The Dancing Screen

Independent Study
Mary Karroqe

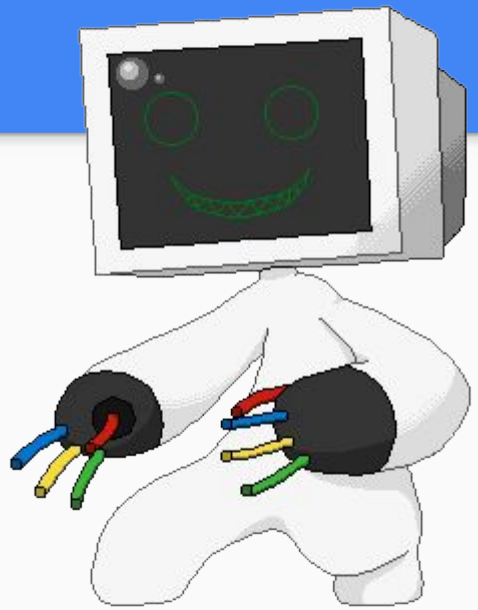
Overview

- Project Description
- Original Approach
- Final Approach
- Data + Pre-Processing
- RNN
- Results
- Future Work

Project Description

The Problem

- My computer doesn't dance
- I wanted to have fun with my senior design project
- Not saving the world here



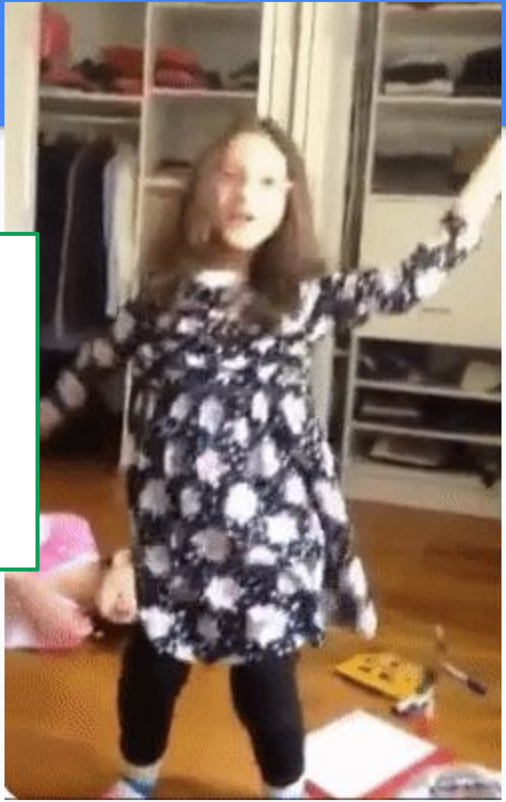
"Monster Dancing Sticker" via [Giphy]

The Project

Can I teach my computer to generate dance videos?

Applications

- Quarantine Dance Parties
- Looking cool in front of the children in your life
- Joy



“DISCO FEVER” Fortnite Dance

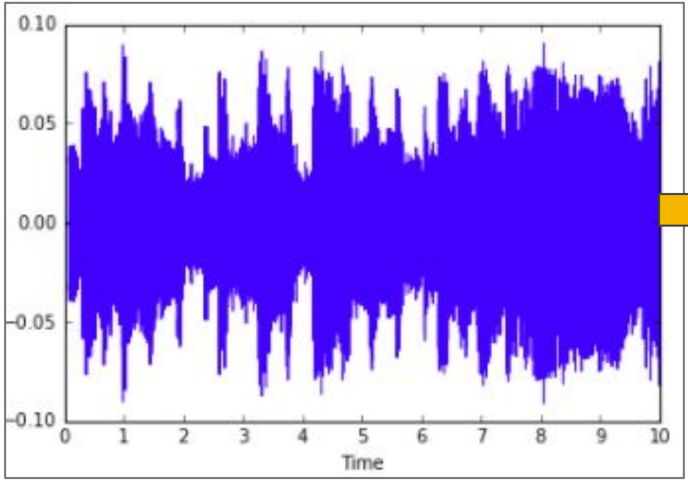
Applications

- Observing how the model breaks down dance moves
- Potential for dance instructors?
- Kids toys?

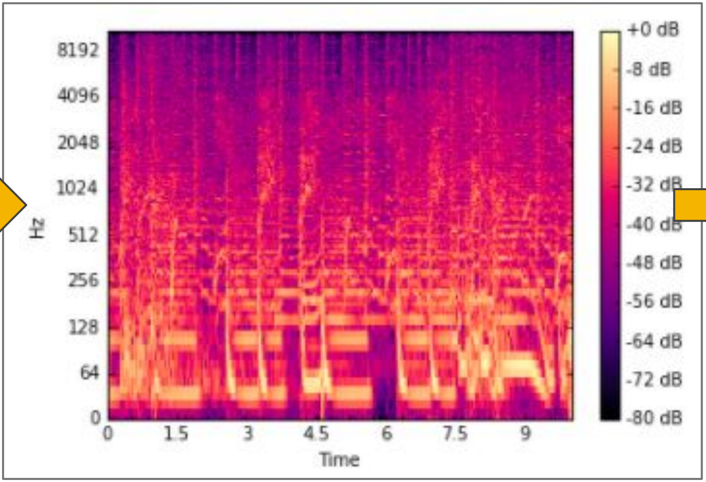
Original Approach

Original Approach

Audio file



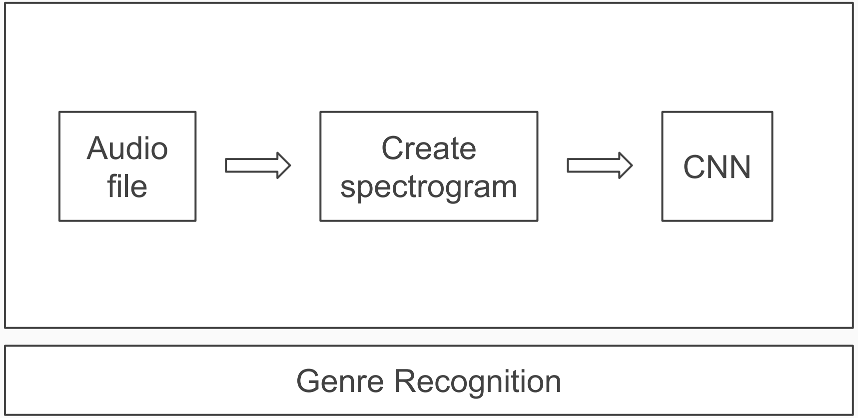
Create spectrogram



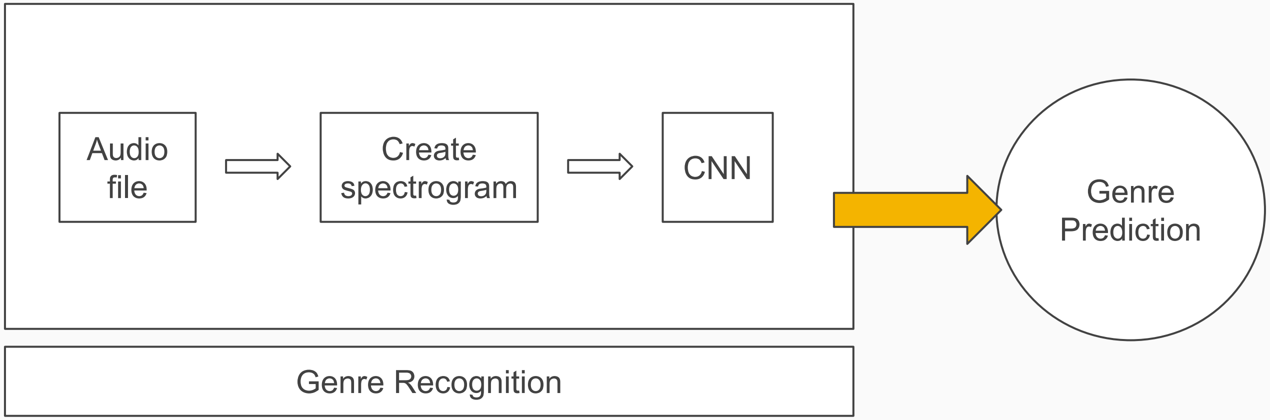
CNN

Genre Recognition

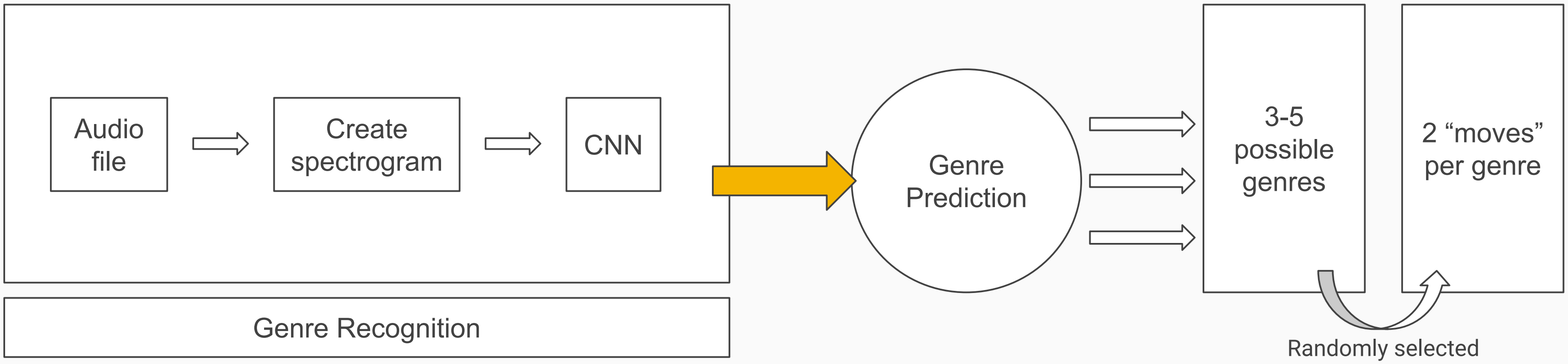
Original Approach



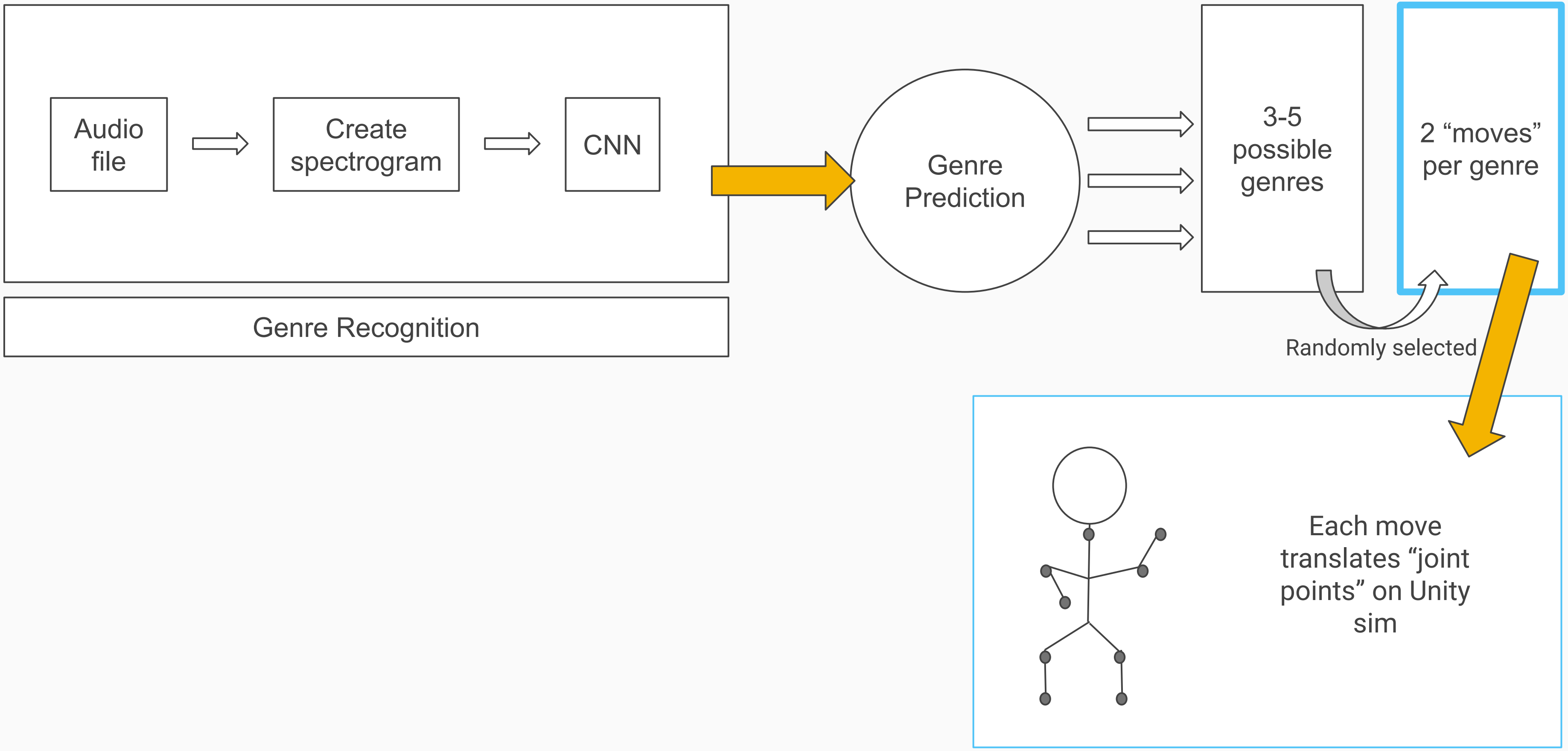
Original Approach



Original Approach



Original Approach

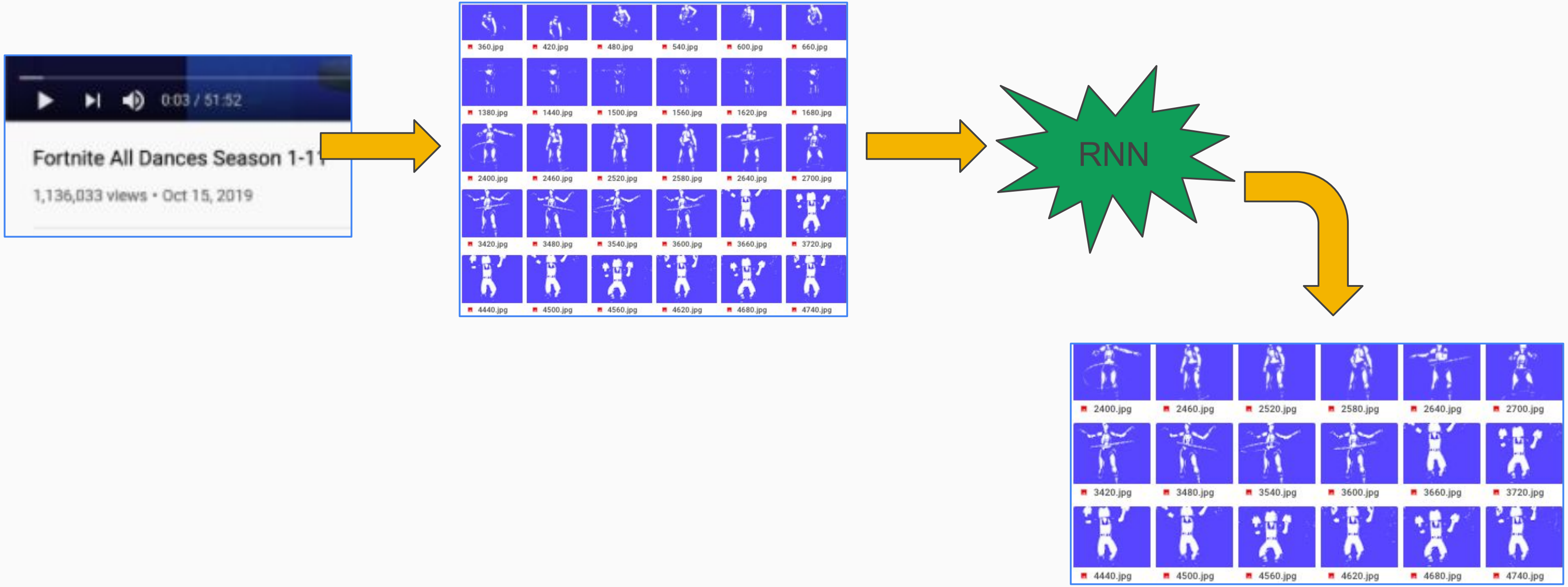


Problems with this Approach

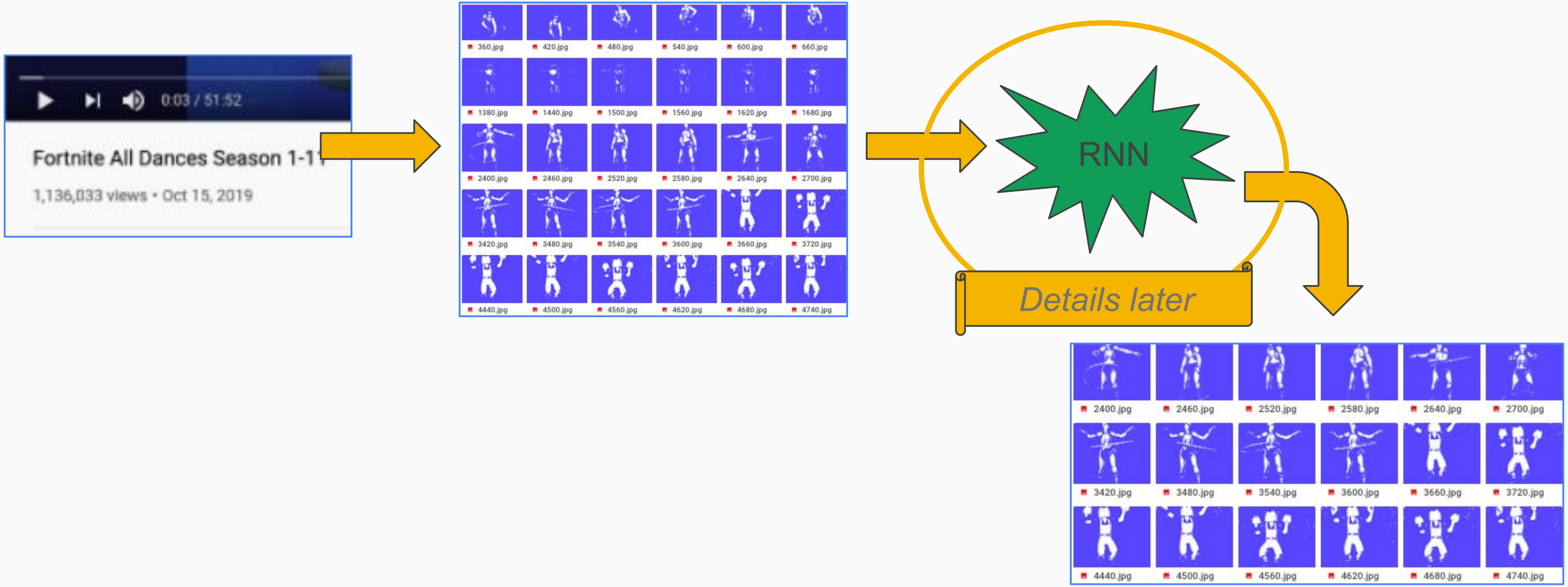
- The decision to start with genre recognition was arbitrarily chosen
- My genre recognition predictions from last semester were only slightly more accurate than random
- Convoluted pipeline; many aspects would be hard-coded
- Limiting the output would result in boring output
- Learning Unity + connecting to CNN output

Final Approach

Final Approach



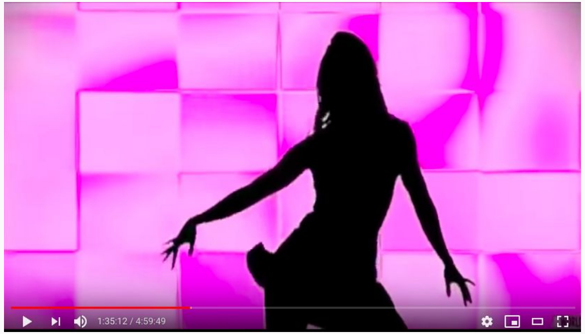
Final Approach



Data +
Pre-Processing

Input Data Options

Green Screen Silhouettes, YouTube playlist

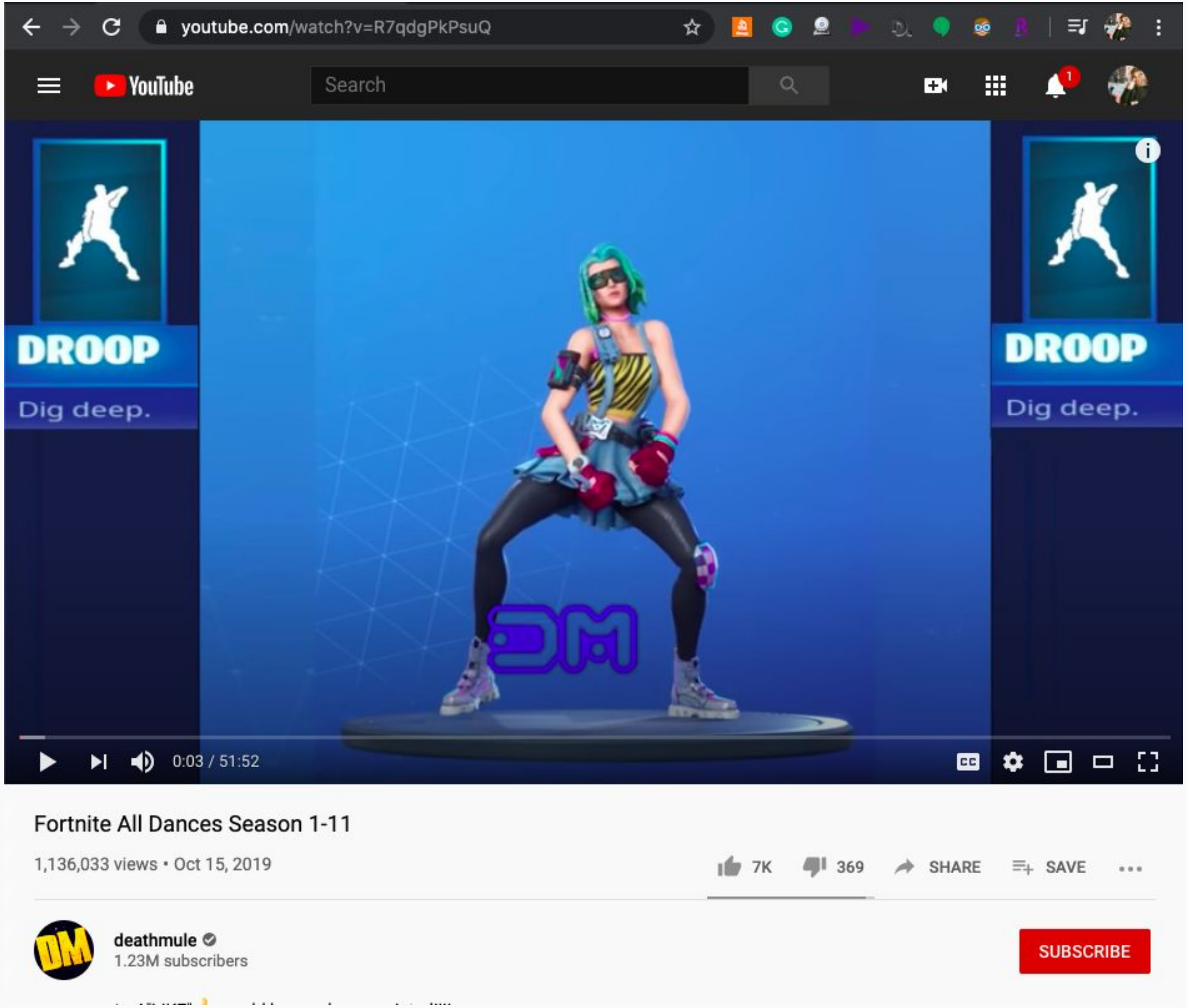


“Shadow Dancers”, YouTube video

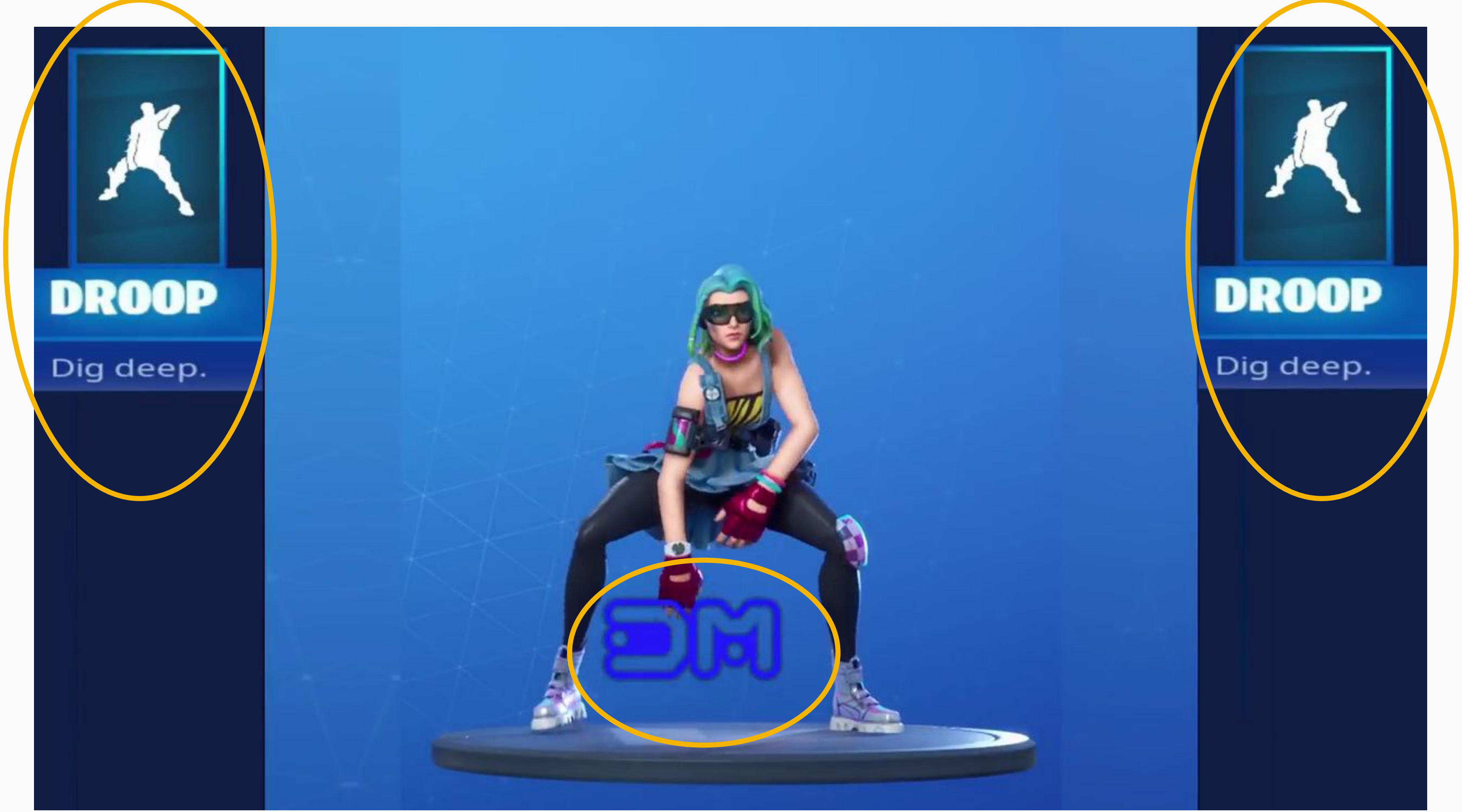
Tik Tok videos filtered by hashtags of dance challenges

Data

- 51:52 Compilation Video
- 146 different dances
- Extracting 1 frame every 2 seconds
- 93,360 total images



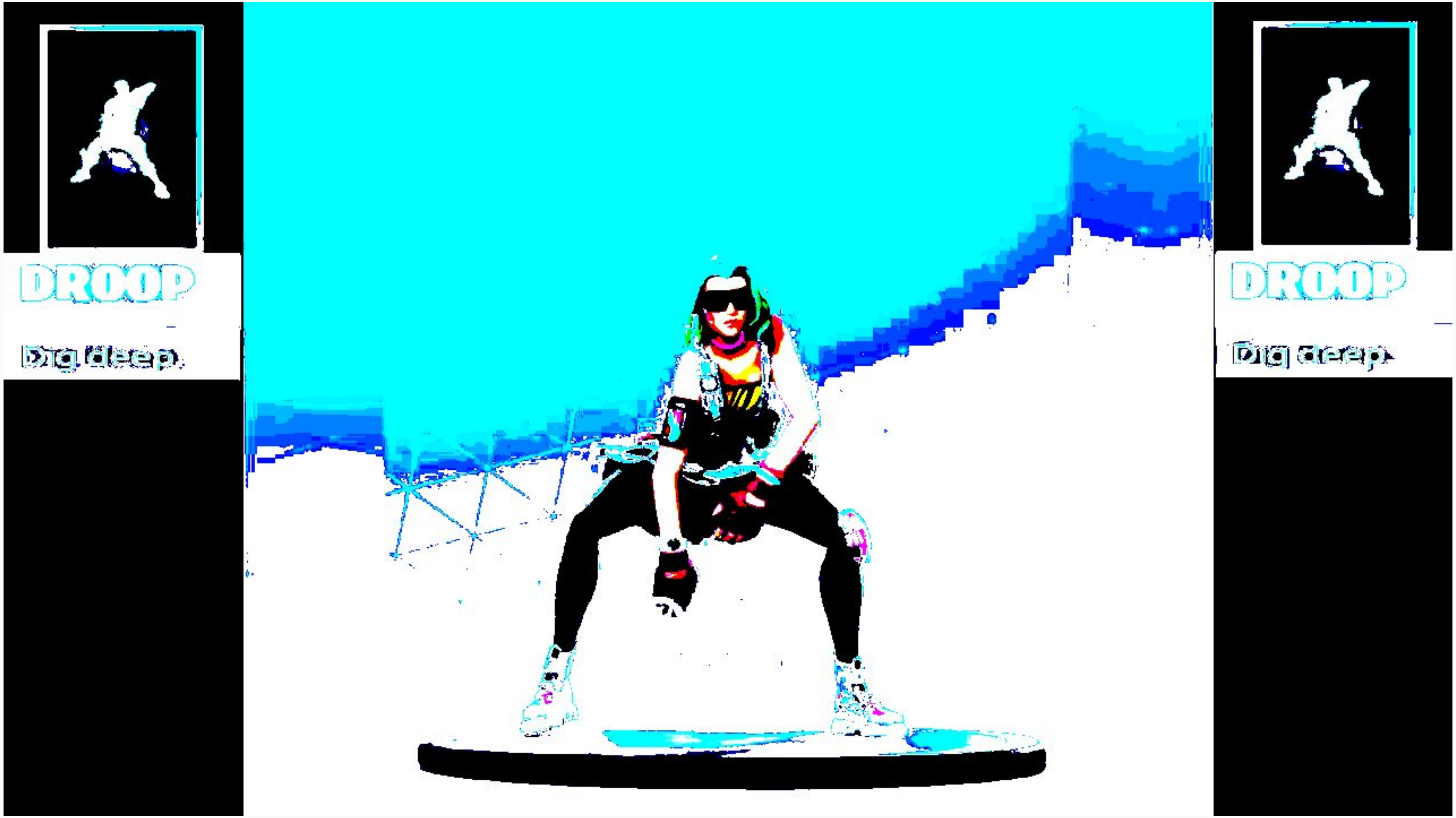
Original Video Frame



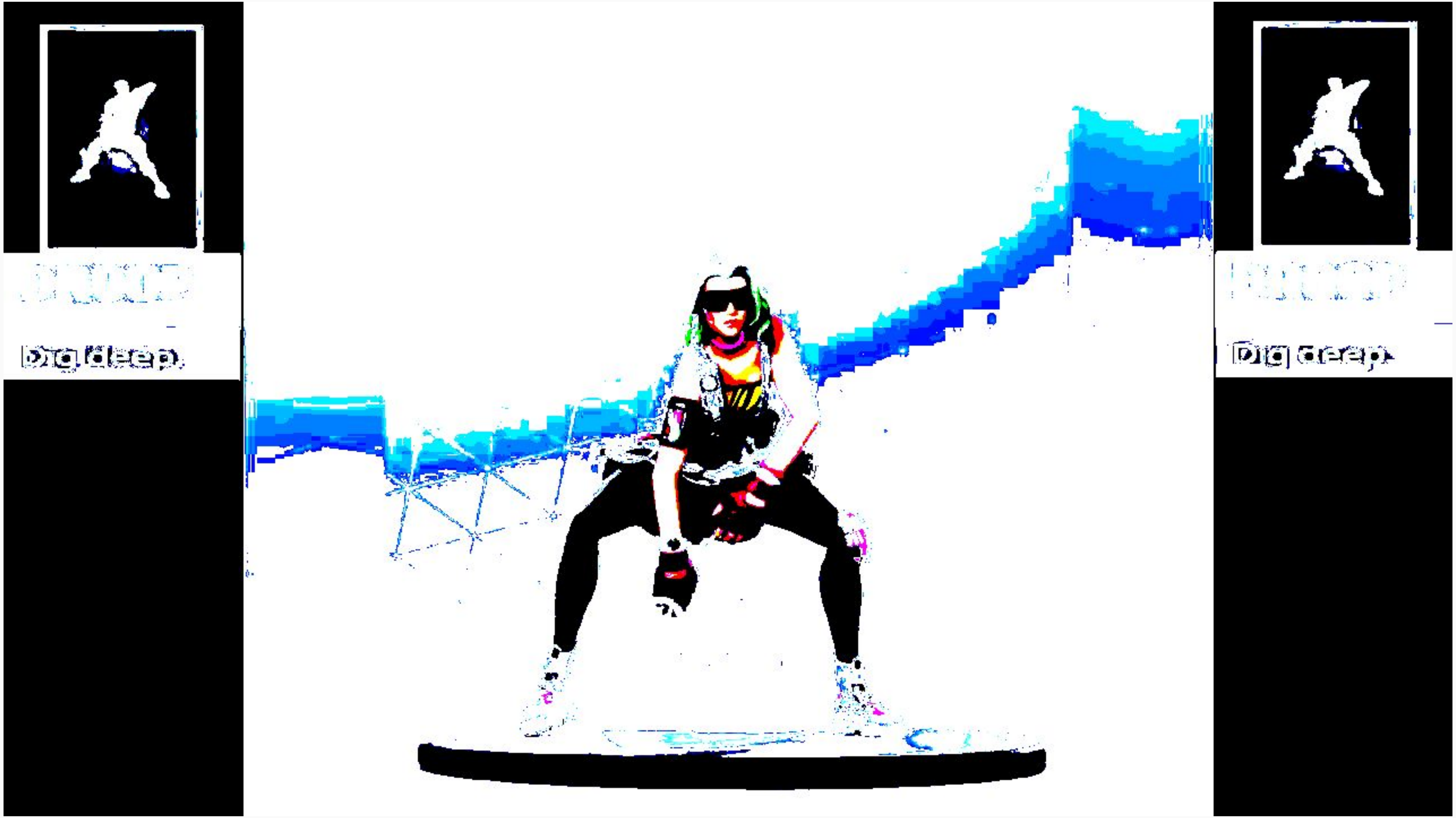
Original Video Frame Problems

Original Video Frame

Transformation 1: Increase contrast



Transformation 2: Remove blue pixels



Transformation 3: Remove cyan pixels

Transformation 4 + 5: Convert all pixels to 1 color + crop

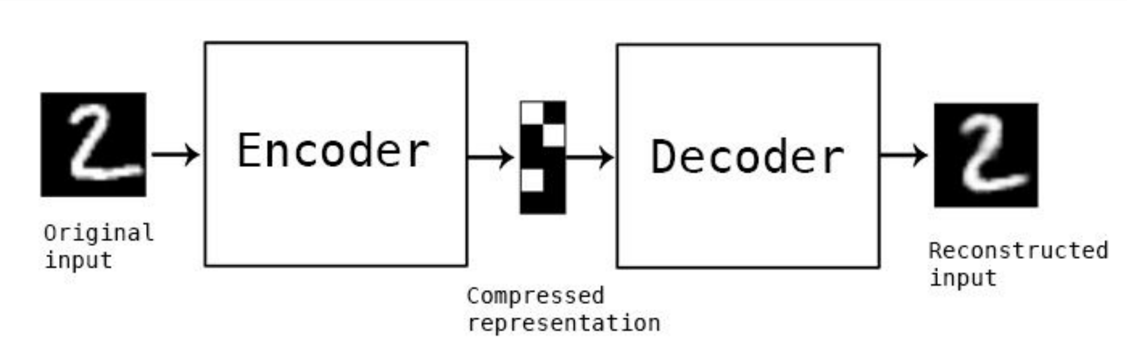

DROOP
Dig deep.
DROOP

Sample dance move

Autoencoder

What are Autoencoders?

- Data compression algorithm
- Learned automatically from examples, usually with neural networks
- Data-specific
- Lossy
- Useful for dimensionality reduction



From Keras Blog

Encoder

- Used to *compress* original images (1,032 x 950) into (128) vectors, losing as little information as possible
- Create dense representation to be fed into RNN

Decoder

- Used to *expand* original (128) vectors into (1,032 x 950) images, *reconstructing* as much information as possible
- “Translate” RNN’s output

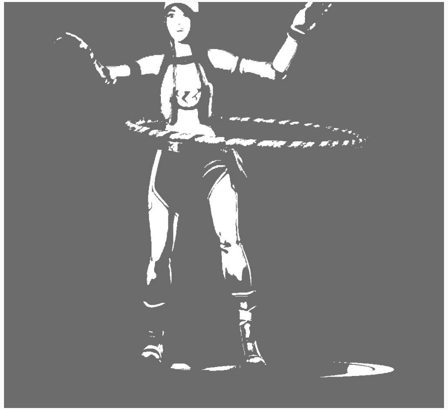
Version 1: Simple

- Single fully-connected layer
- As both encoder and decoder
- Image converted to greyscale
    - Single color channel

```
(703, 950, 1032)
```

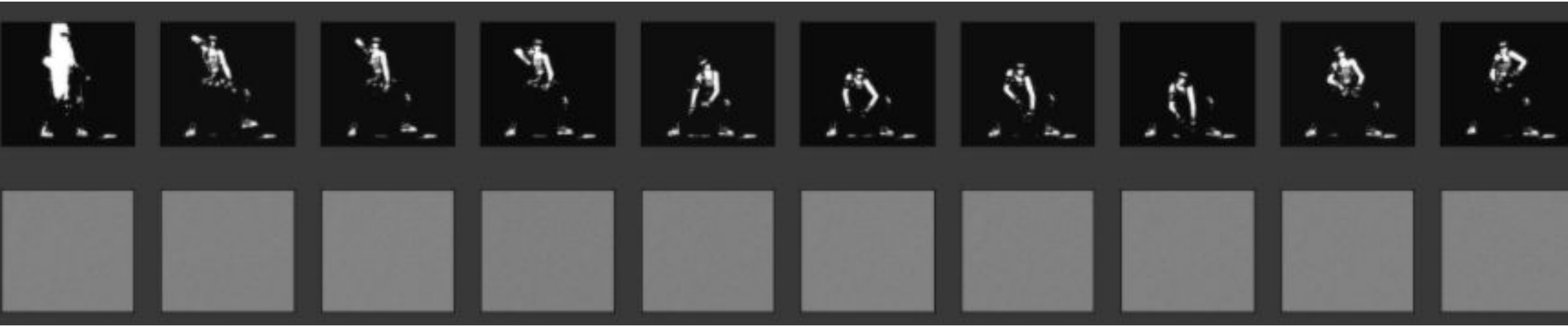
```
(50, 950, 1032)
```

```
(703, 980400)
(50, 980400)
```

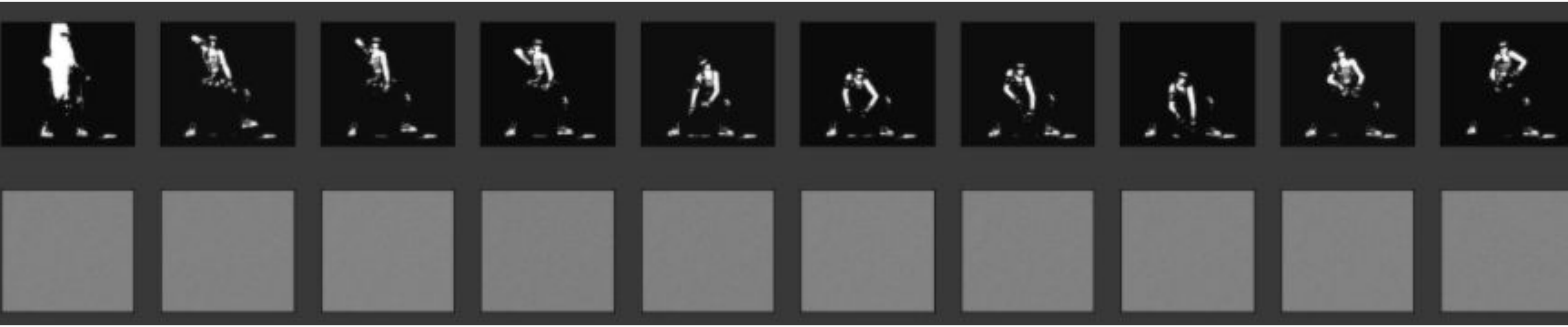


```
Epoch 45/50
60000/60000 [==============================] - 1s 14us/step - loss: 0.1061 - val_loss: 0.1042
Epoch 46/50
60000/60000 [==============================] - 1s 14us/step - loss: 0.1057 - val_loss: 0.1038
Epoch 47/50
60000/60000 [==============================] - 1s 13us/step - loss: 0.1052 - val_loss: 0.1034
Epoch 48/50
60000/60000 [==============================] - 1s 13us/step - loss: 0.1048 - val_loss: 0.1030
Epoch 49/50
60000/60000 [==============================] - 1s 13us/step - loss: 0.1045 - val_loss: 0.1026
Epoch 50/50
60000/60000 [==============================] - 1s 13us/step - loss: 0.1041 - val_loss: 0.1023
```

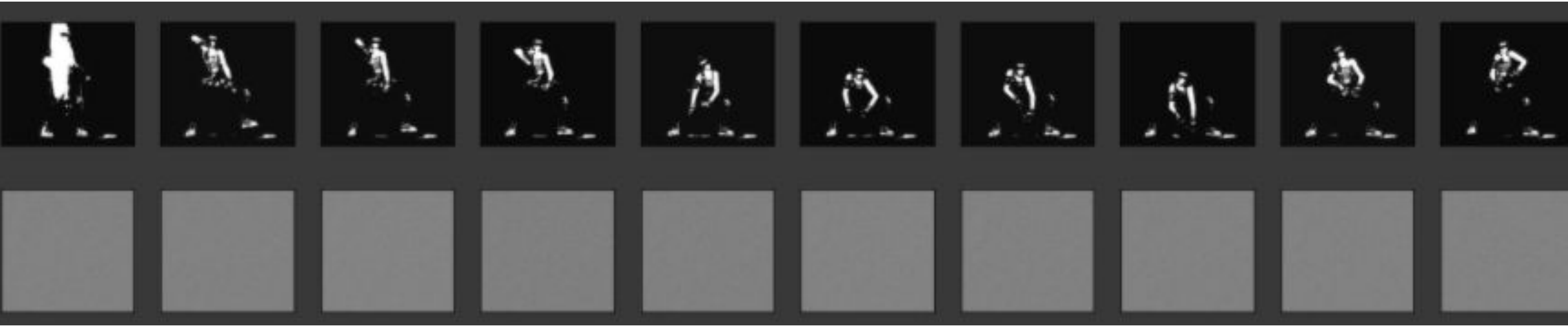
MNIST comparison



10 epcohs



50 epochs



150 epochs

Something’s gotta change

```
Epoch 138/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 139/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 140/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 141/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 142/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 143/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 144/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 145/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 146/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 147/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 148/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 149/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 150/150
703/703 [==============================] - 3s 4ms/step - loss: 0.6931 - val_loss: 0.6931
<keras.callbacks.callbacks.History at 0x7f74b7daec50>
```

Something’s gotta change

Mod 1:

- Realized my compression factor was off (~3%)
- Fixed it to be 24.5%
- ```
  ResourceExhaustedError:
  ```

- It was able to handle `encoding_dim = 512`
- 19.14% compression
- Compiled, but while training:
- ```
  ResourceExhaustedError:  OOM when
           [[node gradients_4/loss_
  Hint: If you want to see a list o
   [Op:__inference_keras_scratch_gr

  Function call stack:
  keras_scratch_graph
  ```

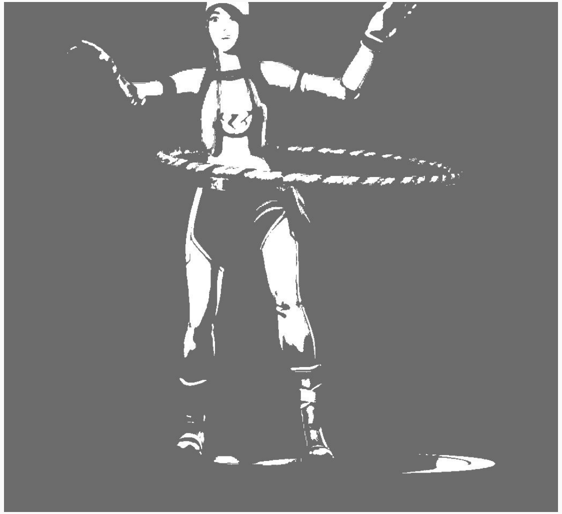
Mod 2:

- Resize images before feeding converting into np `arrays`
- 5% of original
- ```
  (703, 47, 51)
  ```

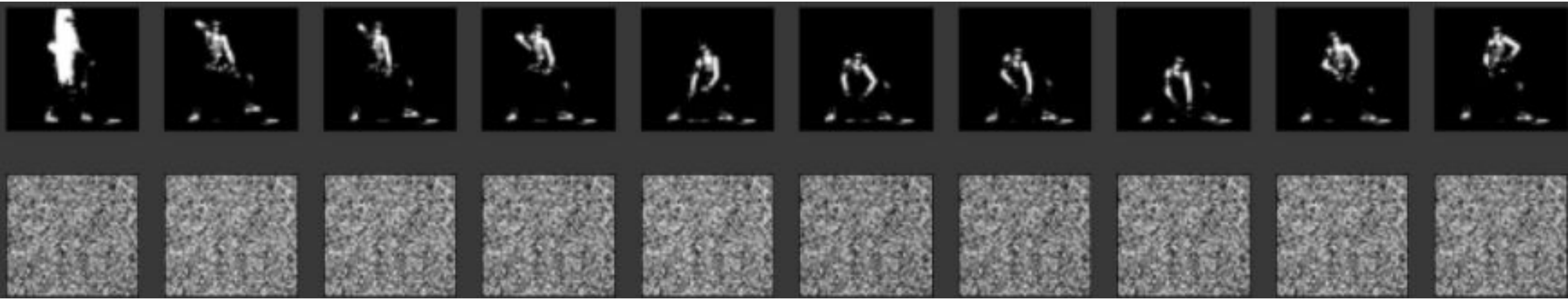
- Previously:

```
(703, 950, 1032)
```

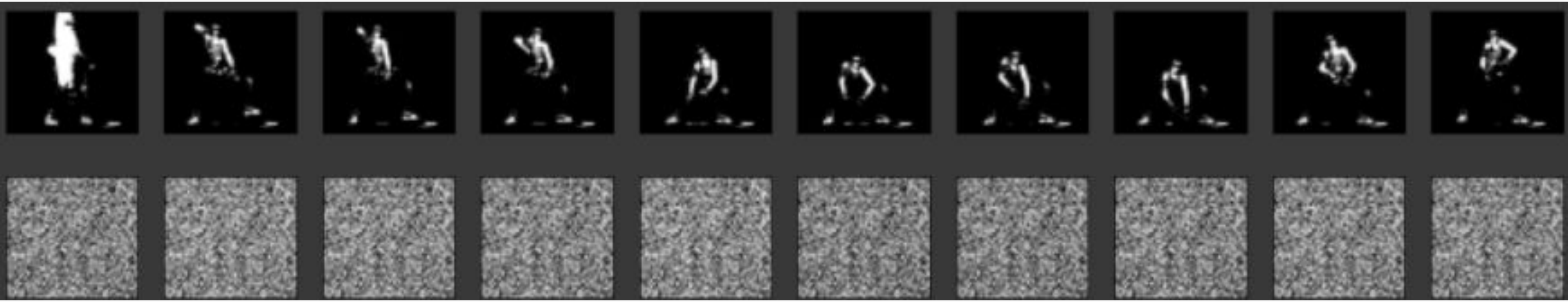
```
(703, 980400)
(50, 980400)
```
```



5% size reduction



50 epochs



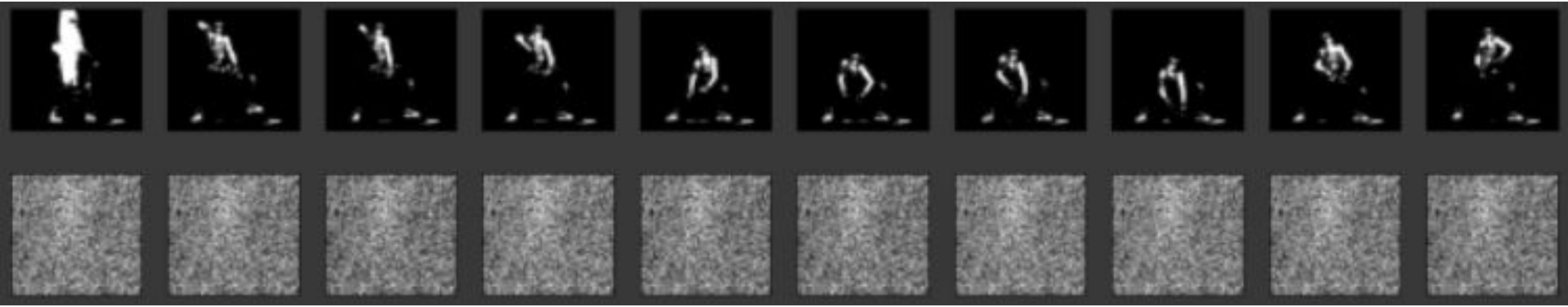
250 epochs

50, 250 epochs w scaled images

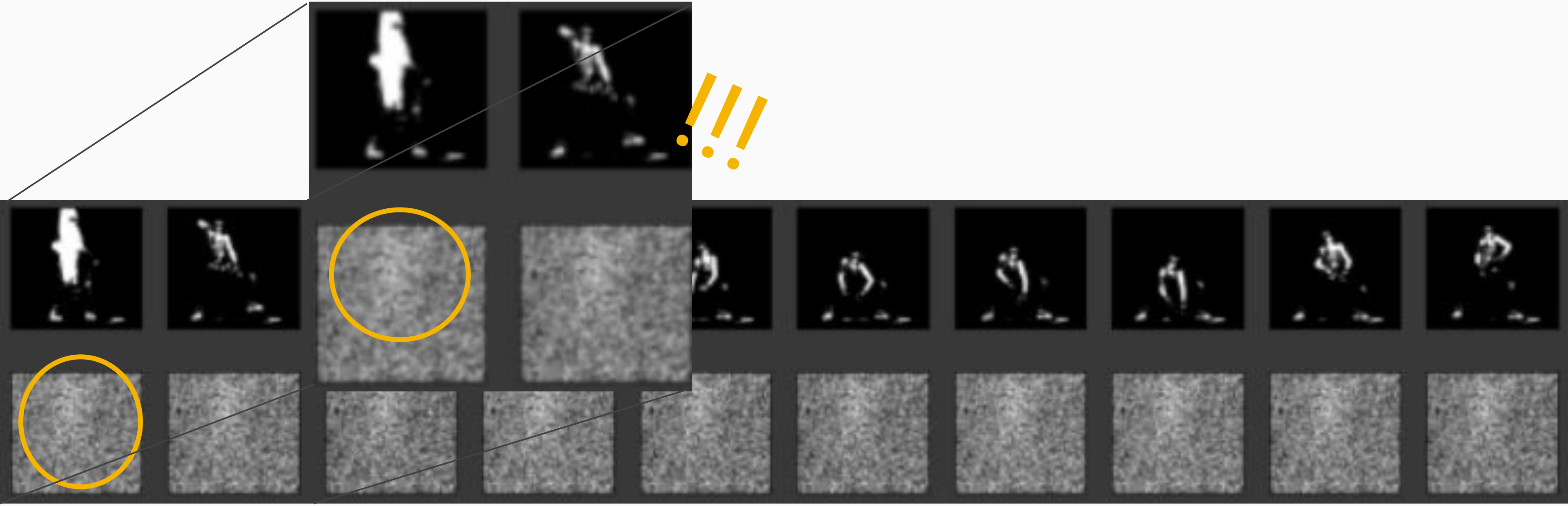
50, 250 epochs w scaled images

Mod 3:

- Time spent training clearly *not* the issue
- Changing the optimizer from `rms_prop` to `adadelta`
- 50 epochs



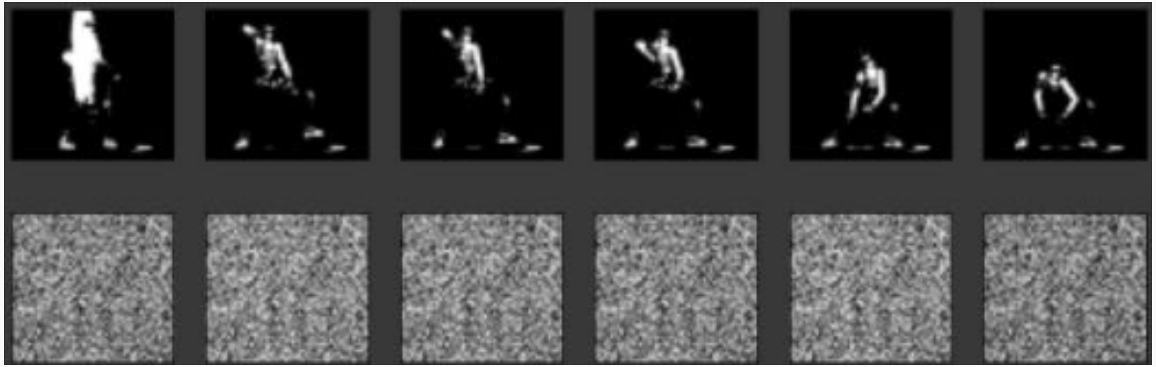
New Simple Encoder Results with adadelta optimizer

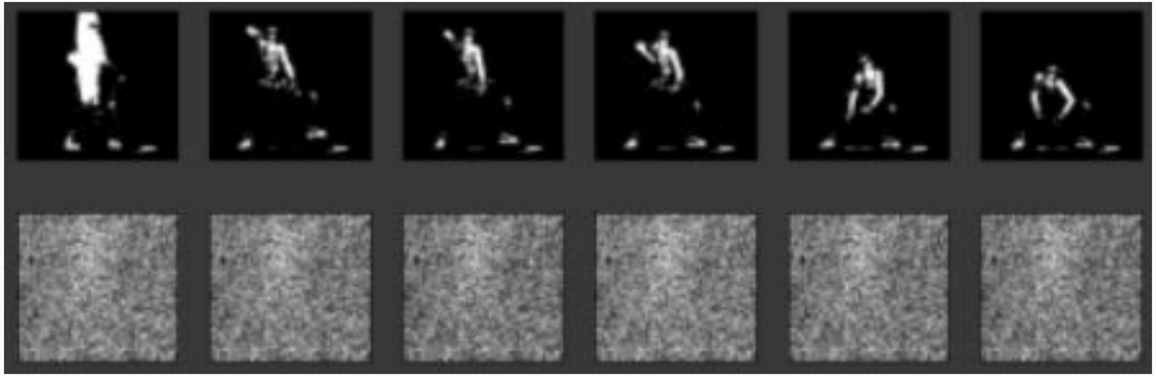


IT’S LIGHTER IN THE MIDDLE!

Mod 2 and Mod 3 Comparison

- Mod 2 more noisy
- Mod 3 has lightness in center where character should be



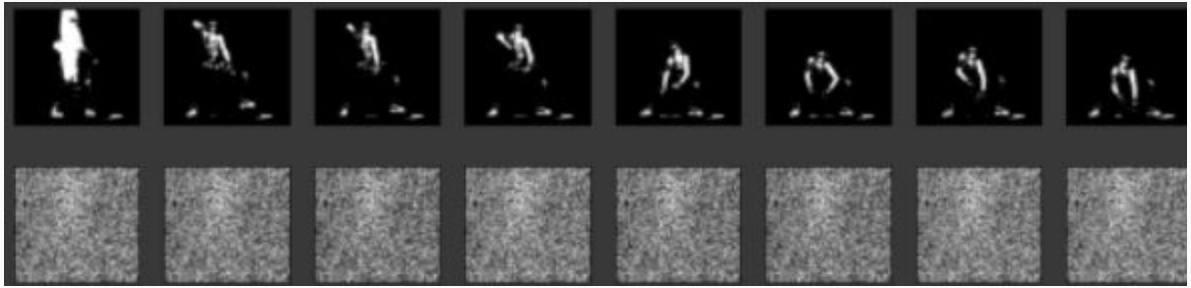


Version 2: Sparsity Constraint

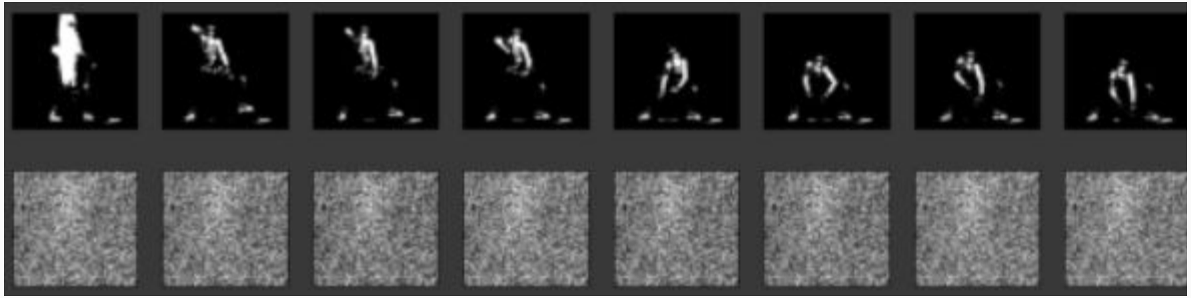
- In V1, reps constrained only by size of hidden layer
- Learns approx of PCA
- Adding sparsity constraint

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 2397)              0
_________________________________________________________________
dense_13 (Dense)             (None, 64)                153472
_________________________________________________________________
dense_14 (Dense)             (None, 2397)              155805
=================================================================
Total params: 309,277
Trainable params: 309,277
Non-trainable params: 0
_________________________________________________________________
```

```
encoded = Dense(encoding_dim, activation='relu',
                activity_regularizer=regularizers.l1(10e-5))(input_img)
```



50 epochs



100 epochs

Version 2: Sparsity Constraint

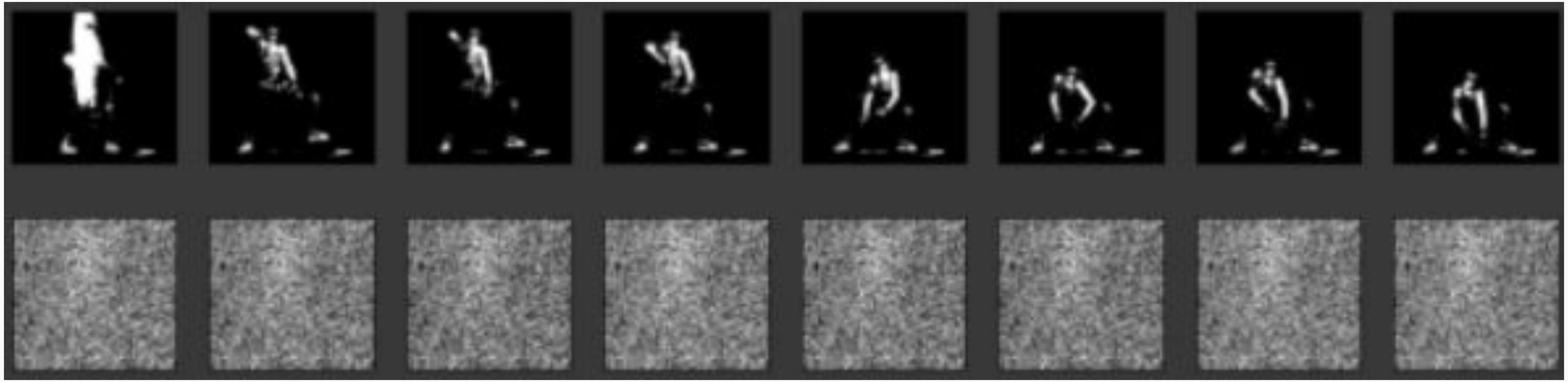
Version 3: Deep Encoder

- Hypothesis:

  more layers == more

  learning?

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 2397)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1227776
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_5 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_6 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_7 (Dense)              (None, 512)               66048
_________________________________________________________________
dense_8 (Dense)              (None, 2397)              1229661
=================================================================
Total params: 2,609,917
Trainable params: 2,609,917
Non-trainable params: 0
_________________________________________________________________
```

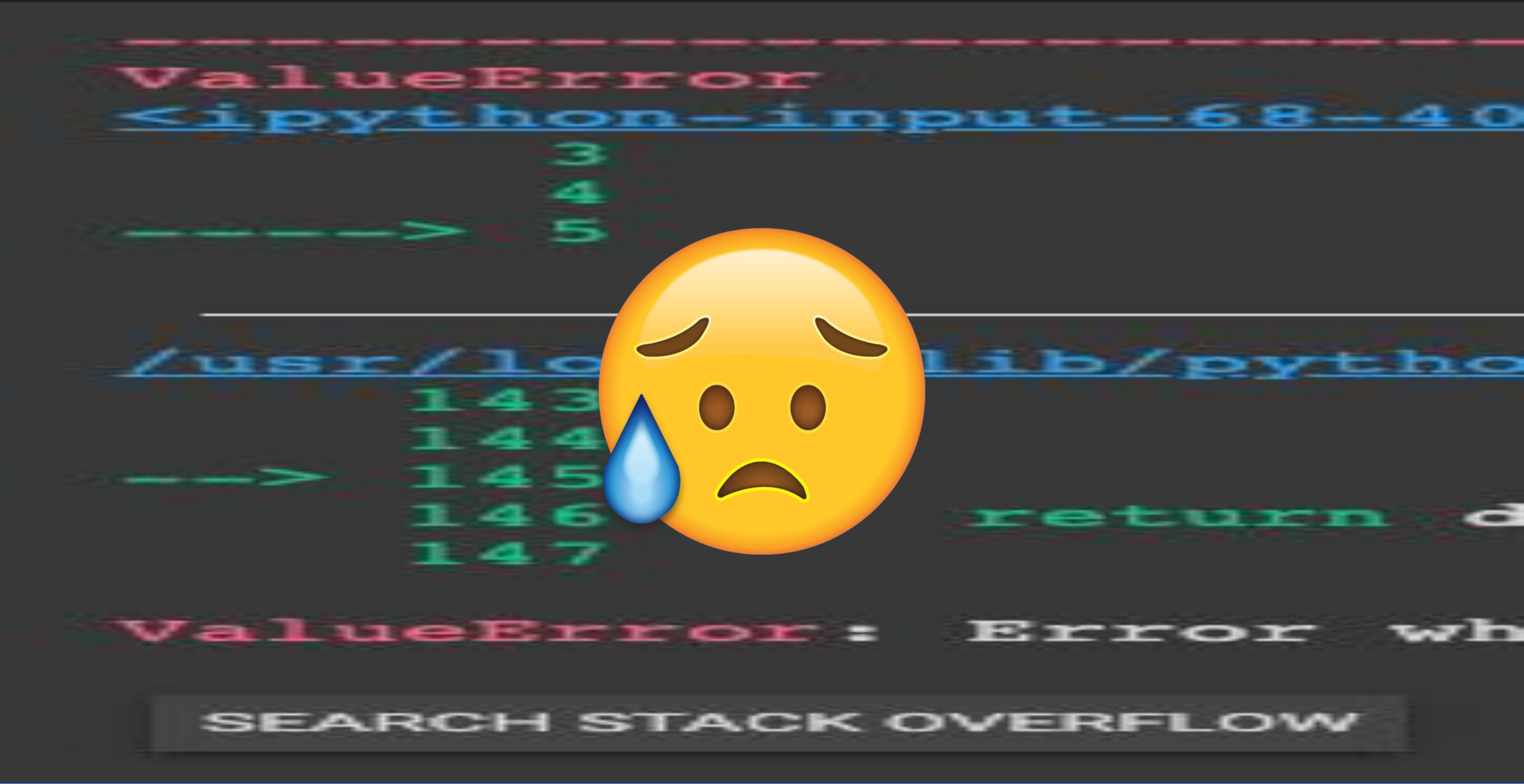


100 epochs

Version 4: Convolutional

- Convolutional neural networks are pretty much standard when it comes to training on images, so it's application here makes sense

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        (None, 47, 51, 1)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 47, 51, 16)        160
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 47, 51, 8)         1160
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 47, 51, 8)         584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 26, 8)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 24, 26, 8)         584
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 24, 26, 8)         584
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 22, 24, 16)        1168
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 22, 24, 1)         145
=================================================================
Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0
_________________________________________________________________
```



Convolutional Autoencoder Results N/A

Autoencoder Results

- Not a massive difference between different versions
- Wondering if there’s

RNN

Why RNN?

- Feedforward networks have no memory
- To process a sequence, the network has to see it all as once
  - Turning sequence into single data point
- A dance is a sequence
  - Of moves, of frames
- A recurrent neural network processes sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far

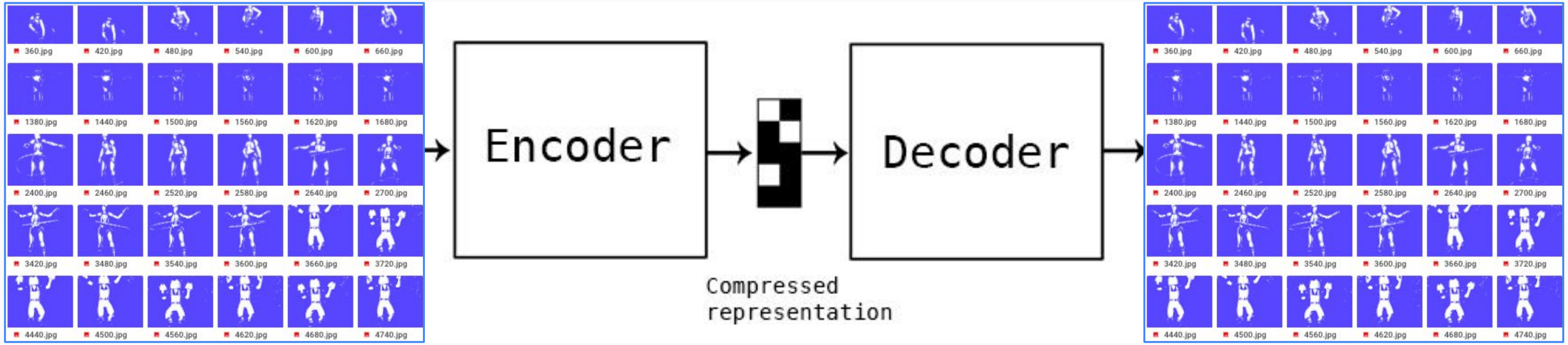
RNN

- Modifying Andrej Karpathy’s RNN
    - Shakespeare creator
    - One-hot encoded characters
        - Input: 95-D vector representing 95 possible chars
    - *How can I do this with images? —> binary images (two colors)*
    - *How can I pass in a*

RNN

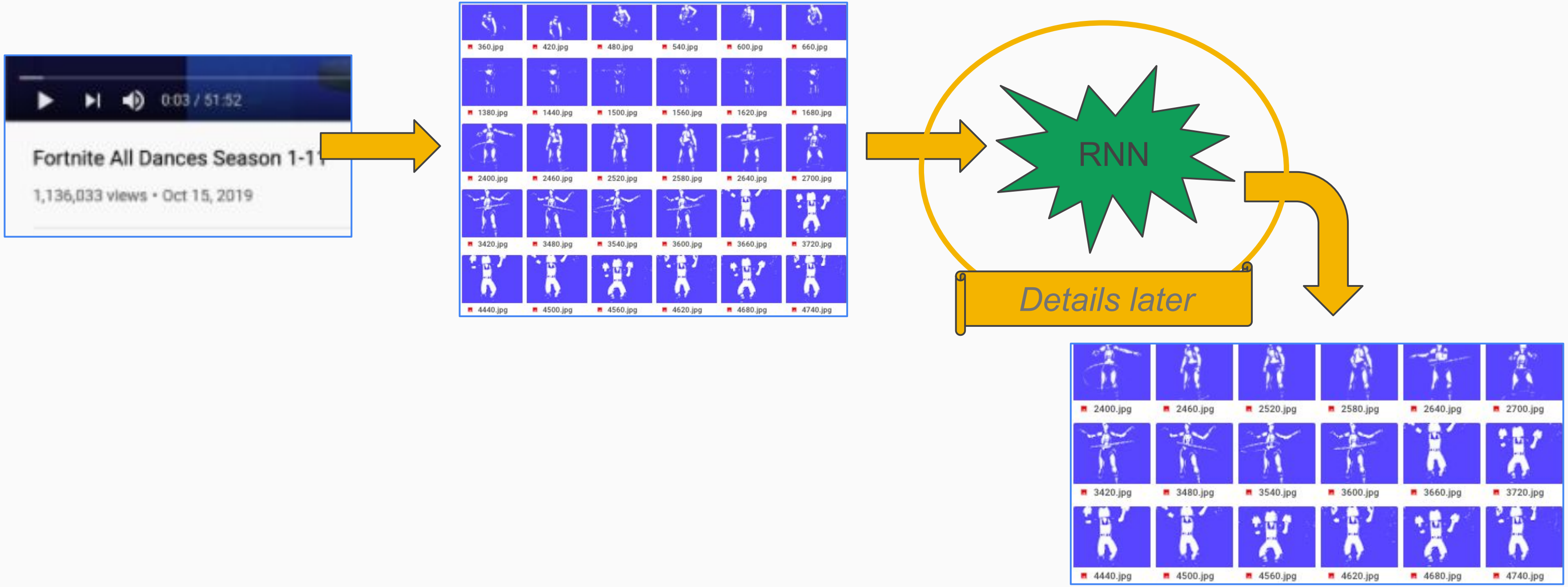
- Compressing them normally is not good enough
  - Ex: regular JPEG compression algorithm is general
- We can use an autoencoder

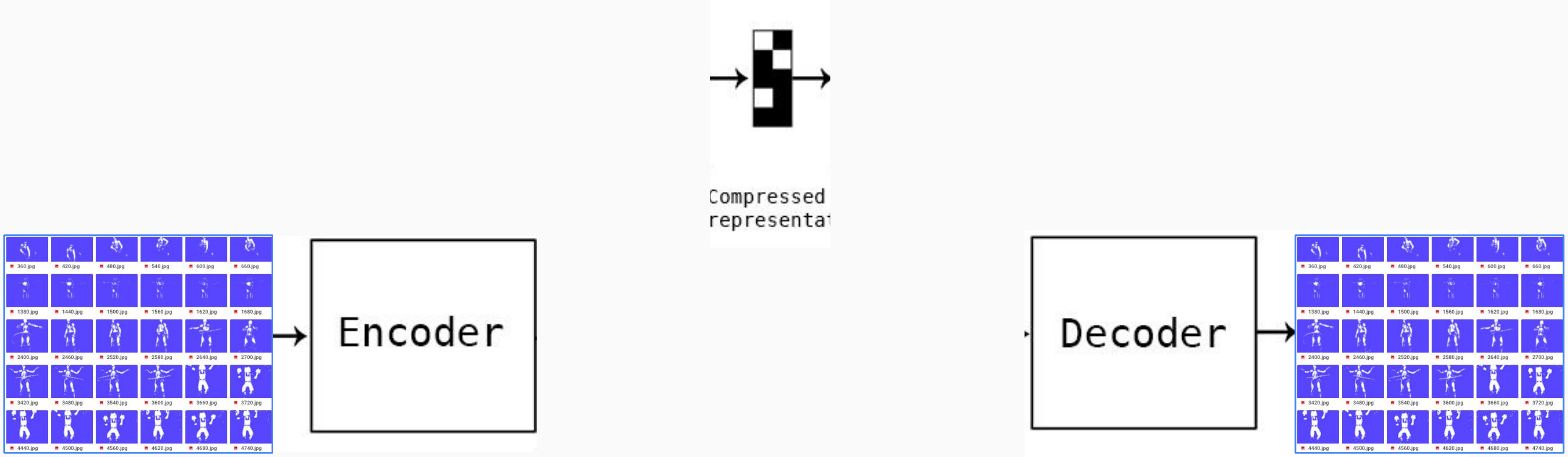
Future Work



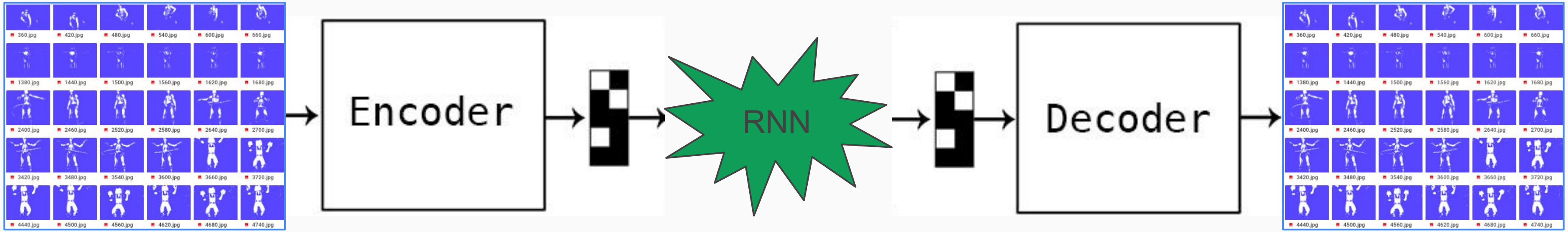
Autoencoder

Final Approach

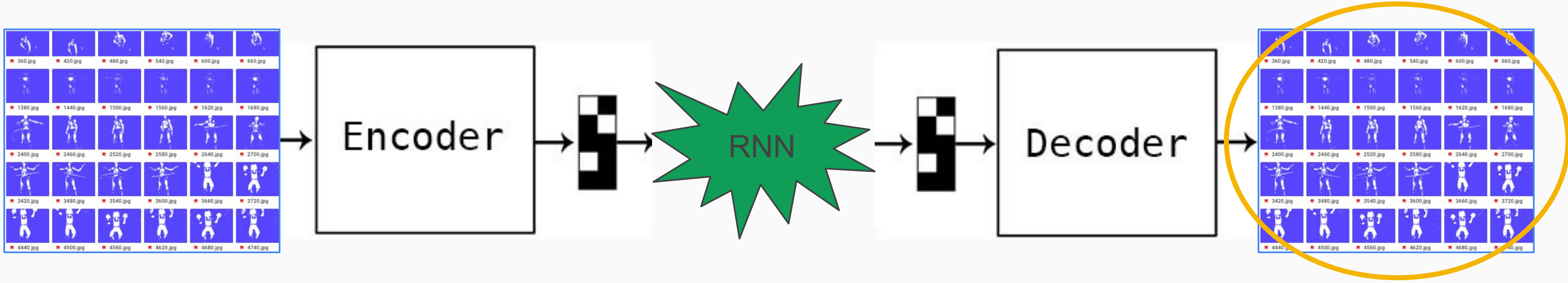




Breaking apart Autoencoder



The Vision


Encoder
RNN
Decoder

Our Baseline:

Future Work

- Incorporating audio files into input
- Preprocessing work has been completed

Thank you! ✌️

The Dancing Screen

Independent Study
Mary Karroqe