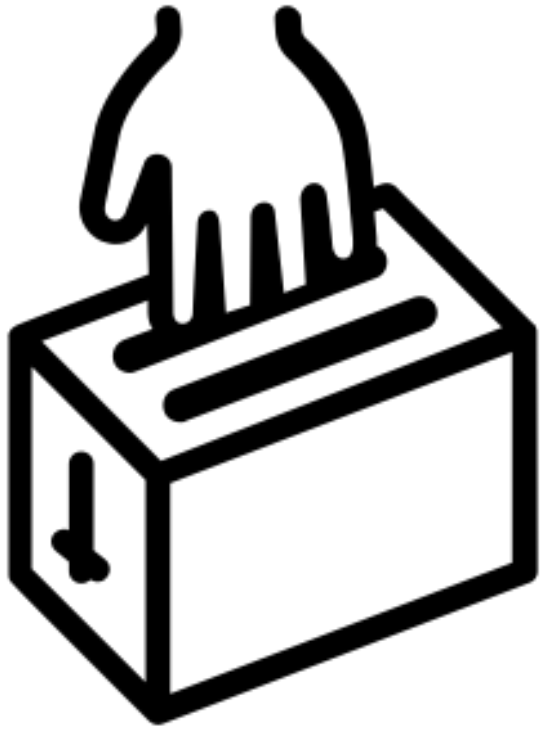
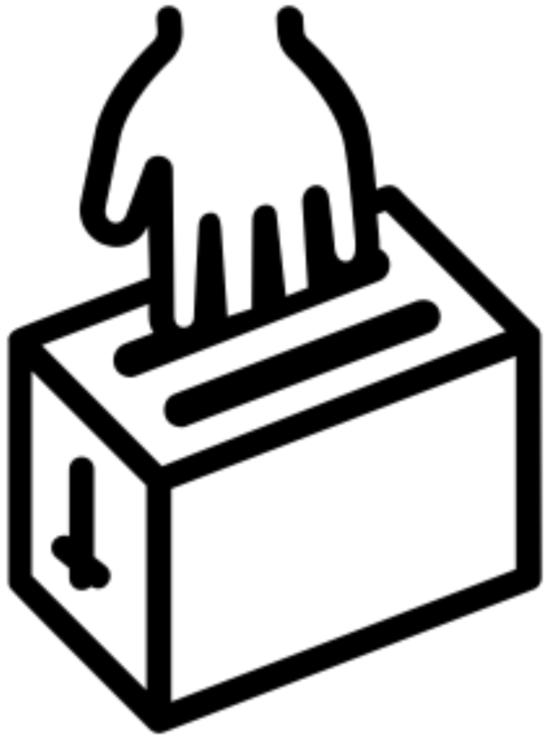
Common Sense
as a Cure against Hype



A.I. Caramba!

Algorithms Can Still be Stupid?



DeepLearning[tm]

versus

JustThinking[tm]

Artificial Stupidity

There's a trend I'm seeing that I do not like.

As it is getting easier to copy/paste code from stack overflow and docs I'm seeing more and more intellectual laziness. It's great that people can get started quickly, but it feels like people focus more on the tools they're using than on the problems they're supposed to be solving.

Hype **really** doesn't help.

Artificial Stupidity

I get it, there's too little time to learn everything and maybe you feel a bit of peer pressure to deliver; copying and pasting code feels easier. Even I do it ... sometimes.

But you should never forget to think!

You might loose touch with the problem you're trying to solve and the system you're trying to improve. You cannot simply have blind faith in the algorithms.

Artificial Stupidity

Tonight I would like to show you some quick examples of what this might lead to in algorithm country. Hopefully I'll be able to remind you that thinking is still a good idea and that there is a lot of benefit in being able to understand your problem before your solution.

If GDPR hasn't encouraged you to be thoughtful, maybe the examples I'll show today will.

There is another way to do ML ... rethinking helps.

this talk has examples

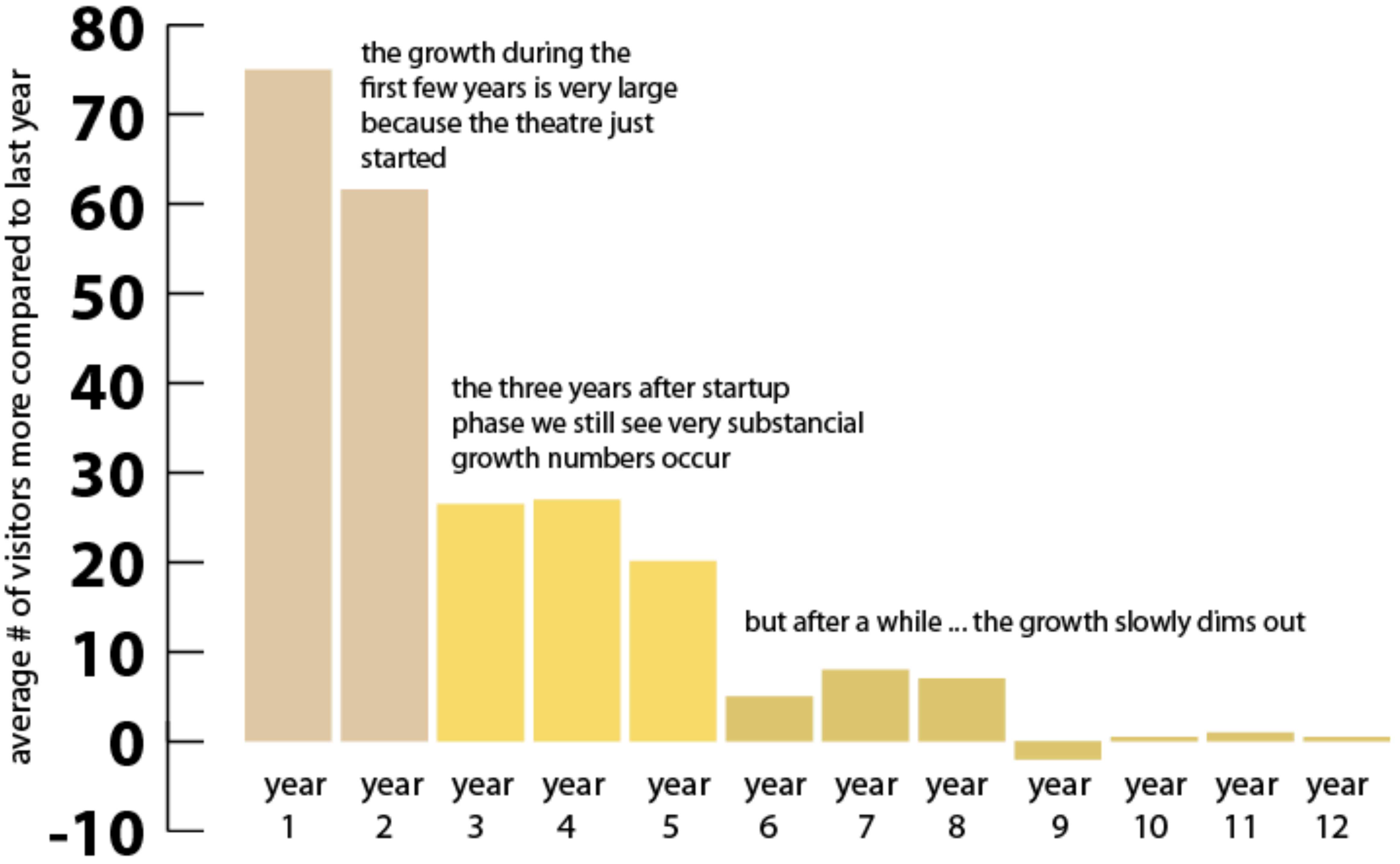
- my first vanity metric [theatre]
- DeepLearning[tm] in vision [faces]
- DeepLearning[tm] in vision [VincentsAE]
- DeepLearning[tm] in timeseries [HMM > DL?]
- conclusion chickens [literally, rethinking]

A Vanity Metric

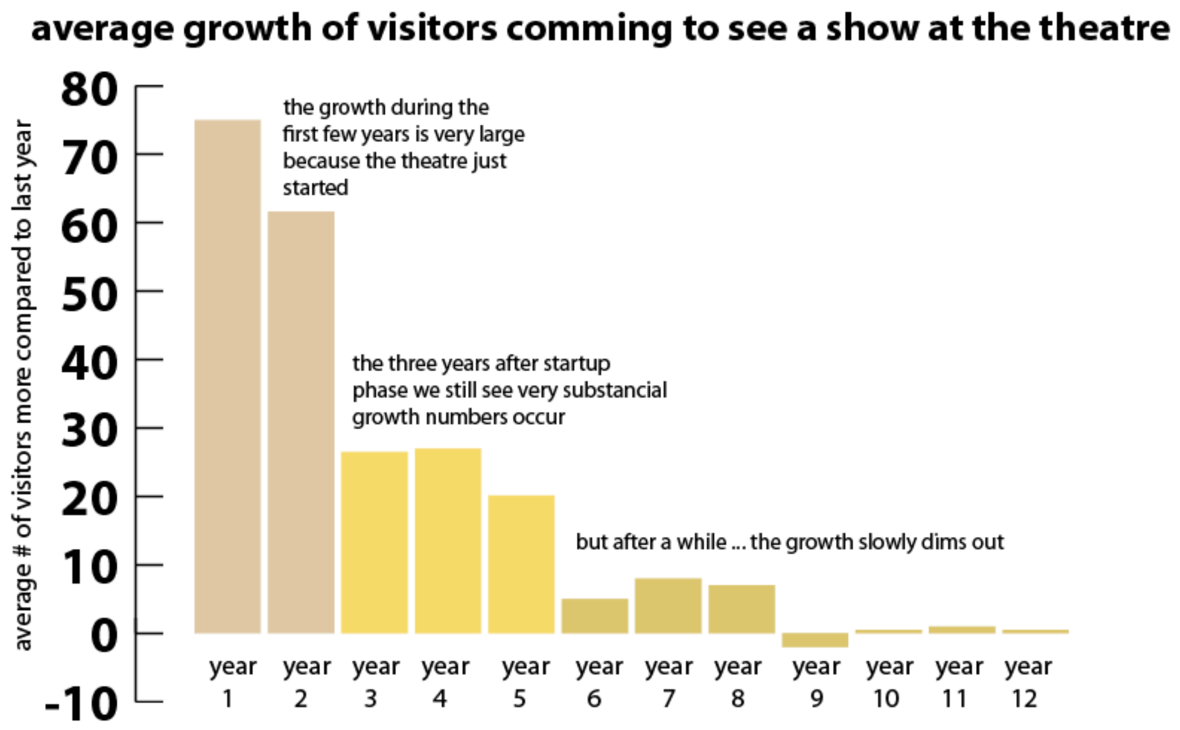
The university asked me to find something to apply statistics to. So I asked my boss for a sales dataset.

There was a usecase ...

average growth of visitors comming to see a show at the theatre



I got an A+ for this work.



But, what do you think went wrong?

Natural Stupidity

The rise in visitors was declining because the theatre was getting full during each and every show, not because the market of visitors had shrunk.

We were measuring the visitors that actually were able to get a seat, not the visitors that wanted to get a seat!

Natural Stupidity

No matter how good you assume you are with the data be sure to keep your eyes open to all things that the data does not tell you.

Do not put blind faith in what numbers seem to tell you until you are quite certain that you know what they don't.

DeepLearning[tm] in vision [faces]

Life at GoDataDriven

GoDataDriven applies data innovation to ensure any business is in front of the wave. From online retail to financial services; public to telco; we transform organizations into data-driven enterprises. We're on the forefront of data technologies and data science, and we're looking for exceptional people to join us.

data engineer | **software engineer** | **lead data engineer cloud** | **data scientist** | **machine learning engineer** | **growth hacker**

WANTED: IT Professionals for Unique And Personalized Data Engineering Training Program

DeepLearning[tm] in vision [faces]

Stijn Tonk

Data Mining Scientist

bio

Robert Rodger

Data Scientist

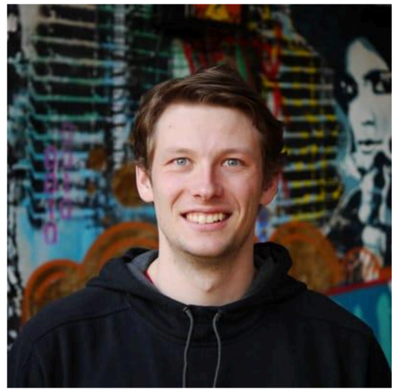
bio

Henk Griffioen

Data Scientist

bio

Bas Harenslak



Big Data Hacker

bio

Jelte Hoekstra

Data Scientist

bio

Rodrigo Agundez

Data Maverick

bio

Fokko Driesprong

Data Evangelist

bio

Nelli Gofman

Data Scientist

bio

$$\{X_n\} = \mathbf{37} \to \mathbf{1000+}$$

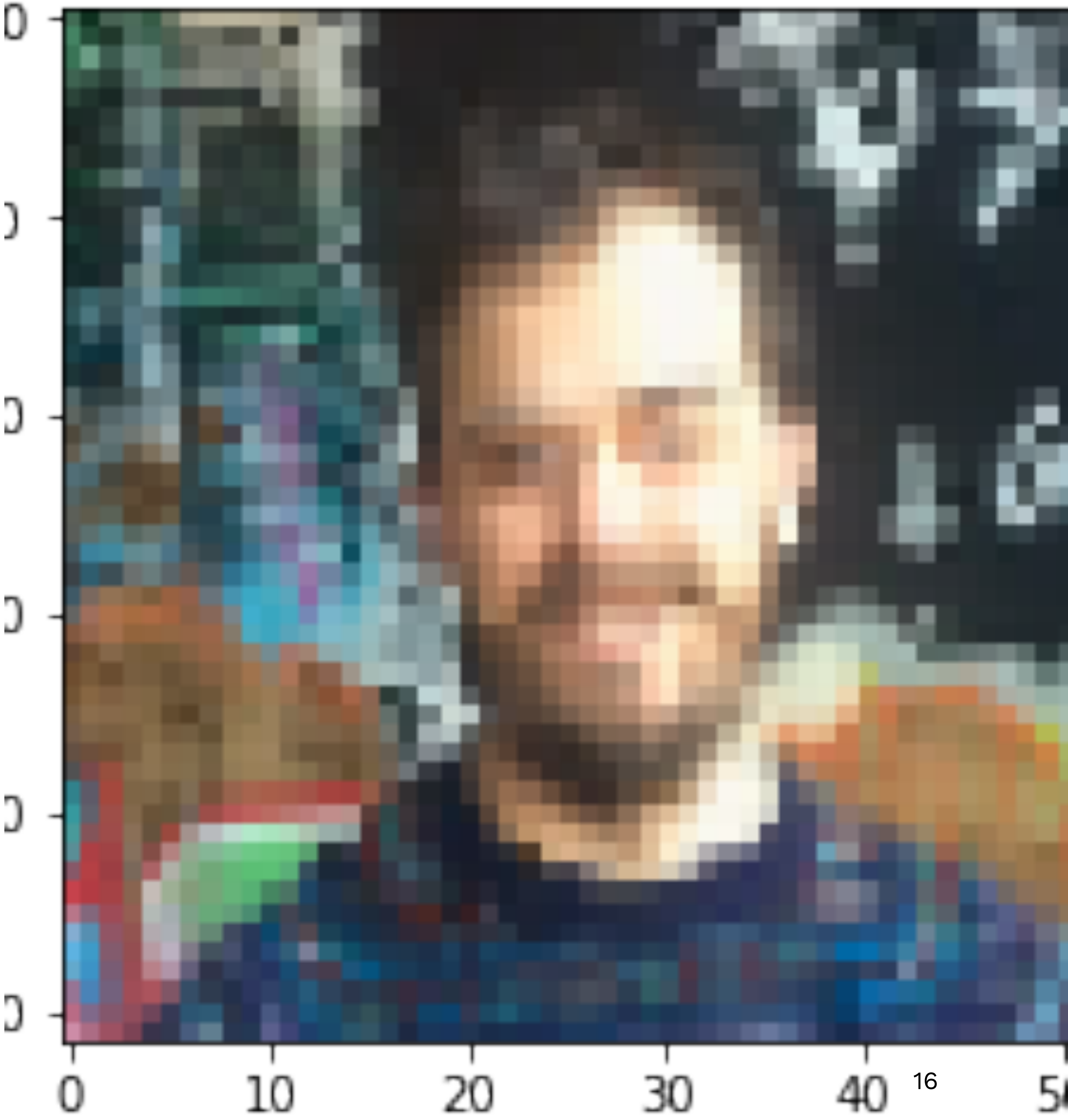
I can generate 1000s of images from my 37 collegues.

```
datagen = ImageDataGenerator(
    horizontal_flip=True,
    rotation_range=25,
    samplewise_std_normalization=True,
    zoom_range=0.2)
```

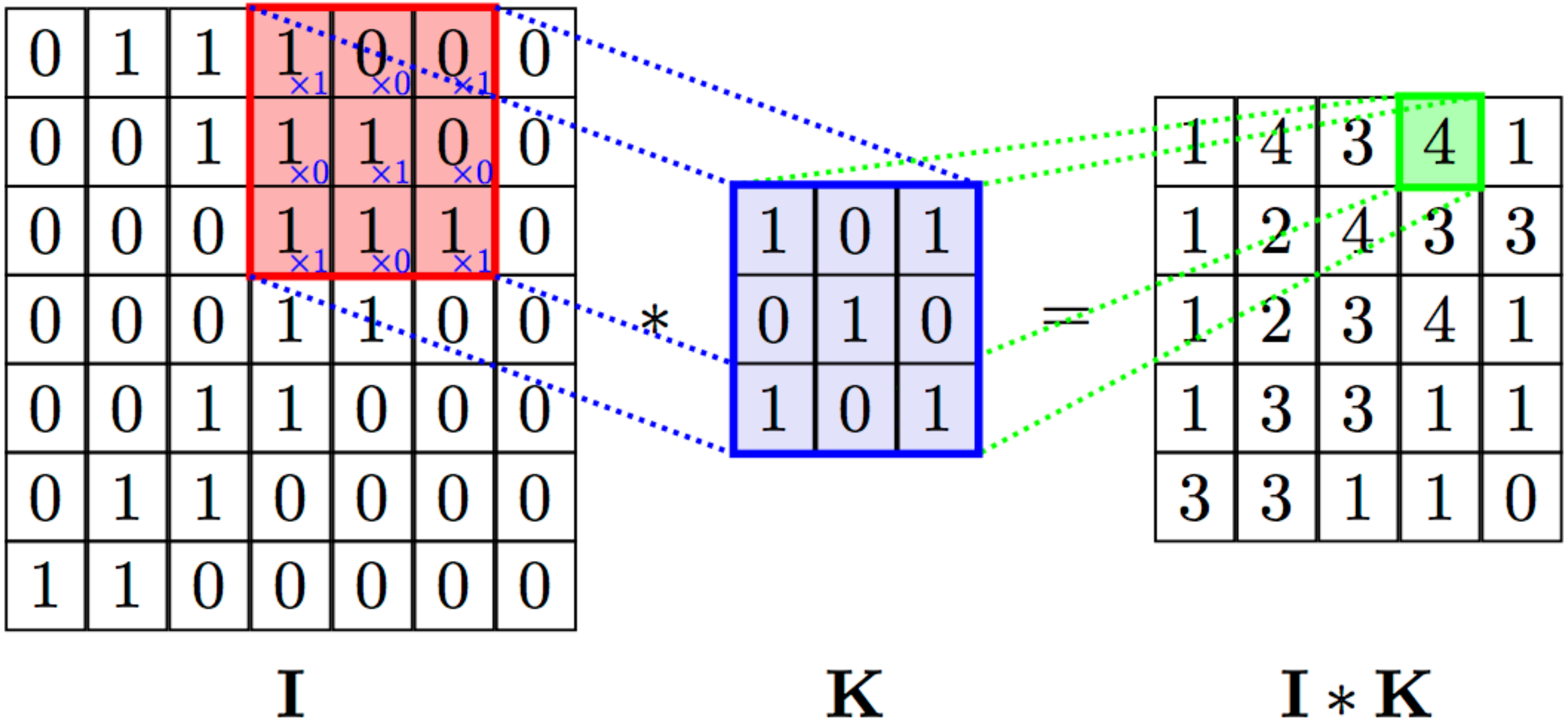
Here's an example from the generator.

With this: let's see if we can generate collegue faces. A good starting point might be to see if we can build a proper encoder.

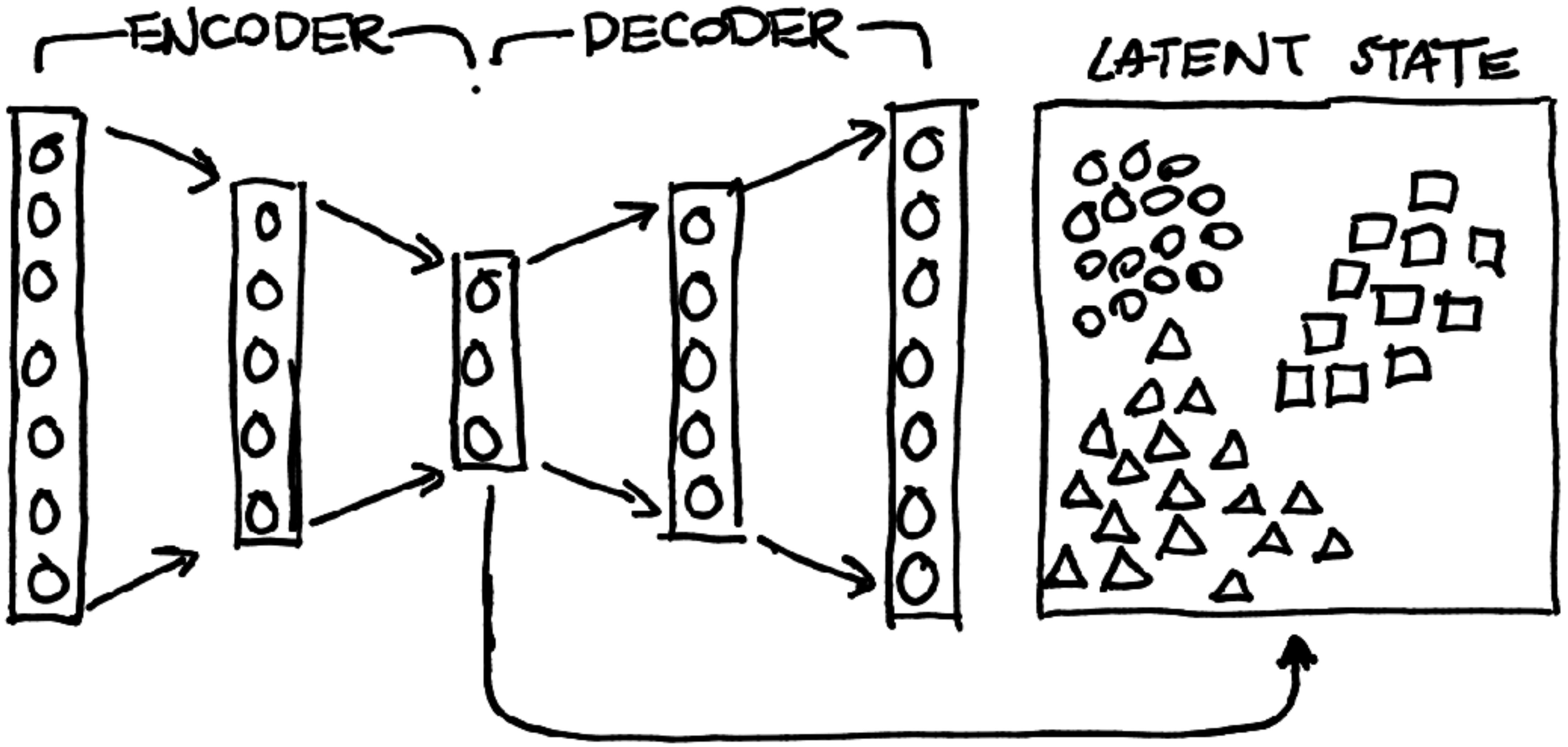
I've decided to build one in keras and after a small afternoon trying to make sure that all the layers fit I had some basic code.



I'll be using Convolutions a lot



And AutoEncoders

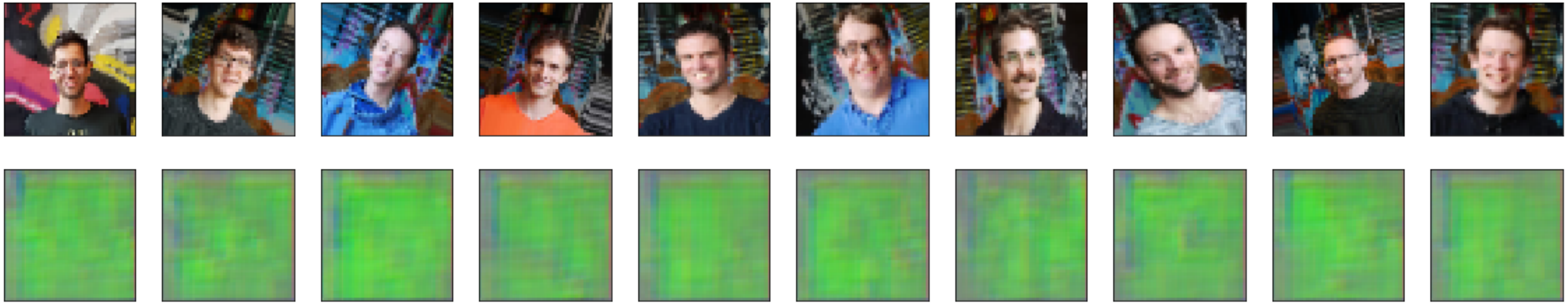


```
x = Conv2D(8, (3, 3), activation='relu', padding='same')(input_img)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same', name='encoded')(x)
encoded = Flatten()(x)
x = Reshape(target_shape = [_.value for _ in x.shape[1:]])(encoded)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=Adam(lr=0.00001), loss='binary_crossentropy')
```

It went ... horribly

So can anybody tell me why?

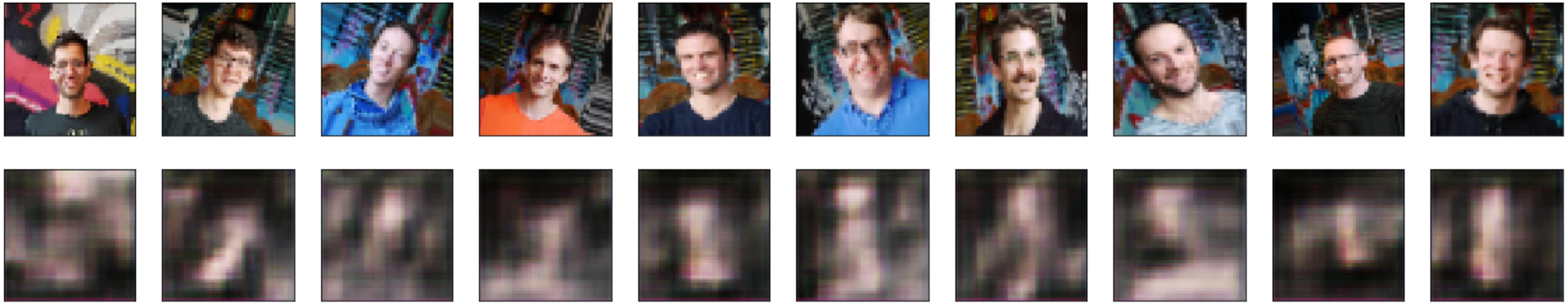


```
x = Conv2D(8, (3, 3), activation='tanh', padding='same')(input_img)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='tanh', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='tanh', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same', name='encoded')(x)
encoded = Flatten()(x)
x = Reshape(target_shape = [_.value for _ in x.shape[1:]])(encoded)
x = Conv2D(4, (3, 3), activation='tanh', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(4, (3, 3), activation='tanh', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='tanh')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=Adam(lr=0.001), loss='binary_crossentropy')
```

It went ... meh

So can anybody tell me why?

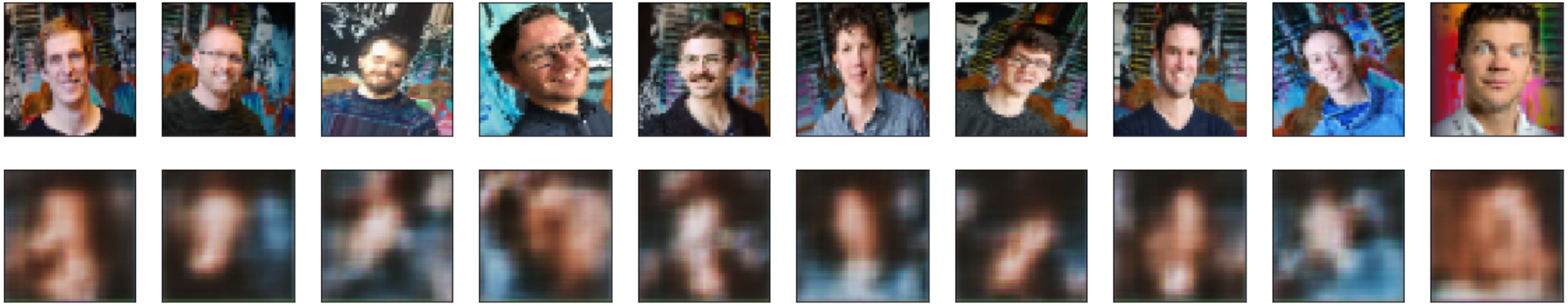


```python
x = Conv2D(8, (3, 3), activation='tanh', padding='same')(input_img)
x = BatchNormalization()(x)
x = AveragePooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='tanh', padding='same')(x)
x = AveragePooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='tanh', padding='same')(x)
x = AveragePooling2D((2, 2), padding='same', name='encoded')(x)
encoded = Flatten()(x)
x = Reshape(target_shape = [_.value for _ in x.shape[1:]])(encoded)
x = Conv2D(4, (3, 3), activation='tanh', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(4, (3, 3), activation='tanh', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='tanh')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=Adam(lr=0.001), loss='binary_crossentropy')
```

It went ... better

So can anybody tell me why?

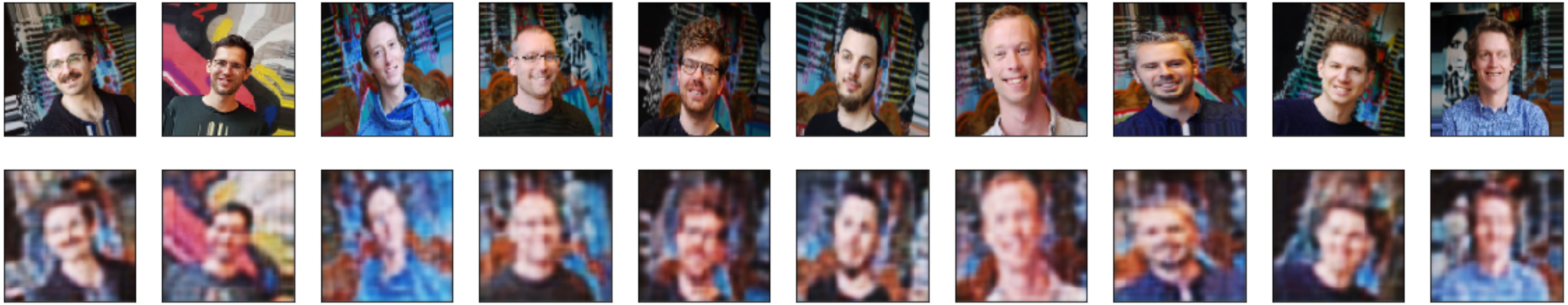


Next up, I figured it might be better to make ever smaller stepsizes as I train more and more epochs.

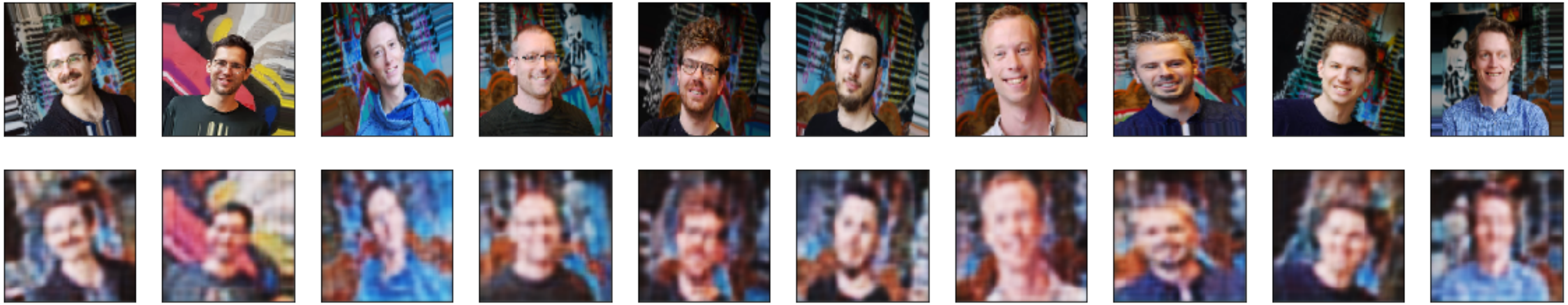
```
adam.lr = 0.01
autoencoder.fit_generator(godata_gen_col, epochs=10, ...)
adam.lr = 0.001
autoencoder.fit_generator(godata_gen_col, epochs=10, ...)
adam.lr = 0.0005
autoencoder.fit_generator(godata_gen_col, epochs=10, ...)
```

It went ... better

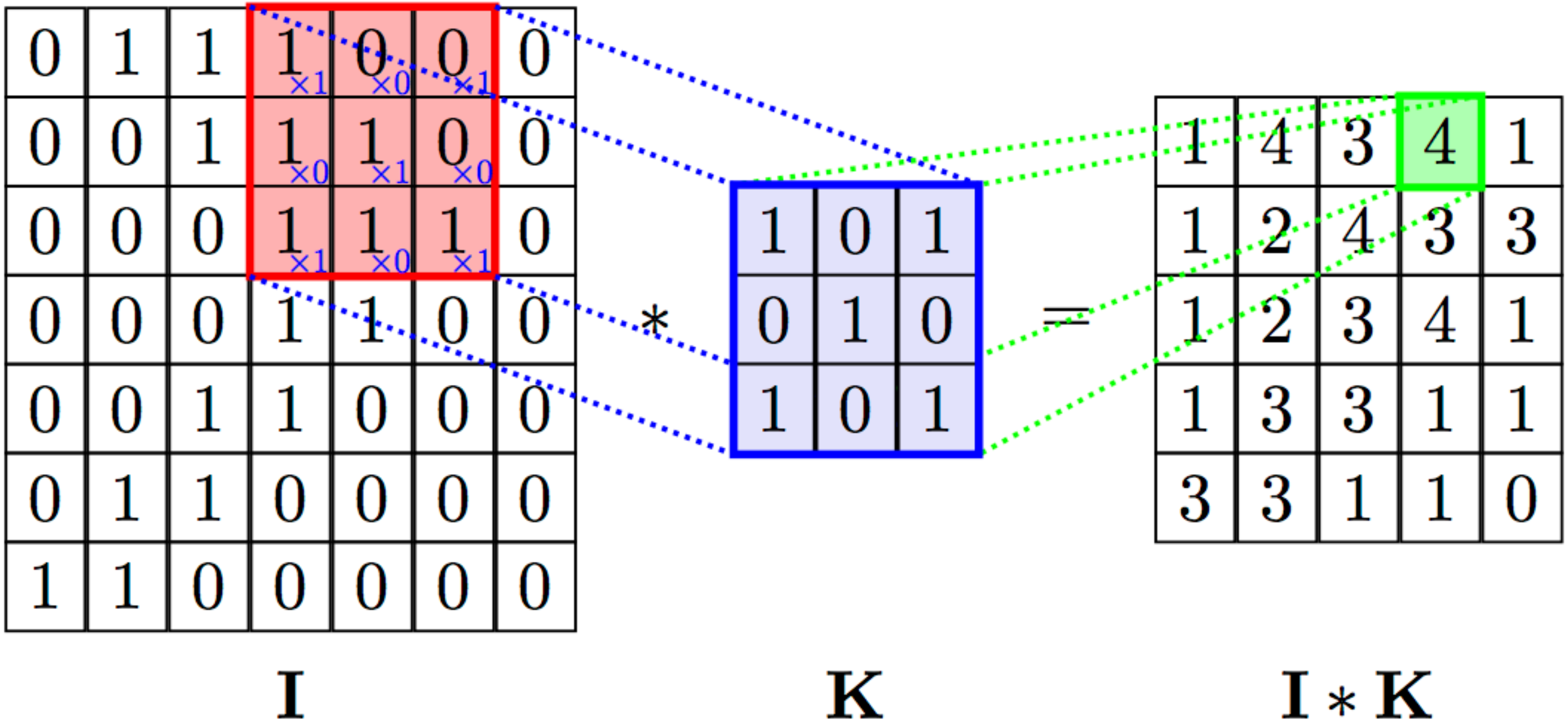
So can anybody tell me why?



It went ... better



Trollolol, actually I increased the input image size!


I
K
I * K

The main thing that made the difference was actually learning what convolutions do. It is very easy, fun even, to download some code from gitlab and paste it in a notebook to see what it does but not understanding it makes it tricky.

If you work with vision tasks and neural networks be sure you at least understand what convolutions are. Spend the few hours learning and you can start to think along much much better.

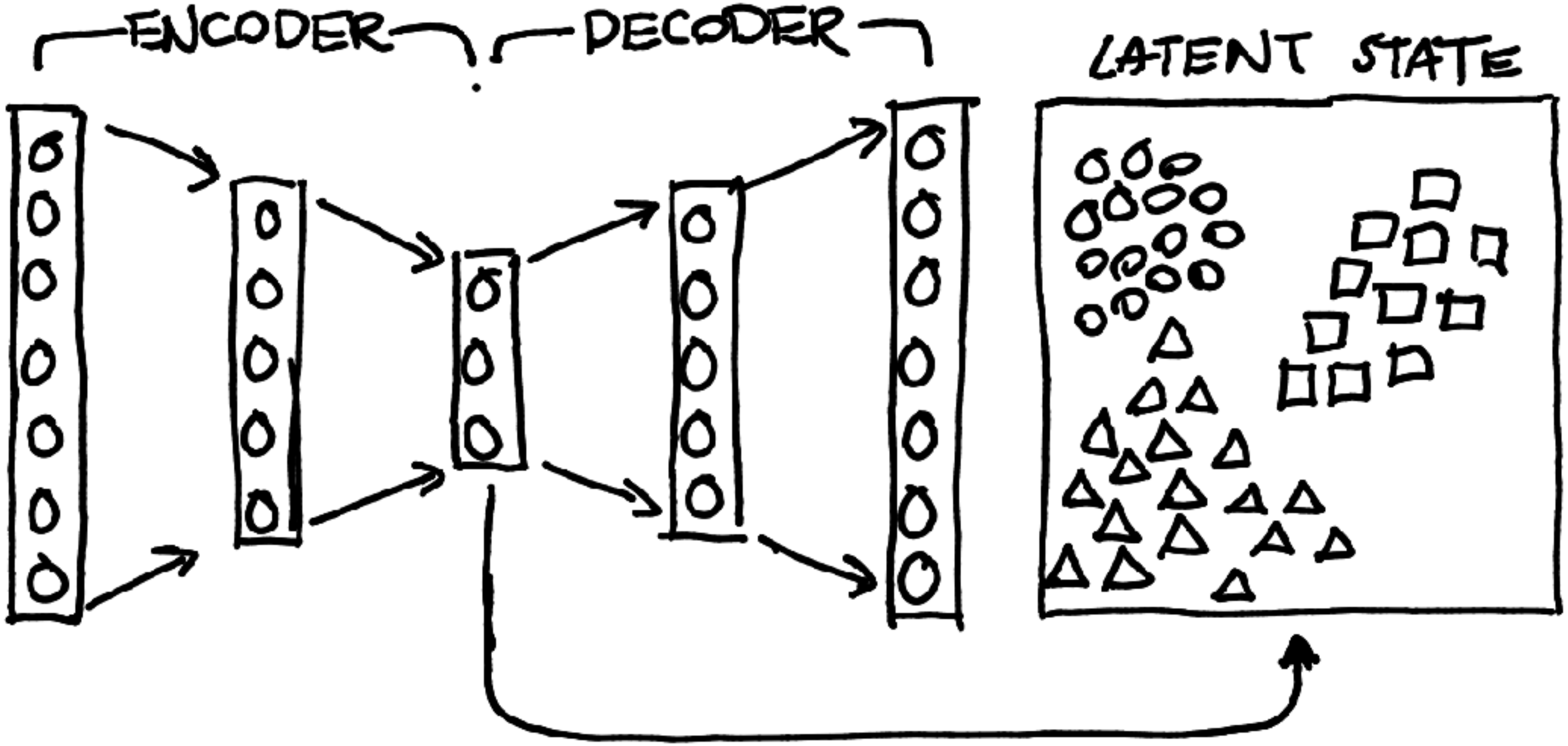
It also helps to maybe start smaller.

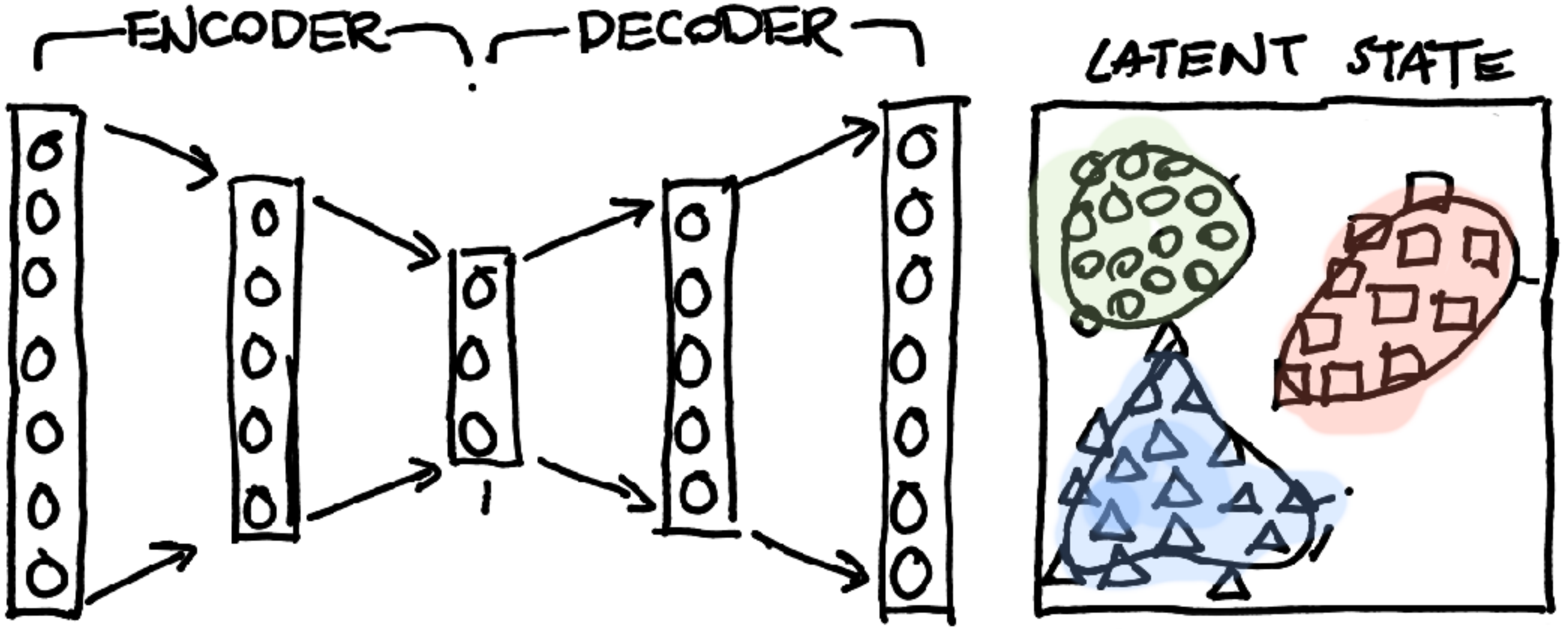
Generating Images

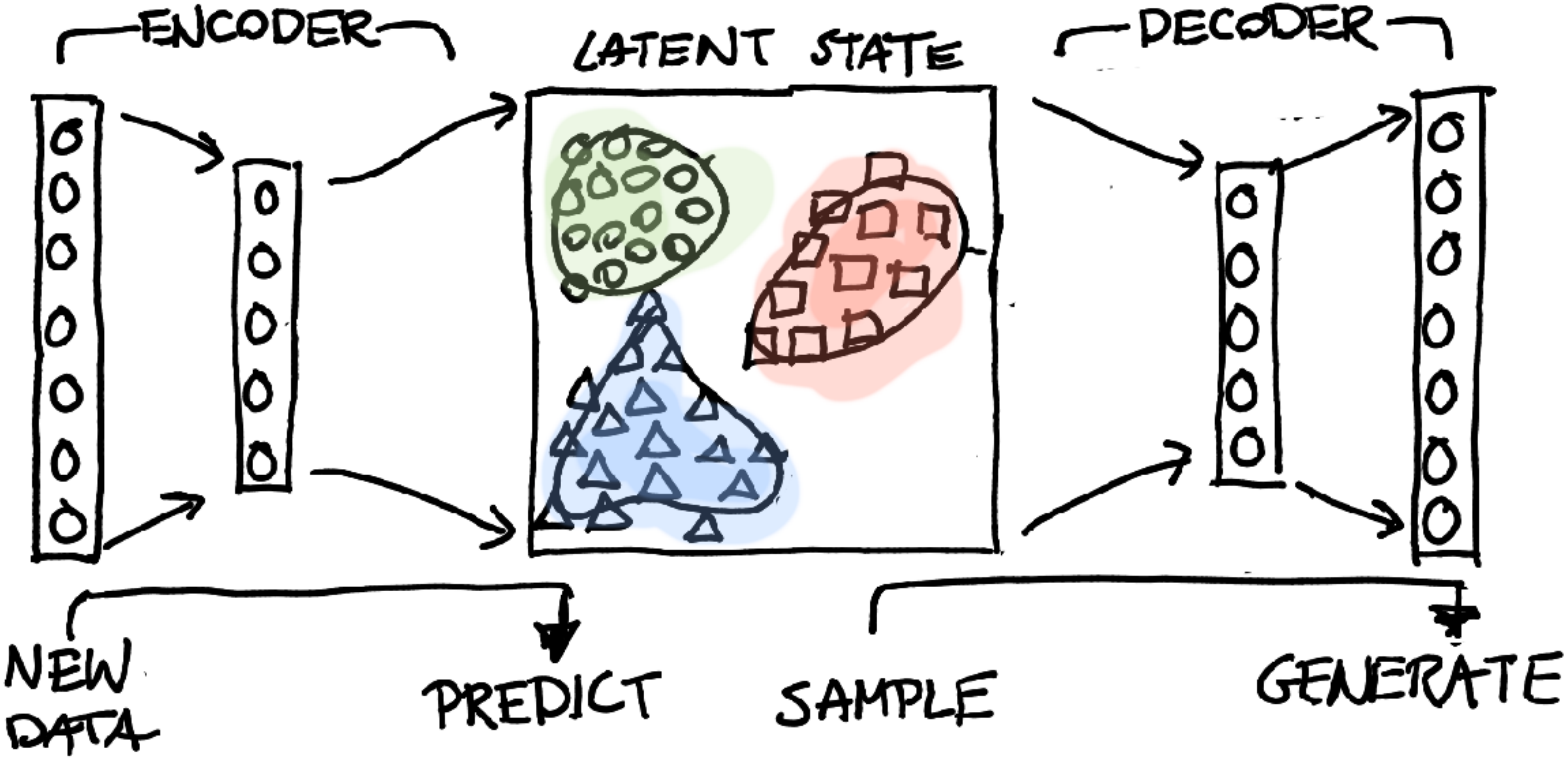
It's great to have an encoder, but how will we generate images with it?

What I am about to demonstrate is still work-in-progress, but it serves as a nice demonstration of really thinking about the problem you're trying to solve.

Note to self: if time is short -> skip this.


ENCODER
DECODER
LATENT STATE

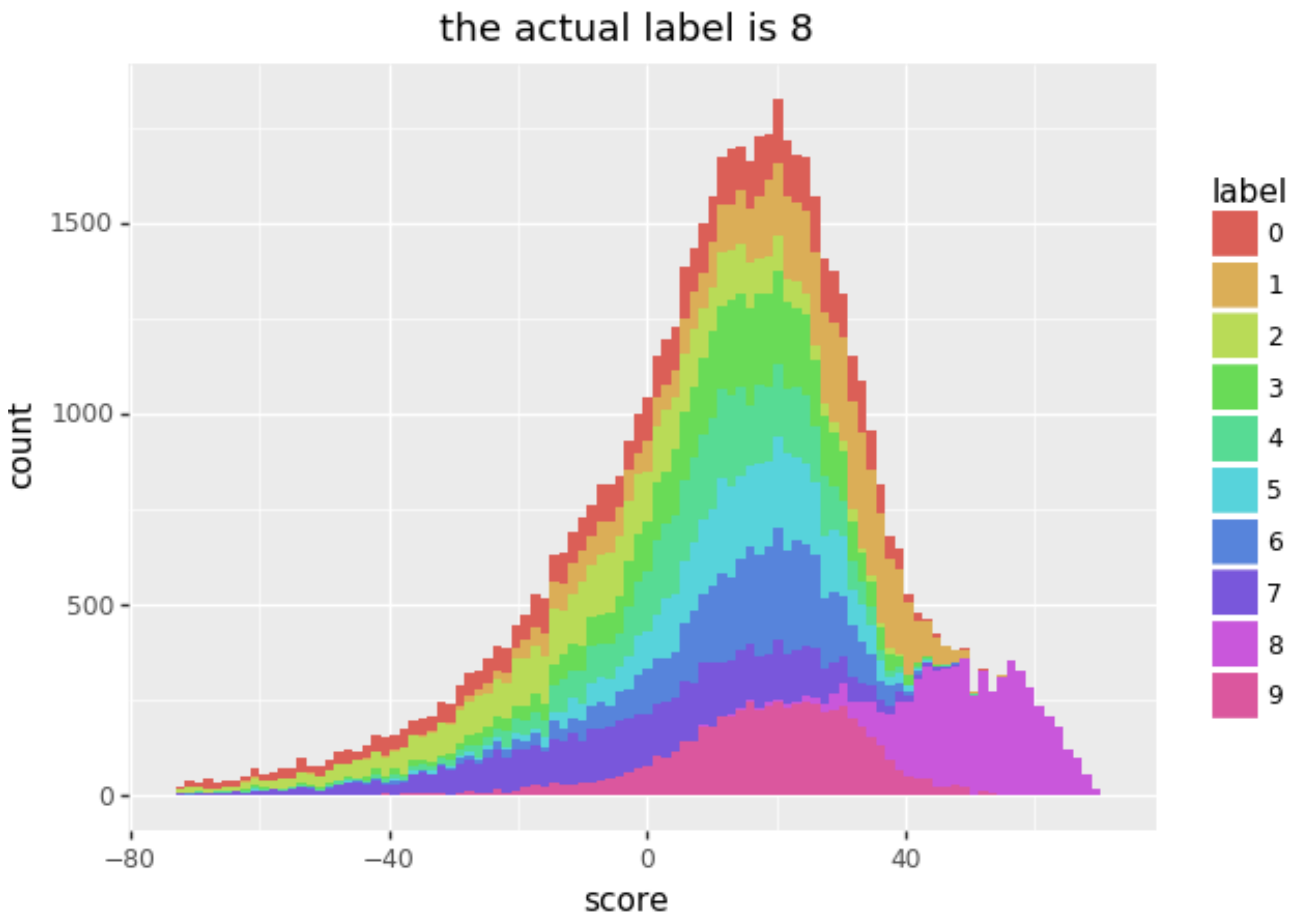

ENCODER
DECODER
LATENT STATE


ENCODER
LATENT STATE
DECODER
NEW DATA
PREDICT
SAMPLE
GENERATE

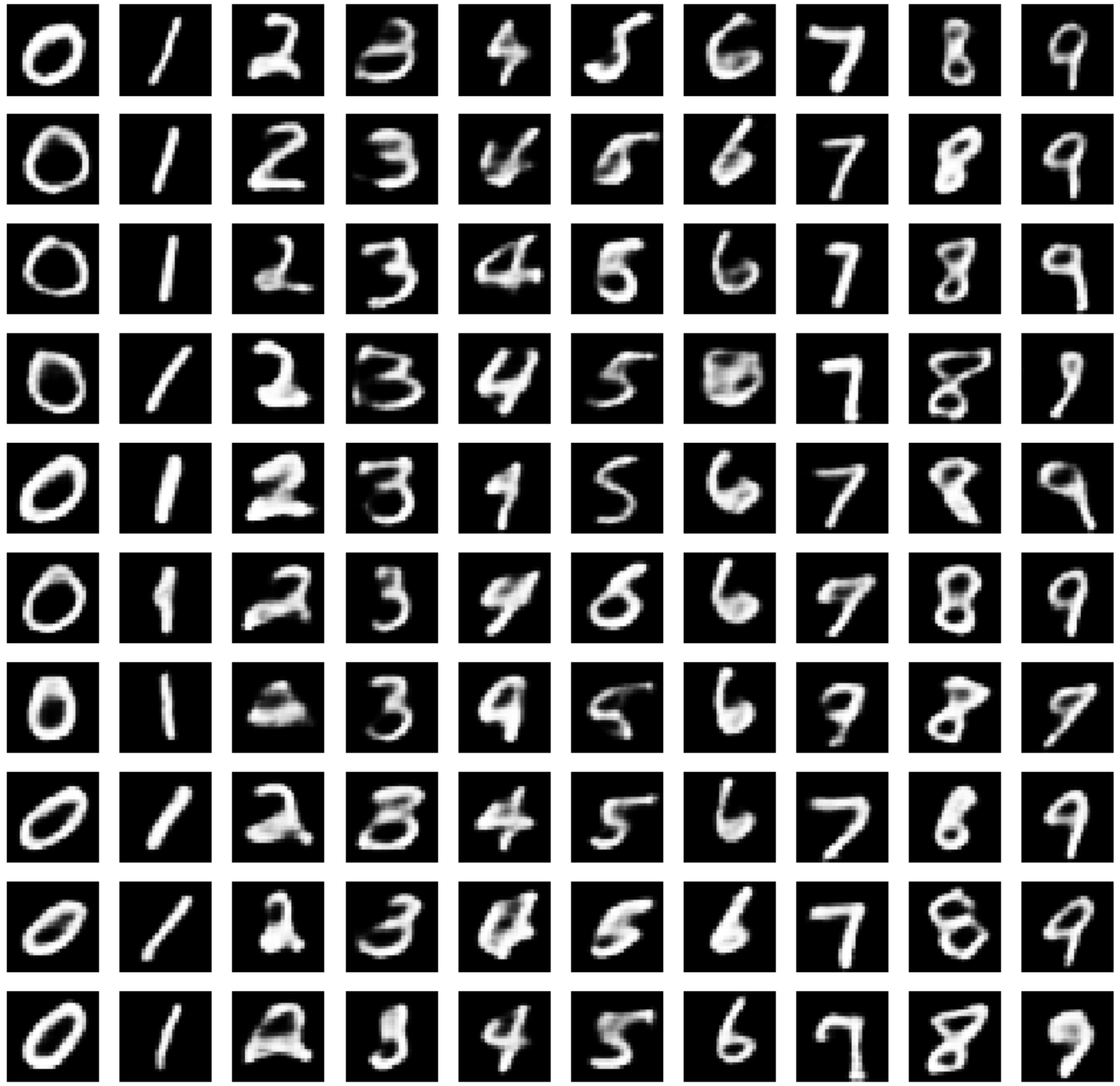
I can haz classification also!

This is the likelihood score given by `GMM[8]` to all samples, the color demonstrates the actual class.

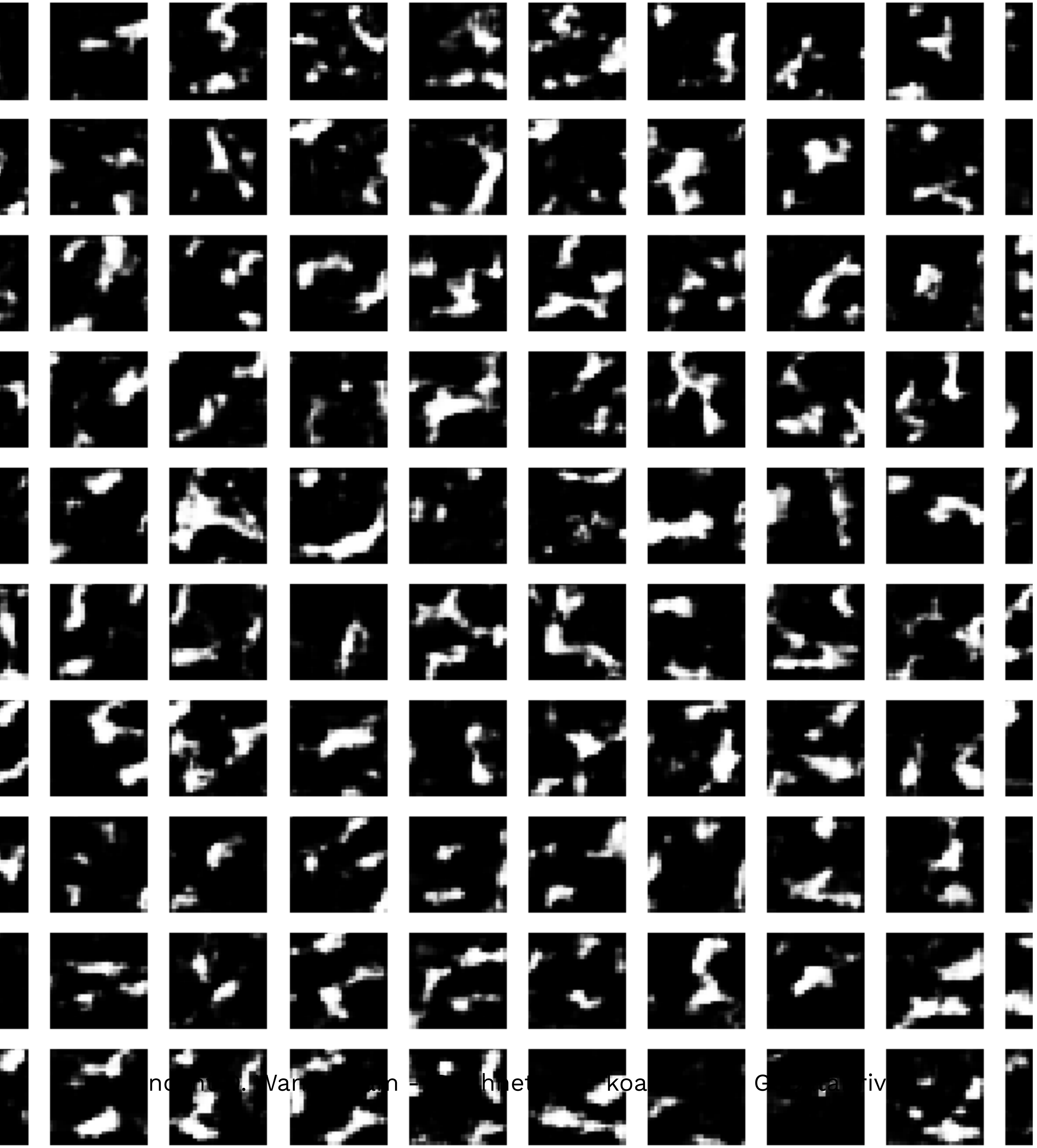
If the label is actually 8, we get a much higher likelihood.



Mnist Example Samples



Mnist Example Samples



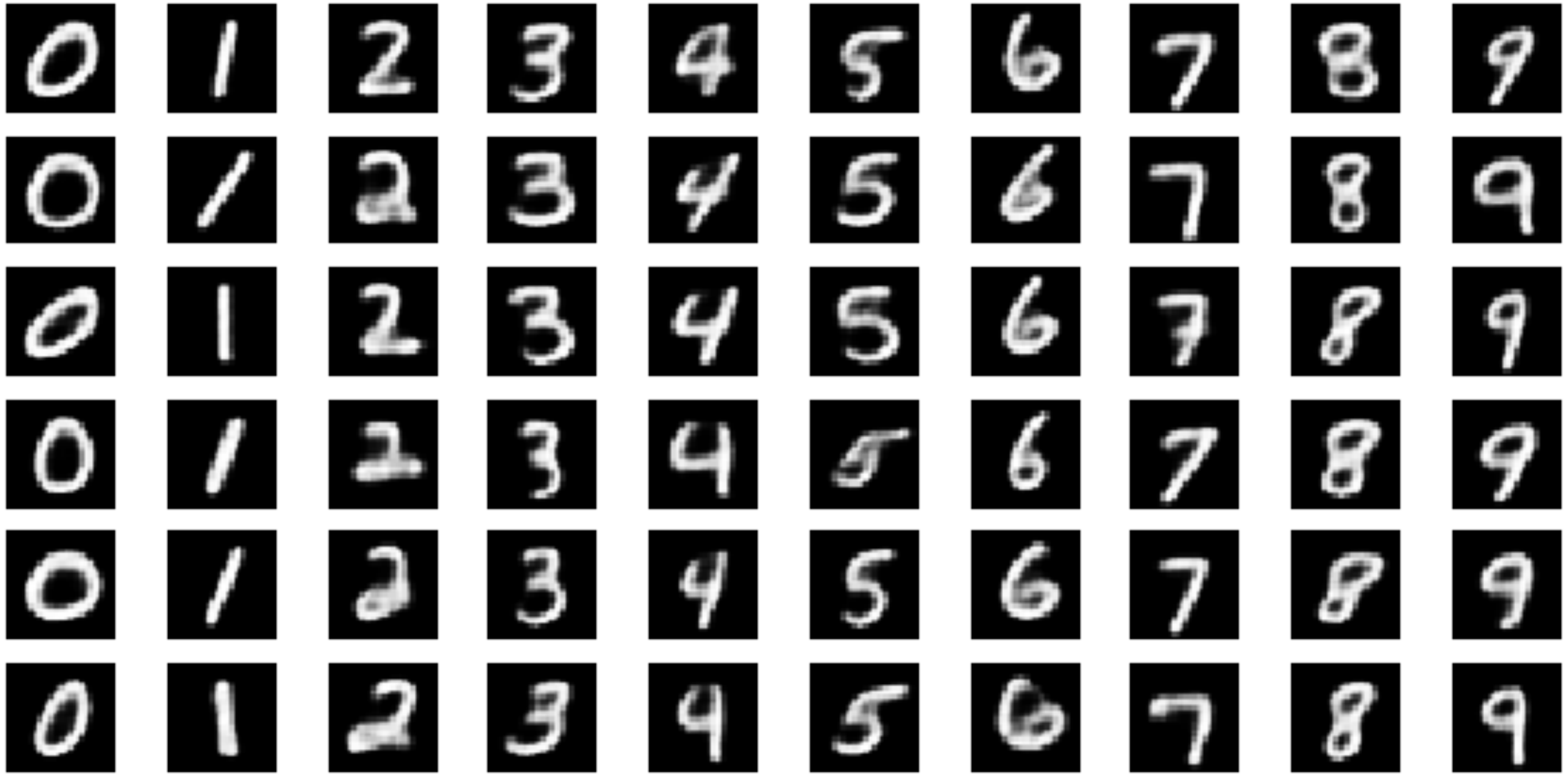
This is random noise

Before was a GMM sample in the decoder, this is pure random.

The decoder doesn't learn from this.

But there's more!

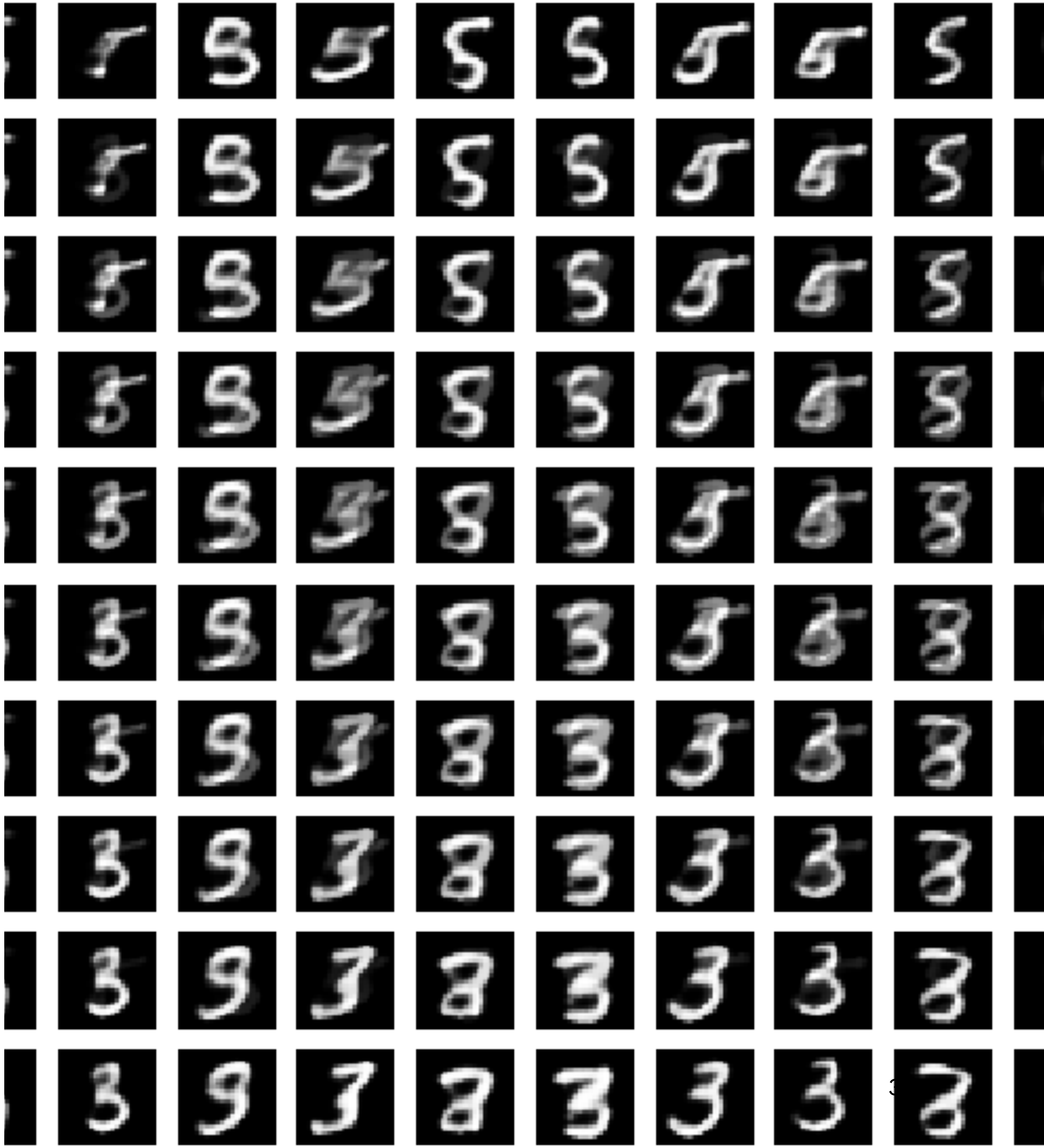
Behold, the GMM means have style interpretation.



But there is more!

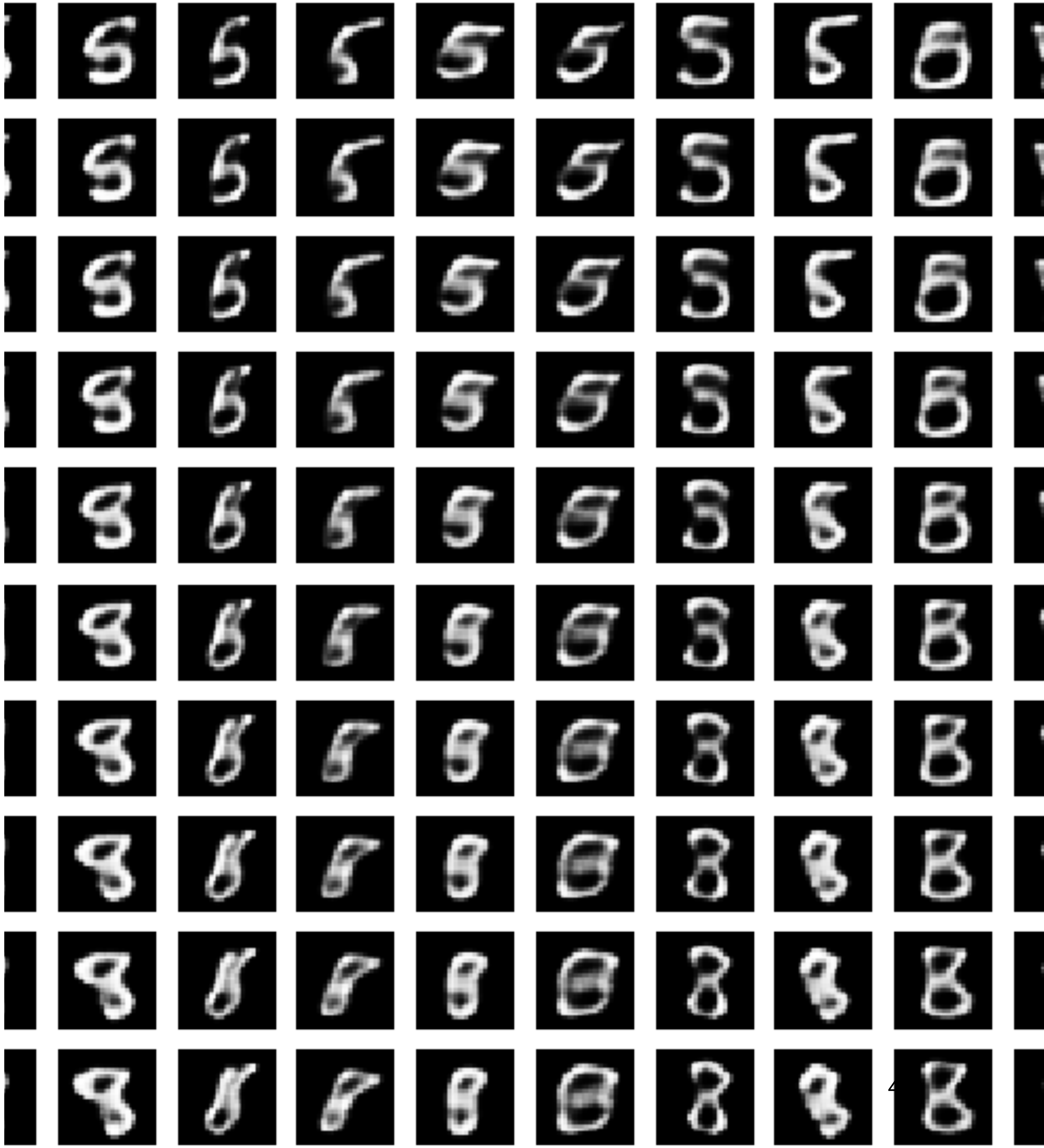
I want to generate one face and move on to another one. The visual must be fluid.

Here I am moving between a 5 and a 3 in encoded space. Note that the I am merely interpolating here.

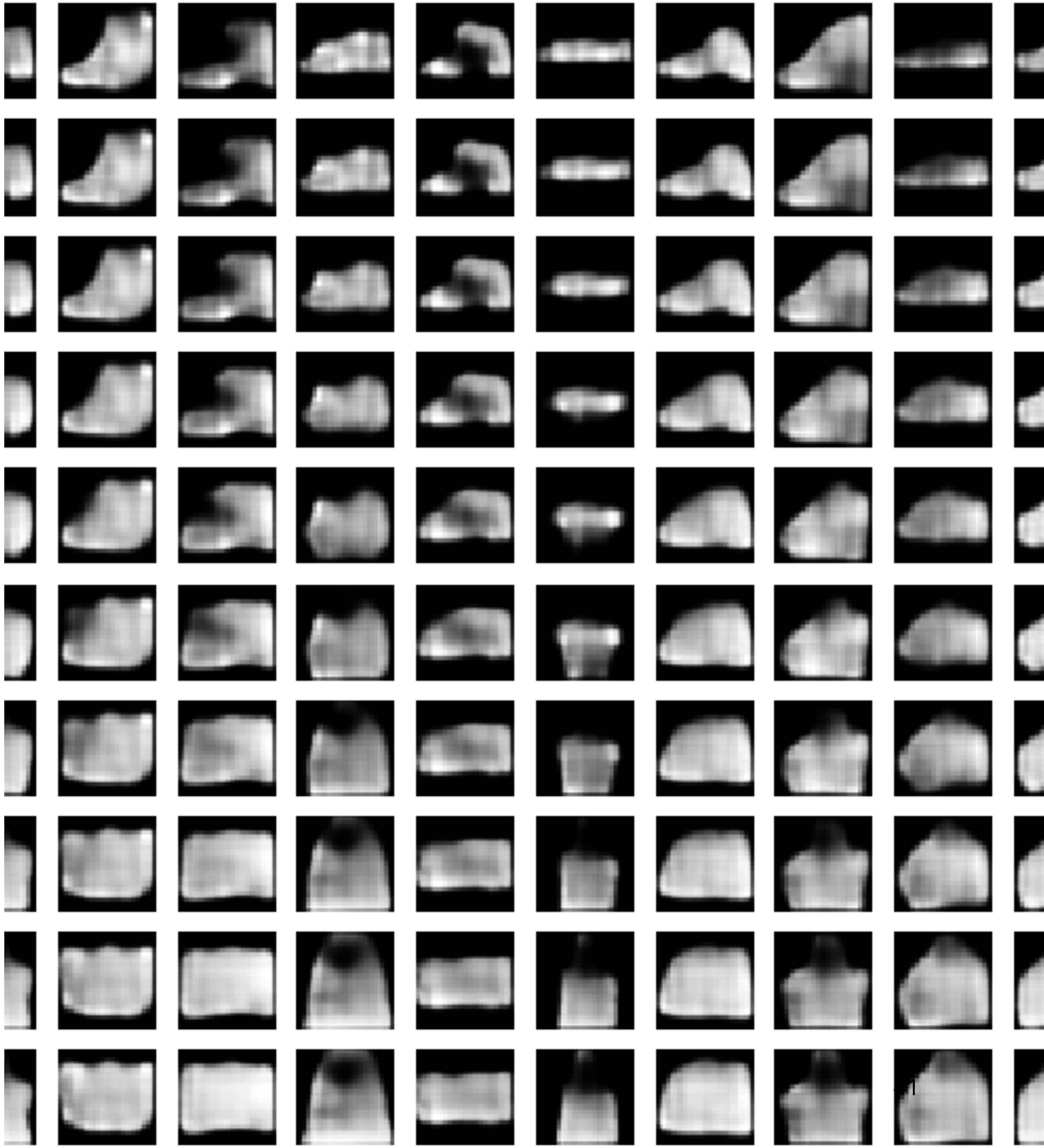


This is the same, but now I am interpolating in encoded space before I give it to the decoder.

It's not bad, but sometimes it seems that the 5 is moving to the 3 before moving to the 8. I'm not asking for it, but this is what it is doing.



Fashion Mnist is not super great either.

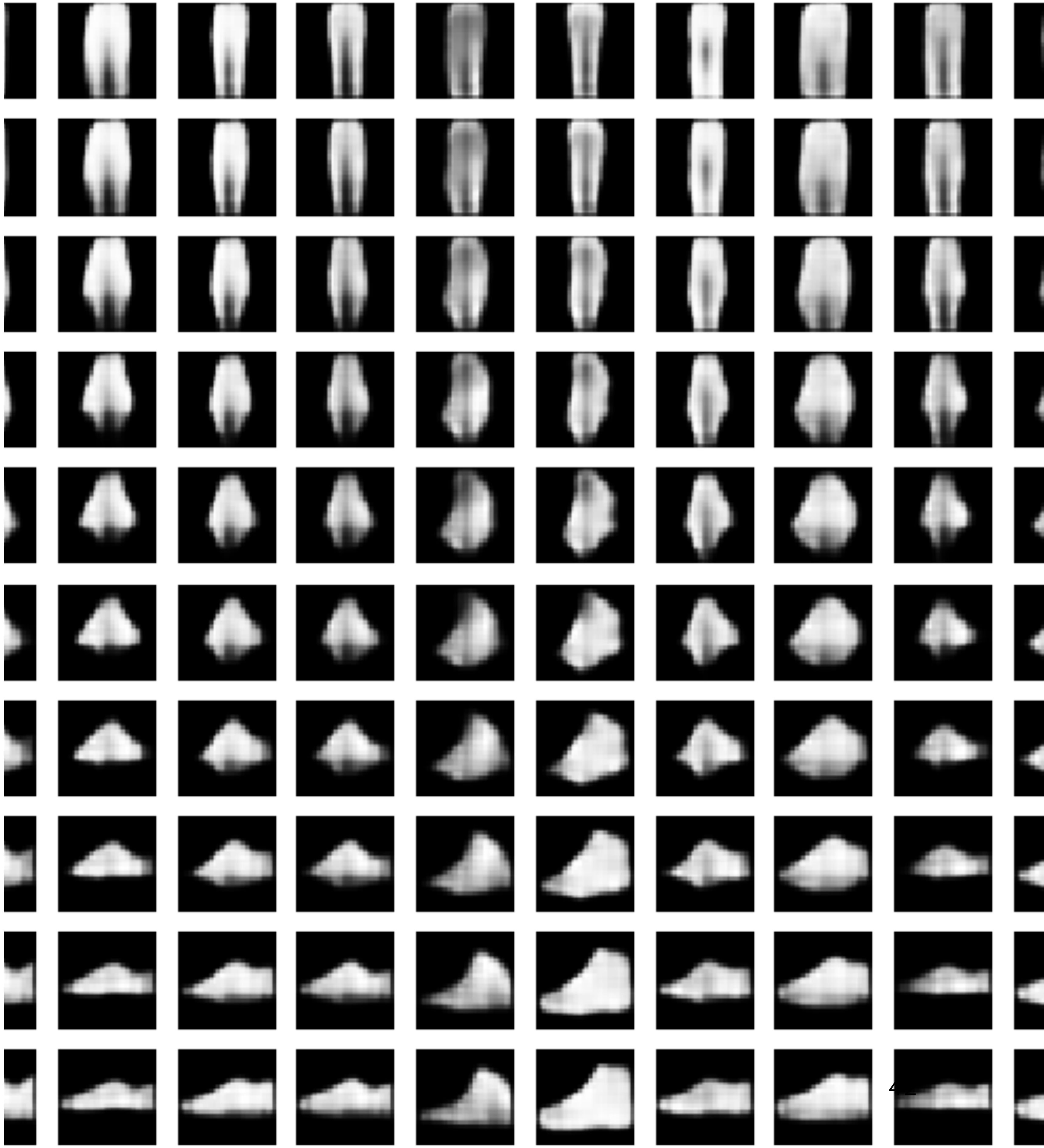


Fashion Mnist is not super great either.

So what do I do?

Do I start a hyperparam search on the decoder?

Or do I think?

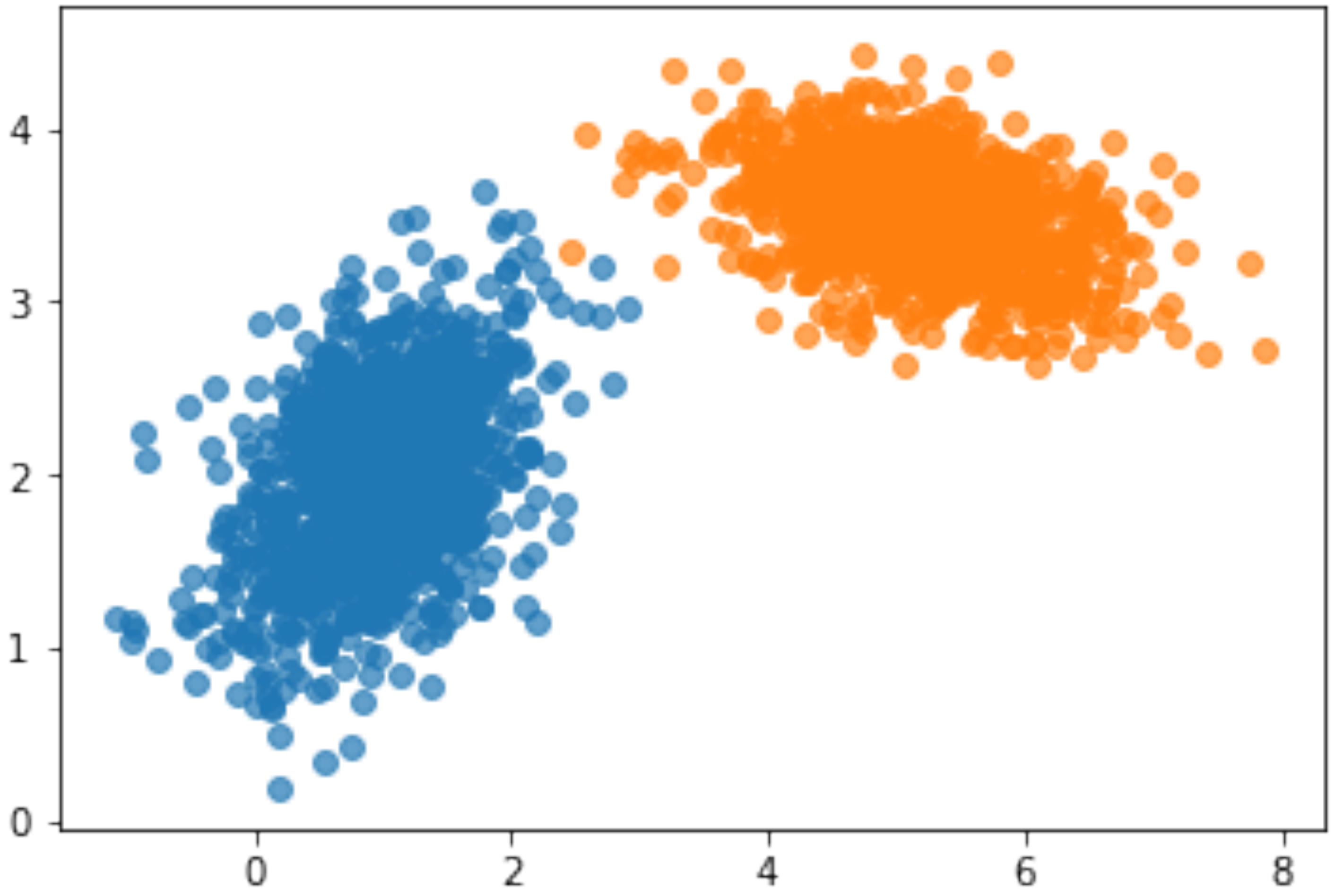


I did thinking

And again, it helped to consider the smaller problem.

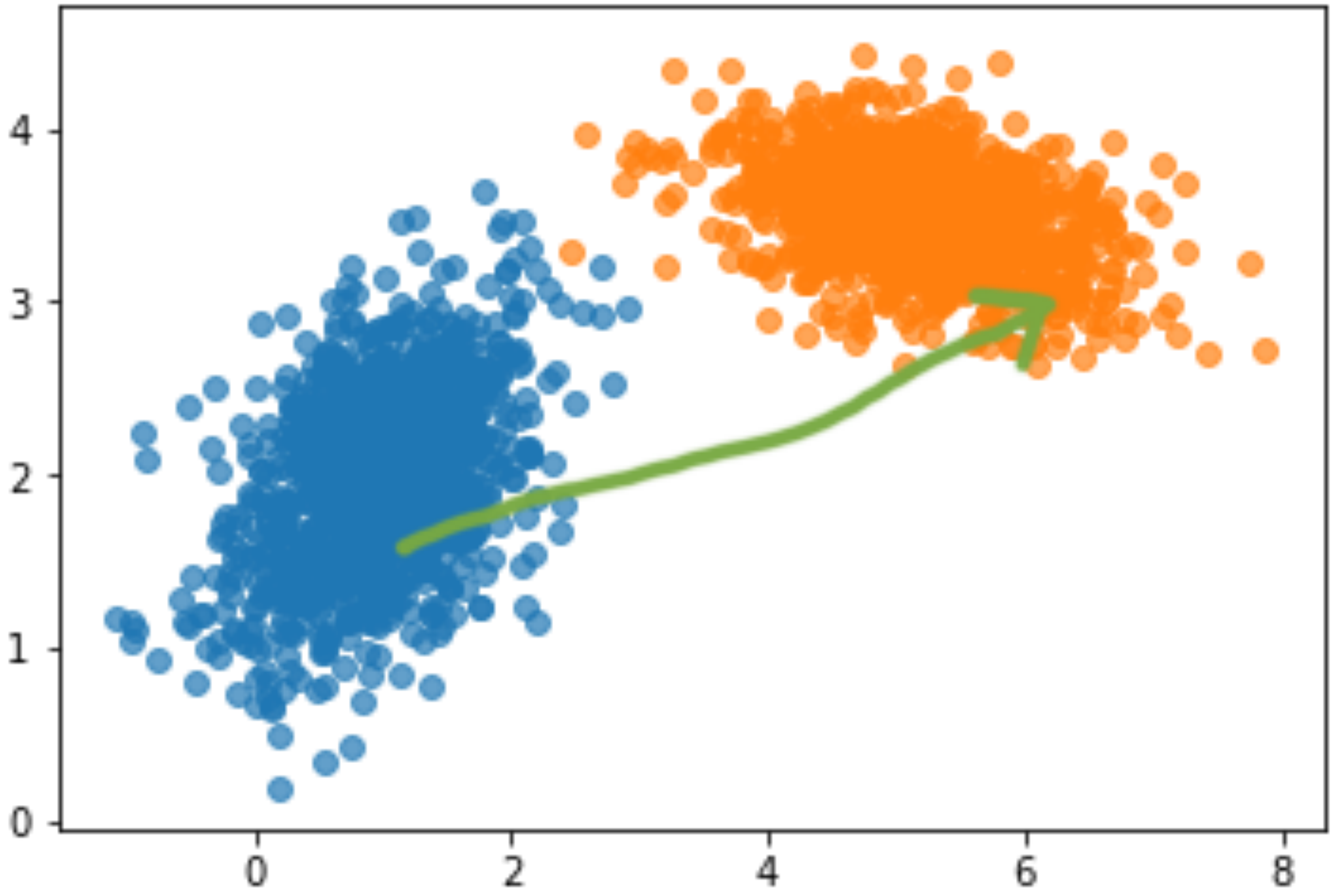
So lets do that.

Let's pretend that the thing to the right is the encoded space. One color represents 5s and the other represents 8s.



I did thinking

What is wrong with this path?

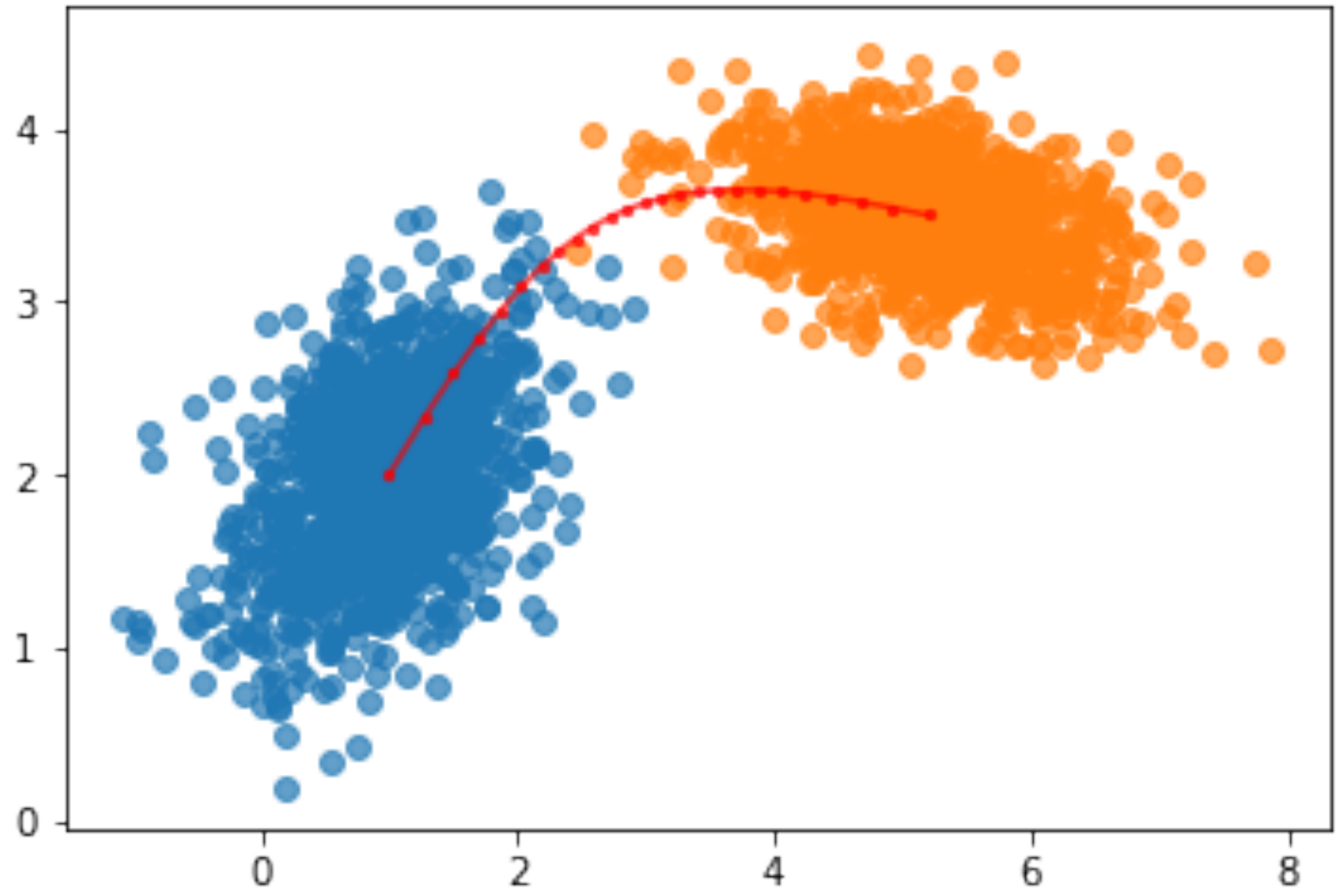


I did thinking

What is good about this path?

Also, do can we decribe this path?

If not with math, what about code?



```python
def likelihood(x):
    alpha = np.linspace(0, 1, alpha_steps)
    return tf.reduce_sum(alpha * dist1.log_prob(x) +
                         (1-alpha) * dist2.log_prob(x))

def grad(f):
    return lambda x: tfe.gradients_function(f)(x)[0]

...
for i in range(n_steps):
    gradient = grad(likelihood)(x)
    x = x + 0.01 * gradient
    ...
path = x.numpy()
```

Thinking is all I got.

I like my own approach here. It's simple and I am in control. I like the interpretations.

Note that I would **not** have come up with this if I merely copied the latest VAE/GAN code snippet off of github.

There's merit in reading other peoples work, but there's bliss in figuring out the problem by yourself too.

Sure takes time though, major downside.

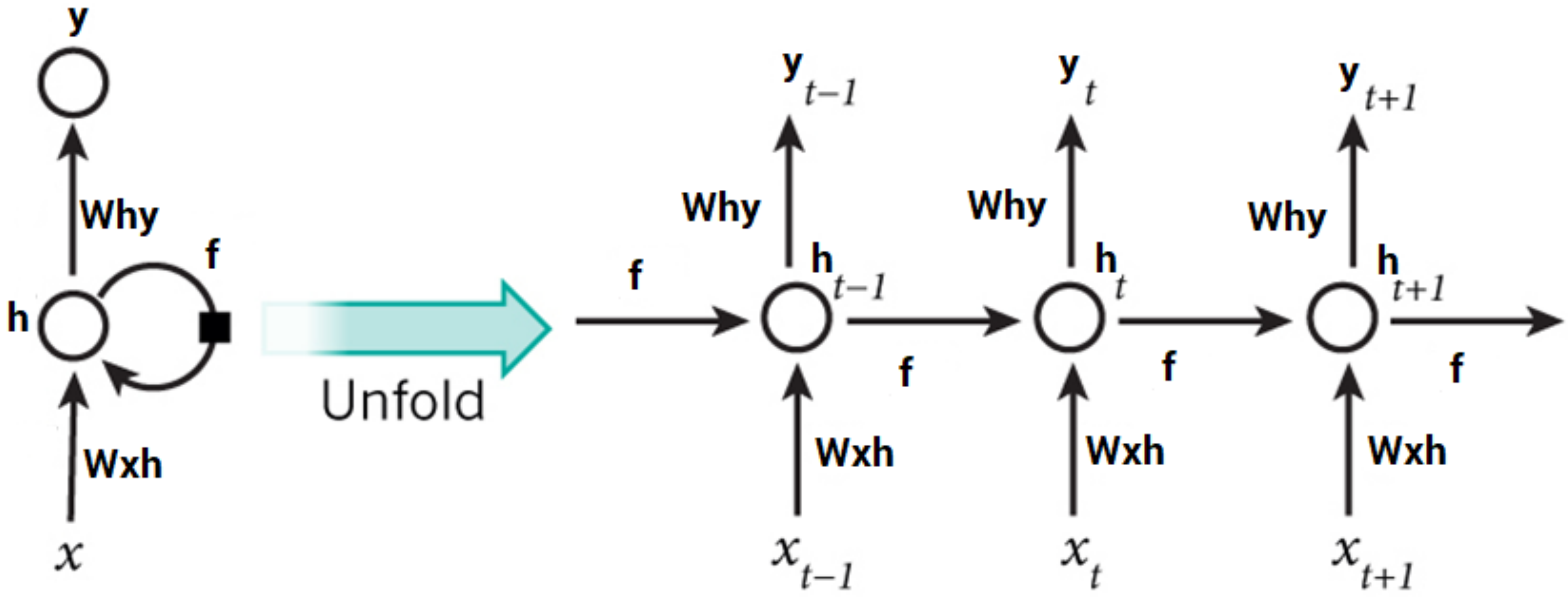
Sometimes I can ask a collegue when I'm stuck. I'm fortunate with my PhD friends who work in computer vision.

But I can either waste my time or spend my time on something worthwhile. It is hard to guess which is best but I hope it is clear that thinking is always a good idea. Massive hyperparameter search can be an act of intellectual desperation.

This is a moment I experience often in day-to-day work. It's a common junior mistake to follow stackoverflow instead of **JustThinking[tm]**. Please try to limit that.

Another Problem
More Hype
TimeSeries!

More Deep Learning Gafuffel



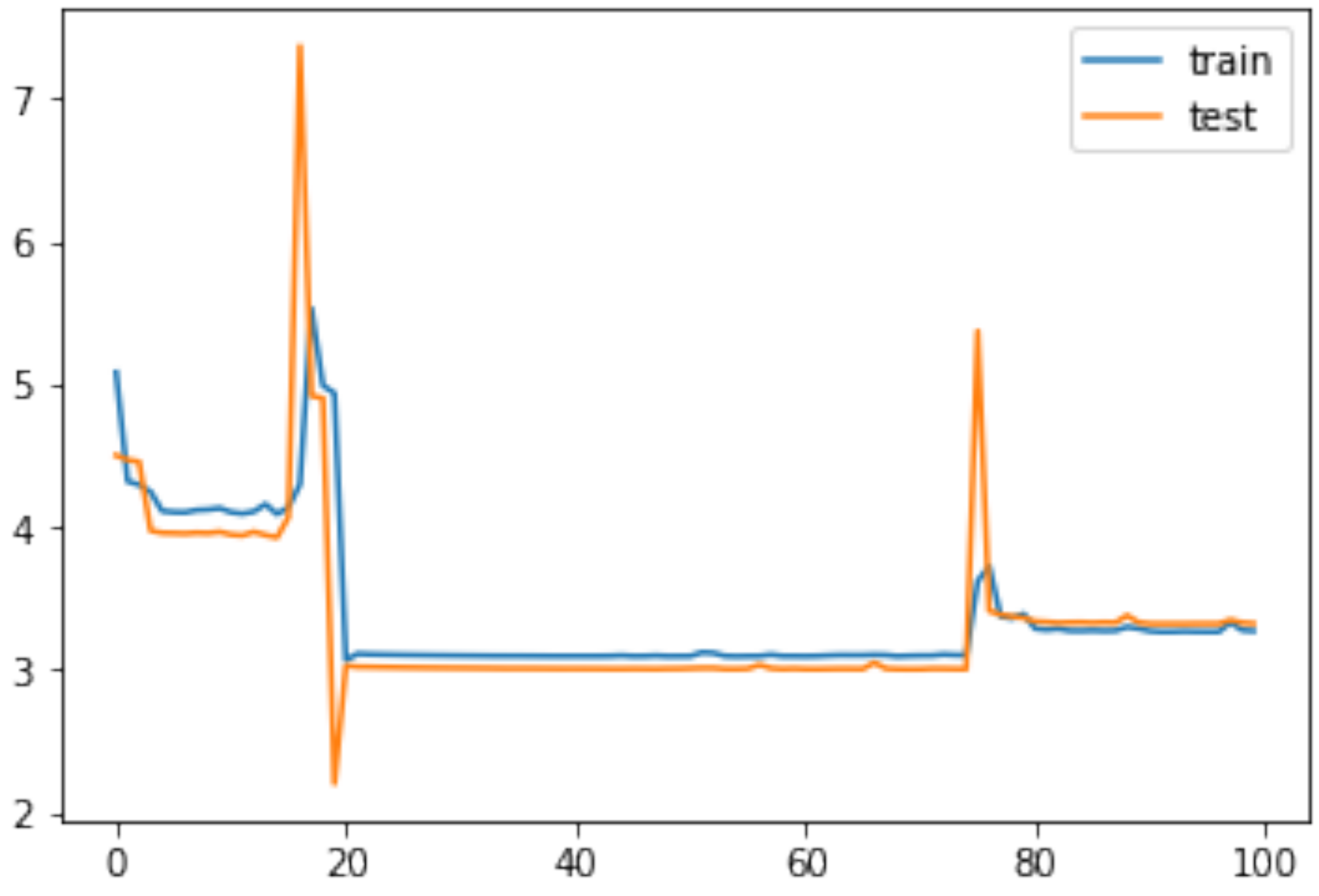
SimpleRNN: Predict Season based on Temperature

```
model = Sequential()
model.add(SimpleRNN(50,
                    input_shape=inpt_shape,
                    return_sequences = True))
model.add(SimpleRNN(50))
model.add(Dense(10))
model.add(Dense(4))
model.compile(loss='binary_crossentropy', optimizer='adam')
```

What could go wrong?

SimpleRNN: Predict Season based on Temperature

The word simple seems misplaced.

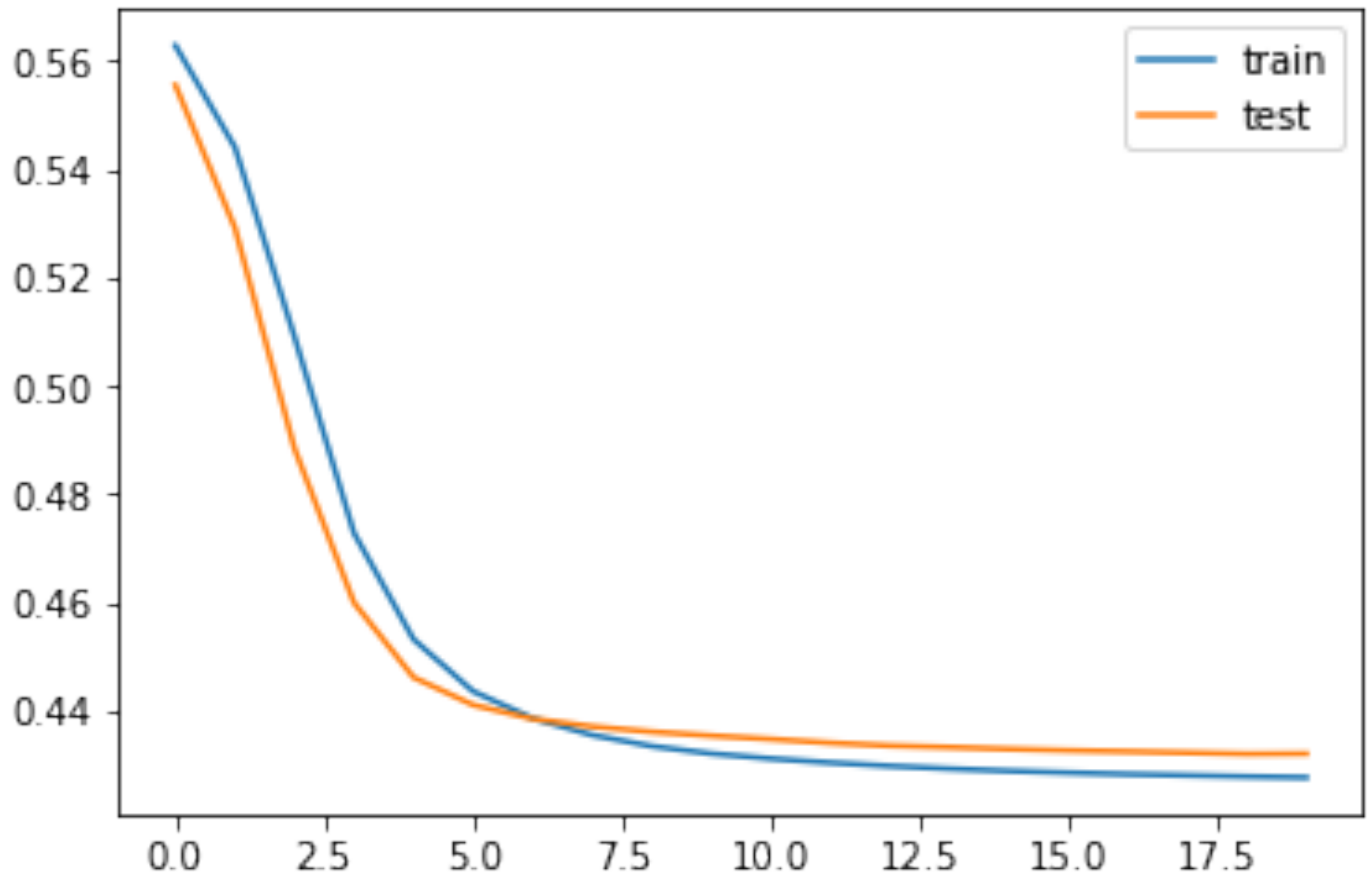


I'll replace it with an LSTM to help a bit with vanishing gradient but the main fix is the final activation (OOps).

```
model = Sequential()
model.add(LSTM(50,
               input_shape=inpt_shape,
               return_sequences = True))
model.add(LSTM(50))
model.add(Dense(10))
model.add(Dense(4, activation='softmax'))
model.compile(loss='binary_crossentropy', optimizer='adam')
```

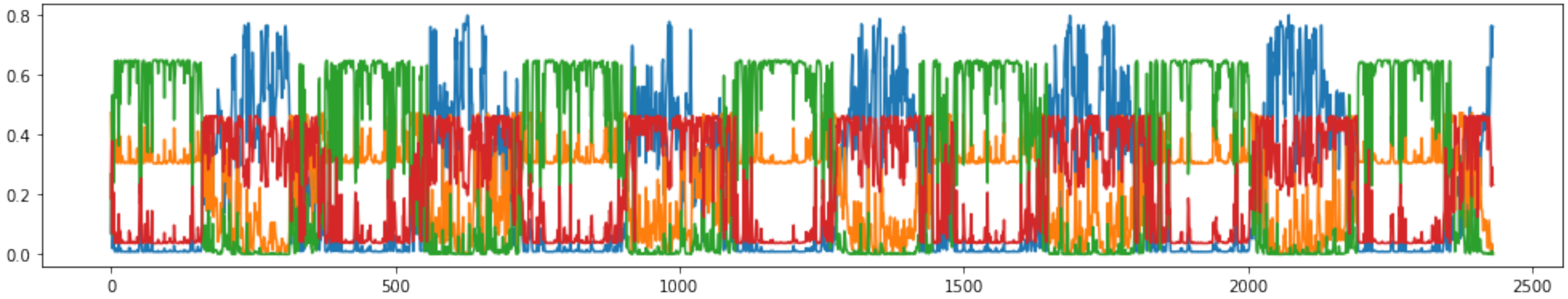
Thinking Seems to Prevail!

The word simple still seems misplaced.



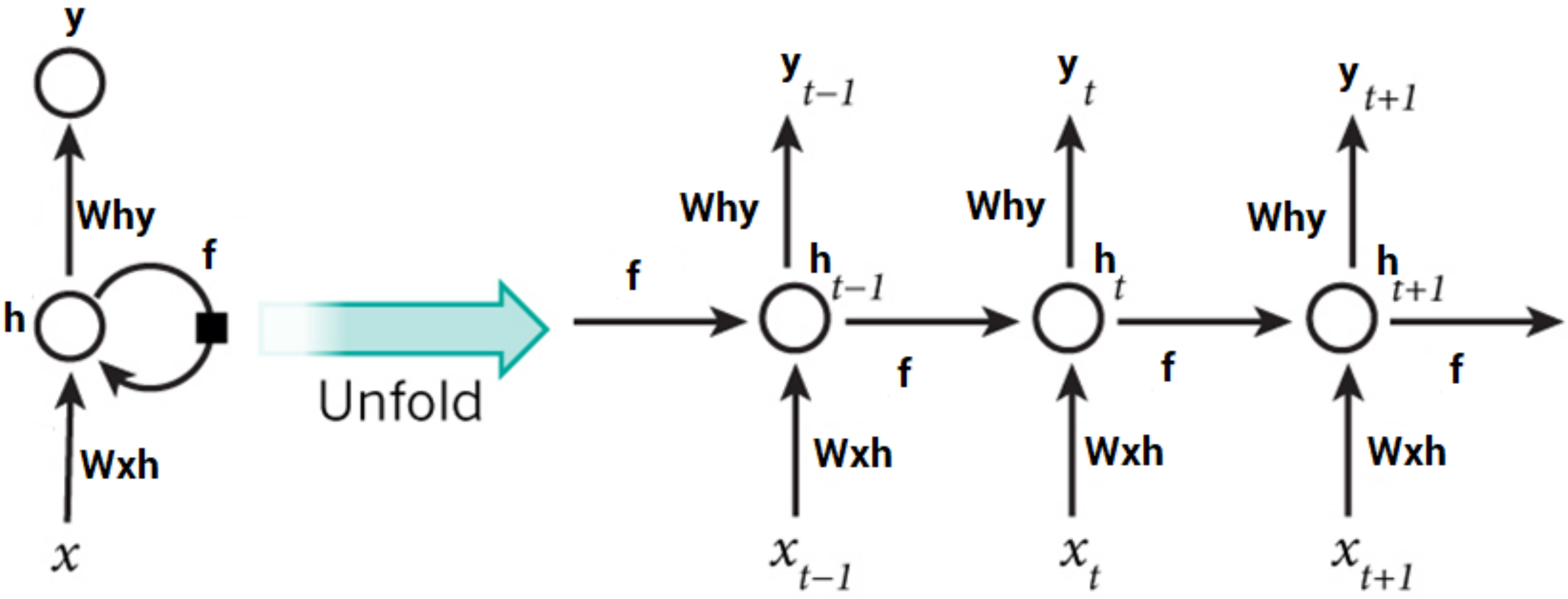
Thinking Seems to Prevail!

Seems is the keyword: this is the output.

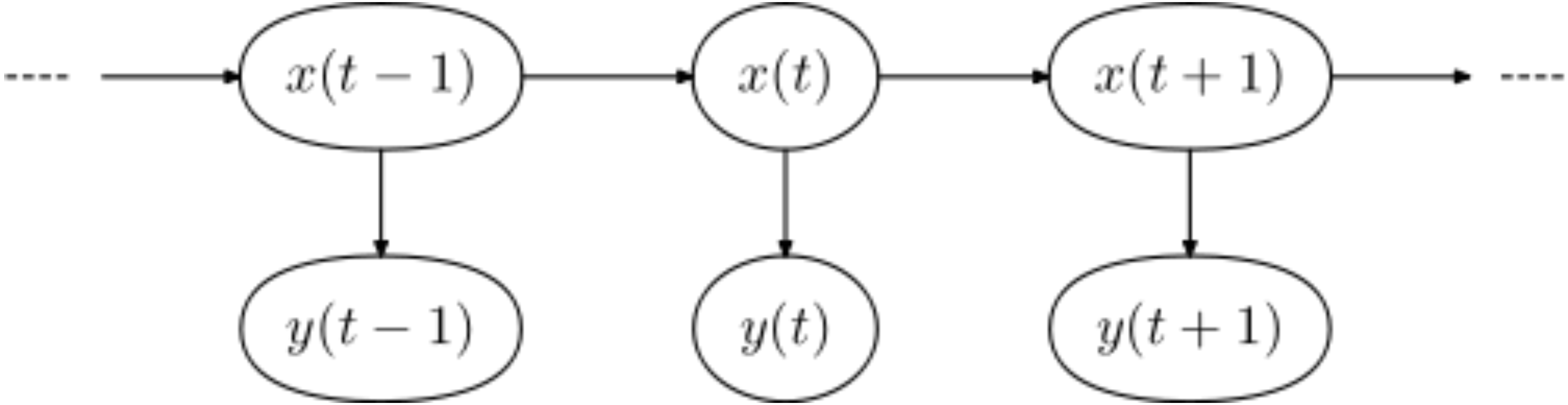


It has little notion of seasons over time. It might overfit but it is super hard for me to push the model in the right direction.

Can I help the model mentally? Yes.



"If only there was a different way of modelling"



Hidden Markov Heroes

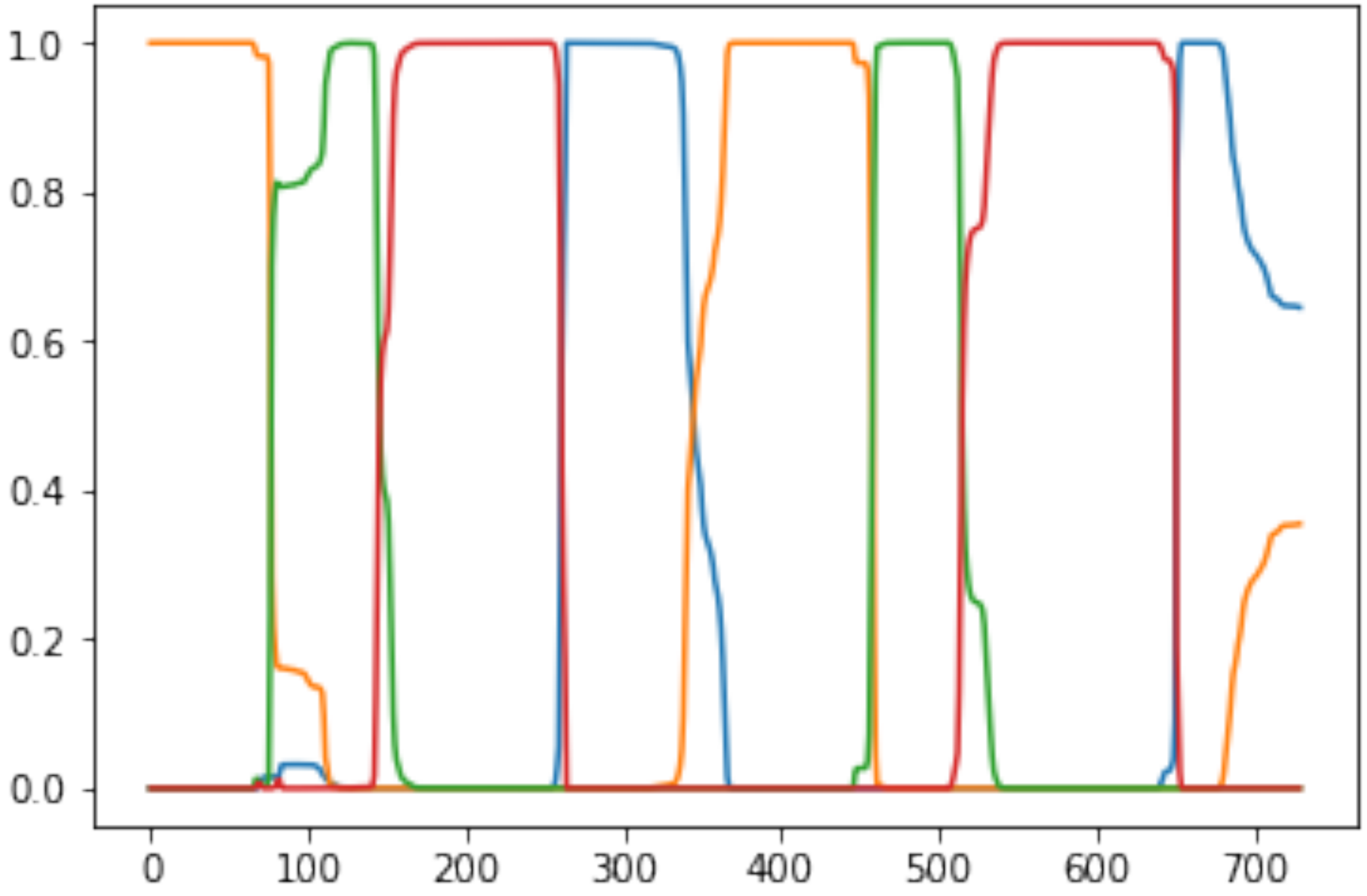
```
import pomegrenate as pg

dists = [pg.NormalDistribution(103, 48),
         pg.NormalDistribution(64, 55),
         pg.NormalDistribution(173, 53),
         pg.NormalDistribution(211, 40)]

trans_mat = np.array([[0.999, 0.001, 0.0, 0.0],
                      [0.0, 0.999, 0.001, 0.0],
                      [0.0, 0.0, 0.999, 0.001],
                      [0.001, 0.0, 0.0, 0.999]])

starts = np.array([0.25, 0.25, 0.25, 0.25])
model = pg.HiddenMarkovModel.from_matrix(trans_mat, dists, starts)
```

I really like this output better



Time of a final, but amazing, demonstRation.

The last example demonstrated something. We wondered what properties the model had to have and we found a model that works very well. Even though keras has an amazing/flexible API, it feels overly verbose at times.

But, what do we do if we don't have a mold that our fits our data? When can we get away with feature engineering and when do we need to design a model? And how do we deal with the implementation?

Conclusion Time

If you understand the solution better than your problem; you're doing it wrong.

Having blind faith in an algorithm is asking for trouble, especially if the algorithm in question isn't articulate. In most situations, just thinking in simple terms and wondering "what might be the main problem and can we test it" is a good place to start.

Lesson: street-knowledge > book smarts.

So what works well?

It helps to prefer systems that are articulate:

- they help you debug
- they help you explain
- they help you monitor
- they help you maintain

Typically these algorithms are also the simpler ones, we might celebrate the moments when we can apply them.

Otherwise, some hints.

- **Admitting to Confusion** If you don't understand whats going on, thats perfectly fine if, and only if, you admit it. When in this state: debug mode!
- **Baby Steps** Don't change everything at the same time, try to be able to explain your incremental improvements.
- **Ask for Critique** You want others to critically look at your work. Ask them "What might go wrong here". You may consider something you missed earlier.
- **Compare to Super Simpel** if only as a benchmark, it is a good sniff test. Start simple and only make it complex when you need to.
- **Visualisation** Plots can show you things you didn't expect and also things the unit-tests don't pick up. Plot early and often where-ever possible.
- **Let it go** Sometimes it helps to just take a step back and worry about it tomorrow. Creativity doesn't work under stress.

Was the real issue in 2016

the fact that we weren't using ApacheSpark[tm]?

Is the real issue in 2018

the fact that we're not using DeepLearning[tm]?

Or is the problem that we don't understand the **actual** system/problem by just staring at a jupyter notebook.

Do you talk with the end-users? Do you regularily drink coffee with the working folks down the factory floor?

The underlying data science is often irrelevant to an end user. Whether inserting your algorithm is O(log n)/O(n^2) is largely irrelevant. Same with ROC curves or a super complex deep tensortorch inplementation. Your customer has a workflow in their mind, and if your software doesn't contribute to it, it's noise that gets in the way.

It's an open door, I know. We may be not be able to solve the anythign for a user if we don't remember to think about the problem.

Where possible, think/learn beforehand/early. Questions?