Final Project: Pokémon pixel art

Presentation by:
Fabrizio D. Brown,
Omar A. Rashdan,
Riz F. Noronha,
and Thomas B. Hansen

Date: 14-06-2023

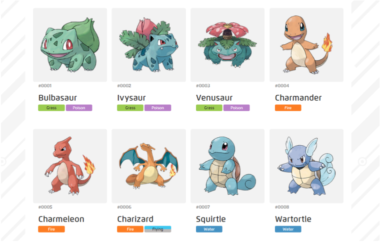
UNIVERSITY OF COPENHAGEN

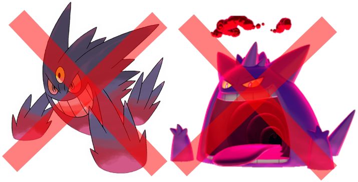
Outline

- Gathering data
- Classifying Pokémon
- Creating Pixel art
  - CNN
  - CycleGAN

Gathering data

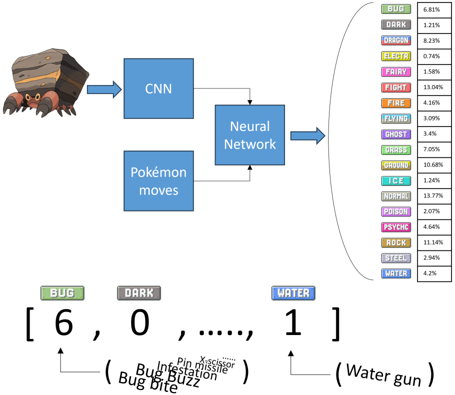
- Gathered images from the official pokemon website and msikma's githubs repository
- Gathered moves and types from pokemondb.net
- Removed temporary evolutions and pokemon forms
- Only work with standard level up moves from generation 9





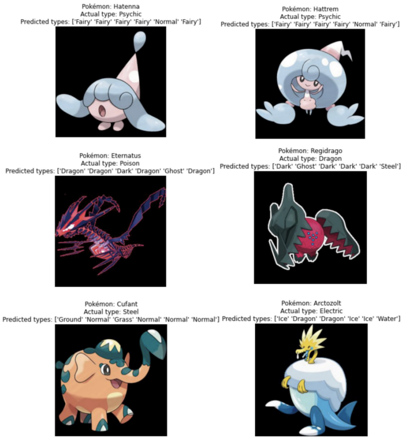
Classifying Pokémon

- Image preprocessing - alpha blending
- Feature vector generated from a pre-trained CNN
- Features passed into a trainable dense network, which predicts a probability per type
- Moves added as a list containing the number of moves for each type



Differences from last year's group

- Predicted (up to two) types for each pokémon
- Implemented 5-fold cross validation
- Optimized hyperparameters with Optuna

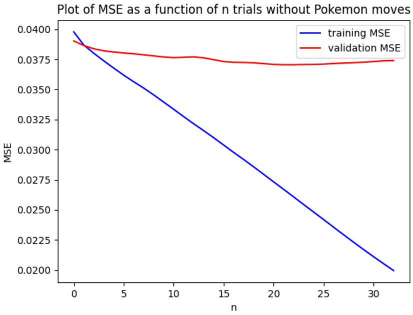


Link

Classifying Pokémon

- MSE was used as the loss function
- MAE caused the algorithm to be more 'confident' (often, confidently wrong!)
- Obtained an accuracy of 17.8%

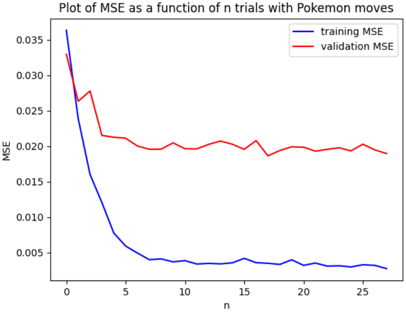
Algorithm	Accuracy
CNN	17.8%



Classifying Pokémon

- Added Pokémon moves as a feature to our neural network
- MSE decreased with an early stopping kicking in at 27'th trial
- An increase of accuracy by 32.2% compared to the CNN algorithm

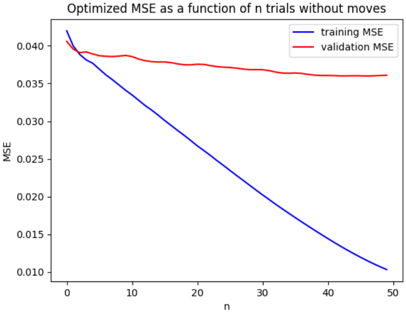
Algorithm	Accuracy
CNN	17.8%
CNN + moves	50%

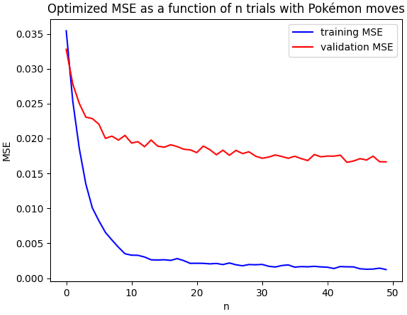


Classifying Pokémon

- Added optimized hyper parameters

Algorithm	Accuracy
CNN	17.8%
CNN + moves	50%
Optimized CNN	20.2%
Optimized CNN + moves	55.9%

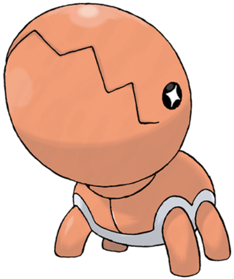




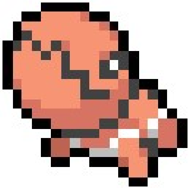
Classifying Pokémon: An example

	Moves	Prediction
BUG	1	1.22%
DARK	0	5.74%
DRAGON	0	1.37%
ELECTR	0	0.27%
FAIRY	0	0.7%
FIGHT	0	0.3%
FIRE	0	0.73%
FLYING	0	0.97%
GHOST	4	29.09%
GRASS	5	54.17%
GROUND	0	0.38%
ICE	0	0.1%
NORMAL	4	0.24%
POISON	0	3.3%
PSYCHC	0	0.25%
ROCK	1	0.86%
STEEL	0	0.12%
WATER	0	0.19%

Creating Pixel art



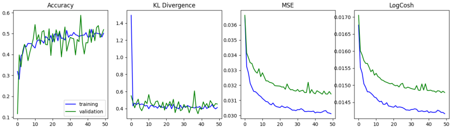
$\longrightarrow$

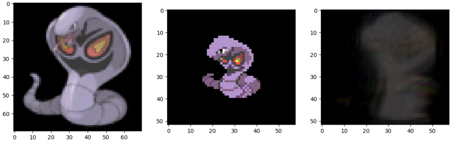


Goal: train a model to transform HD pokémon pictures (left) in sprites (right)

Pixelating with CNN

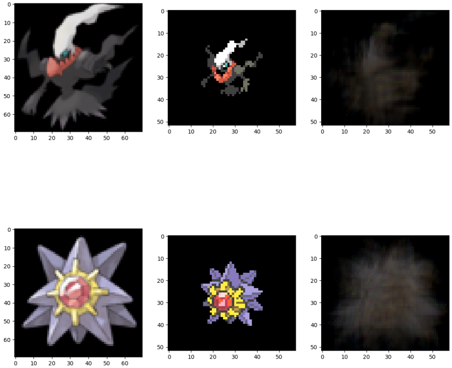
- Take in an RGB image, try to produce an RGB output
- Sprite sizes were made uniform with zero padding, and centered
- Network was a simple feed-forward CNN





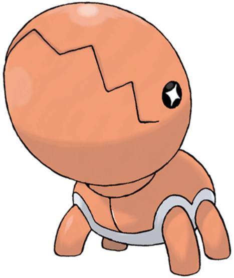
Pixelating with CNN: Issues

- The model did not recognize colours as strongly as expected
- Results were mostly a blurred version of the input
- Image size was not always correlated to sprite size, and thus the model often didn't think of shrinking an image
- Tried modifying hyperparameters and loss functions, working with greyscale, etc to no avail



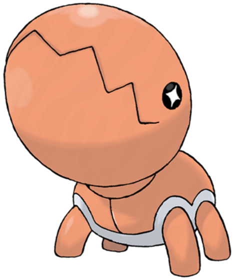
Pixelating with CycleGAN

Pixelating with CycleGAN



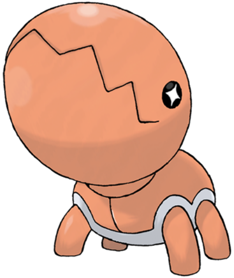
$\longrightarrow$

Pixelating with CycleGAN



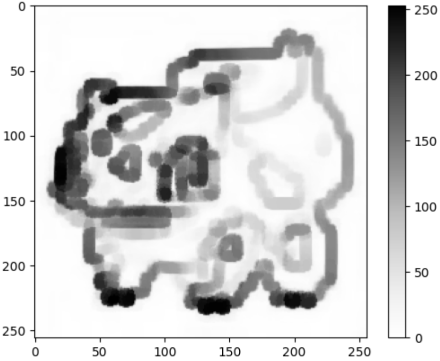
$\longrightarrow$

Pixelating with CycleGAN



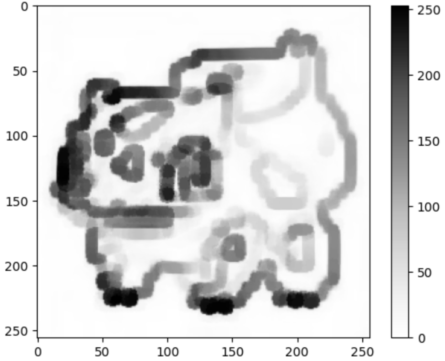
Pixelating with CycleGAN

Sharpness:

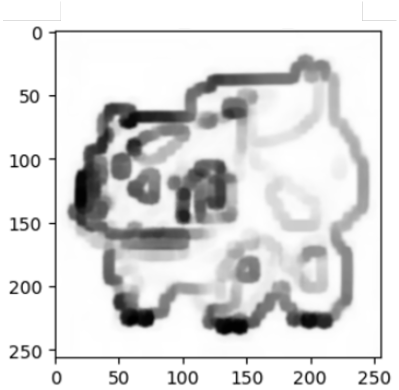


Pixelating with CycleGAN

Sharpness:

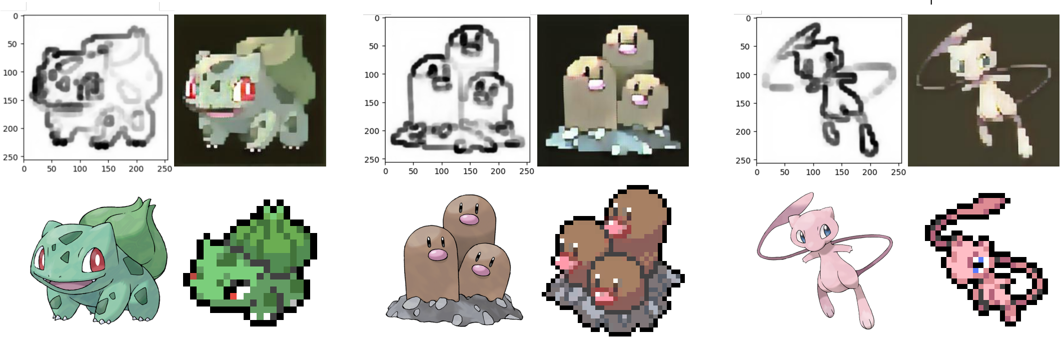


Contrast:



Pixelating with CycleGAN

Contrast	Pixelated image
Official image	Official sprite



Appendix

Classifier model (with only CNN):

- Feature extractor layer (link)
- All nodes are ReLU activated, except the output layer (softmax)
- MSE loss

Optimal hyperparameters found with optuna (400 trials)

- Dense single layer network, with 249 nodes
- Adam learning rate = 0.0005997

Appendix

Classifier model (with moves):

- Feature extractor layer (link)
- All nodes are ReLU activated, except the output layer (softmax)
- MSE loss

Optimal hyperparameters found with optuna (400 trials)

- Feature vector passed through a dense single layer with 501 nodes
- Output is concatenated with move data, and passed through another dense single-layered network with 253 nodes
- Adam learning rate $= 0.000443$

Appendix

CNN model:

- 4 sequential convolutional layers (1,878 params), all nodes are ReLU activated
- Layer1: Conv2D - 5 filters, kernel size (3,3)
- Layer2: Conv2D - 5 filters, kernel size (6,4)
- Layer3: Conv2D - 5 filters, kernel size (6,4)
- Layer4: Conv2D - 3 filters, kernel size (7,5)
- Adam learning rate $= 0.001$
- MSE loss

Hyperparameters were chosen to give the desired output size. Their values were tweaked manually, but they did not get better results. The loss function was modified, and even a custom weighted loss was used (which focused on the center of the image, where the sprite should be) but it did not improve the results.

Appendix

Pixelator model:

- Link to the used pretrained dictionary

The number of blocks was chosen to get better results (a better resolution).

Model layers:

1. ReflectionPad2d: Padding = 3.
2. Conv2d: 64 filters, kernel size (7x7).
3. Conv2d + Downsampling: 128 filters, kernel size (3x3).
4. Conv2d + Downsampling: 256 filters, kernel size (3x3).
5. ResNet Blocks: 512 filters per ResNet block with 9 blocks, kernel size (3x3).
6. ConvTranspose2d + Upsampling: 256 filters, kernel size (3x3).
7. ConvTranspose2d + Upsampling: 128 filters, kernel size (3x3).
8. ReflectionPad2d: Padding = 3.
9. Conv2d: 3 filters, kernel size (7x7).

Appendix

pix2pix model (did not work)
We tried to use this model to make our own trained directory, but model was very computationally heavy, and we did not have the time to prepare our own pixelating dictionary with model, for the deadline.

Making our own CycleGAN (did not work)
We tried to make a CycleGAN model with a generator, that has a encoder with 6 layers, a decoder with 5 layers. The discriminator had 9 layers. The learning rate is 0.0002.
The model was built sub-par and had a problem with reading in the training and validation data. The model could also not produce any desirable result with a test sample of the data that worked; the results were unintelligible compared to the input data (the pictures of the pokémon).

Appendix

ResNet50 + Encoder/Decoder (did not work)
ResNet50 is a pre-trained CNN model used often in image classification. The model didn't work as planned as ResNet output a 1D vector, and so the information on dimensionality is lost.

Encoder/Decoder (not enough time)
We tried building an Encoder/Decoder without using pre-trained models, we didn't have enough time to finish the code of the model.
The idea behind was to build an Encoder with 2D convolution layers, arriving at a latent space smaller than the initial image so that only certain information (hopefully catchable by the model) is saved. Then we would have built a Decoder by using deconvolution layers or upscaling and using 2D convolutional layers.