Generative Networks

ACTL3143 & ACTL5111 Deep Learning for Actuaries

Patrick Laub

Lecture Outline

- **Text Generation**
- Sampling strategy
- Transformers
- Image Generation
- Neural style transfer
- Autoencoders
- Variational Autoencoders
- Diffusion Models

Generative deep learning

- Using AI as augmented intelligence rather than artificial intelligence.
- Use of deep learning to augment creative activities such as writing, music and art, to *generate* new things.
- Some applications: text generation, deep dreaming, neural style transfer, variational autoencoders and generative adversarial networks.

Text generation

> Generating sequential data is the closest computers get to dreaming.

- Generate sequence data: Train a model to predict the next token or next few tokens in a sentence, using previous tokens as input.
- A network that models the probability of the next tokens given the previous ones is called a *language model*.

Source: Alex Graves (2013), Generating Sequences With Recurrent Neural Networks

Word-level language model

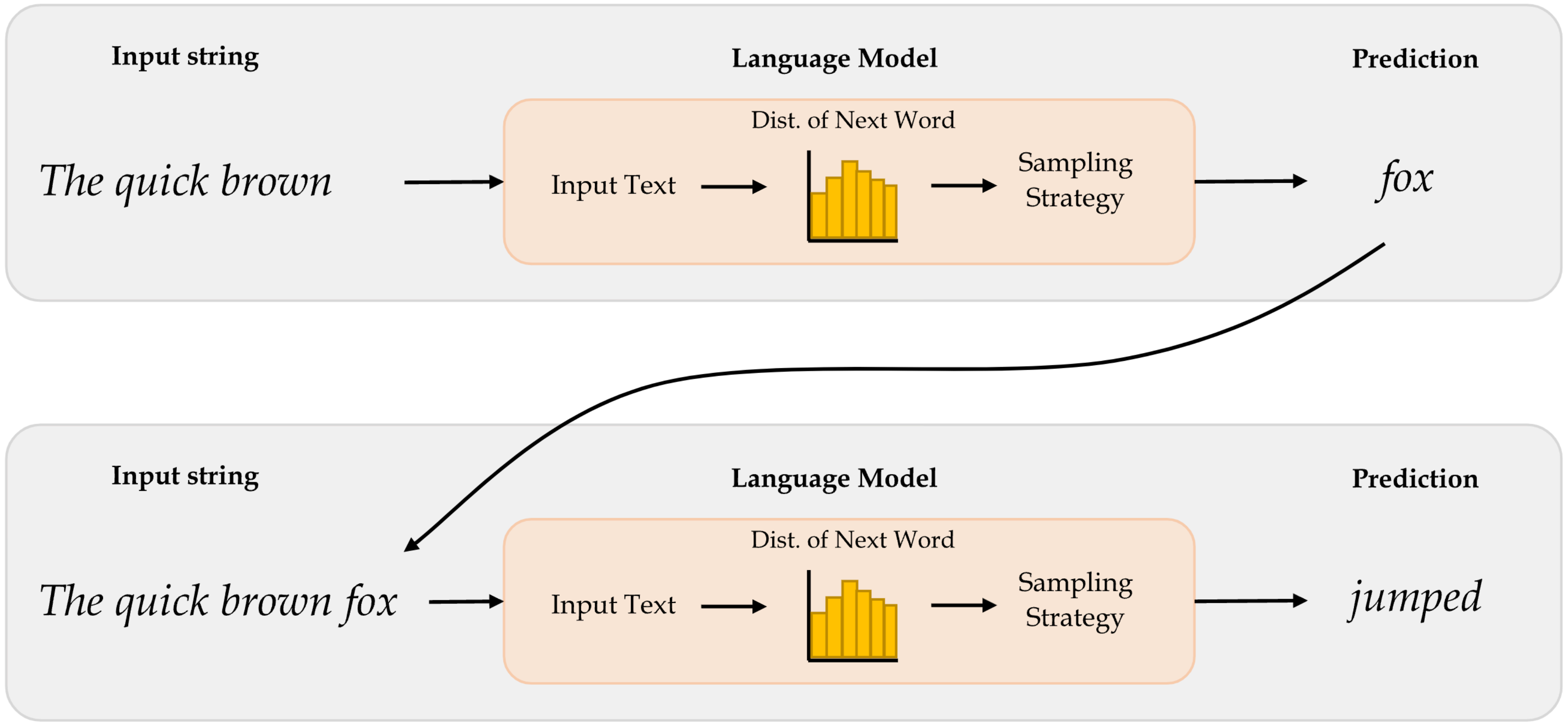


Diagram of a word-level language model.

Source: Marcus Lautier (2022).

Character-level language model

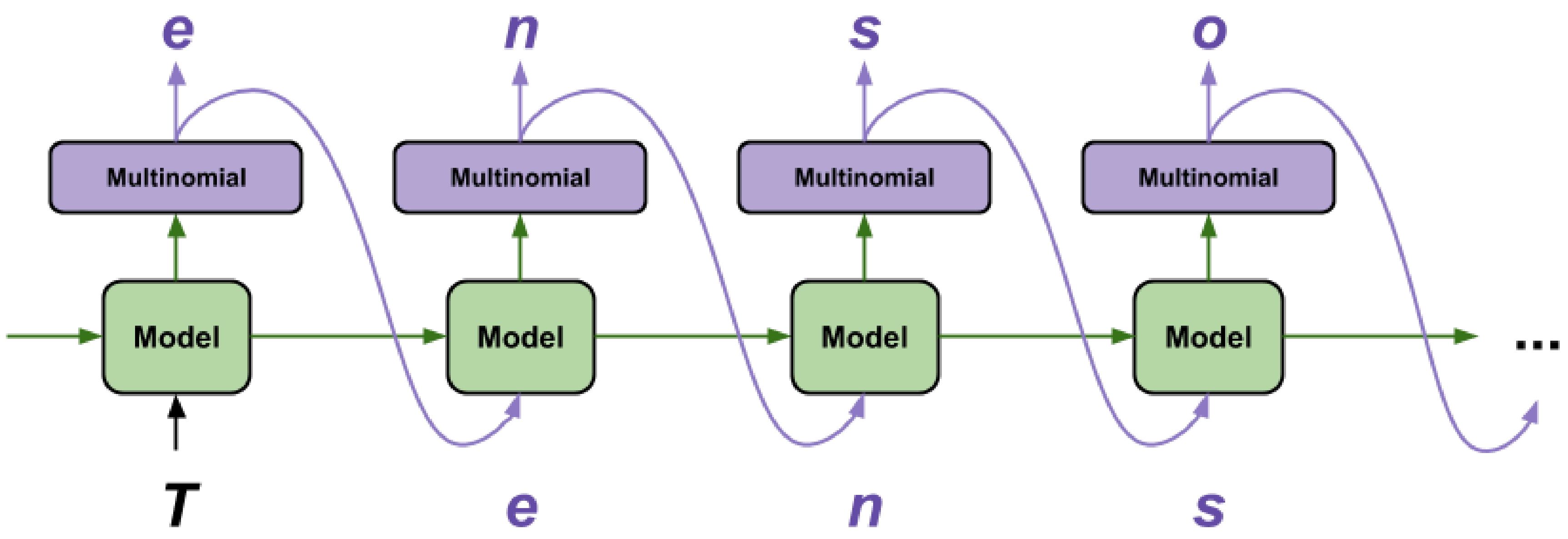


Diagram of a character-level language model (Char-RNN)

Source: Tensorflow tutorial, Text generation with an RNN.

Useful for speech recognition

RNN output	**Decoded Transcription**
what is the weather like in bostin right now	what is the weather like in boston right now
prime miniter nerenr modi	prime minister narendra modi
arther n tickets for the game	are there any tickets for the game

Figure 1: Examples of transcriptions directly from the RNN with errors that are fixed by addition of a language model.

Source: Hannun et al. (2014), Deep Speech: Scaling up end-to-end speech recognition, arXiv:1412.5567, Table 1.

Generating Shakespeare I

> ROMEO:
> Why, sir, what think you, sir?
>
> AUTOLYCUS:
> A dozen; shall I be deceased.
> The enemy is parting with your general,
> As bias should still combit them offend
> That Montague is as devotions that did satisfied;
> But not they are put your pleasure.

Source: Tensorflow tutorial, Text generation with an RNN.

Generating Shakespeare II

> DUKE OF YORK:
> Peace, sing! do you must be all the law;
> And overmuting Mercutio slain;
> And stand betide that blows which wretched shame;
> Which, I, that have been complaints me older hours.
>
> LUCENTIO:
> What, marry, may shame, the forish priest-lay estimest you, sir,
> Whom I will purchase with green limits o' the commons' ears!

Source: Tensorflow tutorial, Text generation with an RNN.

Generating Shakespeare III

> ANTIGONUS:
> To be by oath enjoin’d to this. Farewell!
> The day frowns more and more: thou’rt like to have
> A lullaby too rough: I never saw
> The heavens so dim by day. A savage clamour!
>
> [Exit, pursued by a bear]

Lecture Outline

- Text Generation
- **Sampling strategy**
- Transformers
- Image Generation
- Neural style transfer
- Autoencoders
- Variational Autoencoders
- Diffusion Models

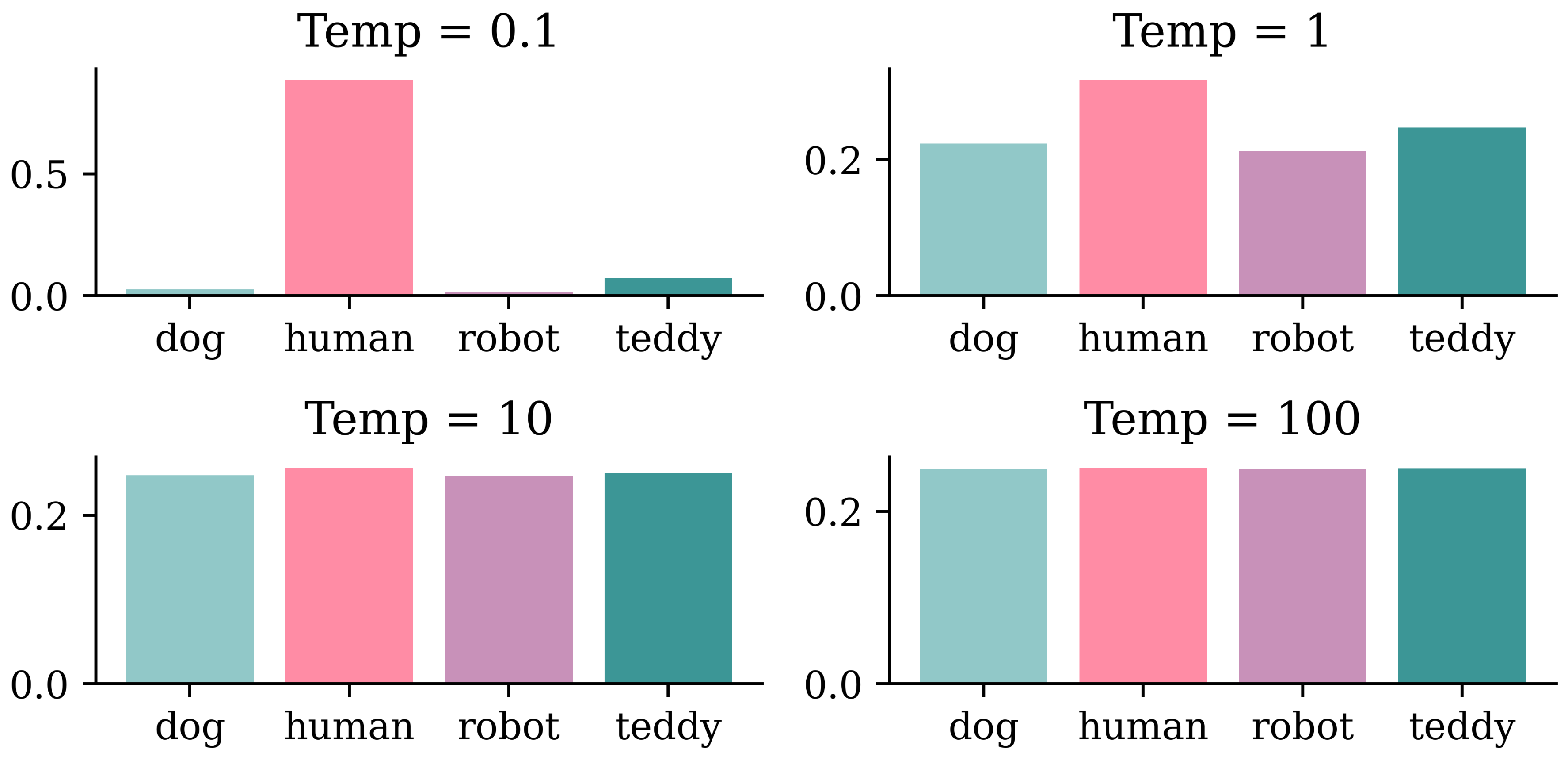
- *Greedy sampling* will choose the token with the highest probability. It makes the resulting sentence repetitive and predictable.
- *Stochastic sampling*: if a word has probability 0.3 of being next in the sentence according to the model, we'll choose it 30% of the time.

Softmax temperature

- The softmax temperature is a parameter that controls the randomness of the next token.
- The formula is:

$$\mathrm{softmax}_{\mathrm{temperature}}(x) = \frac{\exp(x/\mathrm{temperature})}{\sum_i \exp(x_i/\mathrm{temperature})}$$

“I am a” …


Temp = 0.1
0.5
0.0
dog
human
robot
teddy
Temp = 1
0.2
0.0
dog
human
robot
teddy
Temp = 10
0.2
0.0
dog
human
robot
teddy
Temp = 100
0.2
0.0
dog
human
robot
teddy

Generating Laub (temp = 0.01)

> *In today's lecture we will* be different situation. So, next one is what they rective that each commit to be able to learn some relationships from the course, and that is part of the image that it's very clese and black problems that you're trying to fit the neural network to do there instead of like a specific though shef series of layers mean about full of the chosen the baseline of car was in the right, but that's an important facts and it's a very small summary with very scrort by the beginning of the sentence.

UNSW SYDNEY

Generating Laub (temp = 0.25)

> *In today's lecture we will* decreas before model that we that we have to think about it, this mightsks better, for chattely the same project, because you might use the test set because it's to be picked up the things that I wanted to heard of things that I like that even real you and you're using the same thing again now because we need to understand what it's doing the same thing but instead of putting it in particular week, and we can say that's a thing I mainly link it's three columns.

Generating Laub (temp = 0.5)

> *In today's lecture we will* probably the adw n wait lots of ngobs teulagedation to calculate the gradient and then I'll be less than one layer the next slide will br input over and over the threshow you ampaigey the one that we want to apply them quickly. So, here this is the screen here the main top kecw onct three thing to told them, and the output is a vertical variables and Marceparase of things that you're moving the blurring and that just data set is to maybe kind of categorical variants here but there's more efficiently not basically replace that with respect to the best and be the same thing.

Generating Laub (temp = 1)

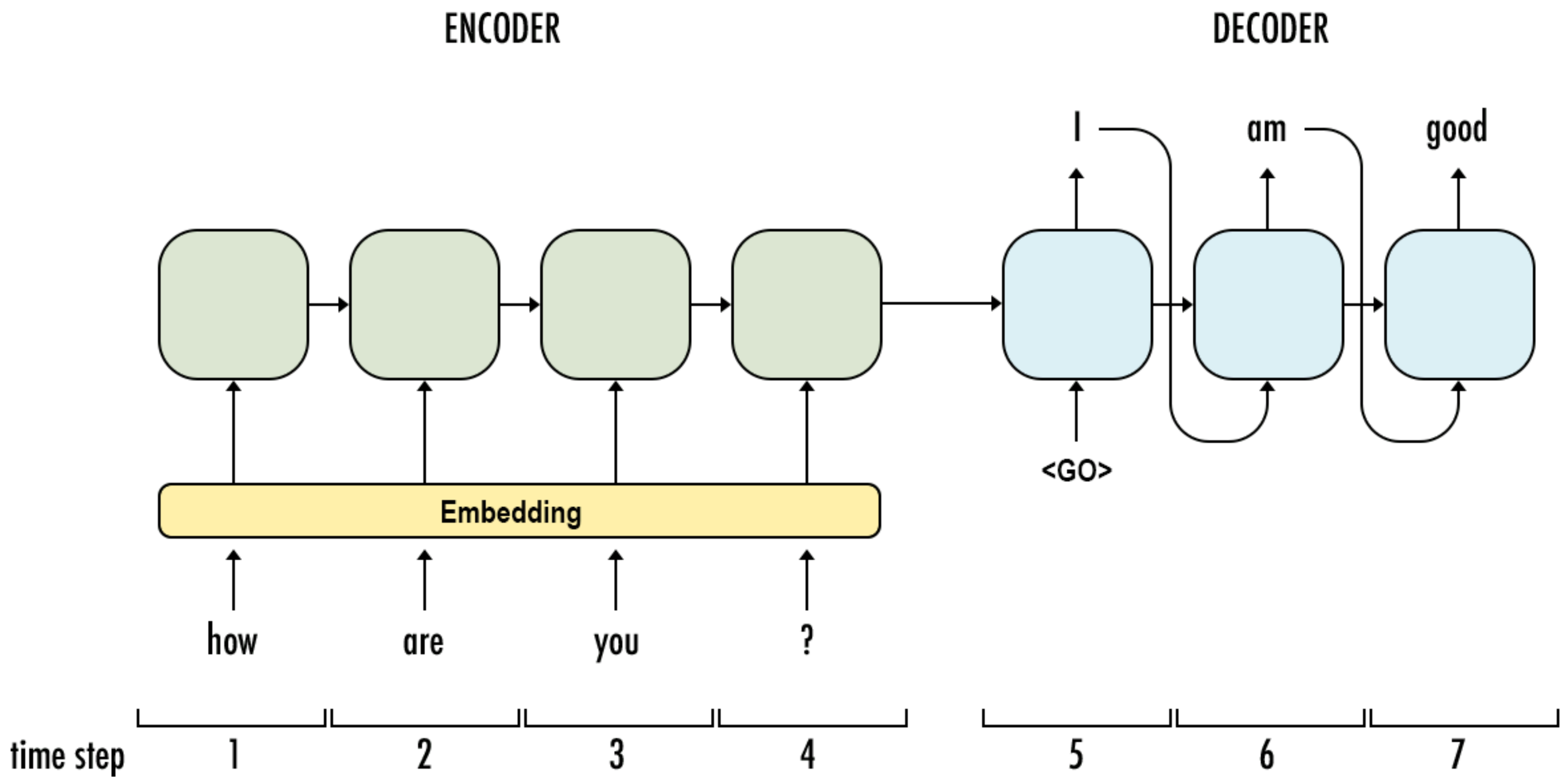
> *In today's lecture we will* put it different shates to touch on last week, so I want to ask what are you object frod current. They don't have any zero into it, things like that which mistakes. 10 claims that the average version was relden distever ditgs and Python for the whole term wo long right to really. The name of these two options. There are in that seems to be modified version. If you look at when you're putting numbers into your, that that's over. And I went backwards, up, if they'rina functional pricing working with.

UNSW
SYDNEY

Generating Laub (temp = 1.5)

> *In today's lecture we will* put it could be bedinnth. Lowerstoriage nruron. So rochain the everything that I just sGiming. If there was a large. It's gonua draltionation. Tow many, up, would that black and 53% that's girter thankAty will get you jast typically stickK thing. But maybe. Anyway, I'm going to work on this libry two, past, at shit citcs jast pleming to memorize overcamples like pre pysing, why wareed to smart a one in this reportbryeccuriay.

Generate the most likely sequence



An example sequence-to-sequence chatbot model.

Source: Payne (2021), What is beam search, Width.ai blog.

Beam search

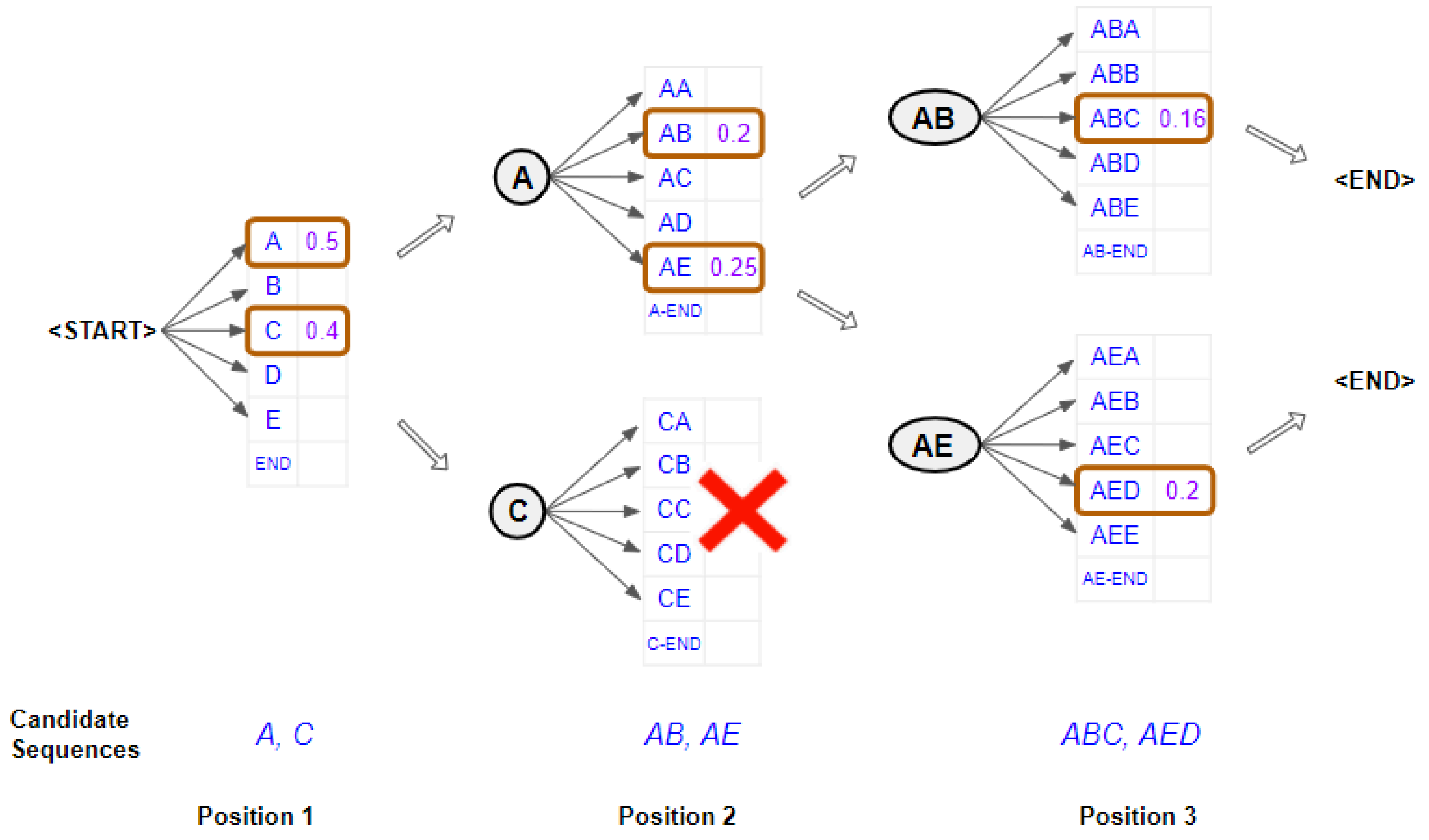


Illustration of a beam search.

Source: Doshi (2021), Foundations of NLP Explained Visually: Beam Search, How It Works, towardsdatascience.com.

Lecture Outline

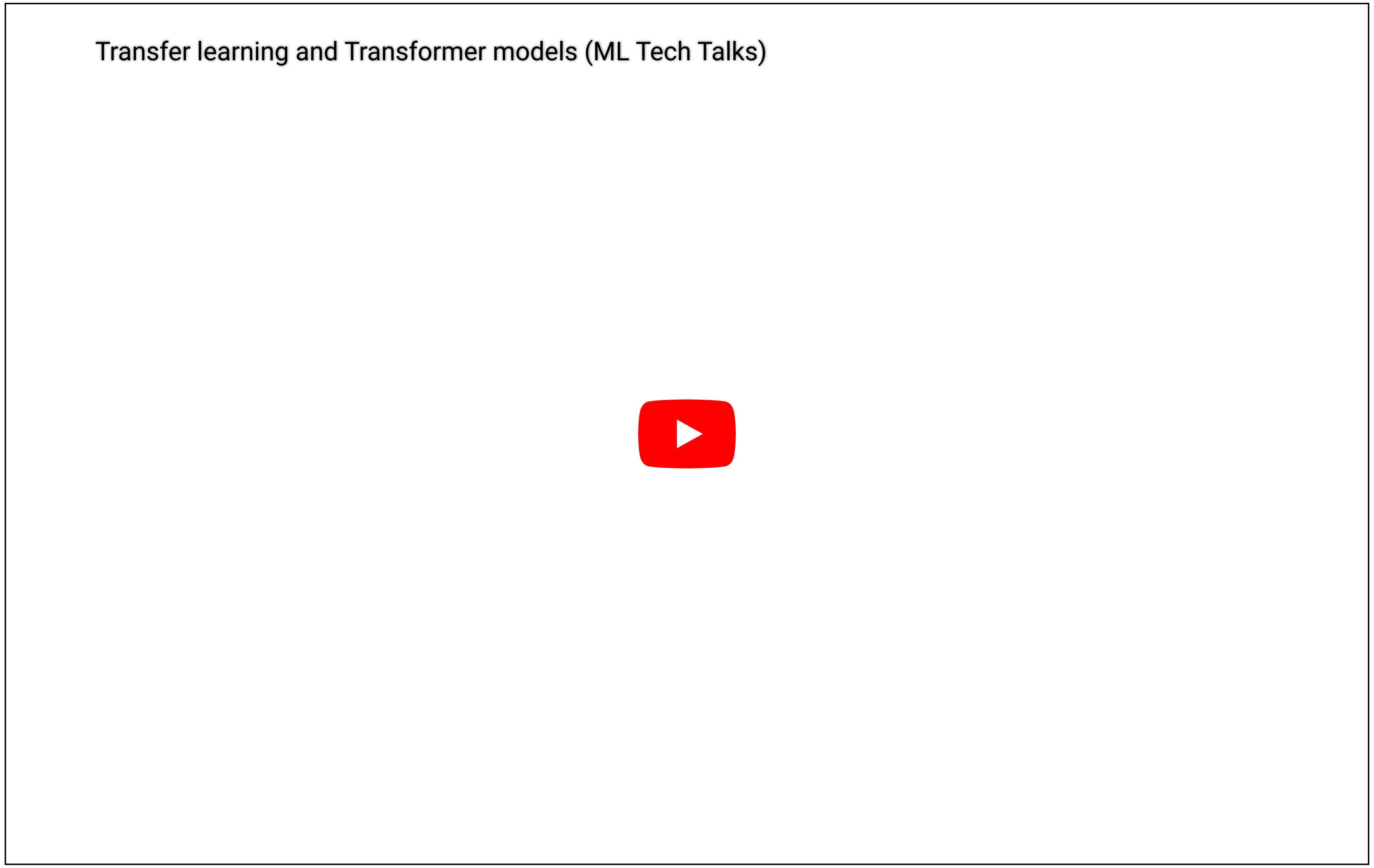
- Text Generation
- Sampling strategy
- **Transformers**
- Image Generation
- Neural style transfer
- Autoencoders
- Variational Autoencoders
- Diffusion Models

Transformer architecture

> GPT makes use of a mechanism known as attention, which removes the need for recurrent layers (e.g., LSTMs). It works like an information retrieval system, utilizing queries, keys, and values to decide how much information it wants to extract from each input token.
>
> Attention heads can be grouped together to form what is known as a multihead attention layer. These are then wrapped up inside a Transformer block, which includes layer normalization and skip connections around the attention layer. Transformer blocks can be stacked to create very deep neural networks.

Source: David Foster (2023), Generative Deep Learning, 2nd Edition, O'Reilly Media, Chapter 9.

Transformer architecture reference


Transfer learning and Transformer models (ML Tech Talks)

🤗 Transformers package

```python
import transformers
from transformers import pipeline
generator = pipeline(task="text-generation", model="gpt2", revision="6c0e608")
```

```python
transformers.set_seed(1)
print(generator("It's the holidays so I'm going to enjoy")[0]["generated_text"])
```

```
It's the holidays so I'm going to enjoy the rest of the time and look forward to this week
with new friends!"
```

```python
transformers.set_seed(2)
print(generator("It's the holidays so I'm going to enjoy")[0]["generated_text"])
```

```
It's the holidays so I'm going to enjoy them again."

I'm not making any assumptions.

I am very happy. It's the holidays so I'm going to enjoy them again. — Daniel Nasser
(@DanielN
```

Reading the course profile

```
context = """
StoryWall Formative Discussions: An initial StoryWall, worth 2%, is due by noon on June
The project will be submitted in stages: draft due at noon on July 1 (10%), recorded pre

As a student at UNSW you are expected to display academic integrity in your work and int
To assist you in understanding what academic integrity means, and how to ensure that you

StoryWall (30%)

The StoryWall format will be used for small weekly questions. Each week of questions wil

Project (40%)

Over the term, students will complete an individual project. There will be a selection o

The deliverables for the project will include: a draft/progress report mid-way through t

Exam (30%)

The exam will test the concepts presented in the lectures. For example, students will be
"""
```

Question answering

```
qa = pipeline("question-answering", model="distilbert-base-cased-distilled-squad", revis
```

```
qa(question="What weight is the exam?", context=context)
```

```
{'score': 0.501968264579773, 'start': 2092, 'end': 2095, 'answer': '30%'}
```

```
qa(question="What topics are in the exam?", context=context)
```

```
{'score': 0.2127578854560852,
 'start': 1778,
 'end': 1791,
 'answer': 'deep learning'}
```

```
qa(question="When is the presentation due?", context=context)
```

```
{'score': 0.5296494364738464,
 'start': 1319,
 'end': 1335,
 'answer': 'Monday at midday'}
```

```
qa(question="How many StoryWall tasks are there?", context=context)
```

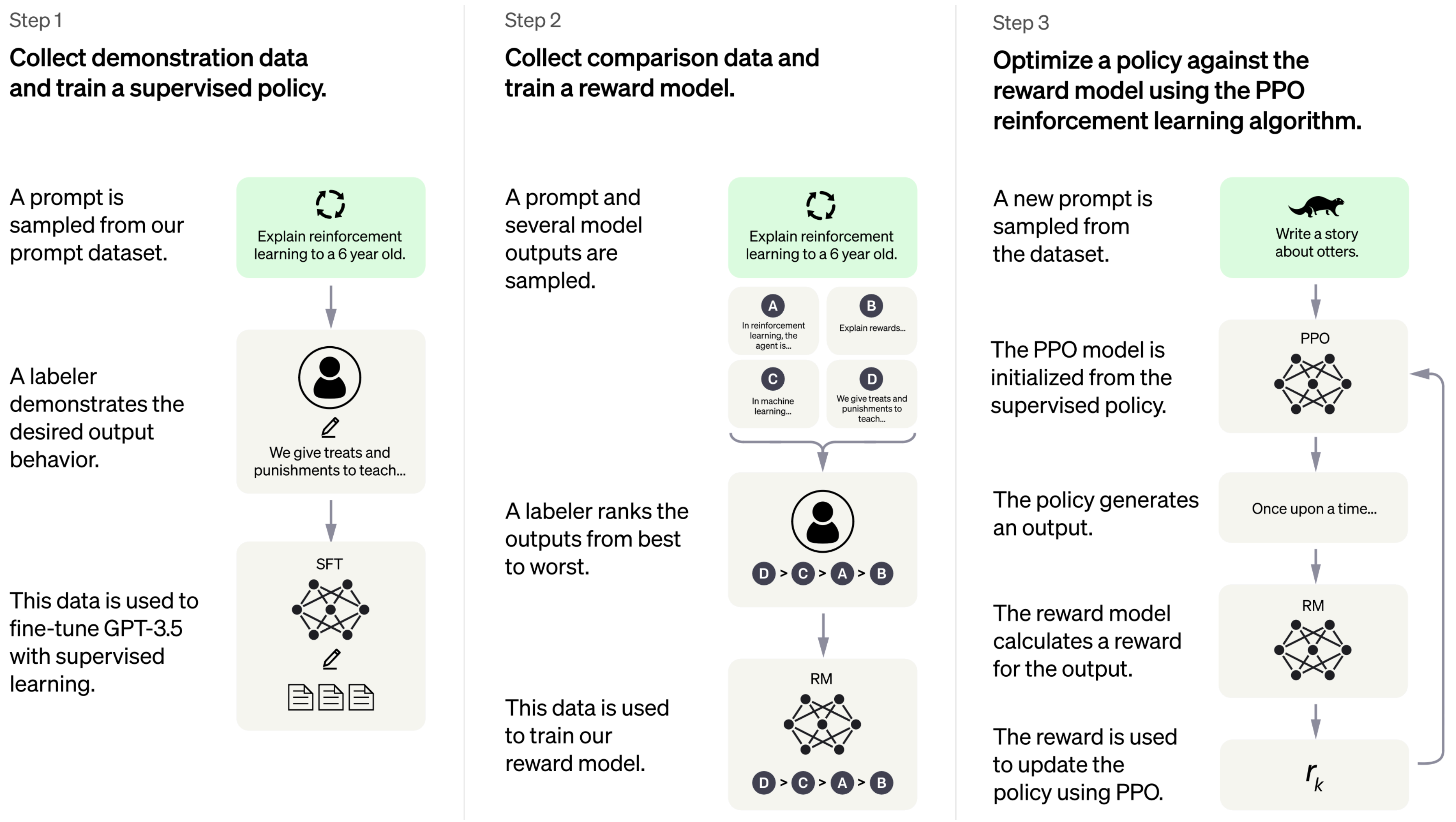
```
{'score': 0.21390870213508606, 'start': 1155, 'end': 1158, 'answer': '30%'}
```

ChatGPT is Transformer + RLHF

> At the time of writing, there is no official paper that describes how ChatGPT works in detail, but from the official blog post we know that it uses a technique called reinforcement learning from human feedback (RLHF) to fine-tune the GPT-3.5 model.

Source: David Foster (2023), Generative Deep Learning, 2nd Edition, O'Reilly Media, Chapter 9.

ChatGPT internals



Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

Explain reinforcement learning to a 6 year old.
We give treats and punishments to teach...
SFT
Explain reinforcement learning to a 6 year old.
A In reinforcement learning, the agent is...
B Explain rewards...
C In machine learning...
D We give treats and punishments to teach...
D > C > A > B
RM
D > C > A > B
Write a story about otters.
PPO
Once upon a time...
RM
r_k

It uses a fair bit of human feedback

Source: OpenAI blog.

ChatGPT

> While ChatGPT still has many limitations (such as sometimes "hallucinating" factually incorrect information), it is a powerful example of how Transformers can be used to build generative models that can produce complex, long-ranging, and novel output that is often indistinguishable from human-generated text. The progress made thus far by models like ChatGPT serves as a testament to the potential of AI and its transformative impact on the world.

Source: David Foster (2023), Generative Deep Learning, 2nd Edition, O'Reilly Media, Chapter 9.

Recommended reading

- The Verge (2022), The Great Fiction of AI: The strange world of high-speed semi-automated genre fiction
- Vaswani et al. (2017), Attention Is All You Need, NeurIPS
- Bommasani et al. (2021), On the Opportunities and Risks of Foundation Models
- Gary Marcus (2022), Deep Learning Is Hitting a Wall, Nautilus article
- SDS 564, Clem Delangue on Hugging Face and Transformers
- SDS 559, GPT-3 for Natural Language Processing
- Computerphile (2019), AI Language Models & Transformers (20m)
- Computerphile (2020), GPT3: An Even Bigger Language Model (25m)
- Nicholas Renotte (2021), AI Blog Post Summarization with Hugging Face Transformers… (33m)
- Seattle Applied Deep Learning (2019), LSTM is dead. Long Live Transformers! (28m)

Lecture Outline

- Text Generation
- Sampling strategy
- Transformers
- **Image Generation**
- Neural style transfer
- Autoencoders
- Variational Autoencoders
- Diffusion Models

Reverse-engineering a CNN

A CNN is a function $f_{\boldsymbol{\theta}}(\mathbf{x})$ that takes a vector (image) $\mathbf{x}$ and returns a vector (distribution) $\widehat{\mathbf{y}}$.

Normally, we train it by modifying $\boldsymbol{\theta}$ so that

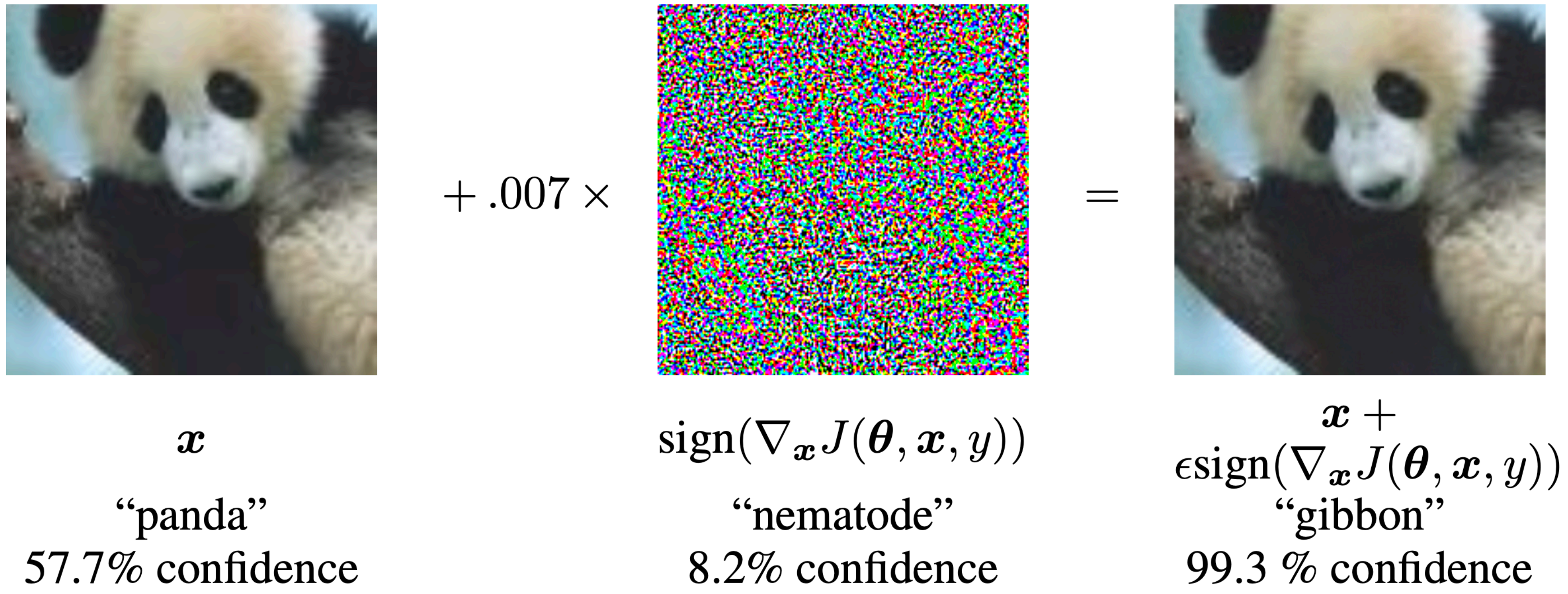
$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \operatorname{Loss}\big(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}\big).$$

However, it is possible to *not train* the network but to modify $\mathbf{x}$, like

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \operatorname{Loss}\big(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}\big).$$

This is very slow as we do gradient descent every single time.

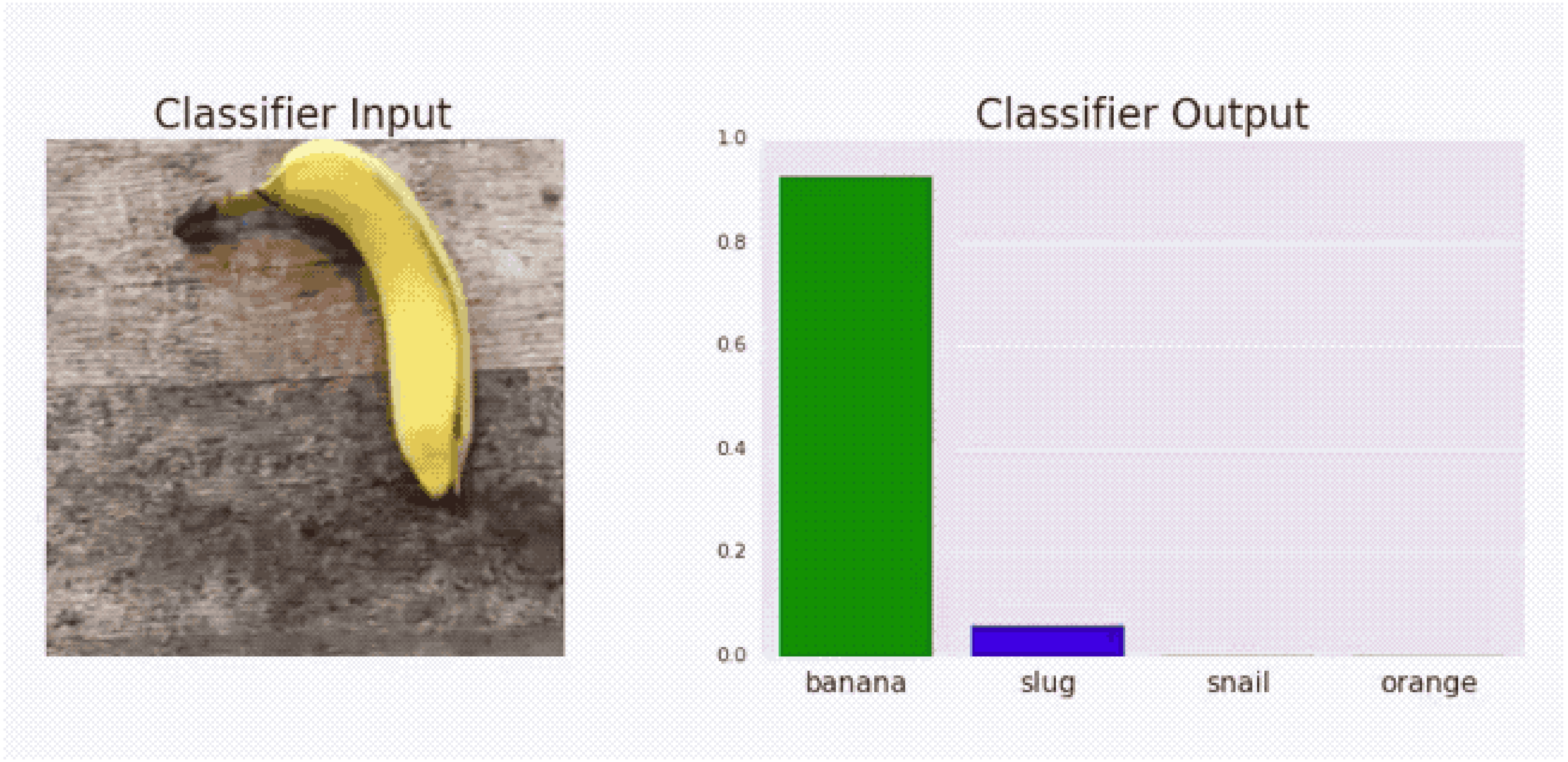
Adversarial examples


$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

A demonstration of fast adversarial example generation applied to GoogLeNet on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image.

Source: Goodfellow et al. (2015), Explaining and Harnessing Adversarial Examples, ICLR.

Adversarial stickers



Adversarial stickers.

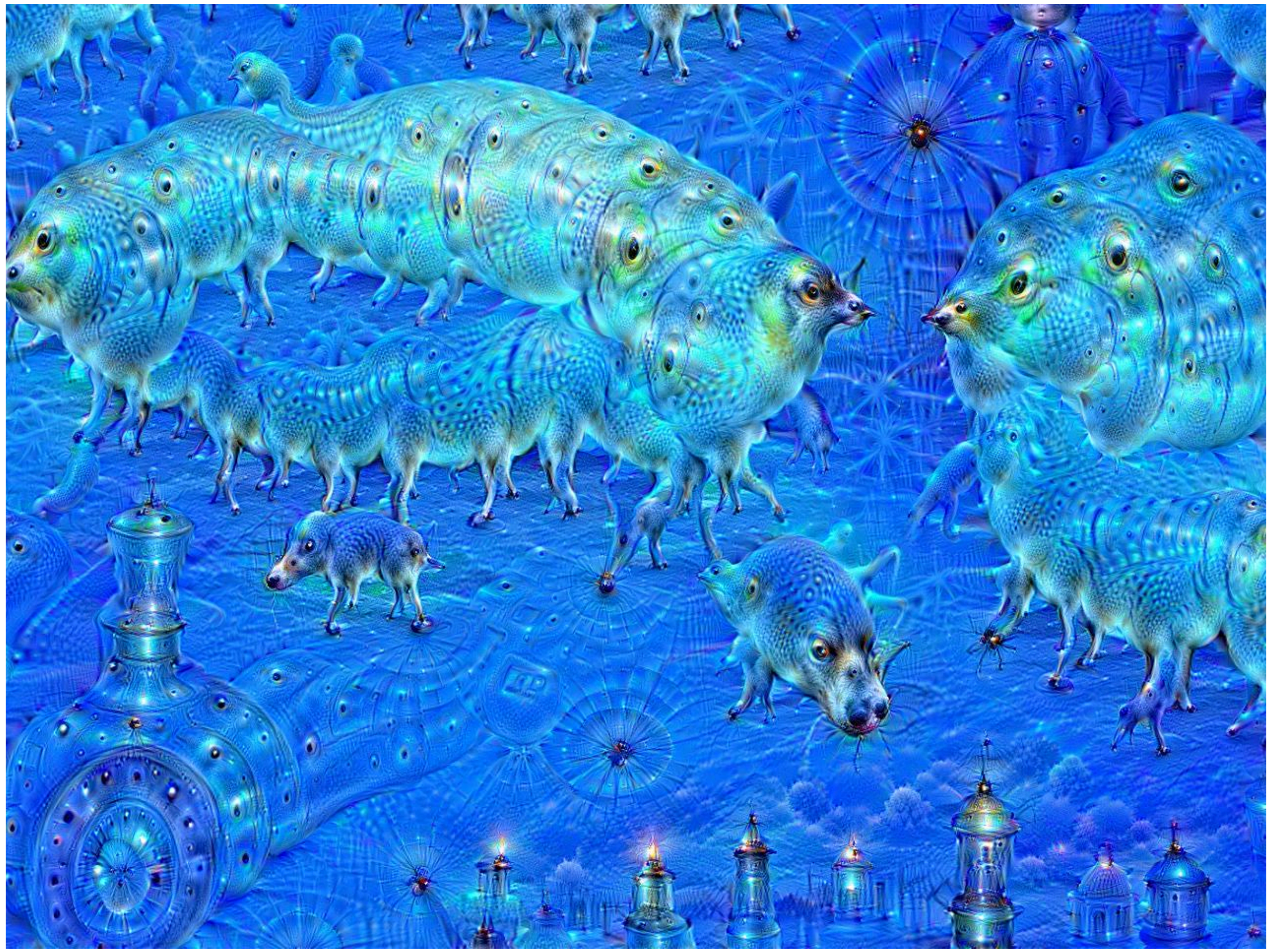
Source: The Verge (2018), These stickers make computer vision software hallucinate things that aren't there.

Adversarial text

"TextAttack 🐙 is a Python framework for adversarial attacks, data augmentation, and model training in NLP"

Demo

Deep Dream



Deep Dream is an image-modification program released by Google in 2015.

Source: Wikipedia, DeepDream page.

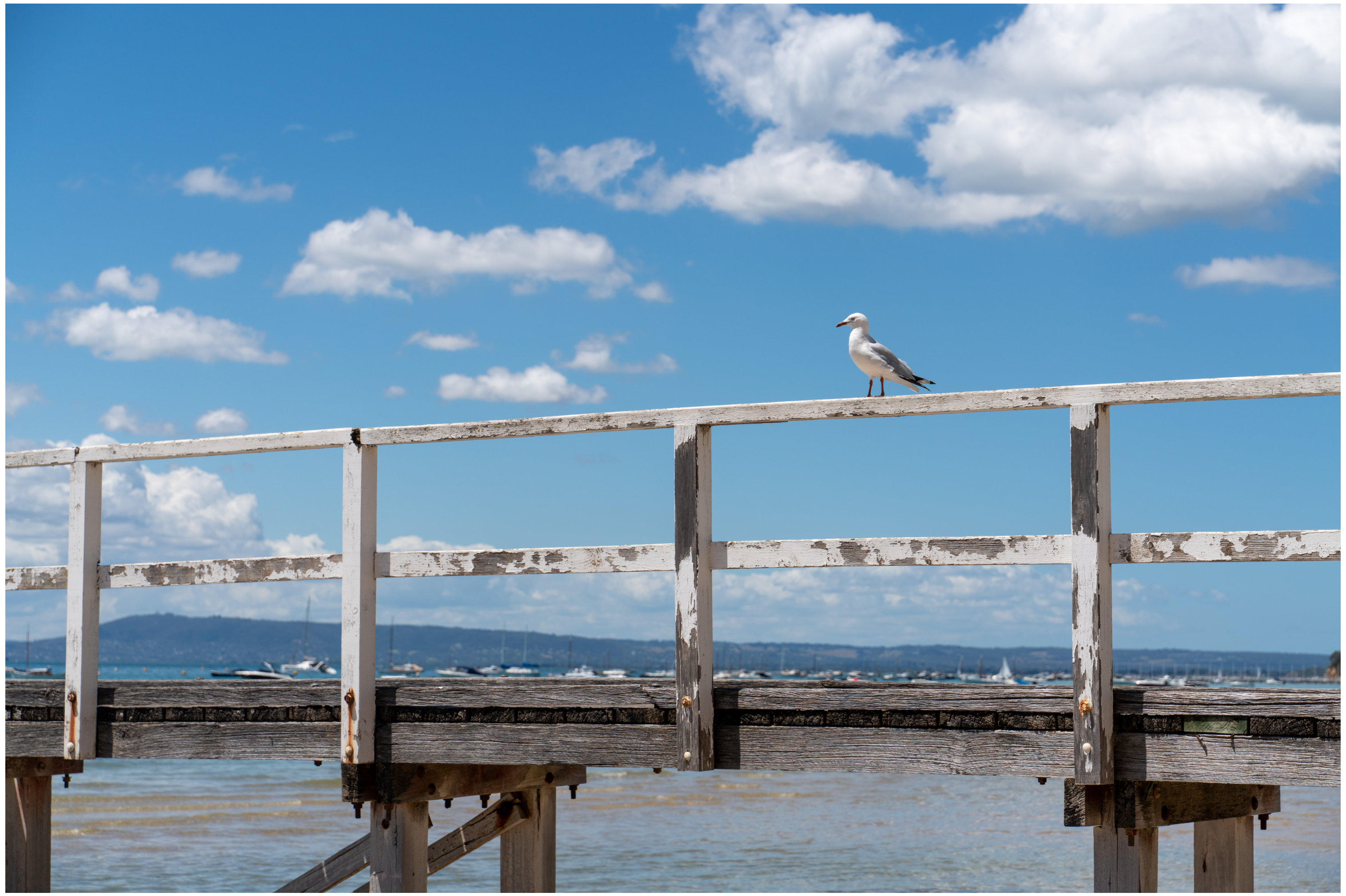
DeepDream

- Even though many deep learning models are black boxes, convnets are quite interpretable via visualization. Some visualization techniques are: visualizing convnet outputs shows how convnet layers transform the input, visualizing convnet filters shows what visual patterns or concept each filter is receptive to, etc.
- The activations of the first few layers of the network carries more information about the visual contents, while deeper layers encode higher, more abstract concepts.

DeepDream

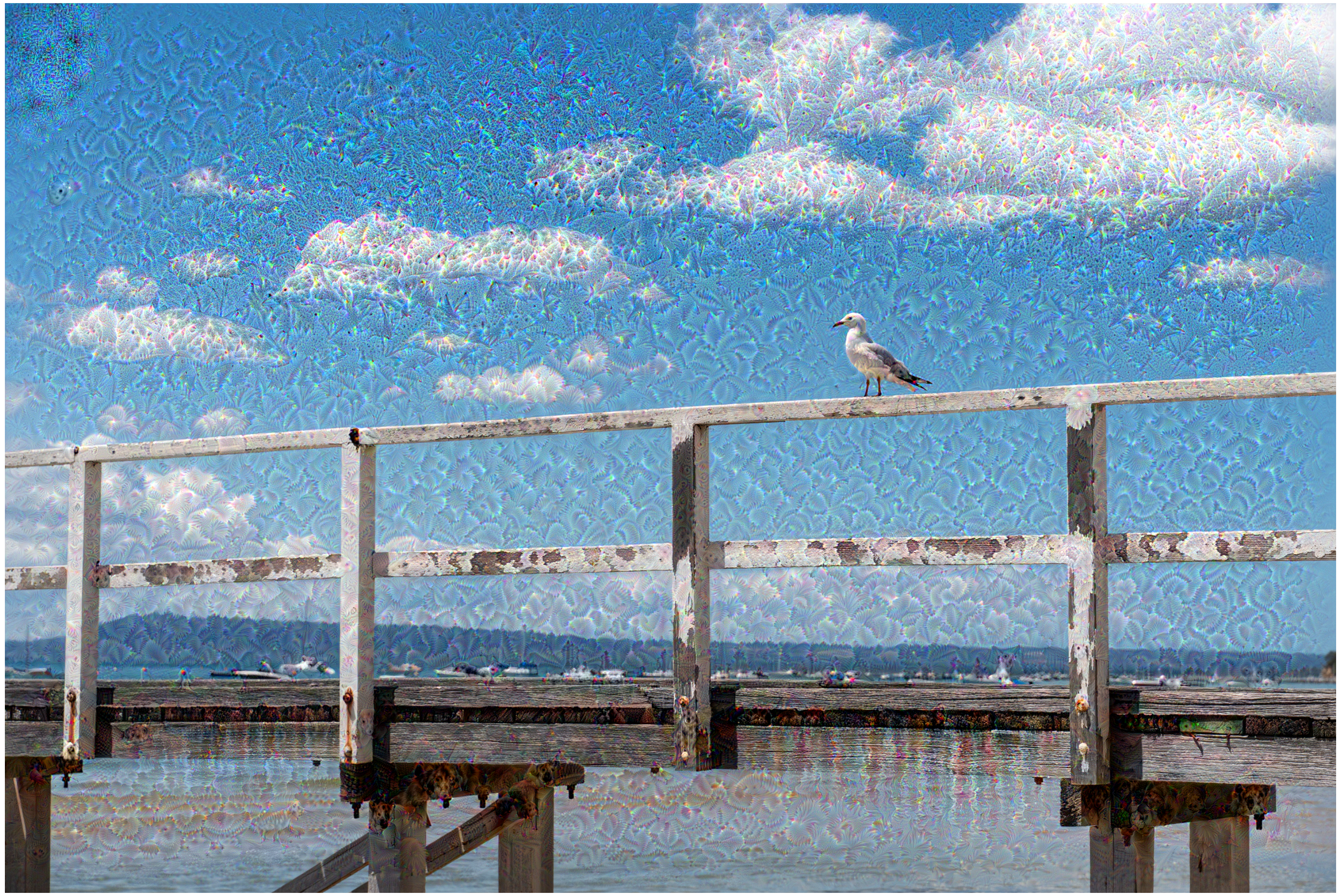
- Each filter is receptive to a visual pattern. To visualize a convnet filter, gradient ascent is used to maximize the response of the filter. Gradient ascent maximize a loss function and moves the image in a direction that activate the filter more strongly to enhance its reading of the visual pattern.
- DeepDream maximizes the activation of the entire convnet layer rather than that of a specific filter, thus mixing together many visual patterns all at once.
- DeepDream starts with an existing image, latches on to preexisting visual patterns, distorting elements of the image in a somewhat artistic fashion.

Original



A sunny day on the Mornington peninsula.

Transformed



Deep-dreaming version.

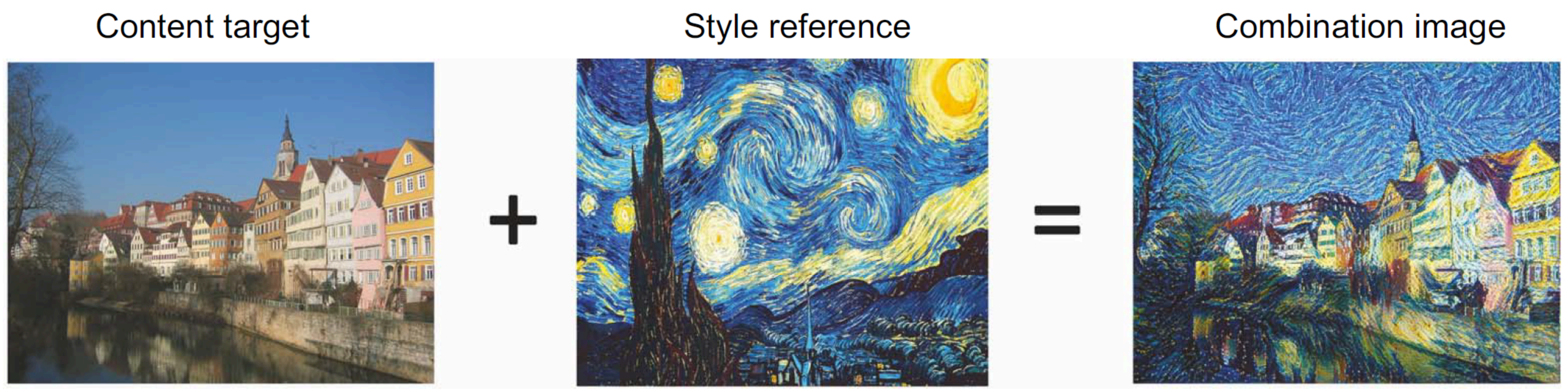
Generated by Keras' Deep Dream tutorial.

Lecture Outline

- Text Generation
- Sampling strategy
- Transformers
- Image Generation
- **Neural style transfer**
- Autoencoders
- Variational Autoencoders
- Diffusion Models

Neural style transfer

Applying the style of a reference image to a target image while conserving the content of the target image.



An example neural style transfer.

Goal of NST

What the model does:

- Preserve content by maintaining similar deeper layer activations between the original image and the generated image. The convnet should “see” both the original image and the generated image as containing the same things.
- Preserve style by maintaining similar correlations within activations for both low level layers and high-level layers. Feature correlations within a layer capture textures: the generated image and the style-reference image should share the same textures at different spatial scales.

A wanderer in Greenland

Content

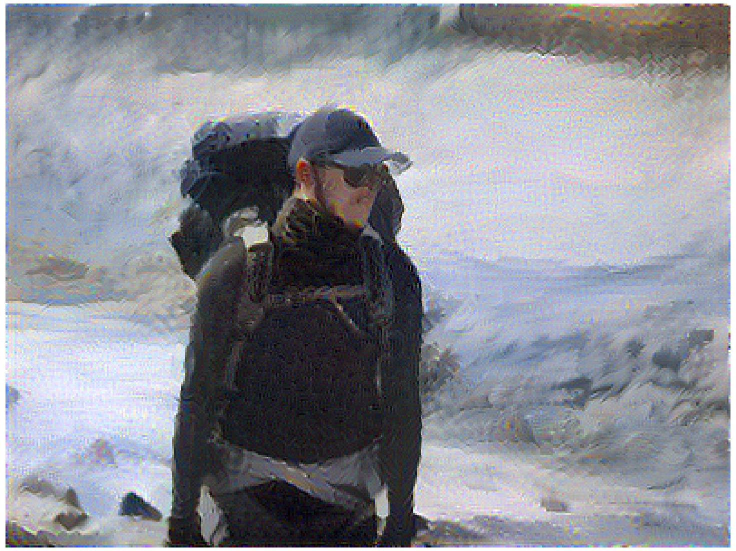
Some striking young hiker in Greenland.

Style

Wanderer above the Sea of Fog by Caspar David Friedrich.

A wanderer in Greenland II

Animation of NST in progress.

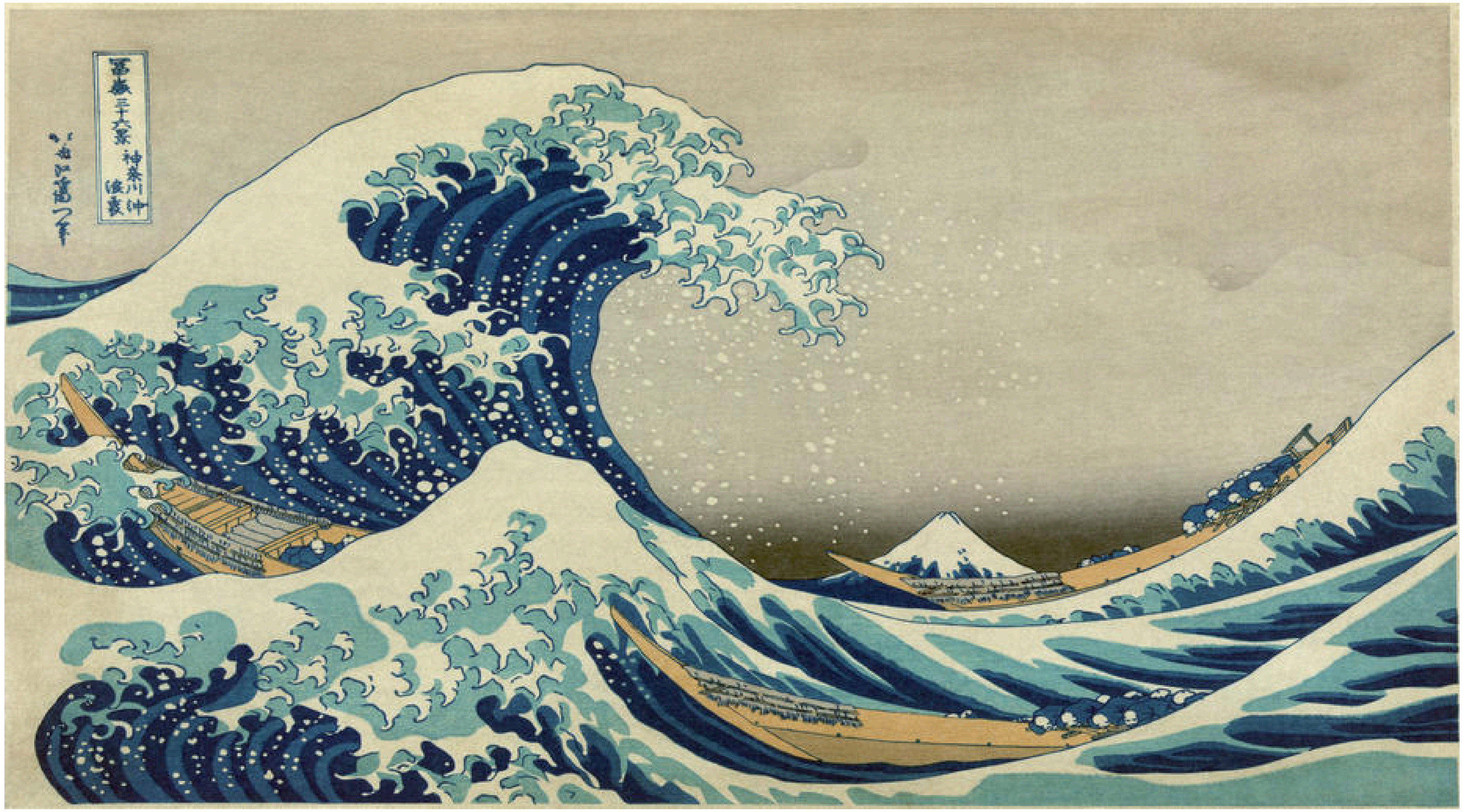


One result of NST.

Question

How would you make this faster for one specific style image?

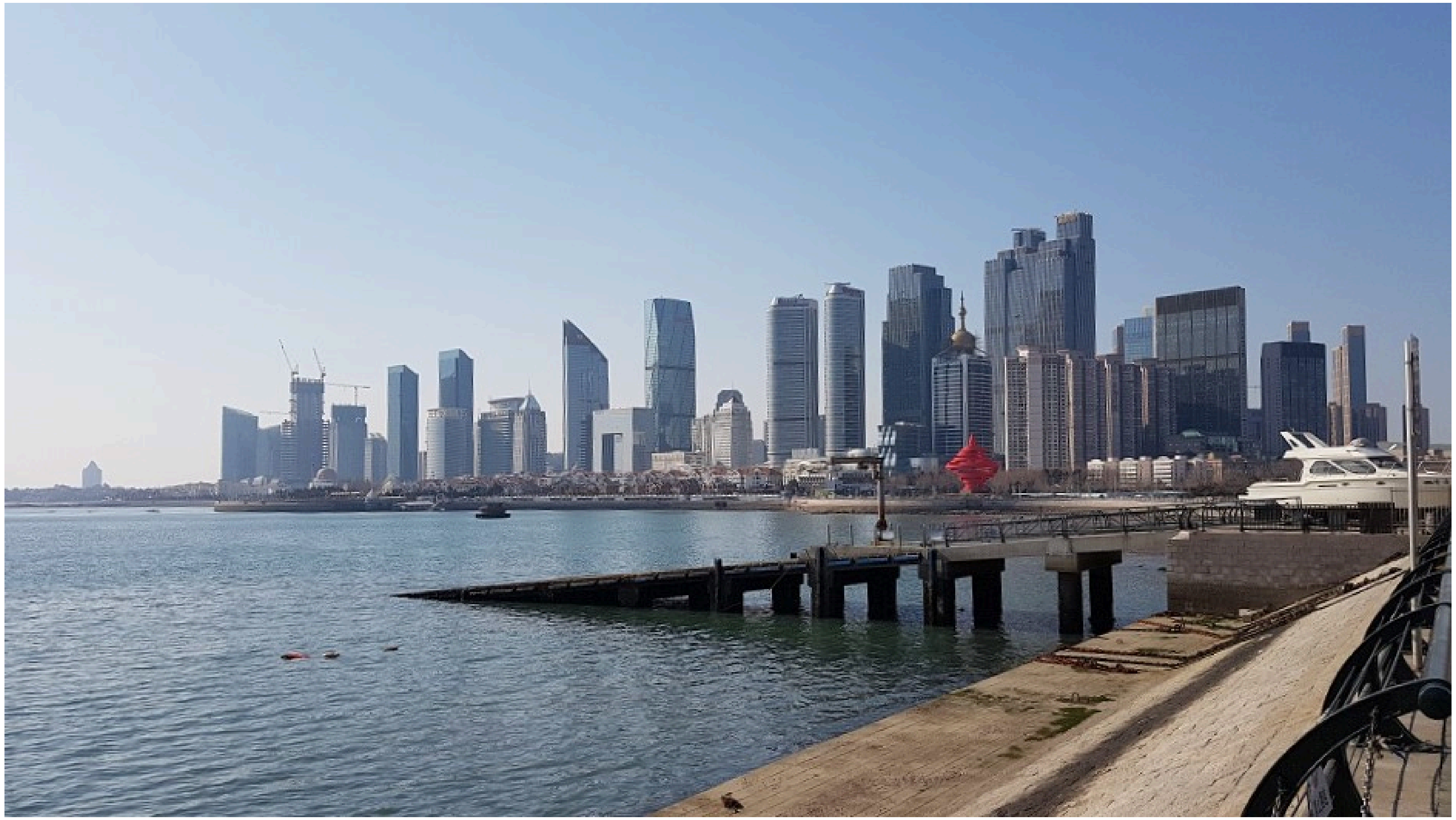
A new style image



Hokusai's Great Wave off Kanagawa

Source: Laub (2018), On Neural Style Transfer, Blog post.

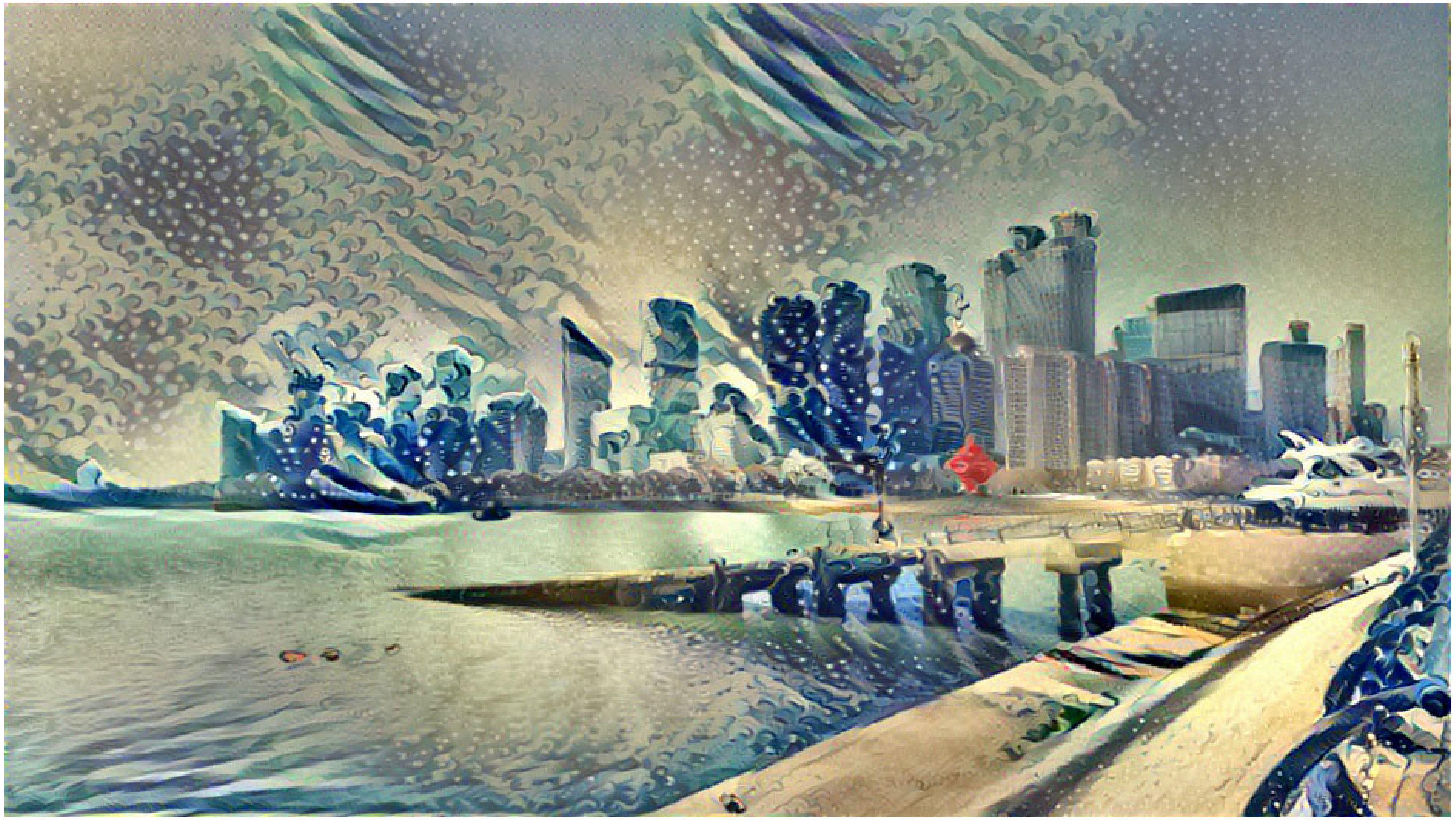
A new content image



The seascape in Qingdao

Source: Laub (2018), On Neural Style Transfer, Blog post.

Another neural style transfer



The seascape in Qingdao in the style of Hokusai's Great Wave off Kanagawa

Source: Laub (2018), On Neural Style Transfer, Blog post.

Why is this important?

Taking derivatives with respect to the input image can be a first step toward explainable AI for convolutional networks.

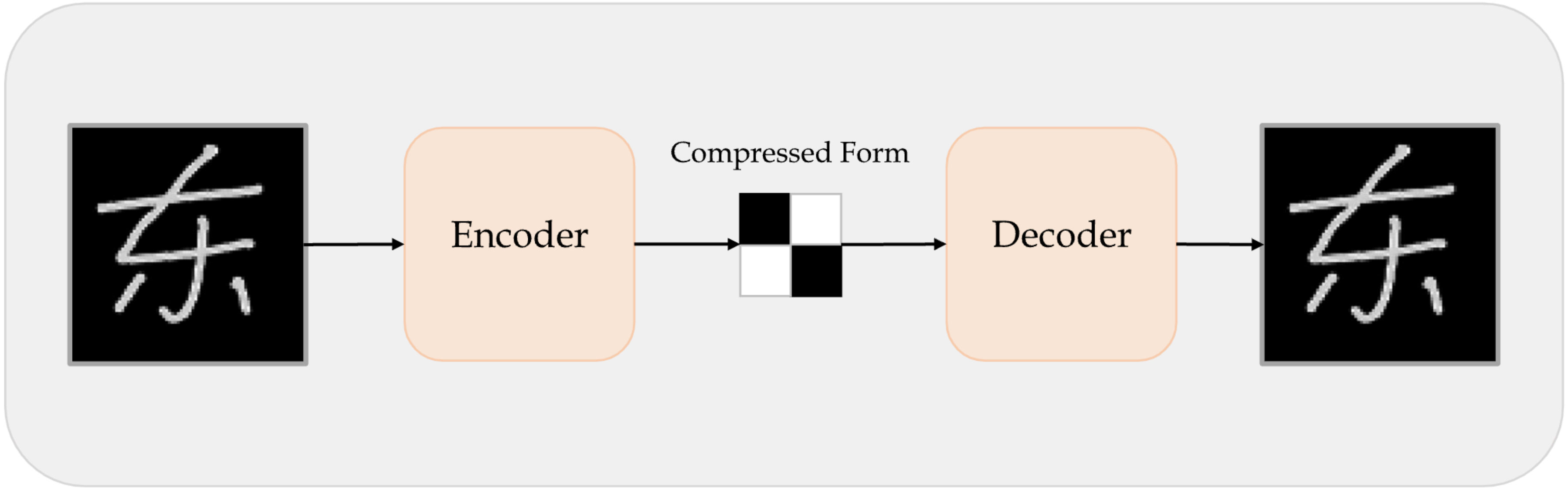
- Saliency maps
- Grad-CAM

Lecture Outline

- Text Generation
- Sampling strategy
- Transformers
- Image Generation
- Neural style transfer
- **Autoencoders**
- Variational Autoencoders
- Diffusion Models

Autoencoder

An autoencoder takes a data/image, maps it to a latent space via an encoder module, then decodes it back to an output with the same dimensions via a decoder module.



Schematic of an autoencoder.

Source: Marcus Lautier (2022).

Autoencoder II

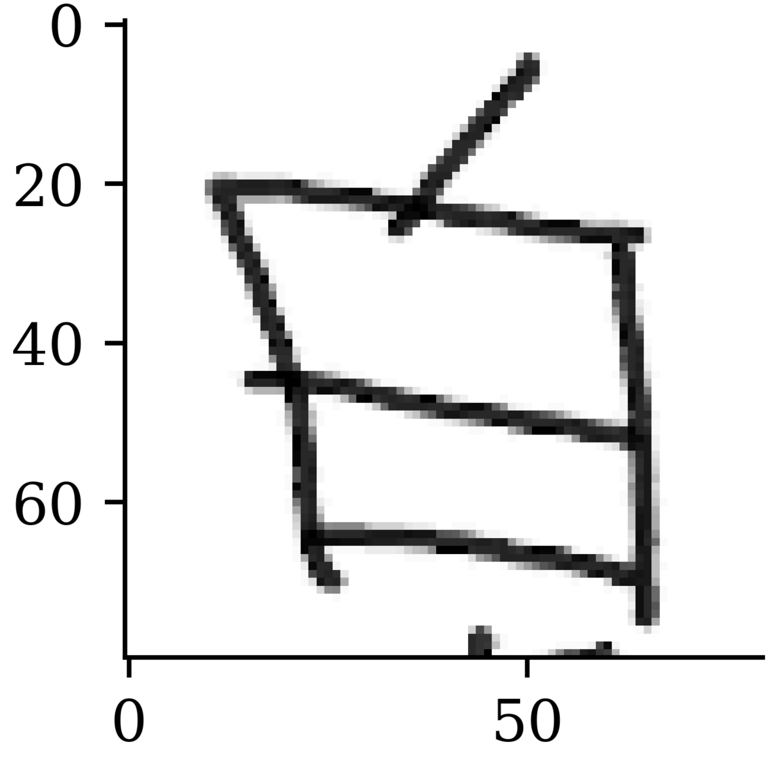
- An autoencoder is trained by using the same image as both the input and the target, meaning an autoencoder learns to reconstruct the original inputs. Therefore it's *not supervised learning*, but *self-supervised learning*.
- If we impose constraints on the encoders to be low-dimensional and sparse, *the input data will be compressed* into fewer bits of information.
- Latent space is a place that stores low-dimensional representation of data. It can be used for *data compression*, where data is compressed to a point in a latent space.
- An image can be compressed into a latent representation, which can then be reconstructed back to a *slightly different image*.

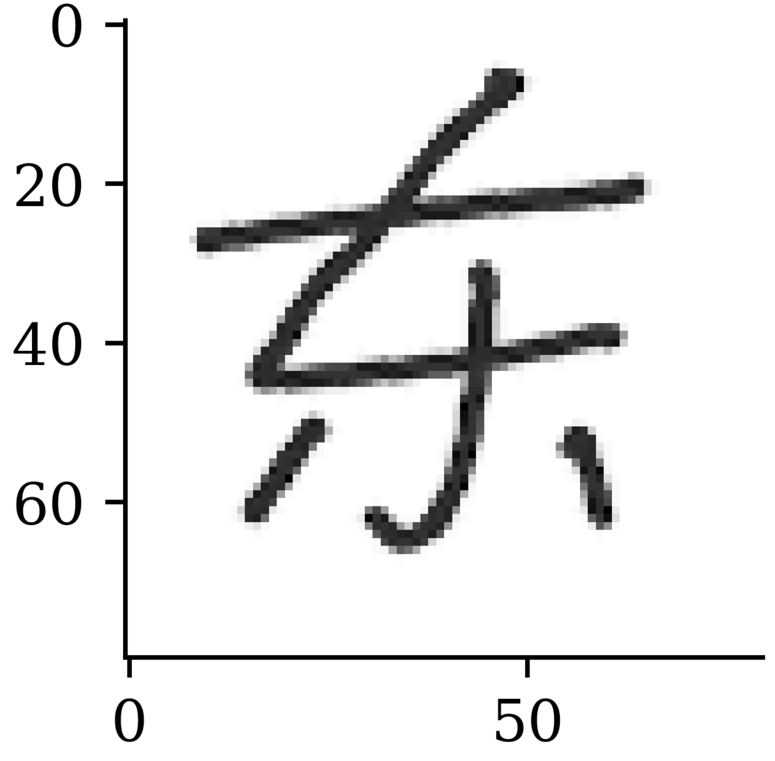
Example: PSAM

Loading the dataset off-screen (using Lecture 6 code).

```
plt.imshow(X_train[0], cmap="gray");
```

```
plt.imshow(X_train[42], cmap="gray");
```

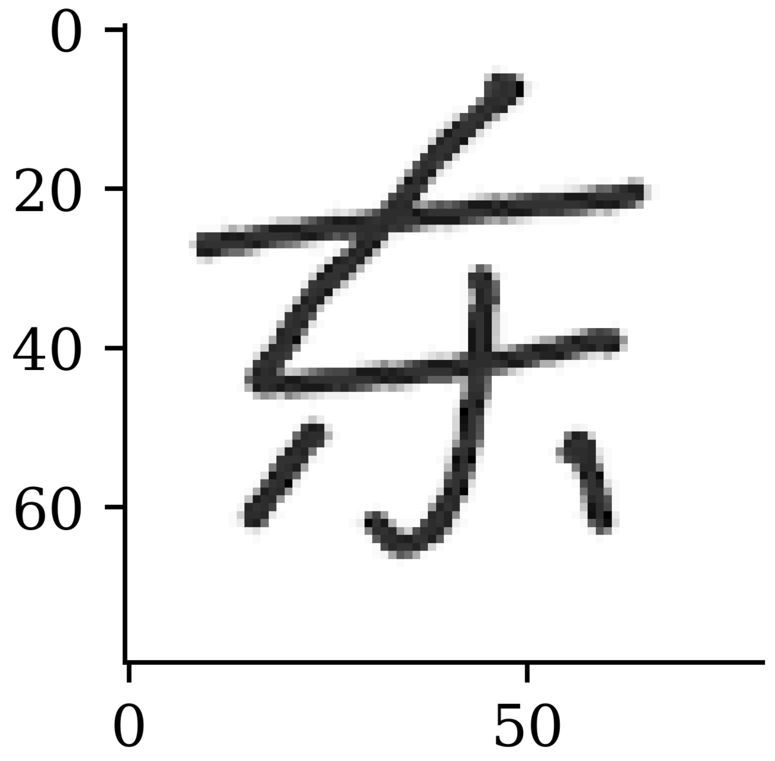




A compression game

```python
plt.imshow(X_train[42], cmap="gray");
print(img_width * img_height)
```

```
6400
```



> *A 4 with a curly foot, a flat line goes across the middle of the 4, two feet come off the bottom.*

96 characters

> *A Dōng character, rotated counterclockwise 15 degrees.*

54 characters

Make a basic autoencoder

```
num_hidden_layer = 400
print(f"Compress from {img_height * img_width} pixels to {num_hidden_layer} latent varia
```

```
Compress from 6400 pixels to 400 latent variables.
```

```
random.seed(123)

model = keras.models.Sequential([
    layers.Input((img_height, img_width, 1)),
    layers.Rescaling(1./255),
    layers.Flatten(),
    layers.Dense(num_hidden_layer, "relu"),
    layers.Dense(img_height*img_width, "sigmoid"),
    layers.Reshape((img_height, img_width, 1)),
    layers.Rescaling(255),
])

model.compile("adam", "mse")
epochs = 1_000
es = keras.callbacks.EarlyStopping(
    patience=5, restore_best_weights=True)
model.fit(X_train, X_train, epochs=epochs, verbose=0,
    validation_data=(X_val, X_val), callbacks=es);
```

The model

```
model.summary()
```

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ rescaling (Rescaling)           │ (None, 80, 80, 1)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 6400)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 400)            │     2,560,400 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 6400)           │     2,566,400 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ reshape (Reshape)               │ (None, 80, 80, 1)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ rescaling_1 (Rescaling)         │ (None, 80, 80, 1)      │             0 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 15,380,402 (58.67 MB)
 Trainable params: 5,126,800 (19.56 MB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 10,253,602 (39.11 MB)
```

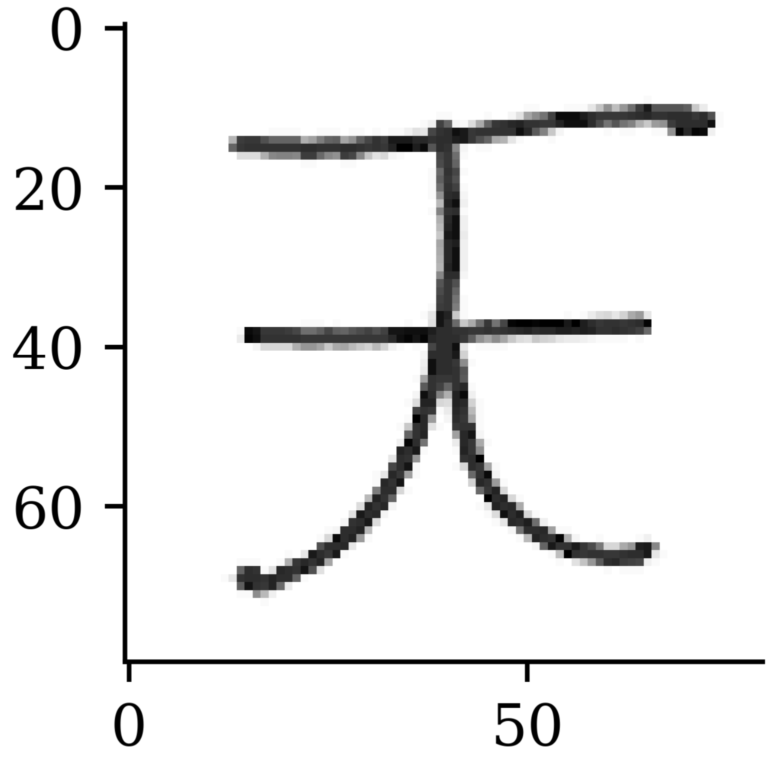
```
model.evaluate(X_val, X_val, verbose=0)
```

```
2261.9765625
```

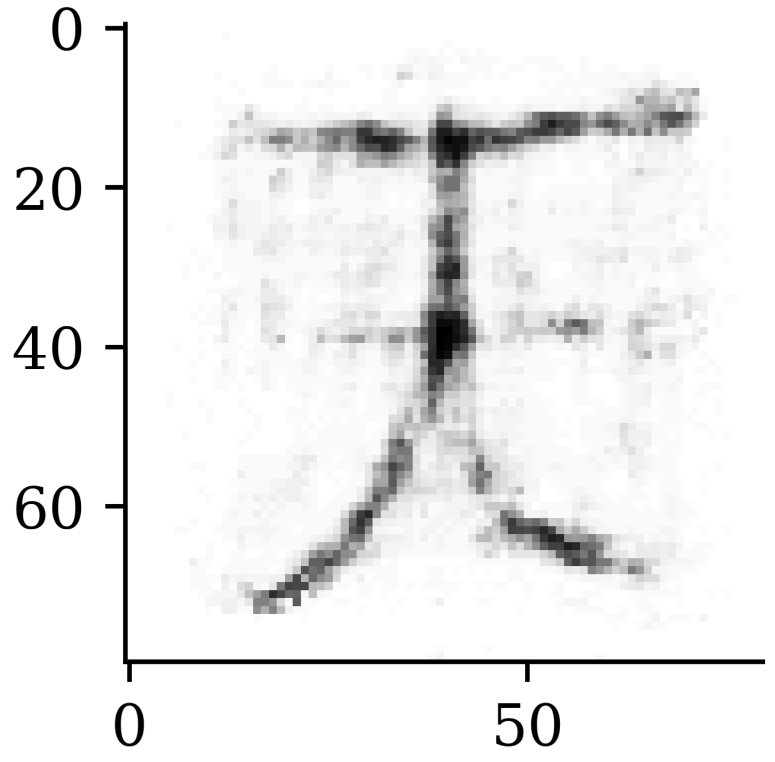
Some recovered image

```python
X_val_rec = model.predict(X_val, verbose=0)
```

```python
plt.imshow(X_val[42], cmap="gray");
```



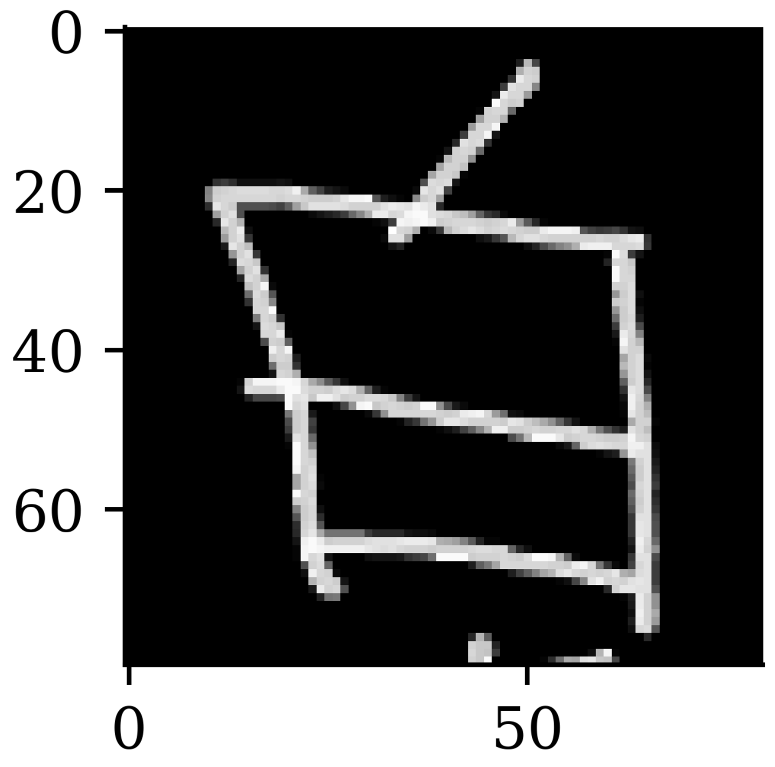
```python
plt.imshow(X_val_rec[42], cmap="gray");
```

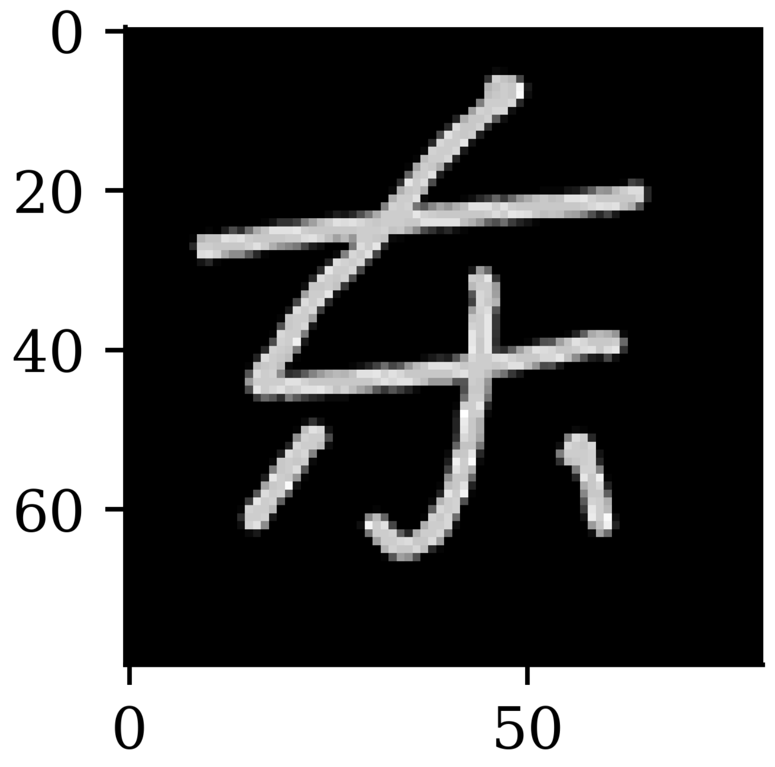


Invert the images

```python
plt.imshow(255 - X_train[0], cmap="gray
```

```python
plt.imshow(255 - X_train[42], cmap="gra
```





Try inverting the images

```
random.seed(123)

model = keras.models.Sequential([
    layers.Input((img_height, img_width, 1)),
    layers.Rescaling(1./255),
    layers.Lambda(lambda x: 1 - x),
    layers.Flatten(),
    layers.Dense(num_hidden_layer, "relu"),
    layers.Dense(img_height*img_width, "sigmoid"),
    layers.Lambda(lambda x: 1 - x),
    layers.Reshape((img_height, img_width, 1)),
    layers.Rescaling(255),
])

model.compile("adam", "mse")
model.fit(X_train, X_train, epochs=epochs, verbose=0,
    validation_data=(X_val, X_val), callbacks=es);
```

The model

```python
model.summary()
```

```
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ rescaling_2 (Rescaling)         │ (None, 80, 80, 1)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lambda (Lambda)                 │ (None, 80, 80, 1)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_1 (Flatten)             │ (None, 6400)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 400)            │     2,560,400 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 6400)           │     2,566,400 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lambda_1 (Lambda)               │ (None, 6400)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ reshape_1 (Reshape)             │ (None, 80, 80, 1)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ rescaling_3 (Rescaling)         │ (None, 80, 80, 1)      │             0 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 15,380,402 (58.67 MB)
 Trainable params: 5,126,800 (19.56 MB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 10,253,602 (39.11 MB)
```

```python
model.evaluate(X_val, X_val, verbose=0)
```

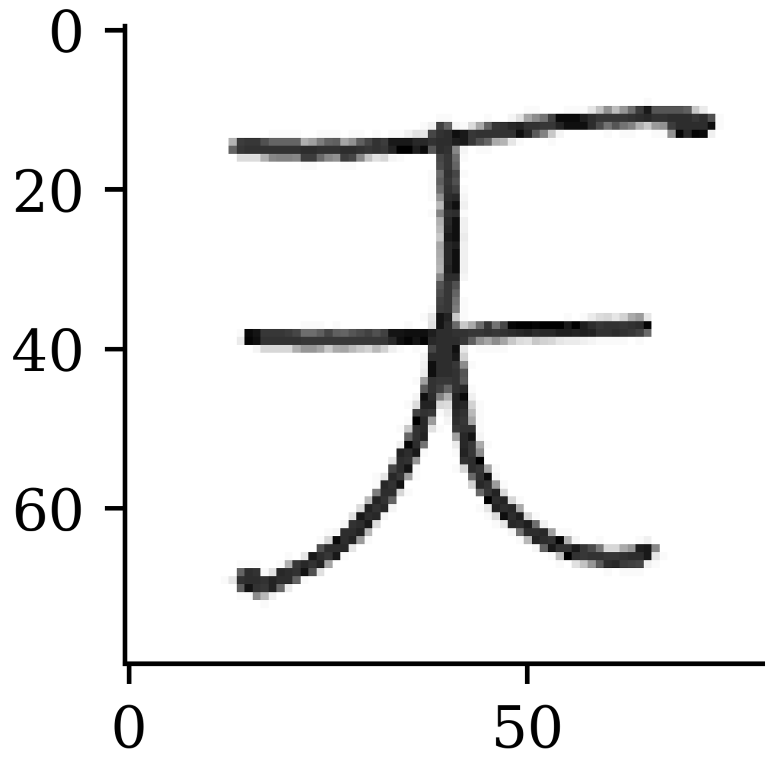
```
4181.1240234375
```

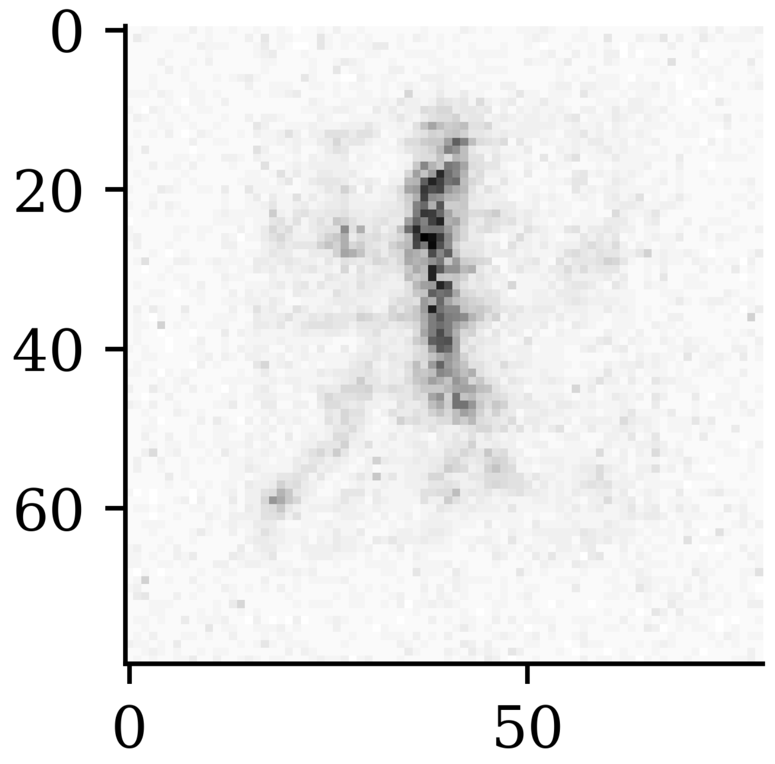
Some recovered image

```python
X_val_rec = model.predict(X_val, verbose=0)
```

```python
plt.imshow(X_val[42], cmap="gray");
```

```python
plt.imshow(X_val_rec[42], cmap="gray");
```





CNN-enhanced encoder

```python
random.seed(123)

encoder = keras.models.Sequential([
    layers.Input((img_height, img_width, 1)),
    layers.Rescaling(1./255),
    layers.Lambda(lambda x: 1 - x),
    layers.Conv2D(16, 3, padding="same", activation="relu"),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding="same", activation="relu"),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding="same", activation="relu"),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(num_hidden_layer, "relu")
])
```

```python
decoder = keras.models.Sequential([
    keras.Input(shape=(num_hidden_layer,)),
    layers.Dense(20*20),
    layers.Reshape((20, 20, 1)),
    layers.Conv2D(128, 3, padding="same", activation="relu"),
    layers.UpSampling2D(),
    layers.Conv2D(64, 3, padding="same", activation="relu"),
    layers.UpSampling2D(),
    layers.Conv2D(1, 1, padding="same", activation="relu"),
    layers.Lambda(lambda x: 1 - x),
    layers.Rescaling(255),
])

```

Encoder summary

```
encoder.summary()
```

```
Model: "sequential_2"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ rescaling_4 (Rescaling)          │ (None, 80, 80, 1)       │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ lambda_2 (Lambda)                │ (None, 80, 80, 1)       │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ conv2d (Conv2D)                  │ (None, 80, 80, 16)      │           160 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 40, 40, 16)      │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)                │ (None, 40, 40, 32)      │         4,640 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)   │ (None, 20, 20, 32)      │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)                │ (None, 20, 20, 64)      │        18,496 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)   │ (None, 10, 10, 64)      │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ flatten_2 (Flatten)              │ (None, 6400)            │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dense_4 (Dense)                  │ (None, 400)             │     2,560,400 │
└──────────────────────────────────┴─────────────────────────┴───────────────┘
 Total params: 2,583,696 (9.86 MB)
 Trainable params: 2,583,696 (9.86 MB)
 Non-trainable params: 0 (0.00 B)
```

Decoder summary

```python
decoder.summary()
```

```
Model: "sequential_3"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense_5 (Dense)                 │ (None, 400)            │       160,400 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ reshape_2 (Reshape)             │ (None, 20, 20, 1)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_3 (Conv2D)               │ (None, 20, 20, 128)    │         1,280 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ up_sampling2d (UpSampling2D)    │ (None, 40, 40, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_4 (Conv2D)               │ (None, 40, 40, 64)     │        73,792 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ up_sampling2d_1 (UpSampling2D)  │ (None, 80, 80, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_5 (Conv2D)               │ (None, 80, 80, 1)      │            65 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lambda_3 (Lambda)               │ (None, 80, 80, 1)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ rescaling_5 (Rescaling)         │ (None, 80, 80, 1)      │             0 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 235,537 (920.07 KB)
 Trainable params: 235,537 (920.07 KB)
 Non-trainable params: 0 (0.00 B)
```

```python
model.evaluate(X_val, X_val, verbose=0)
```

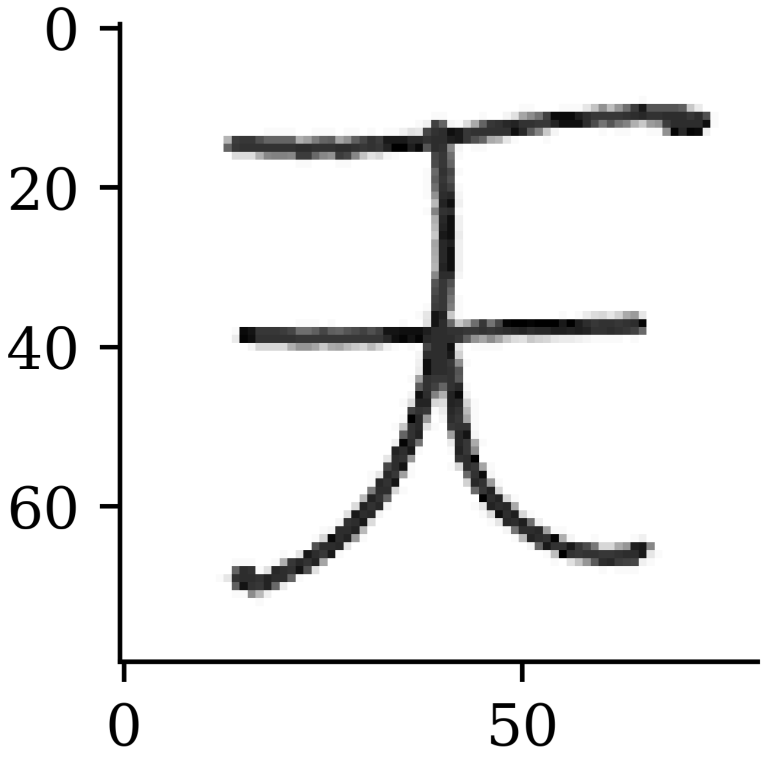
```
3442.788330078125
```

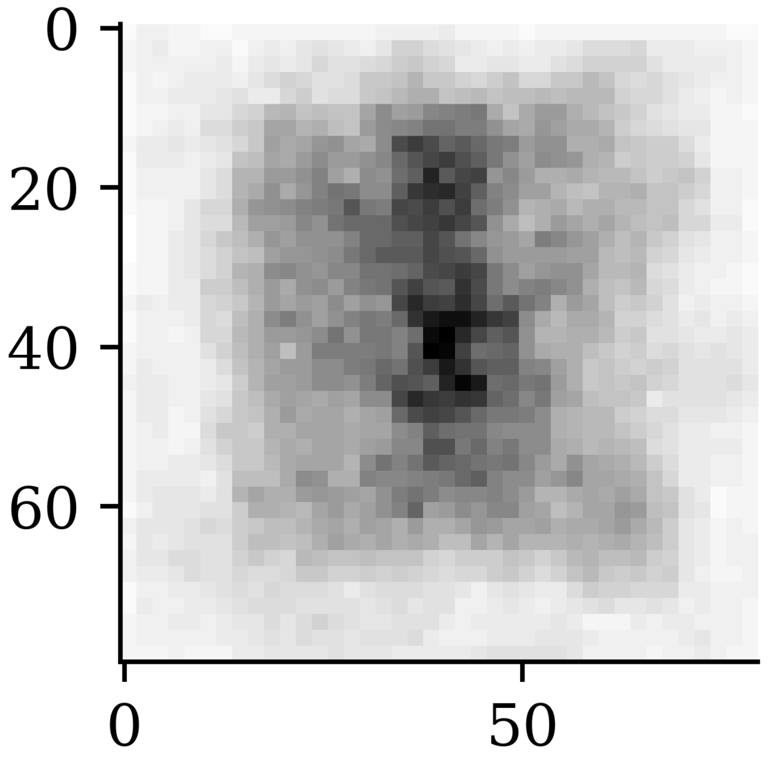
Some recovered image

```python
X_val_rec = model.predict(X_val, verbose=0)
```

```python
plt.imshow(X_val[42], cmap="gray");
```

```python
plt.imshow(X_val_rec[42], cmap="gray");
```





Latent space vs word embedding

- We revisit the concept of word embedding, where words in the vocabulary are mapped into vector representations. Words with similar meaning should lie close to one another in the word-embedding space.
- Latent space contains low-dimensional representation of data. Data/Images that are similar should lie close in the latent space.
- There are pre-trained word-embedding spaces such as those for English-language movie review, German-language legal documents, etc. Semantic relationships between words differ for different tasks. Similarly, the structure of latent spaces for different data sets (humans faces, animals, etc) are different.

Latent space vs word embedding

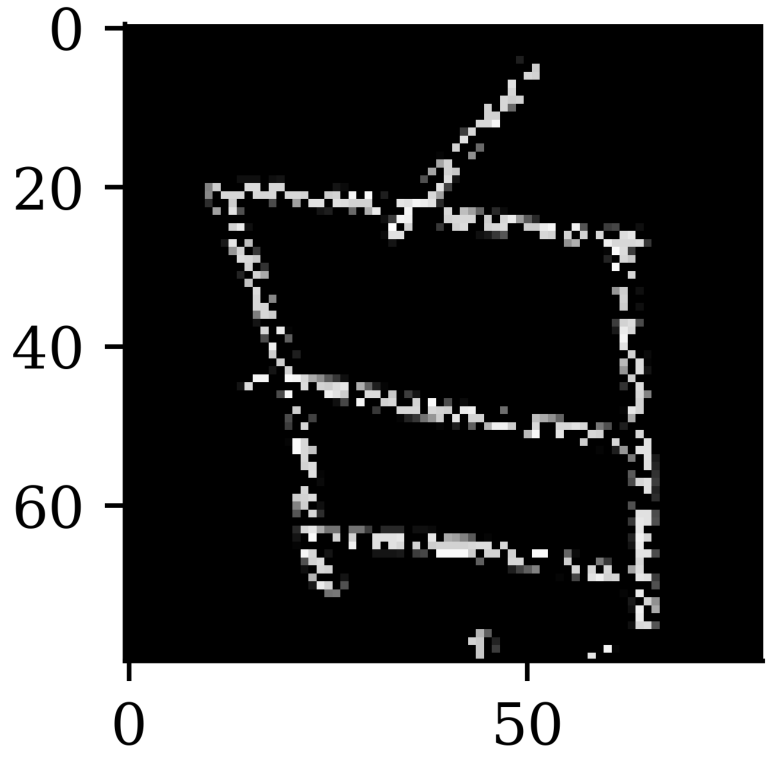
- Given a latent space of representations, or an embedding space, certain directions in the space may encode interesting axes of variation in the original data.
- A **concept vector** is a direction of variation in the data. For example there may be a smile vector such that if z is the latent representation of a face, then $z + s$ is the representation of the same face, smiling. We can generate an image of the person smiling from this latent representation.

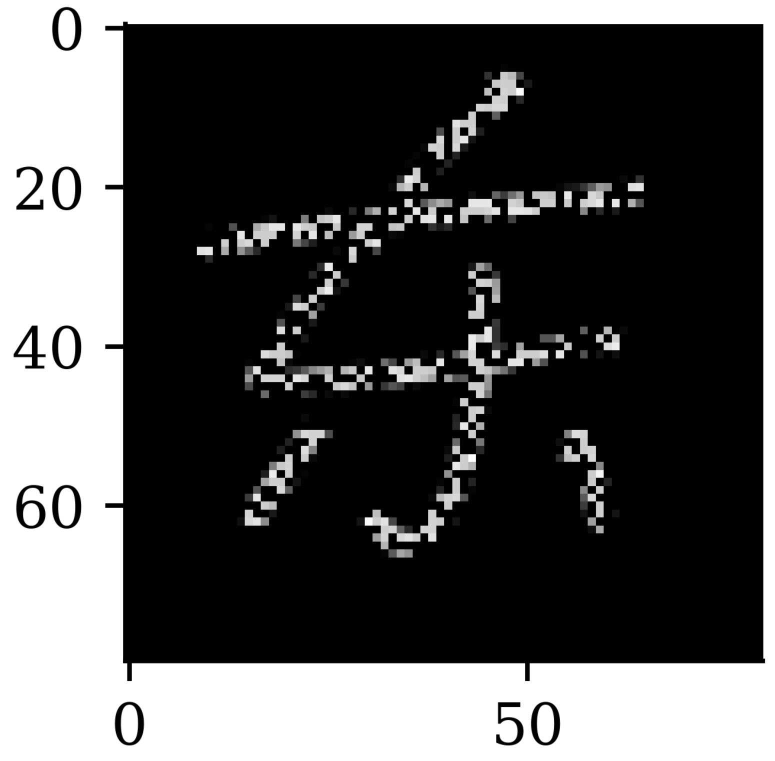
UNSW SYDNEY

Intentionally add noise to inputs

```
mask = rnd.random(size=X_train.shape[1:
plt.imshow(mask * (255 - X_train[0]), c
```

```
mask = rnd.random(size=X_train.shape[1:
plt.imshow(mask * (255 - X_train[42]) *
```





Denoising autoencoder

Can be used to do feature engineering for supervised learning problems

> It is also possible to include input variables as outputs to infer missing values or just help the model "understand" the features – in fact the winning solution of a claims prediction Kaggle competition heavily used denoising autoencoders together with model stacking and ensembling – read more here.

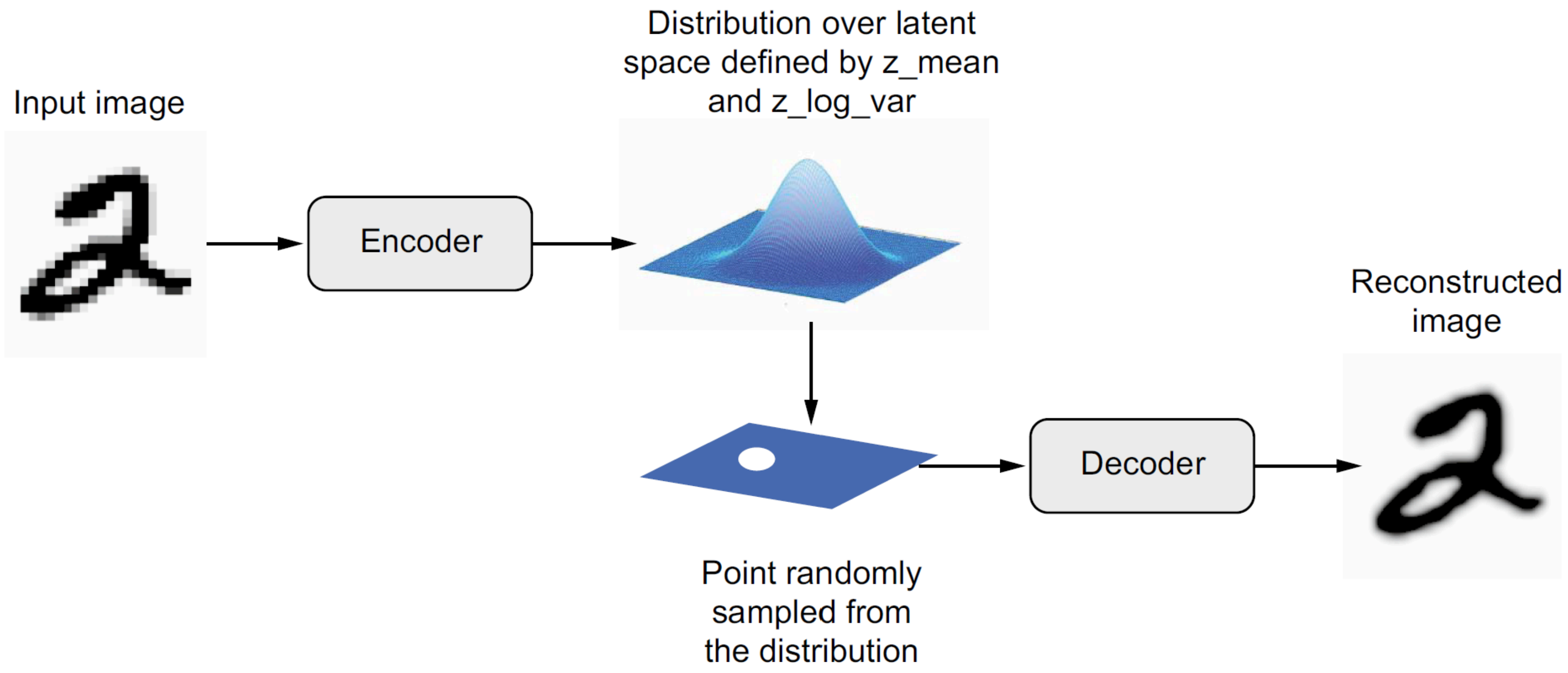
Jacky Poon

Source: Poon (2021), *Multitasking Risk Pricing Using Deep Learning*, Actuaries' Analytical Cookbook.

Lecture Outline

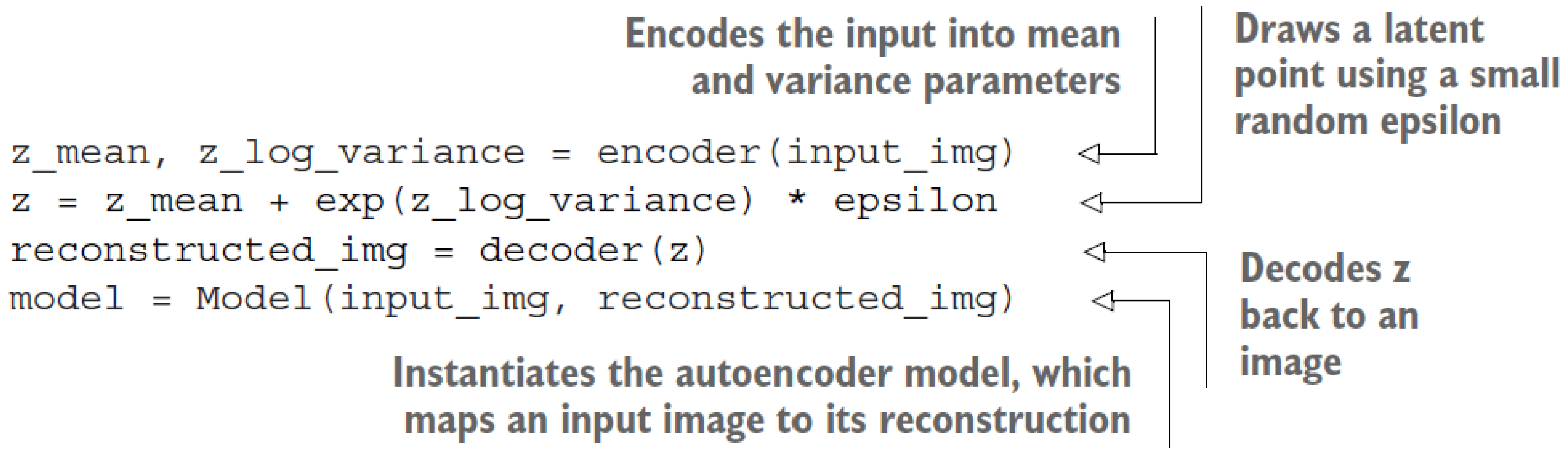
- Text Generation
- Sampling strategy
- Transformers
- Image Generation
- Neural style transfer
- Autoencoders
- **Variational Autoencoders**
- Diffusion Models

Variational autoencoder


Input image
Encoder
Distribution over latent space defined by z_mean and z_log_var
Point randomly sampled from the distribution
Decoder
Reconstructed image

Schematic of a variational autoencoder.

VAE schematic process



```
z_mean, z_log_variance = encoder(input_img)
z = z_mean + exp(z_log_variance) * epsilon
reconstructed_img = decoder(z)
model = Model(input_img, reconstructed_img)
```

Encodes the input into mean and variance parameters

Draws a latent point using a small random epsilon

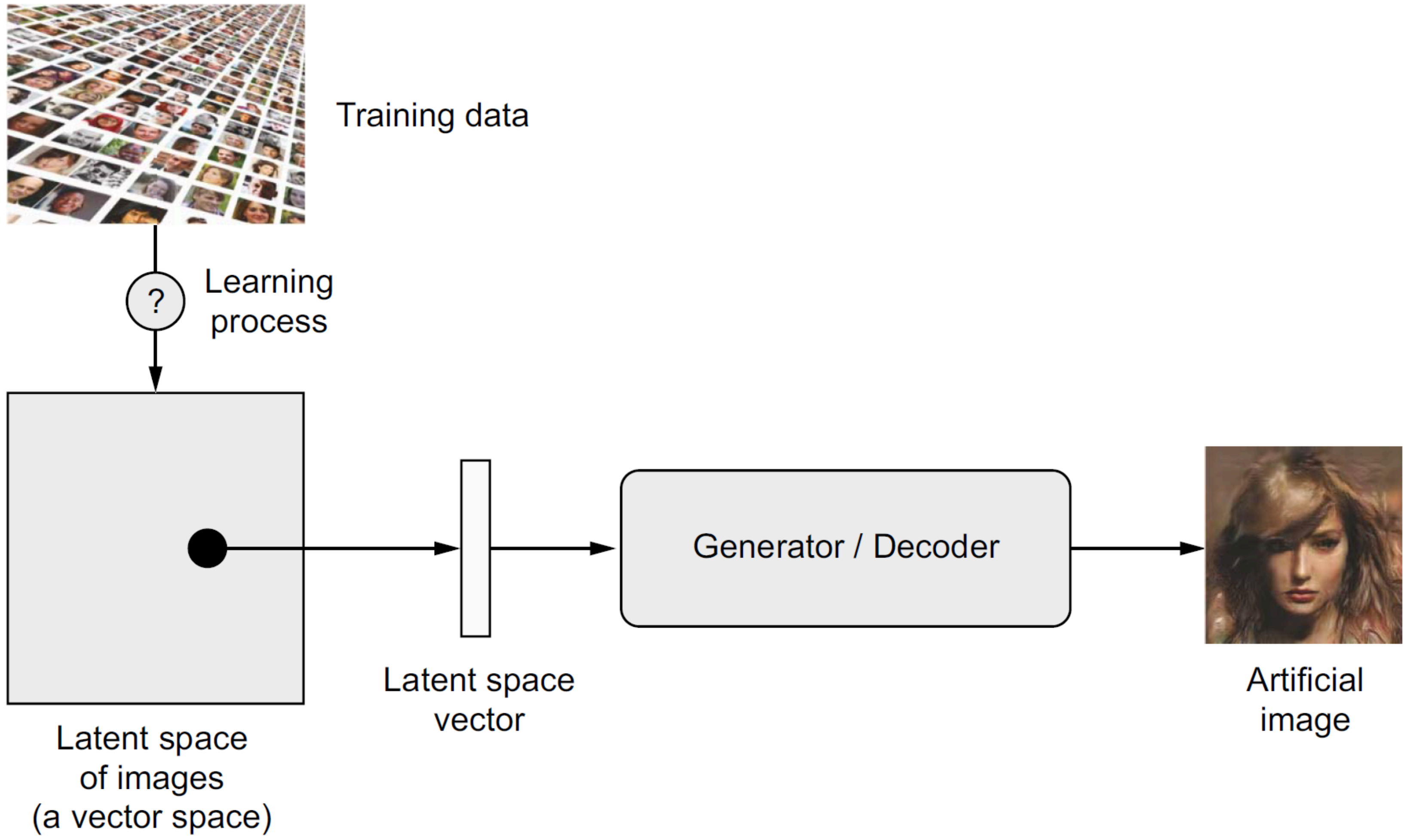
Decodes z back to an image

Instantiates the autoencoder model, which maps an input image to its reconstruction

Keras code for a VAE.

Source: François Chollet (2021), *Deep Learning with Python*, Second Edition, Unnumbered listing in Chapter 12.

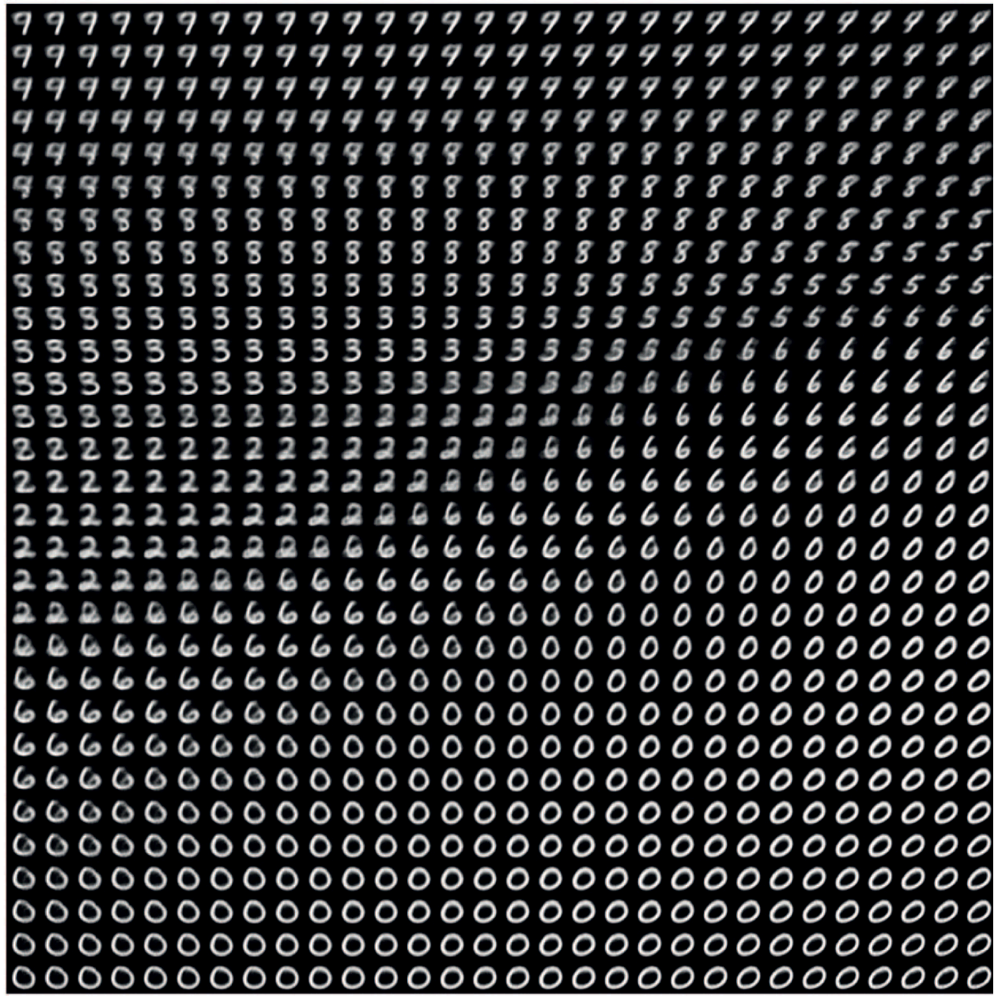
Focus on the decoder



Sampling new artificial images from the latent space.

Source: François Chollet (2021), *Deep Learning with Python*, Second Edition, Figure 12.13.

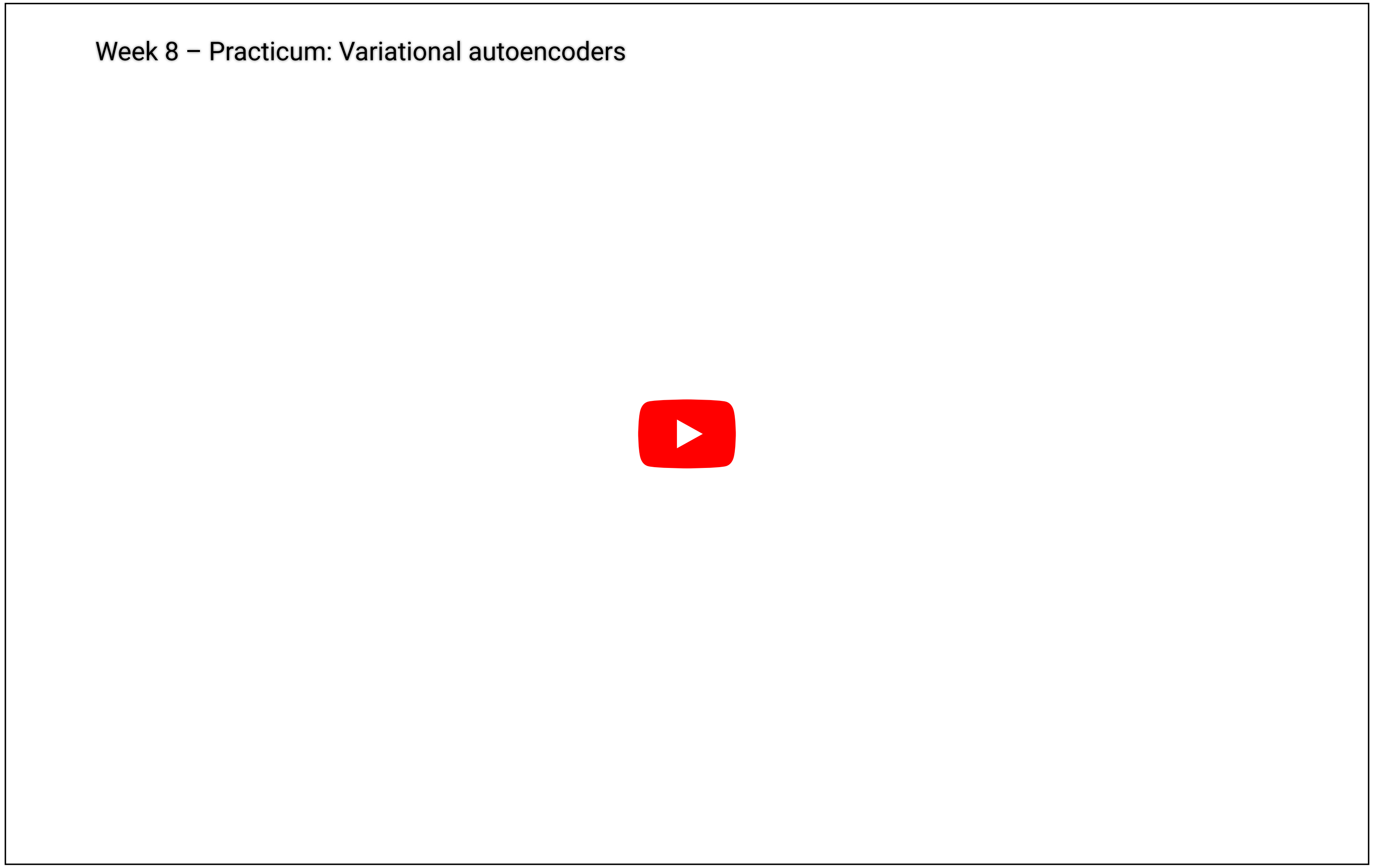
Exploring the MNIST latent space



Example of MNIST-like images generated from the latent space.

Source: François Chollet (2021), *Deep Learning with Python*, Second Edition, Figure 12.18.

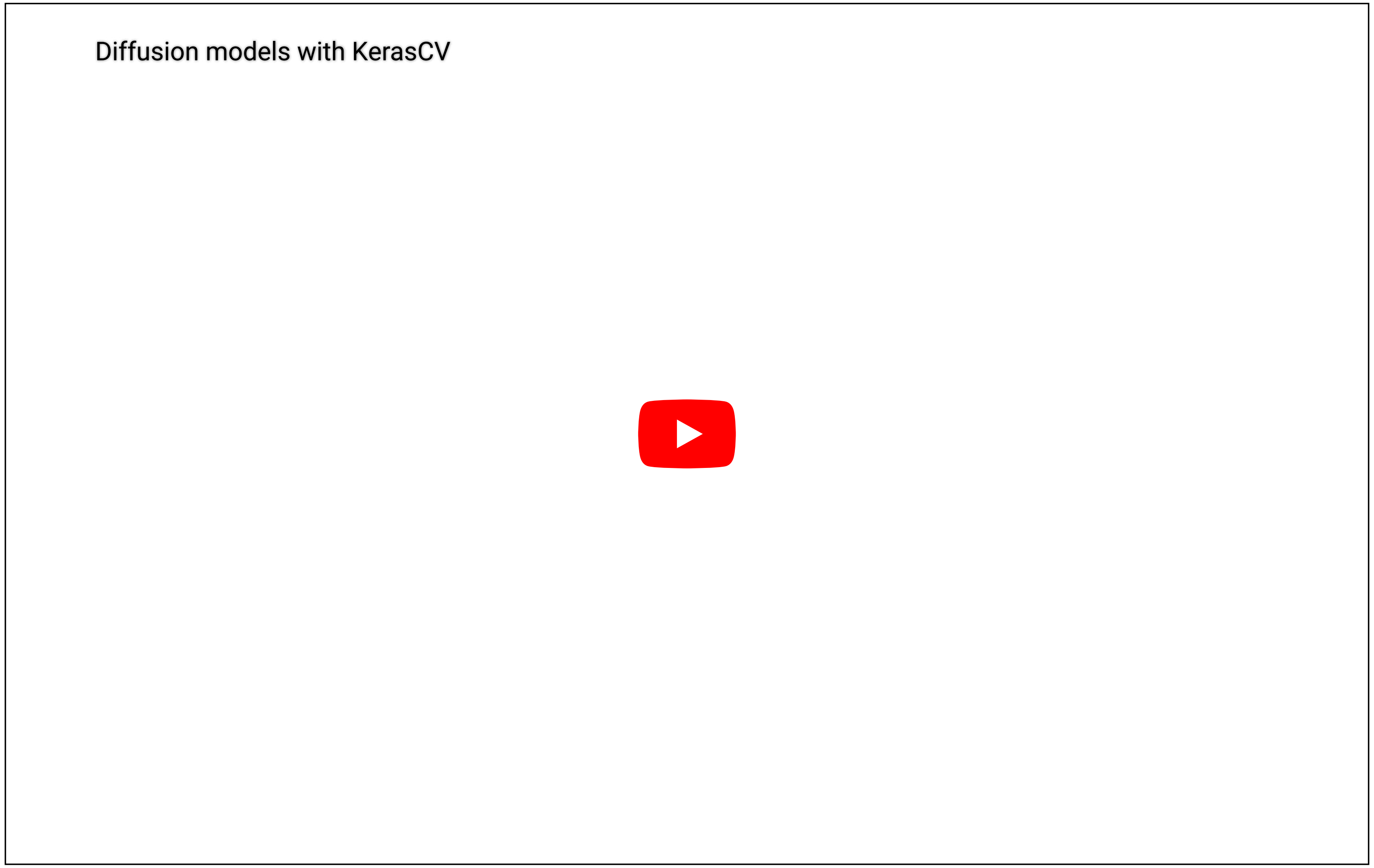
Recommended Viewing


Week 8 – Practicum: Variational autoencoders

Lecture Outline

- Text Generation
- Sampling strategy
- Transformers
- Image Generation
- Neural style transfer
- Autoencoders
- Variational Autoencoders
- **Diffusion Models**

Using KerasCV



Package Versions

```python
from watermark import watermark
print(watermark(python=True, packages="keras,matplotlib,numpy,pandas,seaborn,scipy,torch
```

```
Python implementation: CPython
Python version       : 3.11.8
IPython version      : 8.23.0

keras     : 3.2.0
matplotlib: 3.8.4
numpy     : 1.26.4
pandas    : 2.2.1
seaborn   : 0.13.2
scipy     : 1.11.0
torch     : 2.2.2
tensorflow: 2.16.1
tf_keras  : 2.16.0
```

Glossary

- autoencoder (variational)
- beam search
- bias
- ChatGPT (& RLHF)
- DeepDream
- greedy sampling
- HuggingFace
- language model
- latent space
- neural style transfer
- softmax temperature
- stochastic sampling