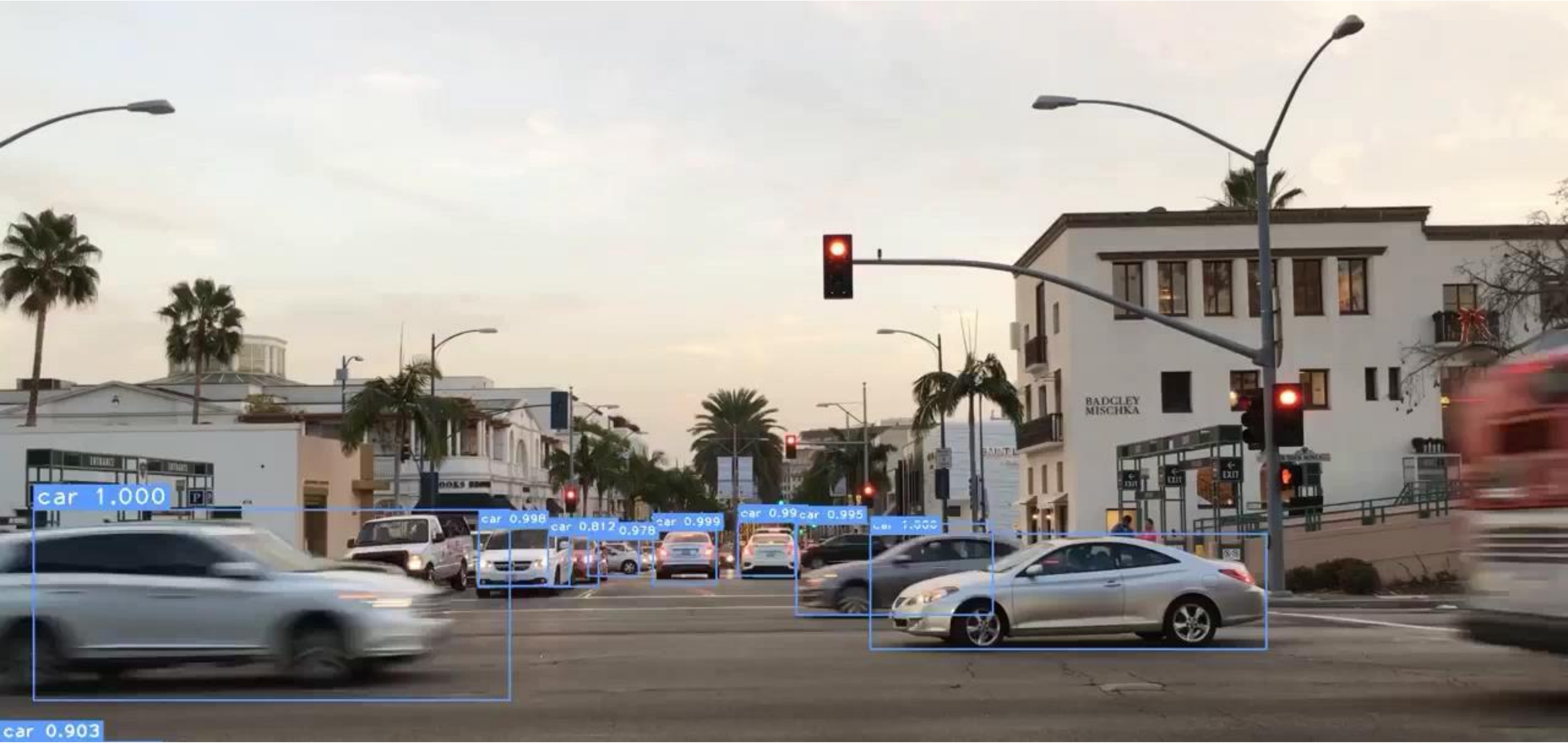
Convolutional neural networks solves the problem of dimensionality



Helgadottir, Argun & Volpe. Optica 6, 506 (2019)

Convolutional neural networks solves the problem of dimensionality




car 1.000
car 0.998
car 0.812
0.978
car 0.999
car 0.99
car 0.995
car 0.903
BADGLEY
MISCHKA
EXIT

Coding example

- How to define a CNN-architecture?
- We need convolutional blocks: Keras: keras.layers.Conv2D
- We need downsampling layers: Keras: keras.layers.MaxPool2D
- We need dense layers: Keras: keras.layers.Dense

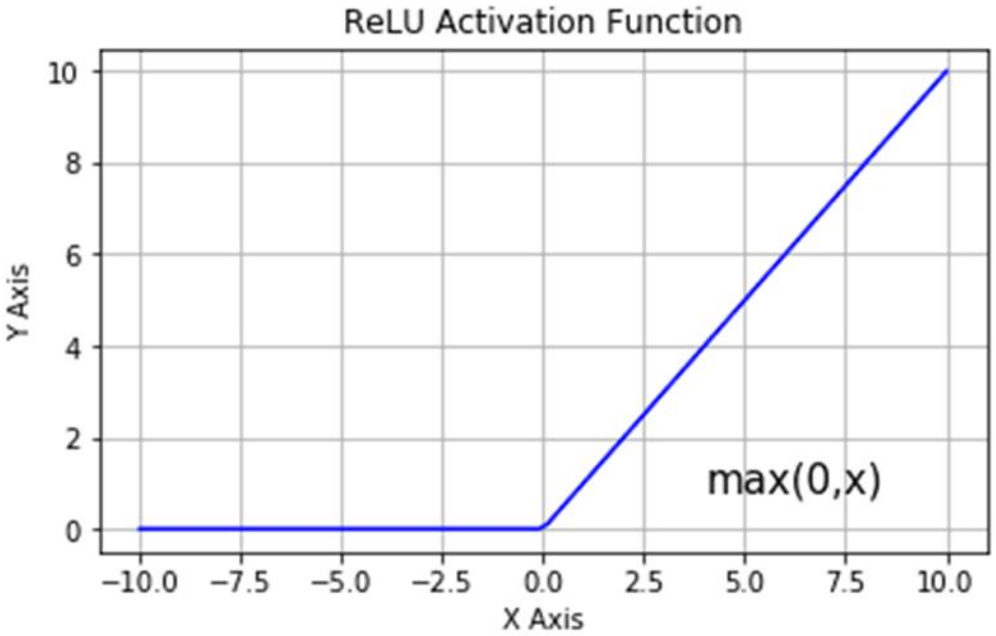
Putting it together:

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten
```

Putting it together:

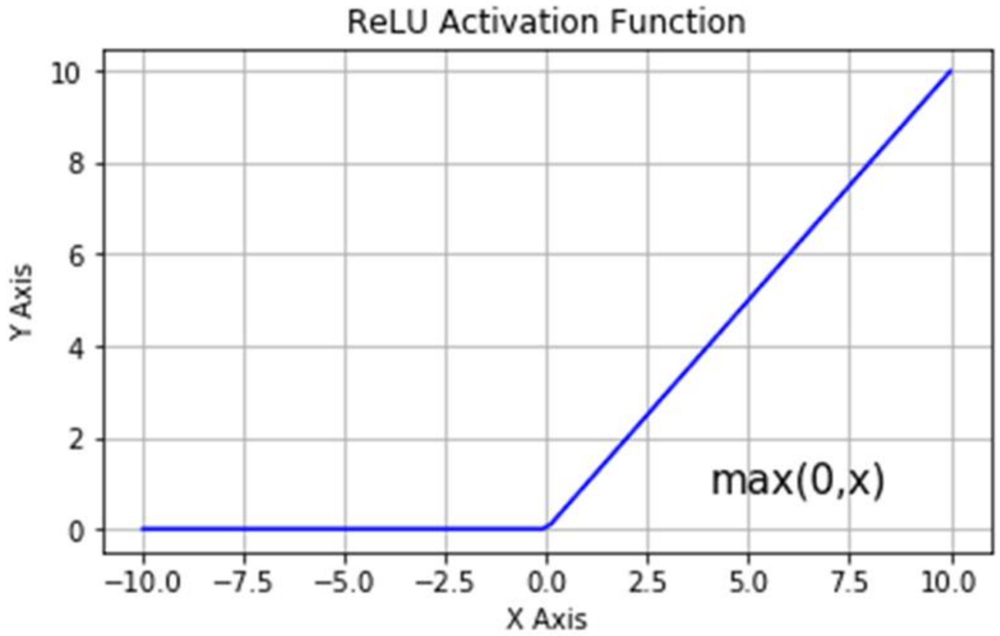
```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

model=keras.models.Sequential() #Defines the model
```

Putting it together:



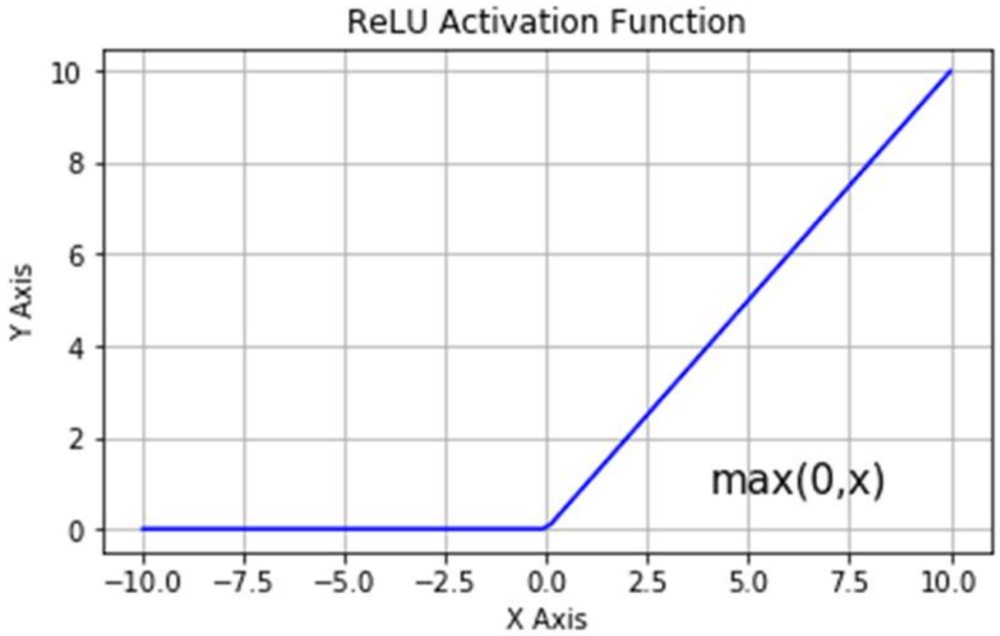

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

model=keras.models.Sequential() #Defines the model
model.add(Conv2D(8,(3,3),activation="relu",padding="same",input_shape=(64,64,1)))
```

Putting it together:



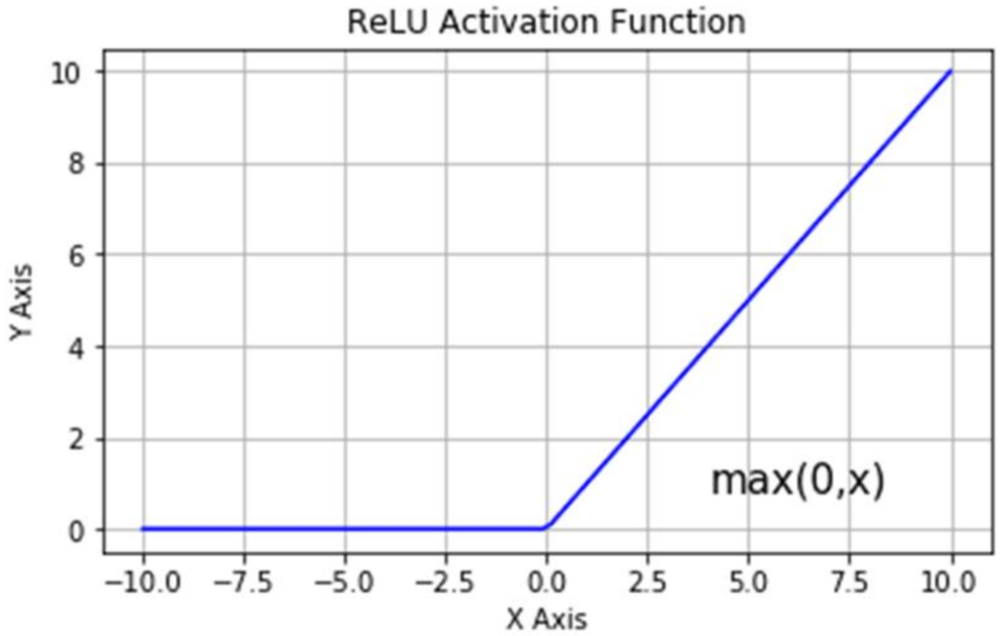

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

model=keras.models.Sequential() #Defines the model
model.add(Conv2D(8,(3,3),activation="relu",padding="same",input_shape=(64,64,1)))
model.add(MaxPool2D(pool_size=(2,2)))
```

Putting it together:



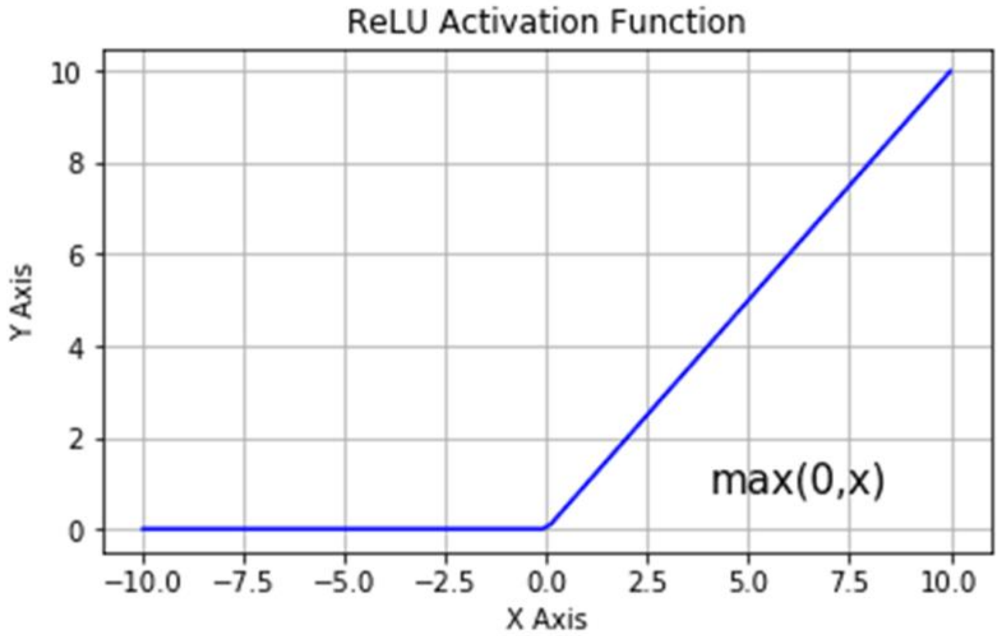

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

model=keras.models.Sequential() #Defines the model
model.add(Conv2D(8,(3,3),activation="relu",padding="same",input_shape=(64,64,1)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(16,(3,3),activation="relu",padding="same"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(32,(3,3),activation="relu",padding="same"))
```

Putting it together:




```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

model=keras.models.Sequential() #Defines the model
model.add(Conv2D(8,(3,3),activation="relu",padding="same",input_shape=(64,64,1)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(16,(3,3),activation="relu",padding="same"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(32,(3,3),activation="relu",padding="same"))
model.add(Flatten())
model.add(Dense(32,activation="relu"))
model.add(Dense(32,activation="relu"))
```

Putting it together:




```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

model=keras.models.Sequential() #Defines the model
model.add(Conv2D(8,(3,3),activation="relu",padding="same",input_shape=(64,64,1)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(16,(3,3),activation="relu",padding="same"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(32,(3,3),activation="relu",padding="same"))
model.add(Flatten())
model.add(Dense(32,activation="relu"))
model.add(Dense(32,activation="relu"))
model.add(Dense(2))
```

Compile the model

optimizer=keras.optimizers.Adam(learning_rate=0.01)

model.compile(optimizer=optimizer,loss="mae")

Mean Square Error (MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Mean Absolute Error (MAE)

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n}$$

Display a summary of the model

```
model.build()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
conv2d (Conv2D) (None, 64, 64, 8) 80
max_pooling2d (MaxPooling2D (None, 32, 32, 8) 0
conv2d_1 (Conv2D) (None, 32, 32, 16) 1168
max_pooling2d_1 (MaxPooling (None, 16, 16, 16) 0
conv2d_2 (Conv2D) (None, 16, 16, 32) 4640
flatten (Flatten) (None, 8192) 0
dense (Dense) (None, 32) 262176
dense_1 (Dense) (None, 32) 1056
dense_2 (Dense) (None, 2) 66
=================================================================
Total params: 269,186
Trainable params: 269,186
Non-trainable params: 0
_________________________________________________________________
```

Create a simple dataset using deeptrack

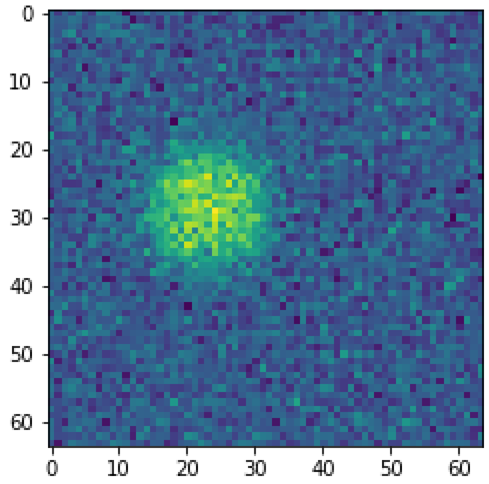
```
import deeptrack as dt
import numpy as np
import matplotlib.pyplot as plt
```

Create a simple dataset using deeptrack

```
import deeptrack as dt
import numpy as np
import matplotlib.pyplot as plt

particle=dt.Sphere(position=lambda: (np.random.uniform(20,40),np.random.uniform(20,40)))
optics=dt.Fluorescence(output_region=(0,0,64,64))
sample=optics(particle)>>dt.Gaussian(sigma=0.1)
```

Create a simple dataset using deeptrack

```
import deeptrack as dt
import numpy as np
import matplotlib.pyplot as plt

particle=dt.Sphere(position=lambda: (np.random.uniform(20,40),np.random.uniform(20,40)))
optics=dt.Fluorescence(output_region=(0,0,64,64))
sample=optics(particle)>>dt.Gaussian(sigma=0.1)

im=sample.update()()
plt.imshow(im[...,0])
```



Create a simple dataset using deeptrack

```
Training_dataset=[sample.update()() for i in range(1000)]
Validation_dataset=[sample.update()() for i in range(100)]
```

Create a simple dataset using deeptrack

```
Training_dataset=[sample.update()() for i in range(1000)]
Validation_dataset=[sample.update()() for i in range(100)]
```

```
def label_function(image):
    position=image.get_property("position")
    return np.array(position)
```

Create a simple dataset using deeptrack

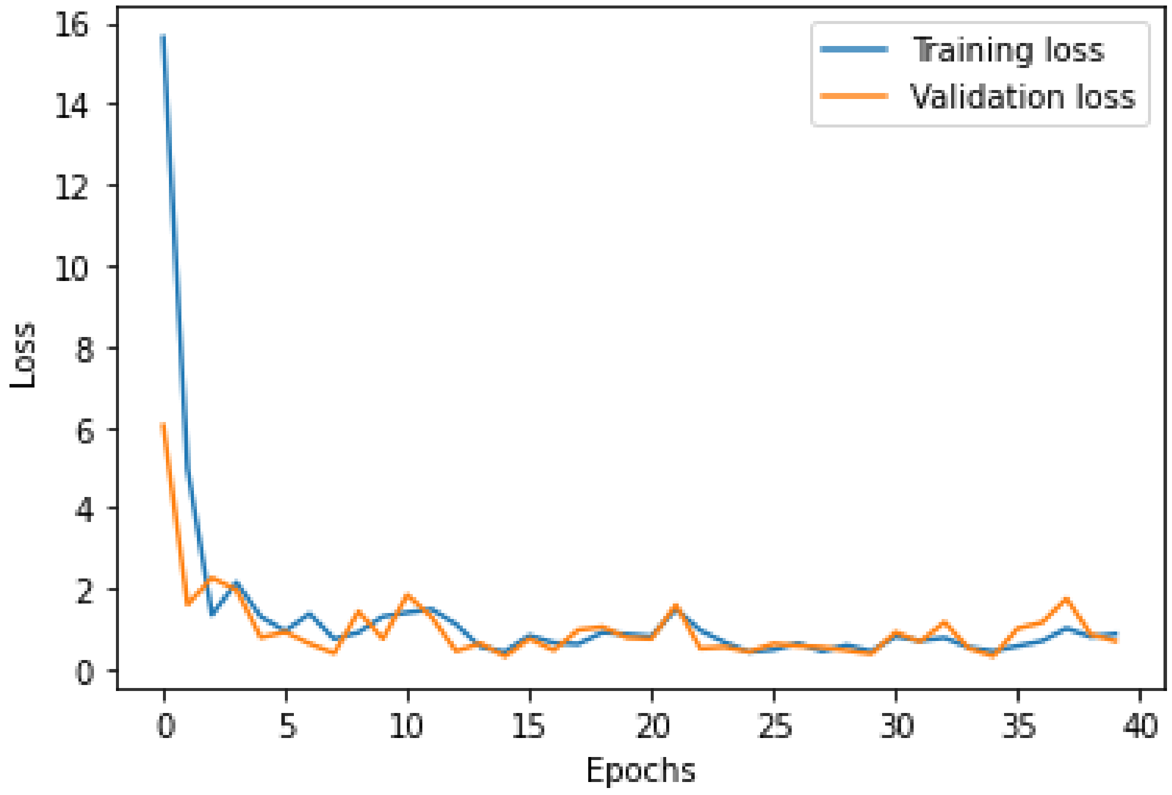
```
Training_dataset=[sample.update()() for i in range(1000)]
Validation_dataset=[sample.update()() for i in range(100)]
```

```
def label_function(image):
    position=image.get_property("position")
    return np.array(position)
```

```
Training_labels=[label_function(Training_dataset[i]) for i in range(len(Training_dataset))]
Validation_labels=[label_function(Validation_dataset[i]) for i in range(len(Validation_dataset))]
```

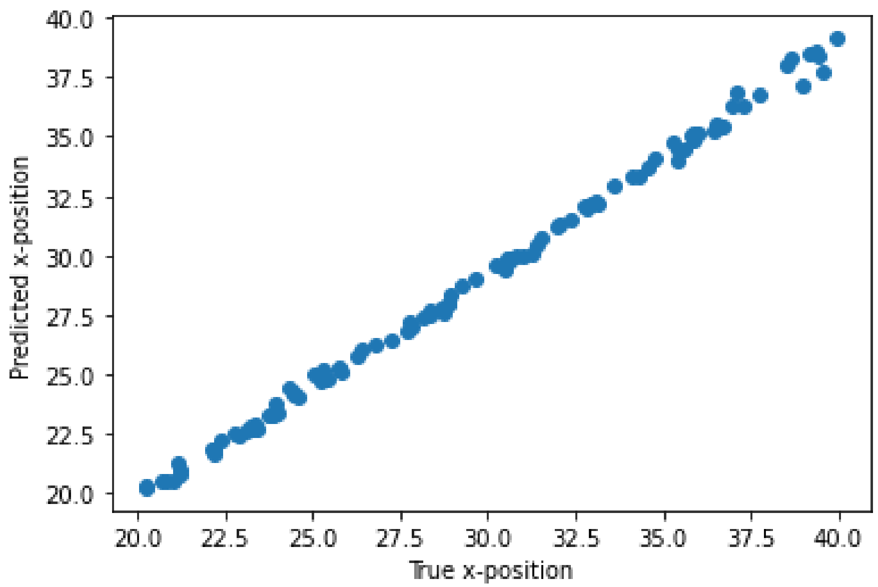
Train and test the model

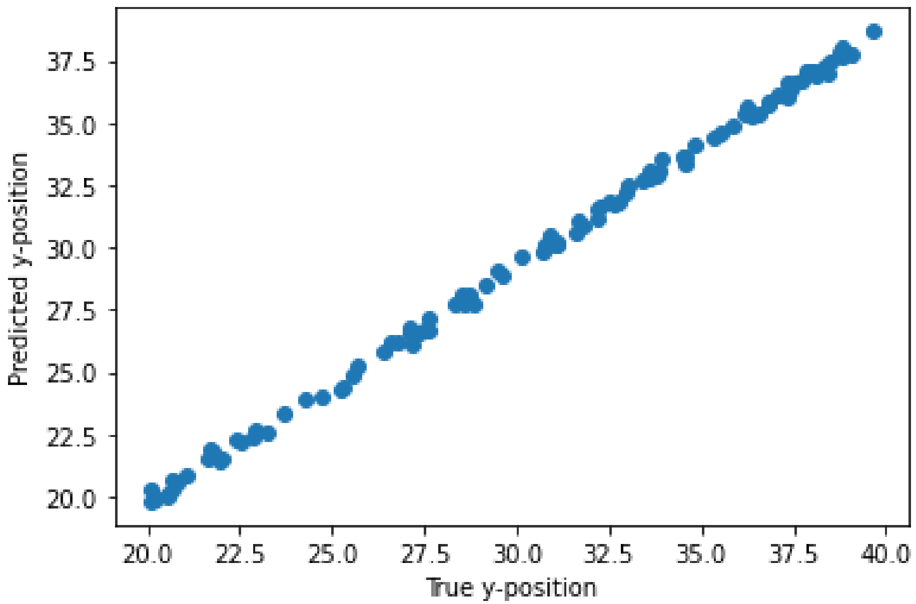
```
h=model.fit(x=np.array(Training_dataset),y=np.array(Training_labels),validation_data=(np.array(Validation_dataset),
np.array(Validation_labels)),epochs=40)
```



Train and test the model

p=model.predict(np.array(Validation_dataset))



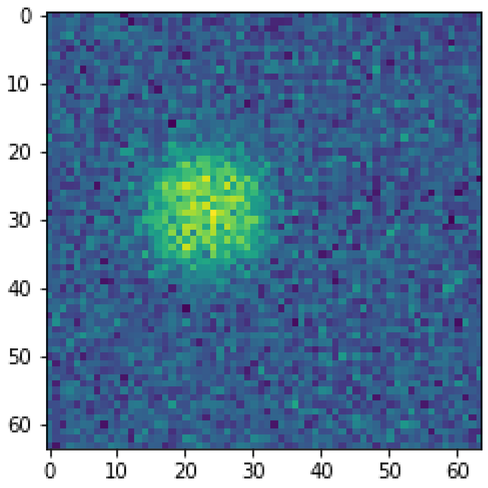


Plot the feature maps

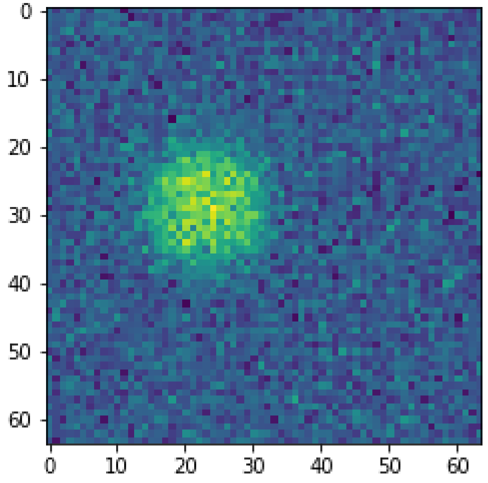
```
models=
[keras.Model(inputs=model.input,outputs=model.layers[0].output),
keras.Model(inputs=model.input,outputs=model.layers[2].output),
keras.Model(inputs=model.input,outputs=model.layers[4].output)]
```

Plot the feature maps

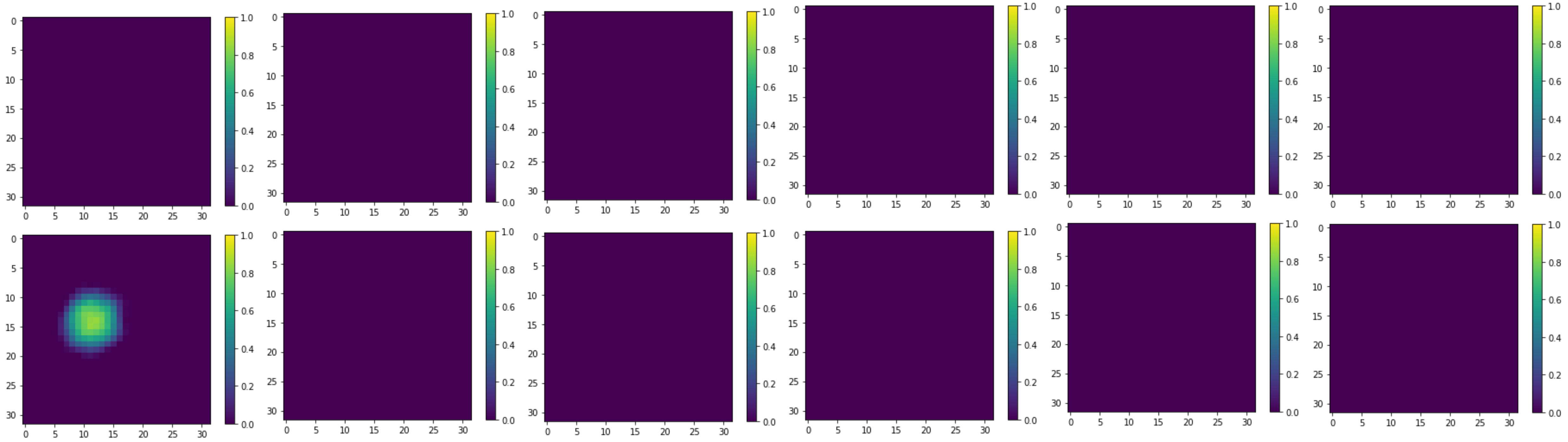
```
models=
[keras.Model(inputs=model.input,outputs=model.layers[0].output),
keras.Model(inputs=model.input,outputs=model.layers[2].output),
keras.Model(inputs=model.input,outputs=model.layers[4].output)]
p2=models[1].predict(im[np.newaxis])
```



Plot the feature maps



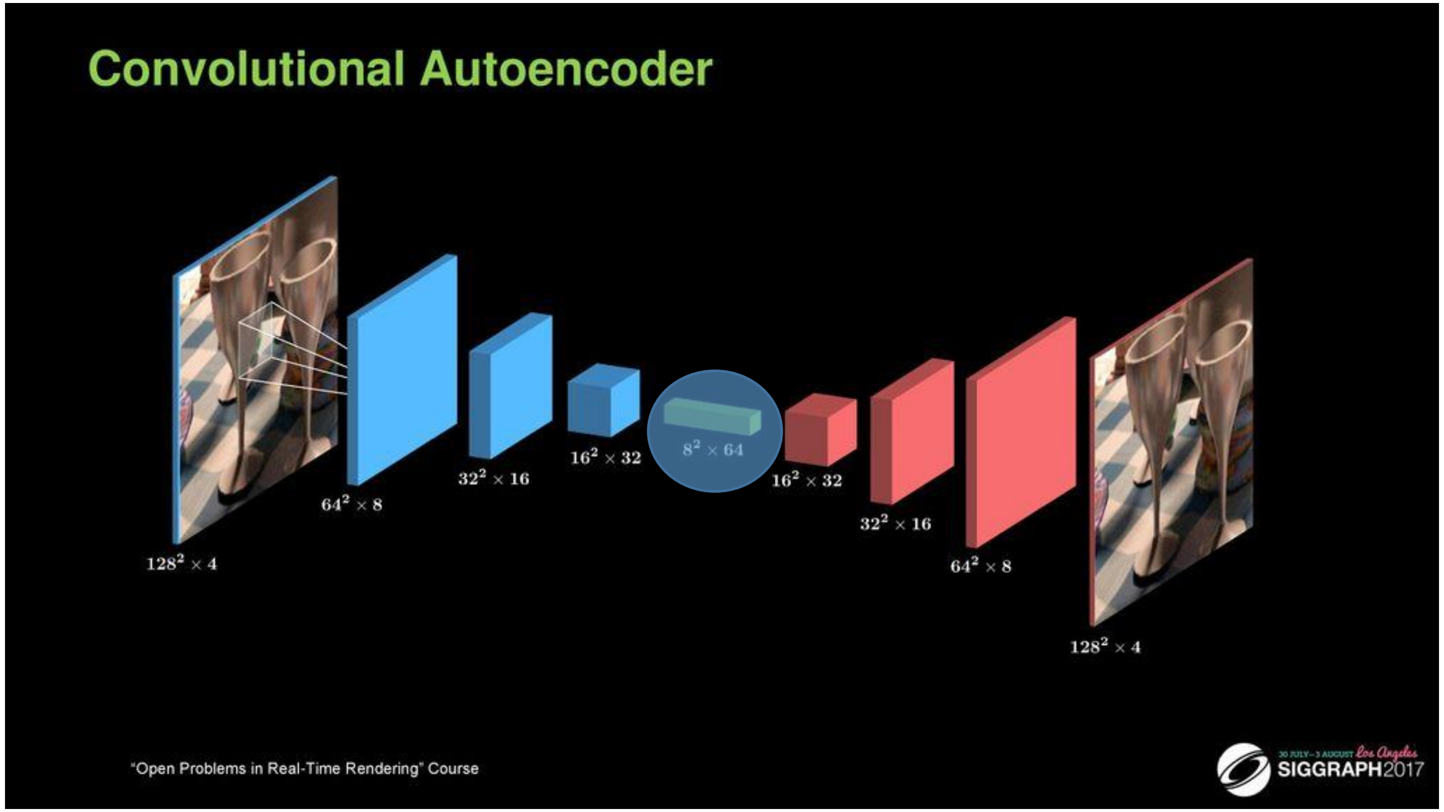
```
models=
[keras.Model(inputs=model.input,outputs=model.layers[0].output),
keras.Model(inputs=model.input,outputs=model.layers[2].output),
keras.Model(inputs=model.input,outputs=model.layers[4].output)]
p2=models[1].predict(im[np.newaxis])
```



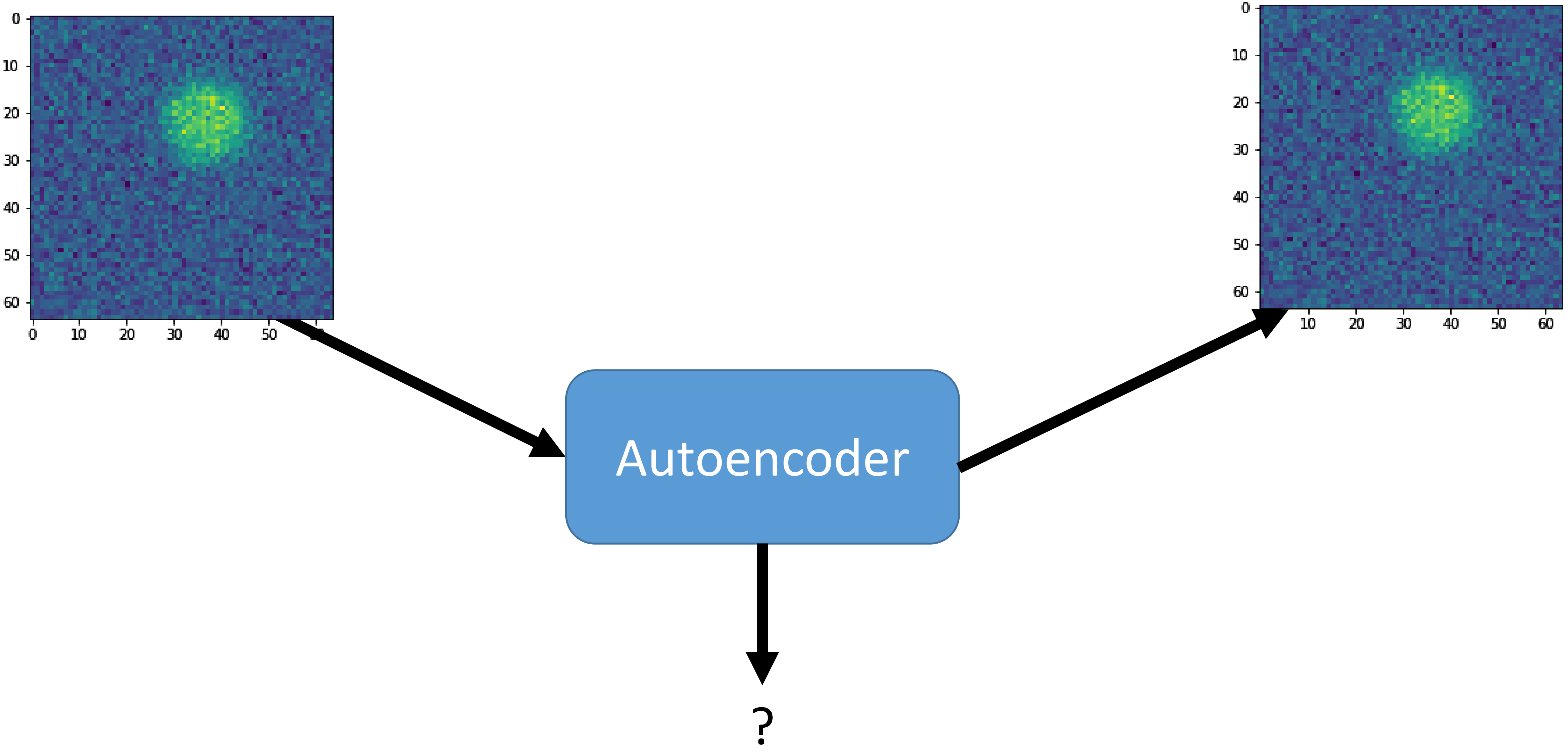
Encoder-decoders

- CNNs requires relatively large amounts of labeled data to work
- What if we have only a small set of training data?
- Encoder-decoder models (or autoencoders) can help solve that problem

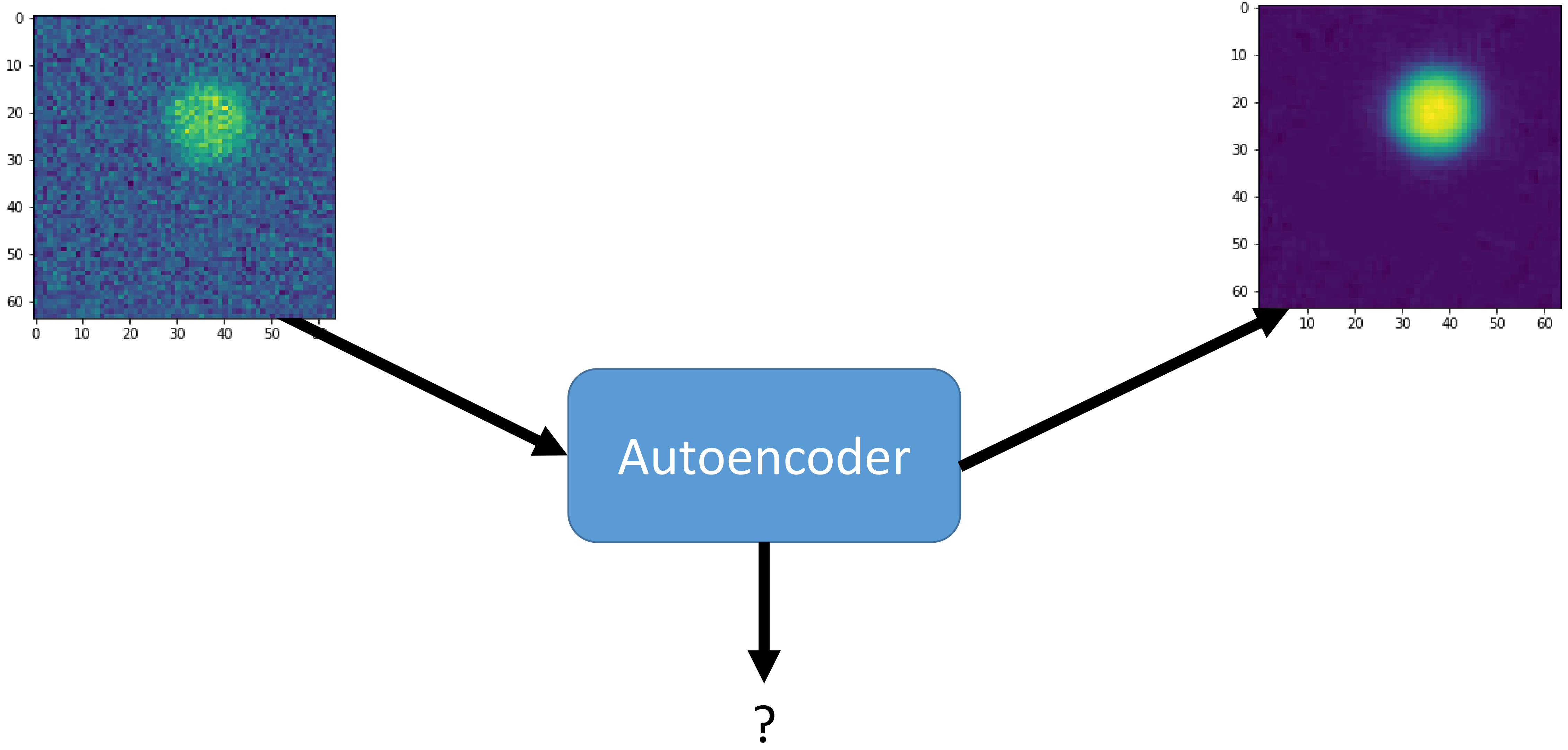
Autoencoders for data compression



Think!



Think!



Autoencoders compress the input data to its essential features!

- Lower-dimensional input space -> less labeled data is required
- Let’s try!

Coding example

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

encoder=keras.models.Sequential()
```

Coding example

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

encoder=keras.models.Sequential()
encoder.add(Conv2D(8,(3,3),activation="relu",padding="same",
input_shape=(64,64,1)))
```

Coding example

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

encoder=keras.models.Sequential()
encoder.add(Conv2D(8,(3,3),activation="relu",padding="same",
input_shape=(64,64,1)))
encoder.add(MaxPool2D(pool_size=(2,2)))
```

Coding example

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

encoder=keras.models.Sequential()
encoder.add(Conv2D(8,(3,3),activation="relu",padding="same",
input_shape=(64,64,1)))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(16,(3,3),activation="relu",padding="same
"))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(32,(3,3),activation="relu",padding="same
"))
encoder.add(Flatten())
encoder.add(Dense(32))
encoder.add(Dense(32))
encoder.add(Dense(2))
```

Coding example

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

encoder=keras.models.Sequential()
encoder.add(Conv2D(8,(3,3),activation="relu",padding="same",
input_shape=(64,64,1)))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(16,(3,3),activation="relu",padding="same
"))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(32,(3,3),activation="relu",padding="same
"))
encoder.add(Flatten())
encoder.add(Dense(32))
encoder.add(Dense(32))
encoder.add(Dense(2))
```

```
Conv2D=keras.layers.Conv2D
Dense=keras.layers.Dense
Reshape=keras.layers.Reshape
Upsample2D=keras.layers.UpSampling2D

decoder=keras.models.Sequential()
```

Coding example

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

encoder=keras.models.Sequential()
encoder.add(Conv2D(8,(3,3),activation="relu",padding="same",
input_shape=(64,64,1)))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(16,(3,3),activation="relu",padding="same
"))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(32,(3,3),activation="relu",padding="same
"))
encoder.add(Flatten())
encoder.add(Dense(32))
encoder.add(Dense(32))
encoder.add(Dense(2))
```

```
Conv2D=keras.layers.Conv2D
Dense=keras.layers.Dense
Reshape=keras.layers.Reshape
Upsample2D=keras.layers.UpSampling2D

decoder=keras.models.Sequential()
decoder.add(Dense(32))
decoder.add(Dense(32))
```

Coding example

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

encoder=keras.models.Sequential()
encoder.add(Conv2D(8,(3,3),activation="relu",padding="same",
input_shape=(64,64,1)))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(16,(3,3),activation="relu",padding="same
"))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(32,(3,3),activation="relu",padding="same
"))
encoder.add(Flatten())
encoder.add(Dense(32))
encoder.add(Dense(32))
encoder.add(Dense(2))
```

```
Conv2D=keras.layers.Conv2D
Dense=keras.layers.Dense
Reshape=keras.layers.Reshape
Upsample2D=keras.layers.UpSampling2D

decoder=keras.models.Sequential()
decoder.add(Dense(32))
decoder.add(Dense(32))
decoder.add(Dense(16*16*32))
decoder.add(Reshape((16,16,32)))
```

Coding example

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

encoder=keras.models.Sequential()
encoder.add(Conv2D(8,(3,3),activation="relu",padding="same",
input_shape=(64,64,1)))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(16,(3,3),activation="relu",padding="same
"))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(32,(3,3),activation="relu",padding="same
"))
encoder.add(Flatten())
encoder.add(Dense(32))
encoder.add(Dense(32))
encoder.add(Dense(2))
```

```
Conv2D=keras.layers.Conv2D
Dense=keras.layers.Dense
Reshape=keras.layers.Reshape
Upsample2D=keras.layers.UpSampling2D

decoder=keras.models.Sequential()
decoder.add(Dense(32))
decoder.add(Dense(32))
decoder.add(Dense(16*16*32))
decoder.add(Reshape((16,16,32)))
decoder.add(Conv2D(32,(3,3),activation="relu",padding="same
"))
decoder.add(Upsample2D((2,2)))
```

Coding example

```
Conv2D=keras.layers.Conv2D
MaxPool2D=keras.layers.MaxPooling2D
Dense=keras.layers.Dense
Flatten=keras.layers.Flatten

encoder=keras.models.Sequential()
encoder.add(Conv2D(8,(3,3),activation="relu",padding="same",
input_shape=(64,64,1)))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(16,(3,3),activation="relu",padding="same
"))
encoder.add(MaxPool2D(pool_size=(2,2)))
encoder.add(Conv2D(32,(3,3),activation="relu",padding="same
"))
encoder.add(Flatten())
encoder.add(Dense(32))
encoder.add(Dense(32))
encoder.add(Dense(2))
```

```
Conv2D=keras.layers.Conv2D
Dense=keras.layers.Dense
Reshape=keras.layers.Reshape
Upsample2D=keras.layers.UpSampling2D

decoder=keras.models.Sequential()
decoder.add(Dense(32))
decoder.add(Dense(32))
decoder.add(Dense(16*16*32))
decoder.add(Reshape((16,16,32)))
decoder.add(Conv2D(32,(3,3),activation="relu",padding="same
"))
decoder.add(Upsample2D((2,2)))
decoder.add(Conv2D(16,(3,3),activation="relu",padding="same
"))
decoder.add(Upsample2D((2,2)))
decoder.add(Conv2D(8,(3,3),activation="relu",padding="same"
))
decoder.add(Conv2D(1,(3,3),padding="same"))
```

Coding example

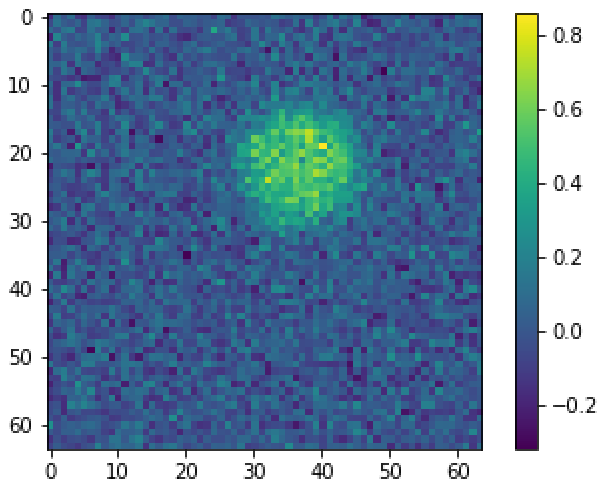
```
Input=keras.layers.Input((64,64,1))
autoencoder=keras.models.Model(inputs=Input,outputs=decoder(encoder(Input)))
```

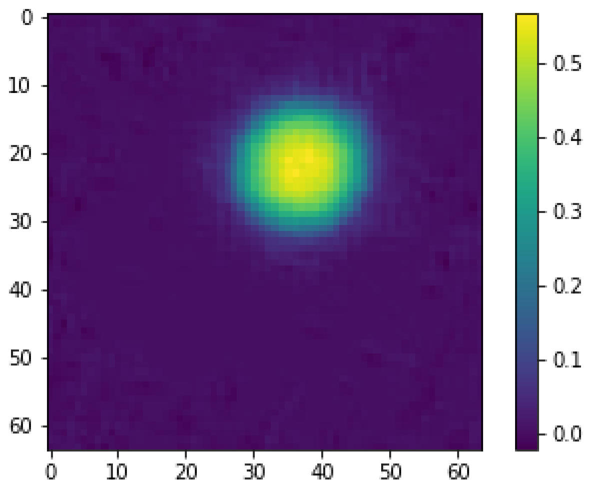
Coding example

```
Input=keras.layers.Input((64,64,1))
autoencoder=keras.models.Model(inputs=Input,outputs=decoder(encoder(Input)))

optimizer=keras.optimizers.Adam(learning_rate=0.0001)
autoencoder.compile(optimizer=optimizer,loss="mae")
autoencoder.fit(x=np.array(Training_dataset),y=np.array(Training_dataset),epochs=50)
```

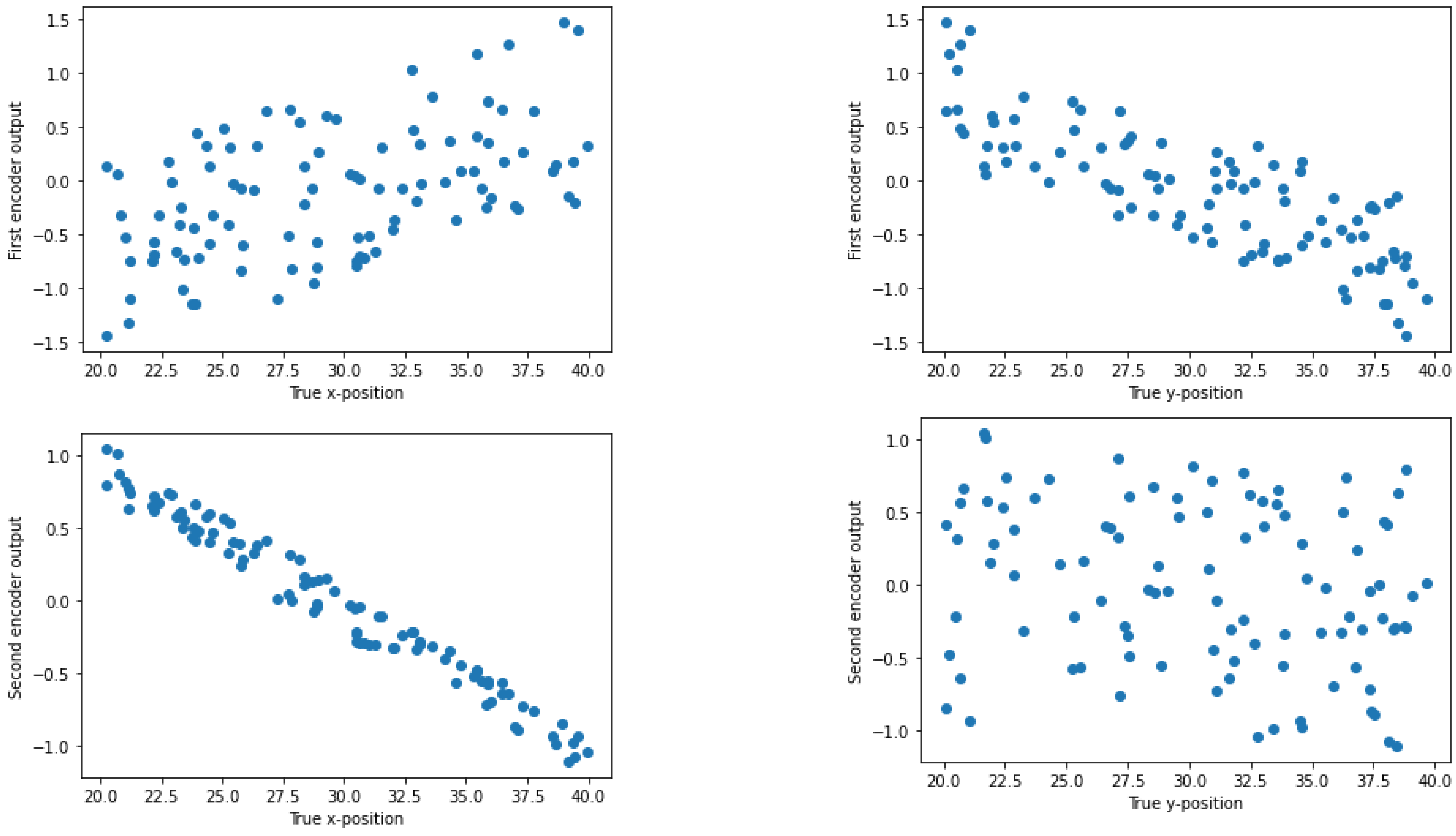
Coding example

```
Input=keras.layers.Input((64,64,1))
autoencoder=keras.models.Model(inputs=Input,outputs=decoder(encoder(Input)))
optimizer=keras.optimizers.Adam(learning_rate=0.0001)

autoencoder.compile(optimizer=optimizer,loss="mae")
autoencoder.fit(x=np.array(Training_dataset),y=np.array(Training_dataset),epochs=50)
```





Encoder output vs position



Optimization

```
small_model=keras.models.Sequential()
small_model.add(Dense(8,activation="tanh"))
small_model.add(Dense(2,activation="tanh"))
```

Optimization

```
small_model=keras.models.Sequential()
small_model.add(Dense(8,activation="tanh"))
small_model.add(Dense(2,activation="tanh"))
```

```
optimizer=keras.optimizers.Adam(learning_rate=0.01)

small_model.compile(optimizer=optimizer,loss="mae")
```

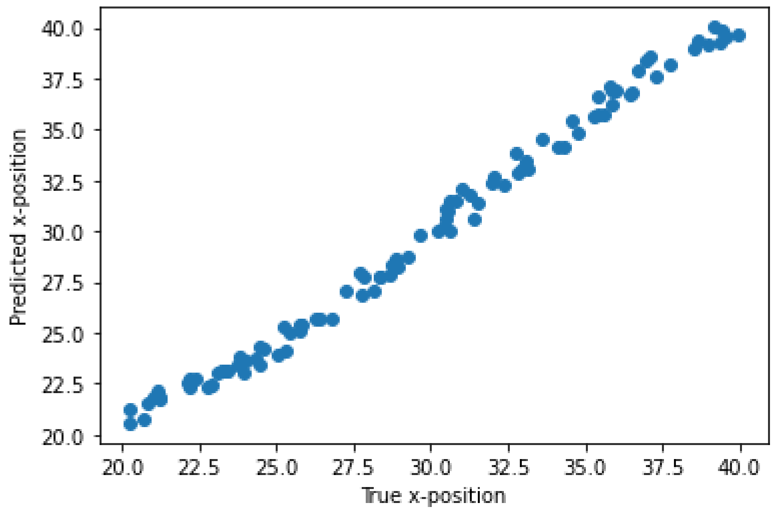
Optimization

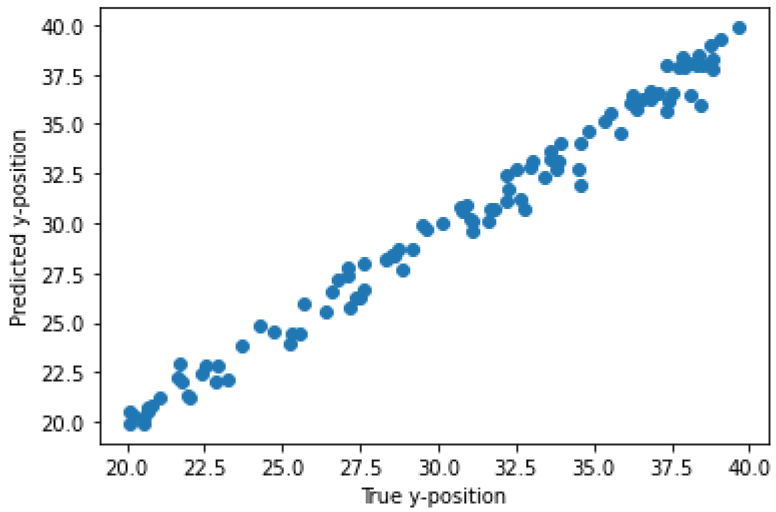
```
small_model=keras.models.Sequential()
small_model.add(Dense(8,activation="tanh"))
small_model.add(Dense(2,activation="tanh"))
```

```
optimizer=keras.optimizers.Adam(learning_rate=0.01)

small_model.compile(optimizer=optimizer,loss="mae")

input_data=encoder.predict(np.array(Training_dataset)[:20])

label_data=np.array(Training_labels)[:20]

small_model.fit(x=input_data,y=label_data,epochs=20)
```

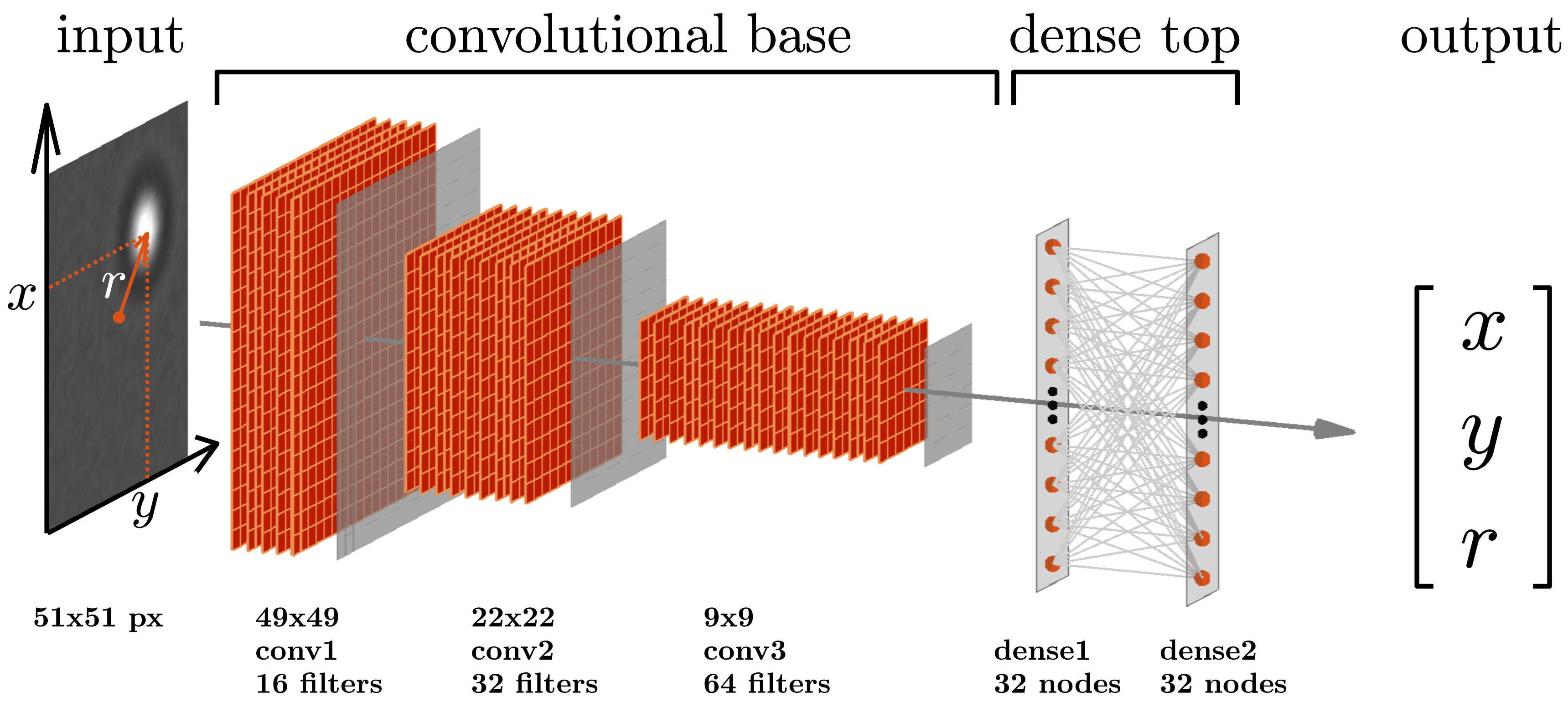
Results

```
val_inputs=encoder.predict(np.array(Validation_dataset))
val_labels=np.array(Validation_labels)

p=small_model.predict(val_inputs)
```





Attention!



Convolutional neural networks probe the image locally. Initially at a fine scale, scanning across the image, then progressively at larger and larger scales as the image is downsampled.

Think!

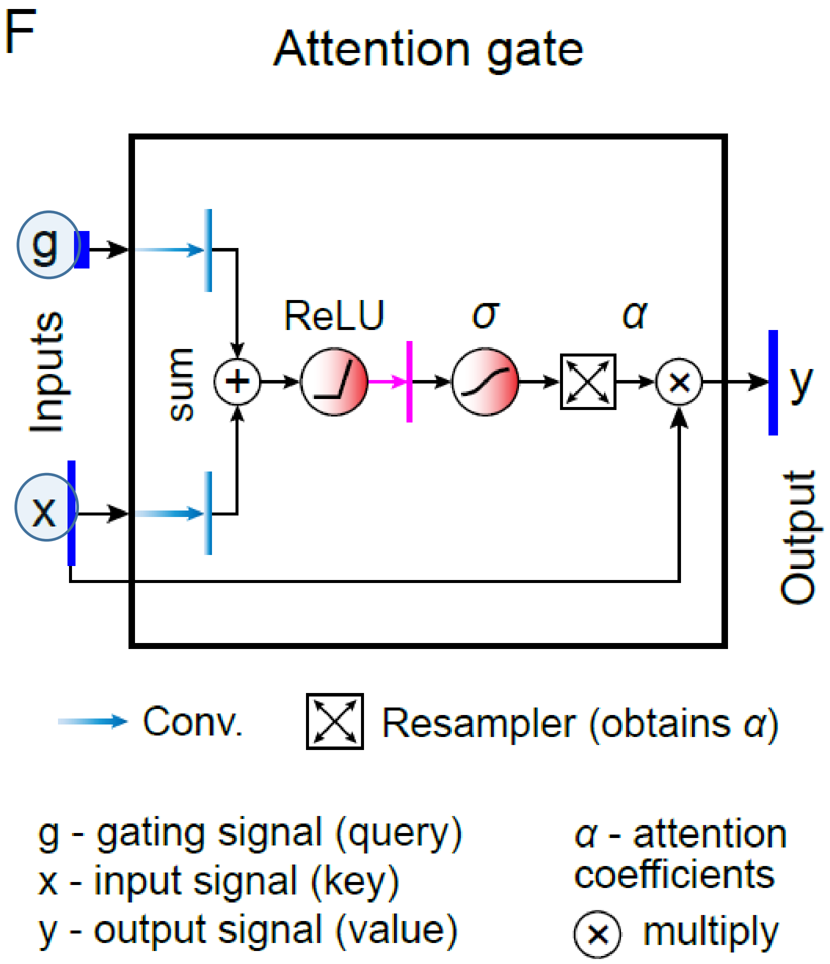
”Convolutional neural networks probe the image locally. Initially at a fine scale, scanning across the image, then progressively at larger and larger scales as the image is downsampled.”

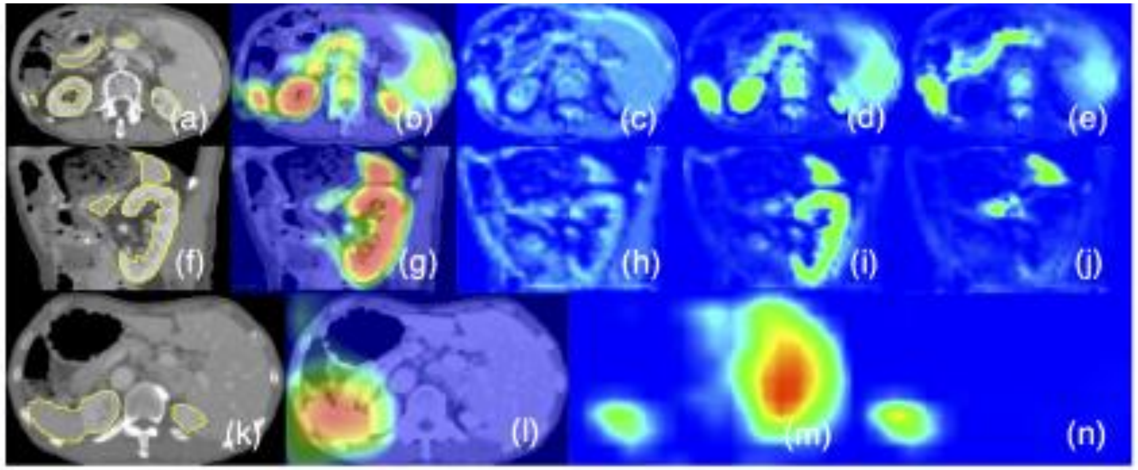
- Does your brain process images this way?

Attention!


HUBLOT
The New York Times
TIMES DIGITAL SUBSCRIPTIONS
MacBook

Spatial attention gate!

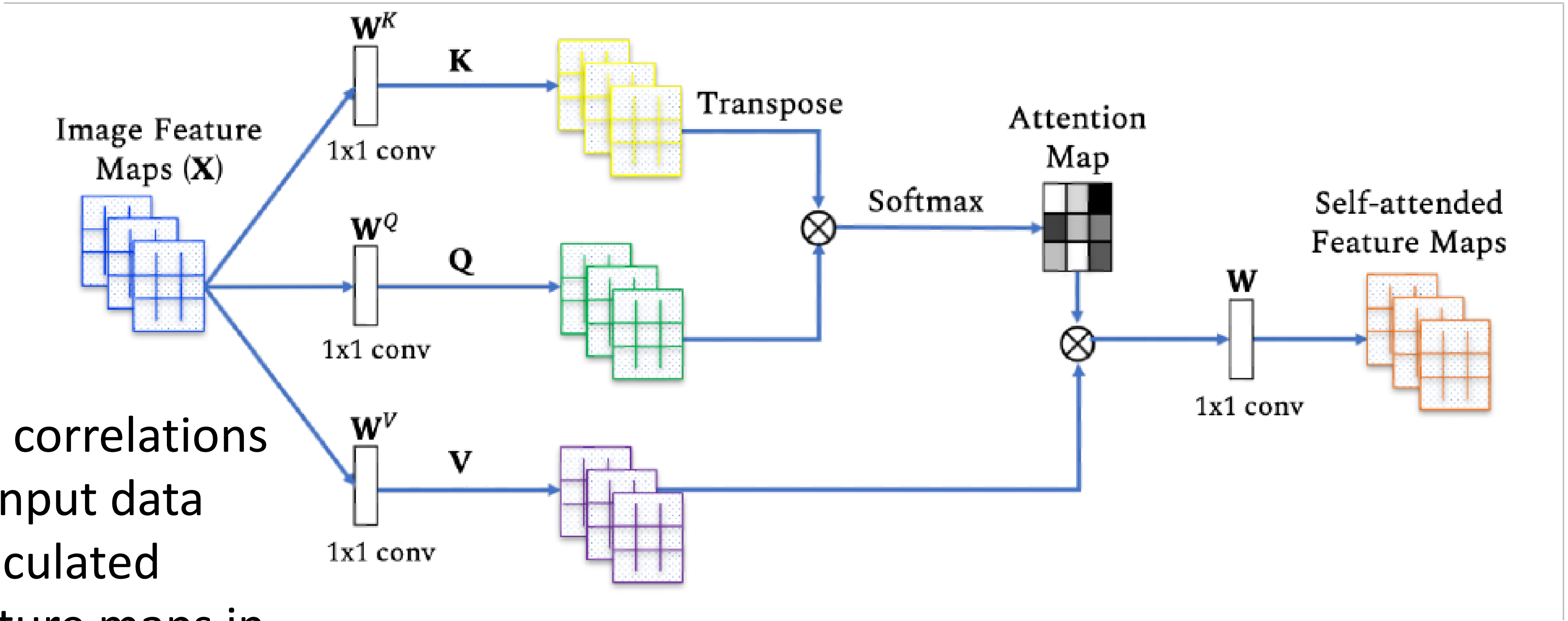




Oktay et al. Attention U-net: learning where to look for the pancreas

g is a coarse grained representation of the image
x is sampled on a finer scale

Self-attention



- Calculates correlations between input data
- Can be calculated across feature maps in latent space, or across pixels

Khan et al. Transformers in Vision: A Survey

Think!

- What are the benefits of using attention in image processing?
- Any drawbacks?

In-depth view of the attention gate

- Attention was first developed for natural language processing
- Let’s use that as an example!

Attention

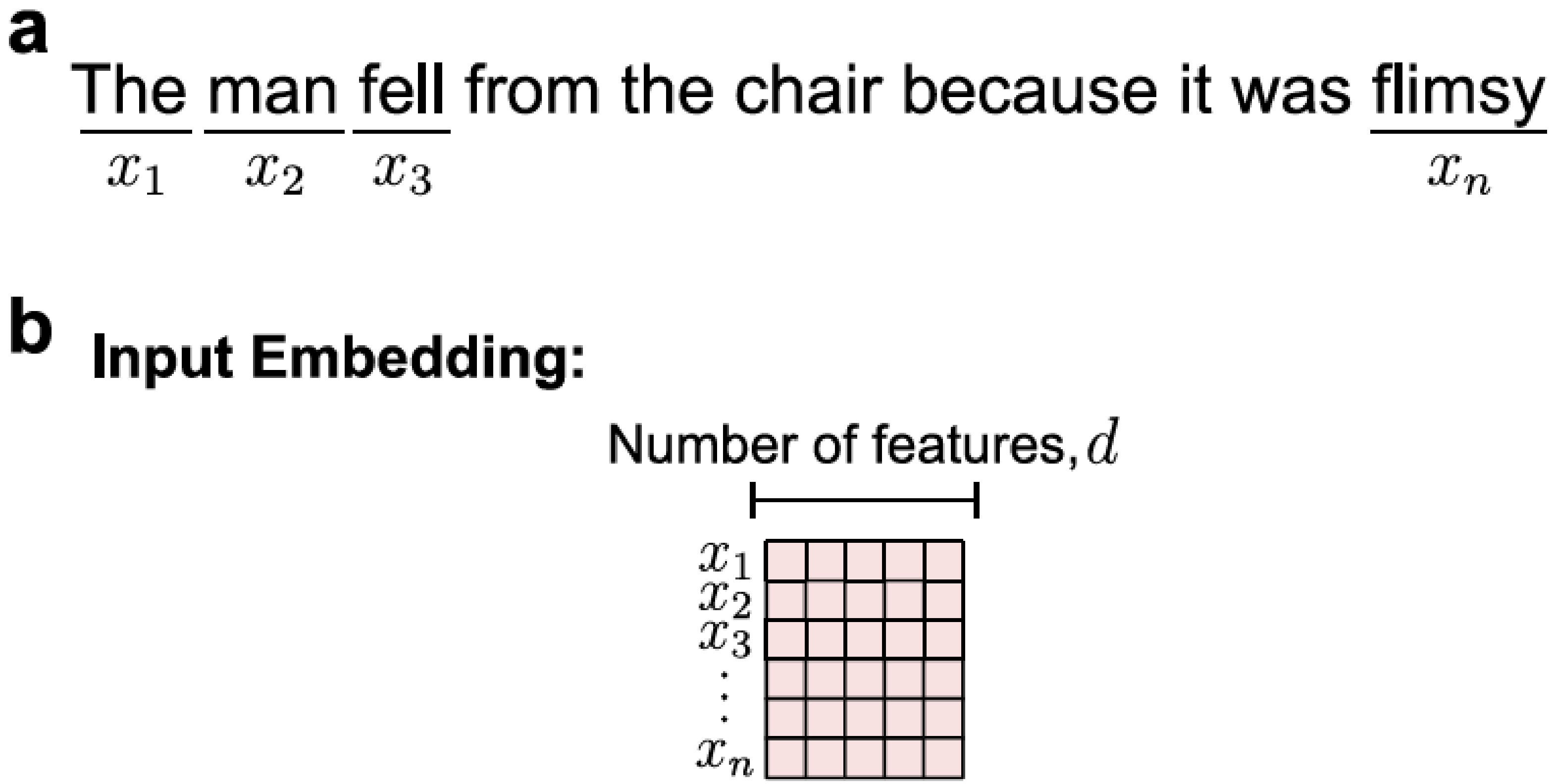
a

The man fell from the chair because it was flimsy

$\underbrace{\text{The}}_{x_1}\ \underbrace{\text{man}}_{x_2}\ \underbrace{\text{fell}}_{x_3}\ \text{from the chair because it was}\ \underbrace{\text{flimsy}}_{x_n}$

Does ”flimsy” refer to the word ”man” or the word ”chair”?
This is where attention comes into play! Let’s see how.

Attention

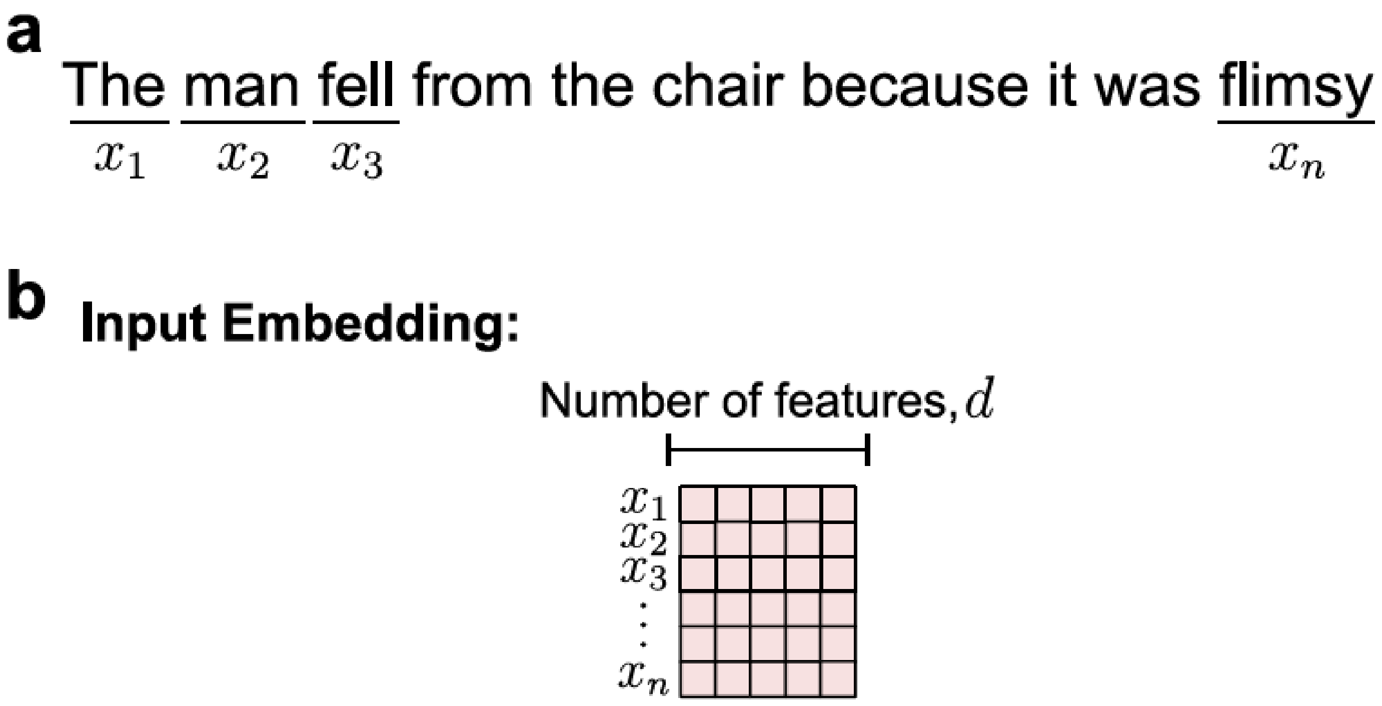


Does ”flimsy” refer to the word ”man” or the word ”chair”?
This is where attention comes into play! Let’s see how.

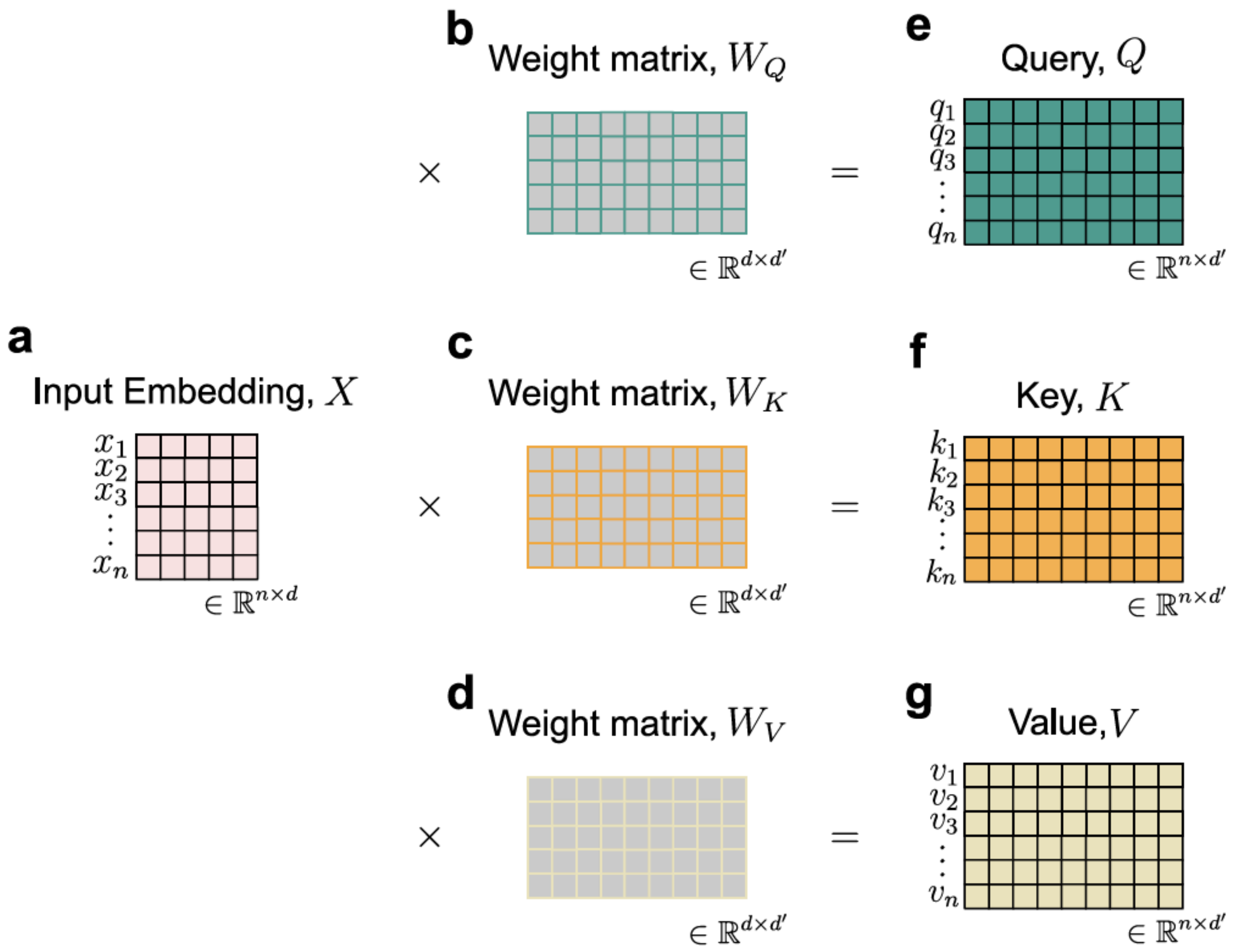
In Keras:

Indicates that the input can be of any length,
useful for language processing and time series!

```
d=5
X=keras.layers.Input(input_shape=(None,d))
```



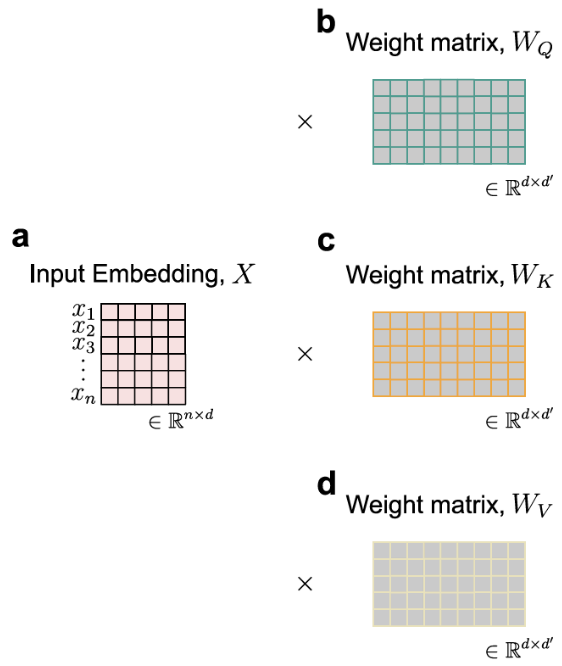
Attention



a
Input Embedding, X
x_1
x_2
x_3
x_n
$\in \mathbb{R}^{n\times d}$
b
Weight matrix, W_Q
$\in \mathbb{R}^{d\times d'}$
c
Weight matrix, W_K
$\in \mathbb{R}^{d\times d'}$
d
Weight matrix, W_V
$\in \mathbb{R}^{d\times d'}$
$\times$
$=$
e
Query, Q
q_1
q_2
q_3
q_n
$\in \mathbb{R}^{n\times d'}$
f
Key, K
k_1
k_2
k_3
k_n
$\in \mathbb{R}^{n\times d'}$
g
Value, V
v_1
v_2
v_3
v_n
$\in \mathbb{R}^{n\times d'}$

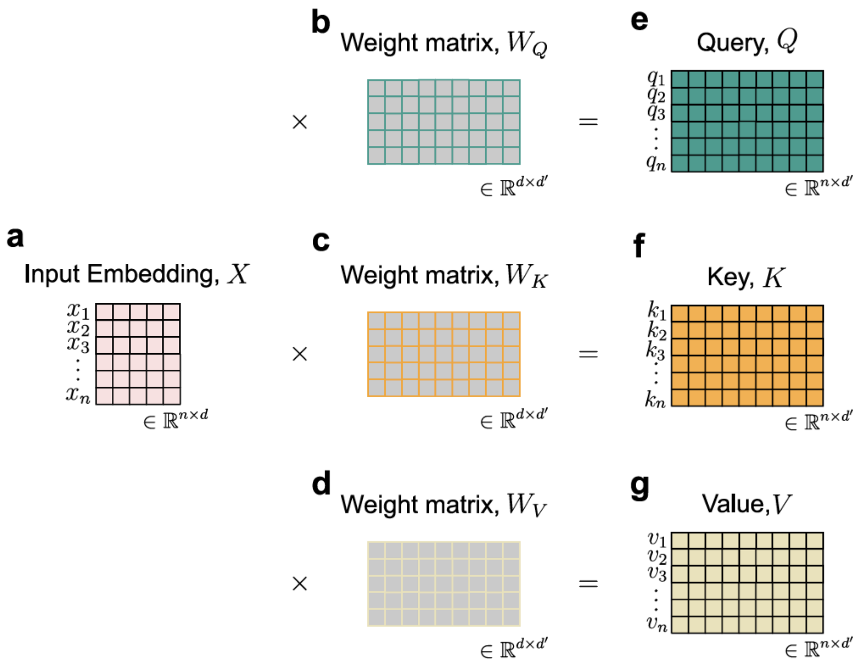
In Keras:

```
d=5
X=keras.layers.Input(input_shape=(None,d))

dp=9
query_dense=Dense(dp,activation="linear",use_bias=False)
key_dense=Dense(dp,activation="linear",use_bias=False)
value_dense=Dense(dp,activation="linear",use_bias=False)
```



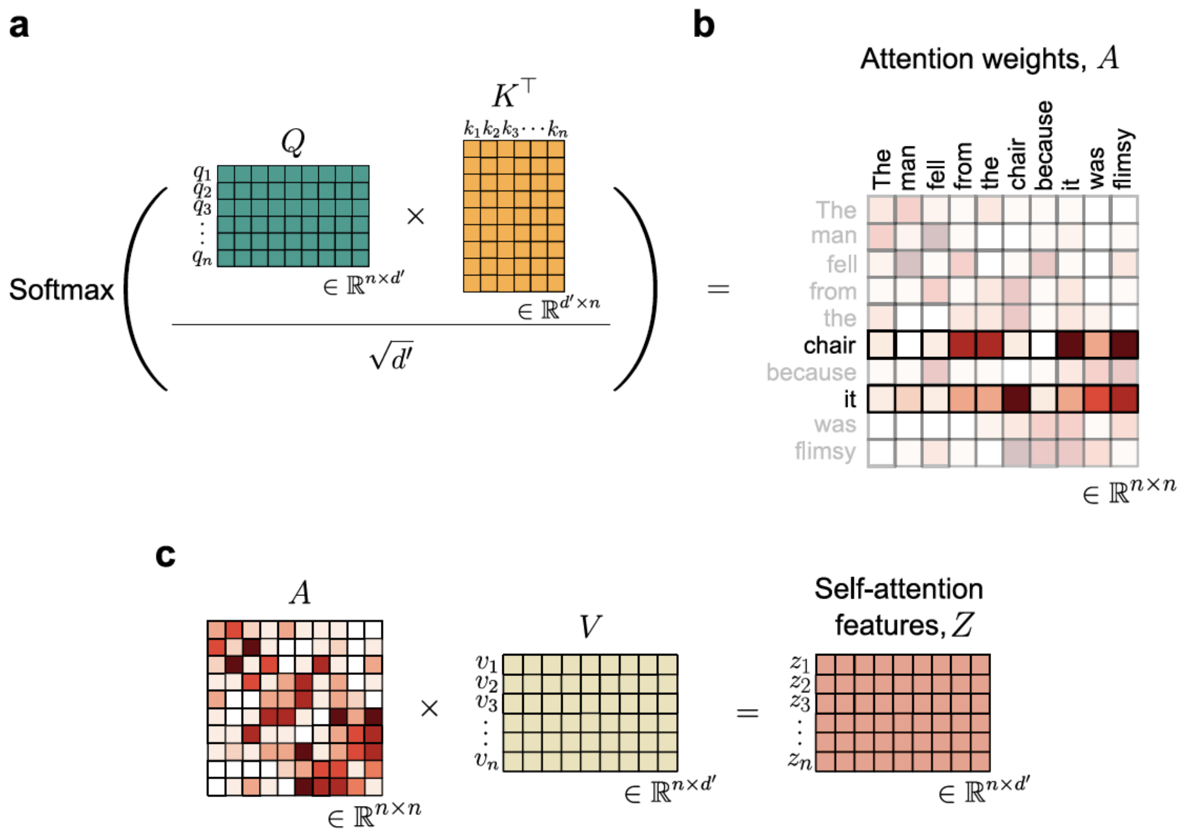
In Keras:

```
d=5
X=keras.layers.Input(input_shape=(None,d))

dp=9
query_dense=Dense(dp,activation="linear",use_bias=False)
key_dense=Dense(dp,activation="linear",use_bias=False)
value_dense=Dense(dp,activation="linear",use_bias=False)


Q=query_dense(X)
K=key_dense(X)
V=value_dense(X)
```



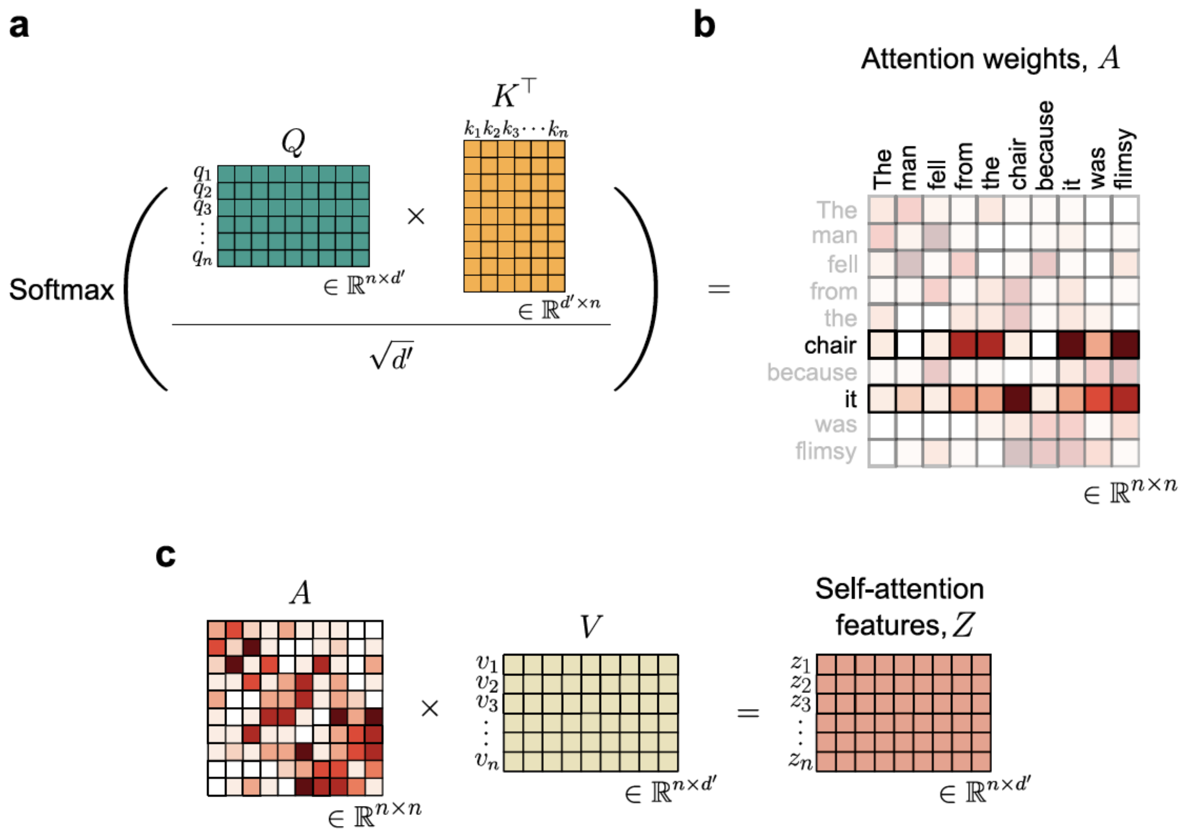
Attention

- Attention weights are, after training of the attention module, a map of the correlation between different input elements
- ”Self-attention”



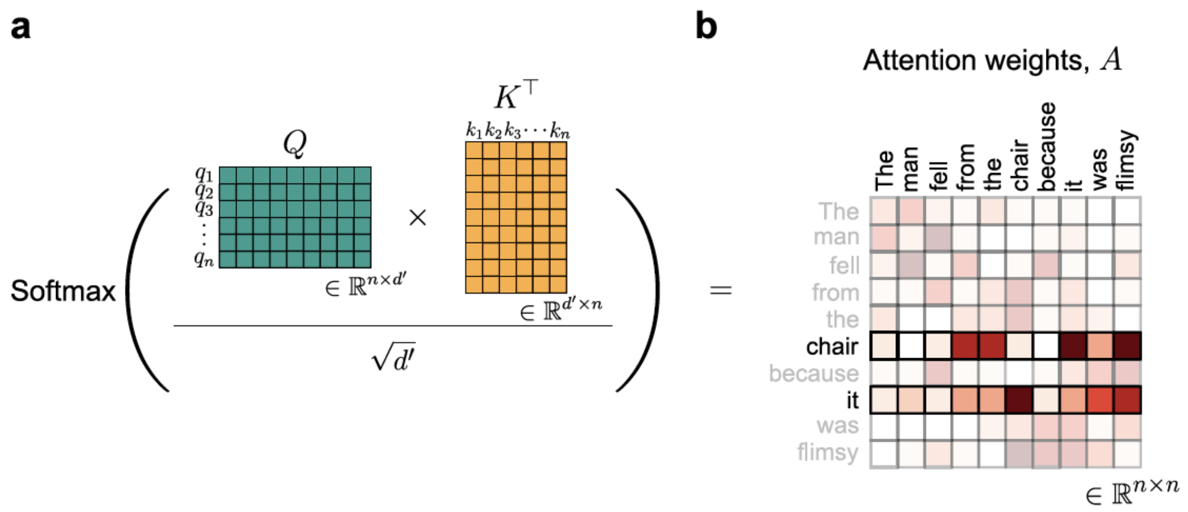
Attention

- After multipication with V the output is an nxd’ matrix in which each element represents a linear combination of the input elements, via the correlation matrix A



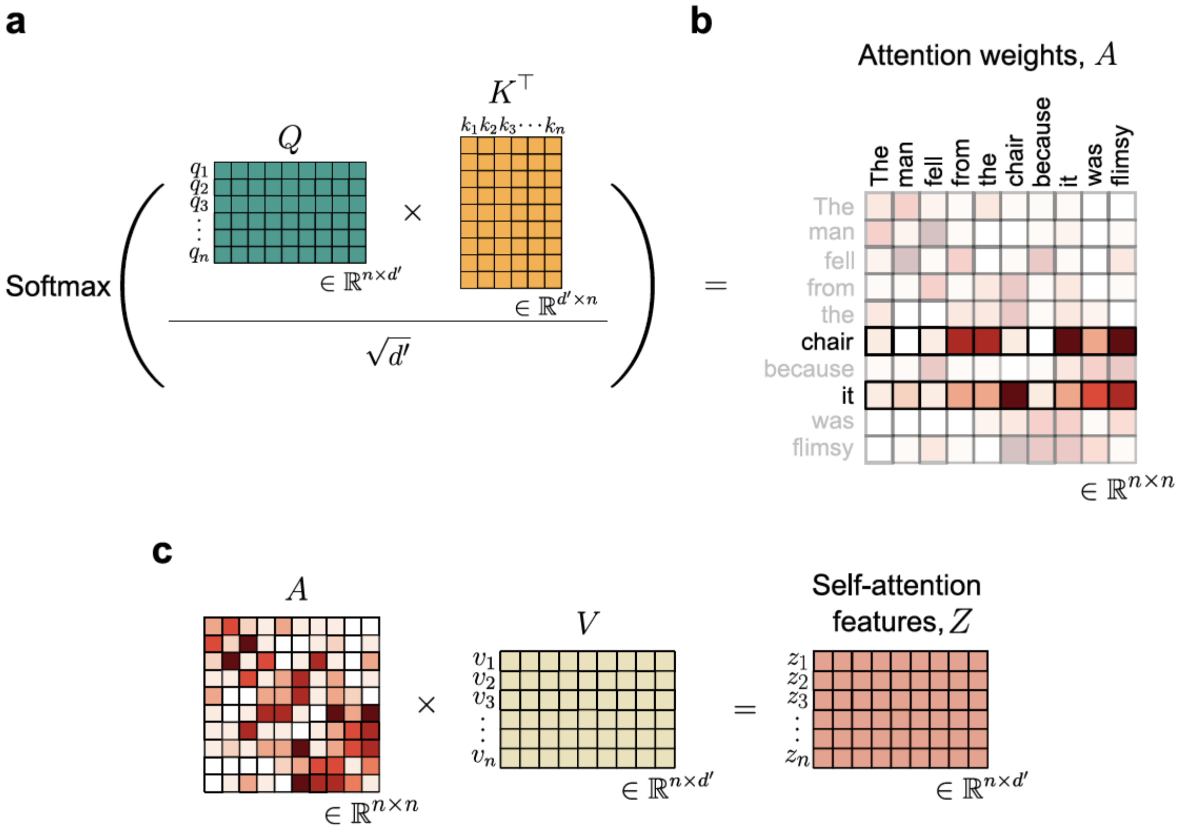
In Tensorflow:

```
import tensorflow as tf
score=tf.matmul(Q,K,transpose_b=True)
A=tf.nn.softmax(score/tf.math.sqrt(dp*1.),axis=-1)
```

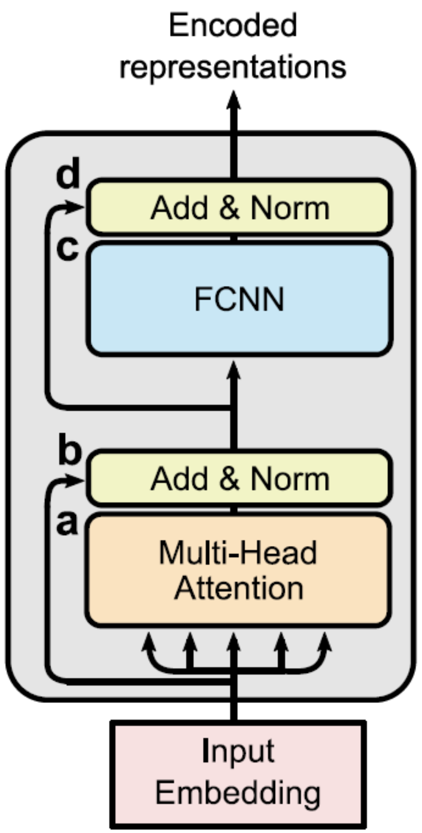


In Tensorflow:

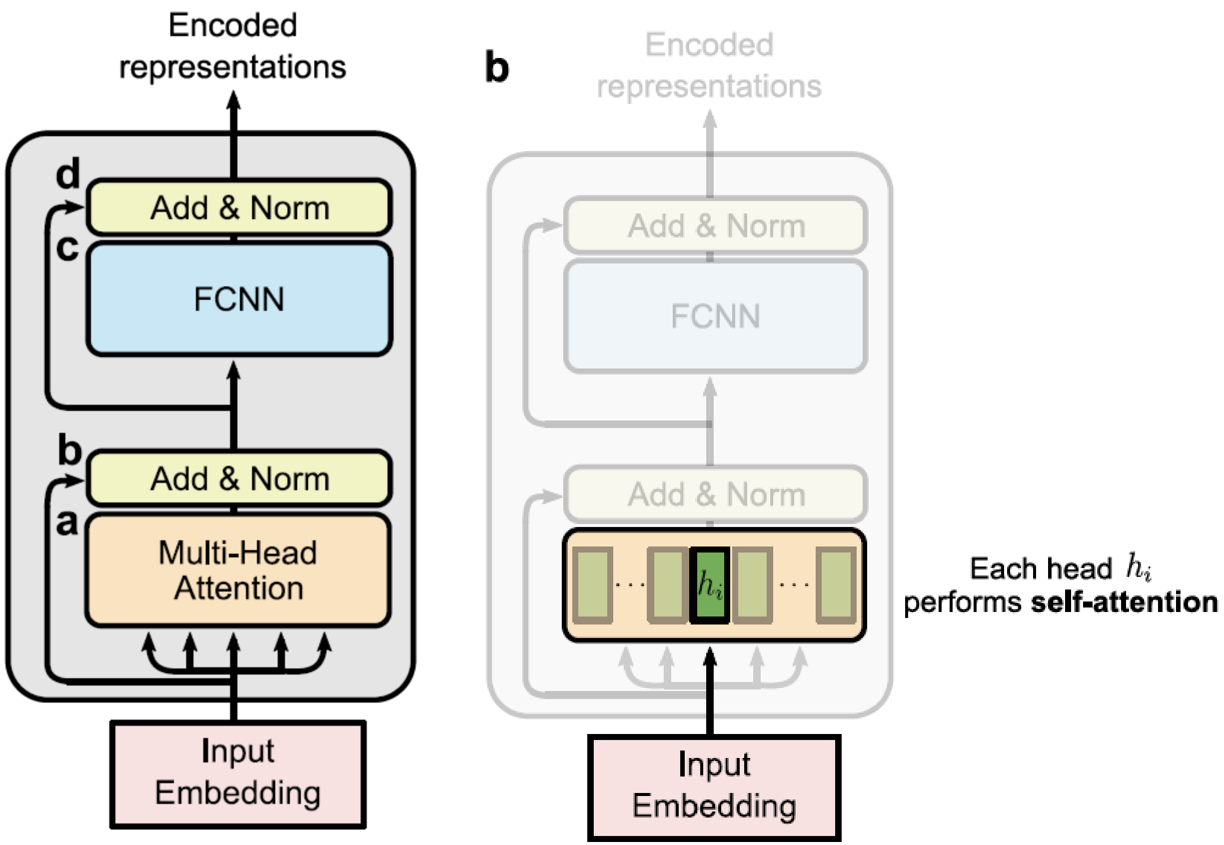
```
import tensorflow as tf
score=tf.matmul(Q,K,transpose_b=True)
A=tf.nn.softmax(score/tf.math.sqrt(dp*1.),axis=-1)

self_attention_features=tf.matmul(A,V)
```



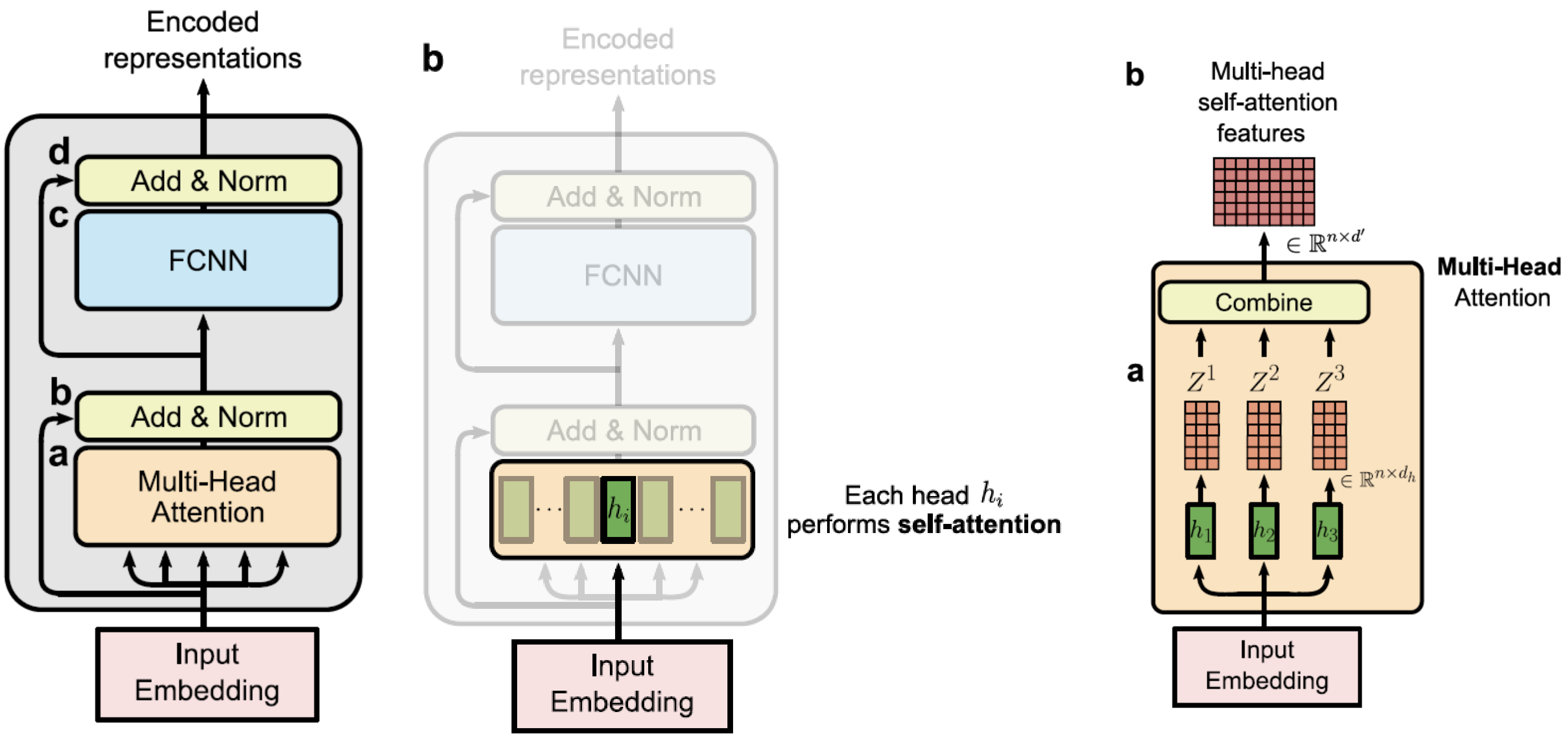
Transformer



Transformer



Transformer



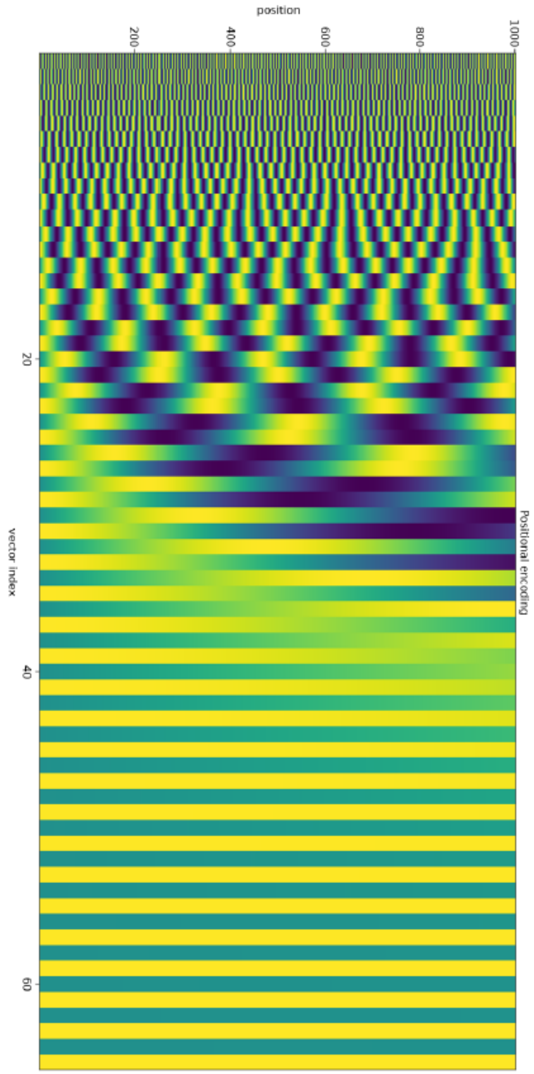
Temporal attention

- Temporal self-attention creates a correlation map between different time points.
- However, the network has no sense of temporal order
- When predicting future outcomes based on past observations, likely attention should be placed primarily on datapoints late in the time series

- This requries *positional embedding* of the input

Positional embedding (time 2 vector)

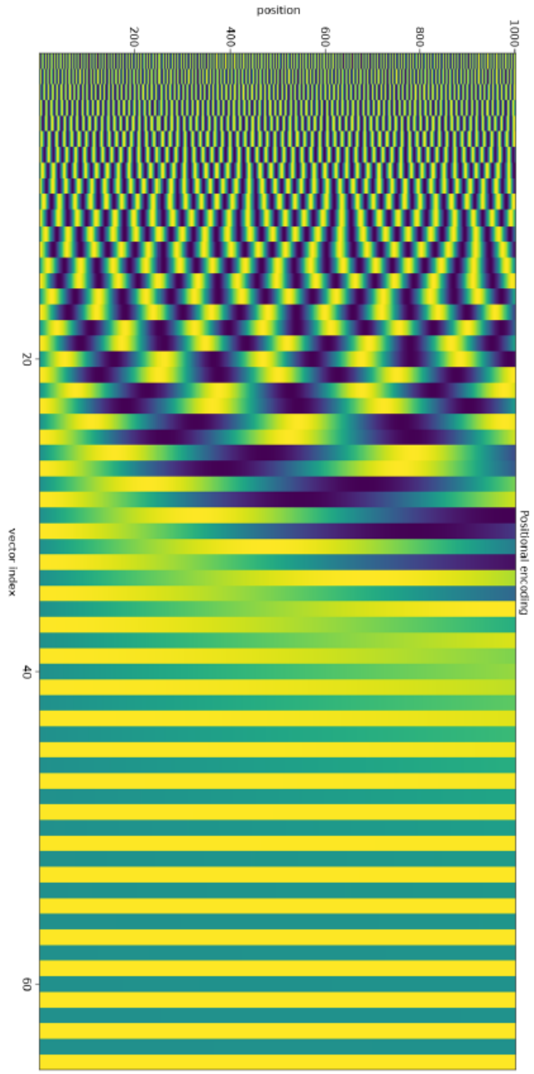
$$\mathbf{t2v}(\tau)[i] = \begin{cases} \omega_i \tau + \varphi_i, & \text{if } i = 0 \\ \mathcal{F}\left(\omega_i \tau + \varphi_i\right), & \text{if } 1 \leq i \leq k \end{cases}$$



Positional embedding (time 2 vector)

$$\mathbf{t2v}(\tau)[i] = \begin{cases} \omega_i \tau + \varphi_i, & \text{if } i = 0 \\ F\left(\omega_i \tau + \varphi_i\right), & \text{if } 1 \leq i \leq k \end{cases}$$

F is a periodic function
ω_i and φ_i are learnable parameters

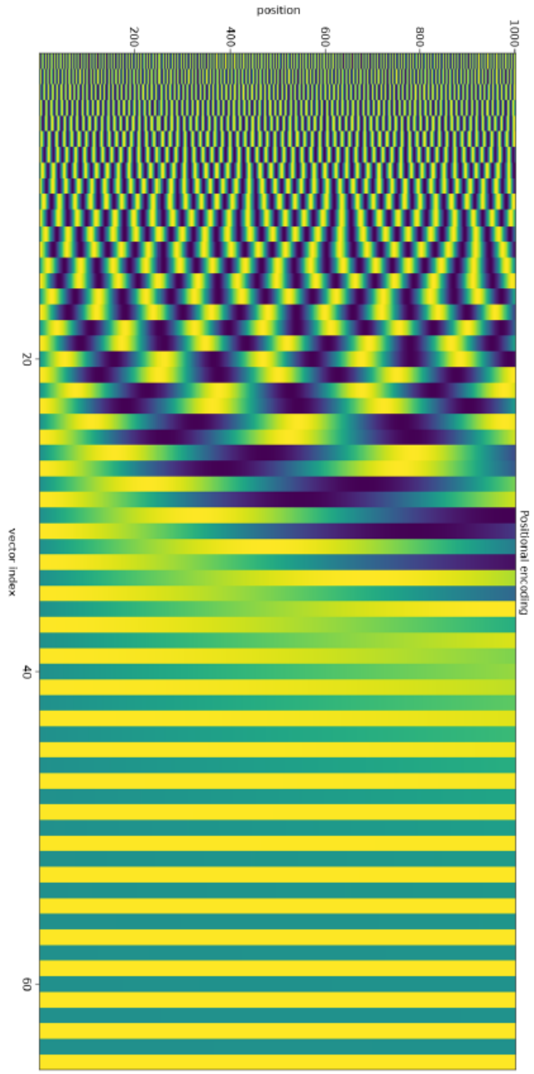


Positional embedding (time 2 vector)

$$\mathbf{t2v}(\tau)[i] = \begin{cases} \omega_i \tau + \varphi_i, & \text{if } i = 0 \\ F(\omega_i \tau + \varphi_i), & \text{if } 1 \le i \le k \end{cases}$$

F is a periodic function
ω_i and φ_i are learnable parameters

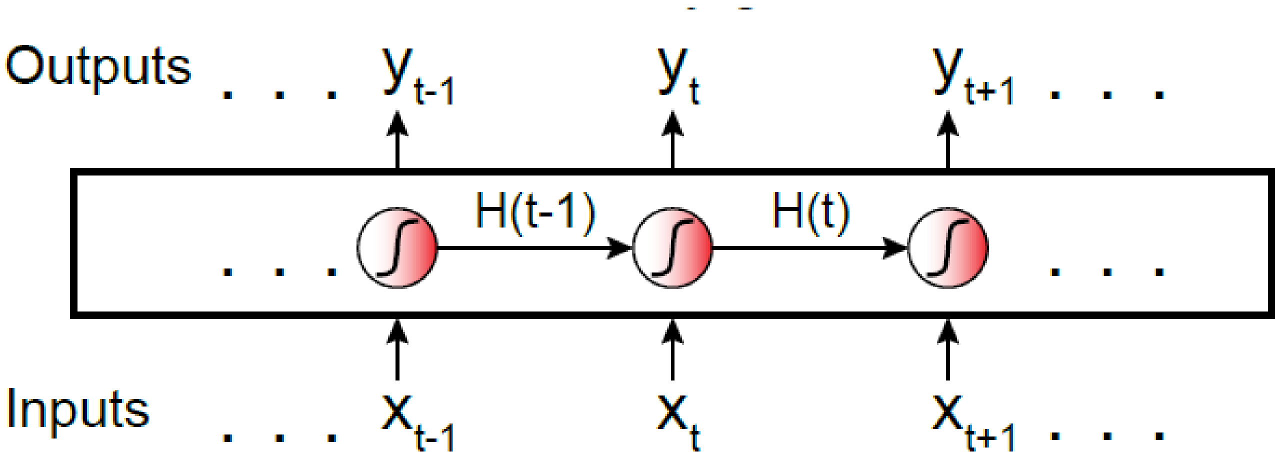
Enables the network to learn the progression of time (i=0) and periodic features of the data (through the function F)



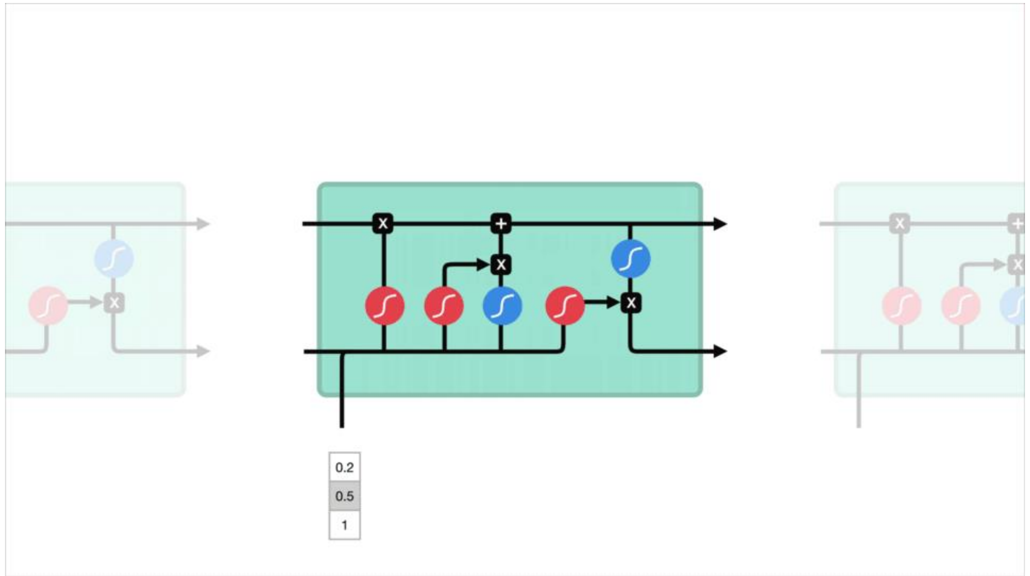
Other ways of encoding time in neural networks

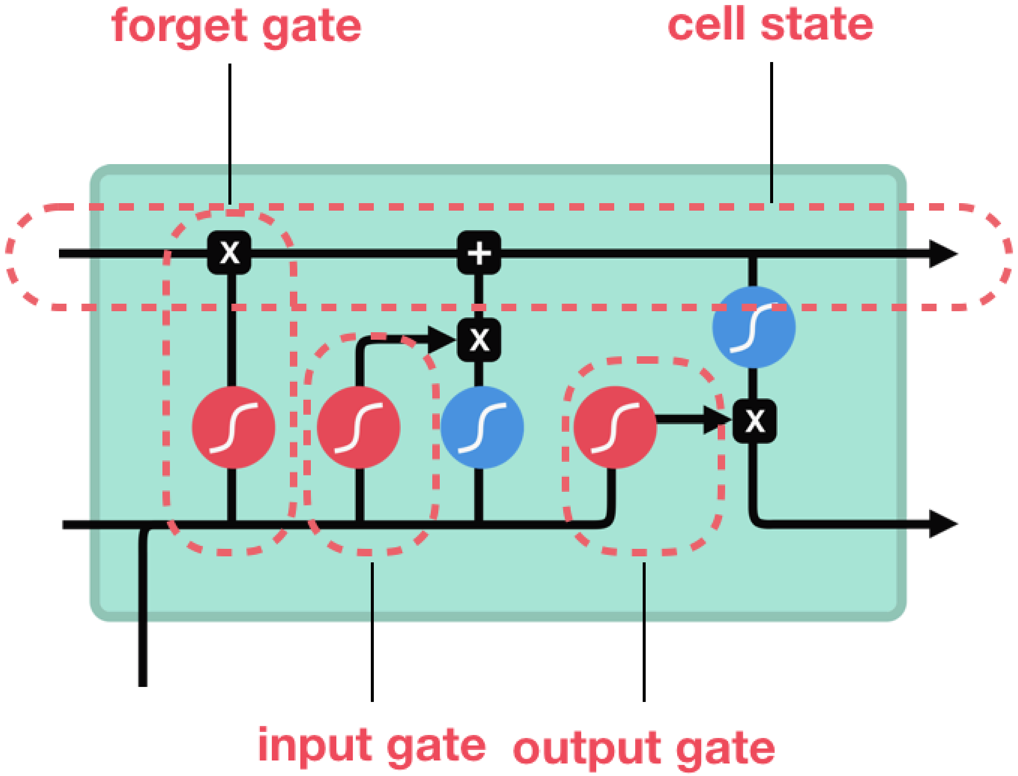
Memory in neural networks

- Data is analysed sequentially, and output from current step is used in subsequent steps
- Difficult to learn long range dependencies
- Vanishing/exploding gradient



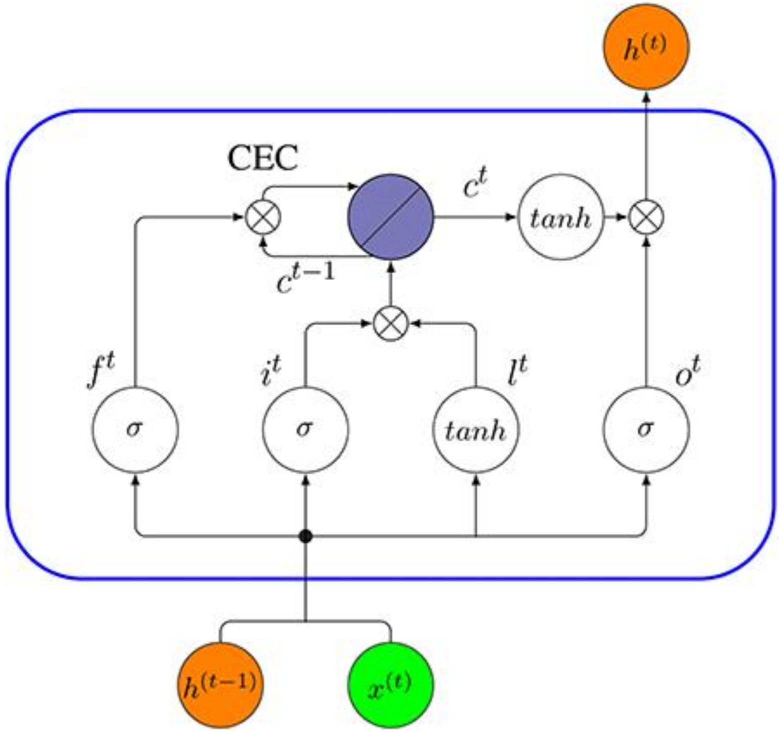
Long short term memory (LSTM)





Long short-term memory layers

- They work, but they take a long time to train
- Still difficult with long range dependencies



https://doi.org/10.3389/frai.2020.00004