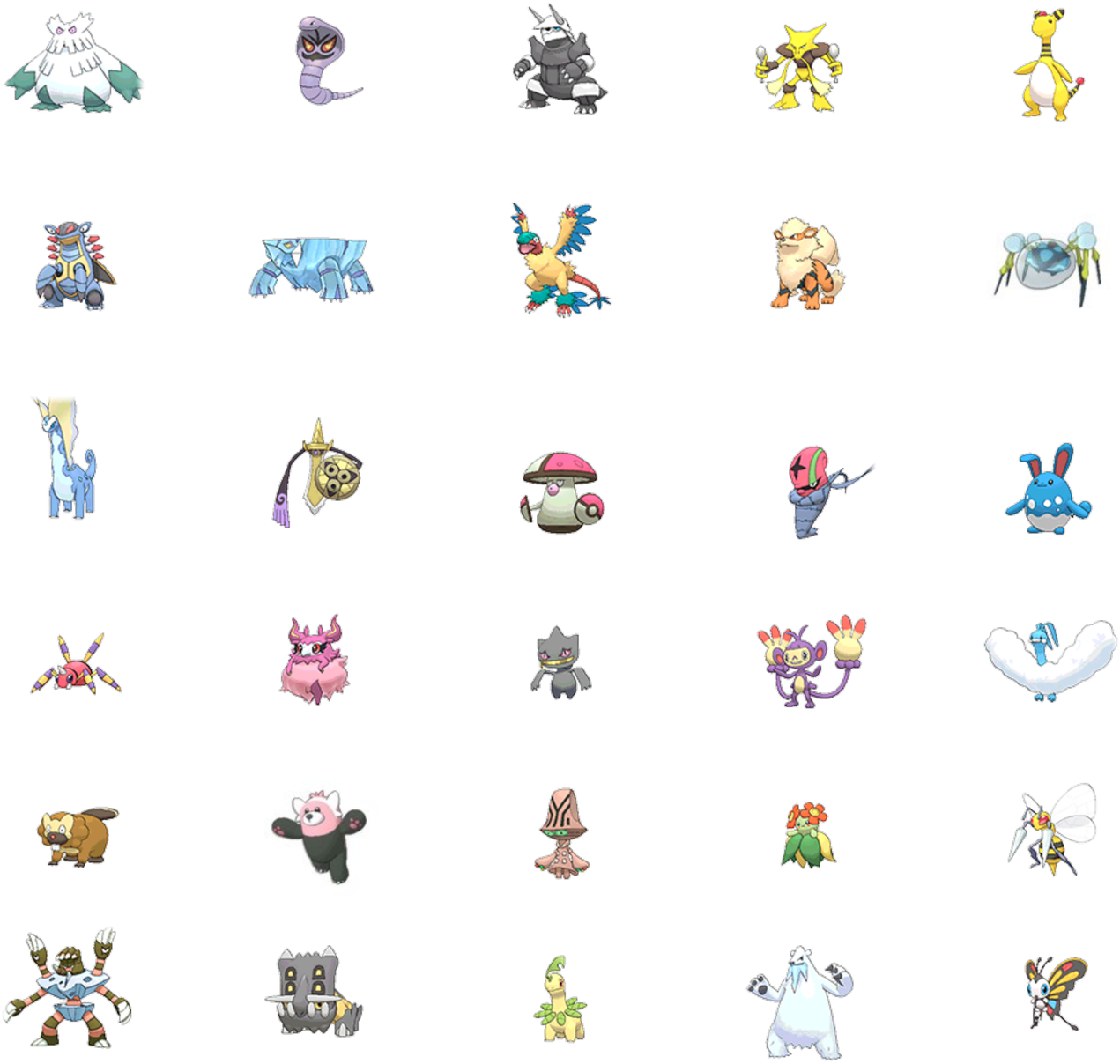
Predicting Evolution

Creative classification and ideation with machine learning

Aug 2020 | Tom © ph[0]ton

Purpose:

- Design and innovation firms are asked to come up with mock ups and pitches in a fast moving environment
- This requires in-house or freelance resources that are taken away from current paying work
- It’s possible that these inefficiencies could be addressed with machine learning
- **Can classifying evolved Pokémon be used to as a proof of concept for design consistency?**
- **Can generating Pokémon be used as a proof of concept for rapid ideation and character prototyping?**



NumPy

pandas

bokeh

OpenCV

Data:

- Data obtained from Kaggle and can be found here
- Contains images of all Pokémon from generation 1 to generation 7
- Total images: 809
- Libraries used for modeling:
  - Numpy
  - Pandas
  - Matplotib
  - Bokeh
  - SKlearn
  - OpenCV
  - Tensorflow
  - Keras

#116
Horsea
Water

→ (Level 32)

#117
Seadra
Water

#072
Tentacool
Water · Poison

→ (Level 30)

#073
Tentacruel
Water · Poison

#058
Growlithe
Fire

→ (use Fire Stone)

#059
Arcanine
Fire

#030
Nidorina
Poison

→ (use Moon Stone)

#031
Nidoqueen
Poison · Ground

#021
Spearow
Normal · Flying

→ (Level 20)

#022
Fearow
Normal · Flying

#172
Pichu
Electric

→ (high Friendship)

#025
Pikachu
Electric

Phase 1: Gotta classify them all

#005
Charmeleon
Fire

→ (Level 36)

#006
Charizard
Fire · Flying

#001
Bulbasaur
Grass · Poison

→ (Level 16)

#002
Ivysaur
Grass · Poison

#123
Scyther
Bug · Flying

→ (trade holding Metal Coat)

#212
Scizor
Bug · Steel

#054
Psyduck
Water

→ (Level 33)

#055
Golduck
Water

#050
Diglett
Ground

→ (Level 26)

#051
Dugtrio
Ground

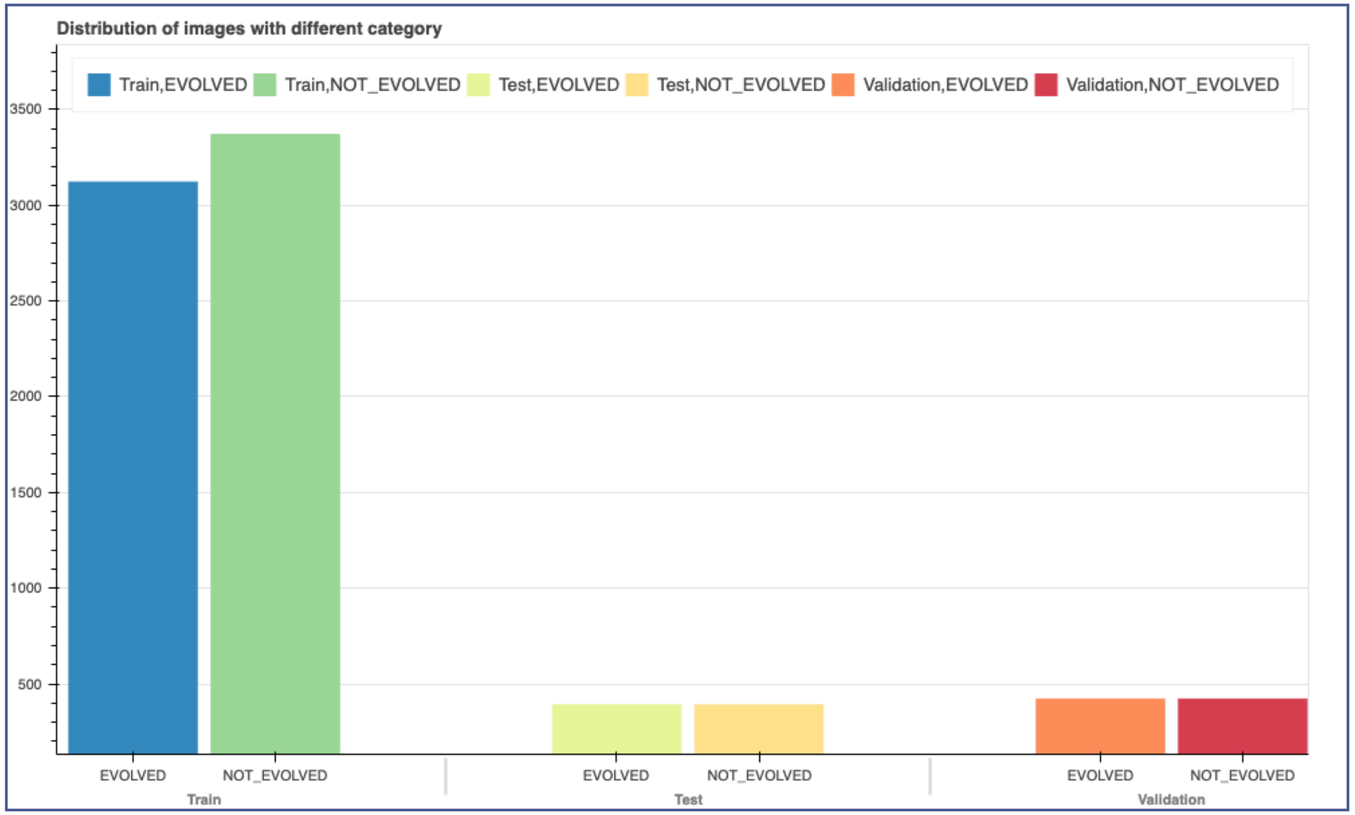
#037
Vulpix
Fire

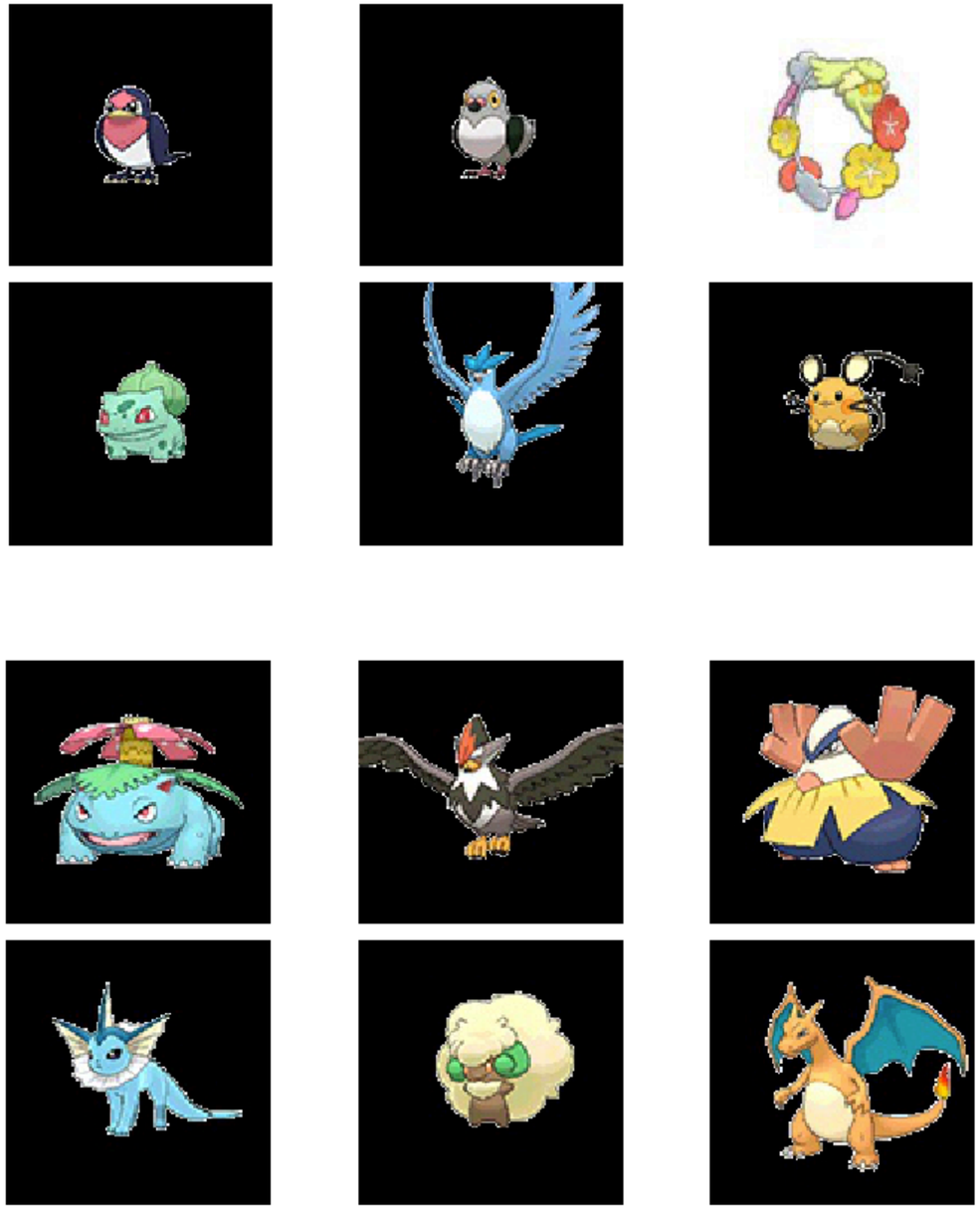
→ (use Fire Stone)

#038
Ninetales
Fire

Data prep:

- Datasets were built by hand - sorting evolved and not evolved into two separate folders
- Images were reproduced 7 times in order to create the training data
- Low class imbalance retained as a result of this approach
- Images were converted from PNG to JPG to address issue with the transparency layer
- All images 120x120



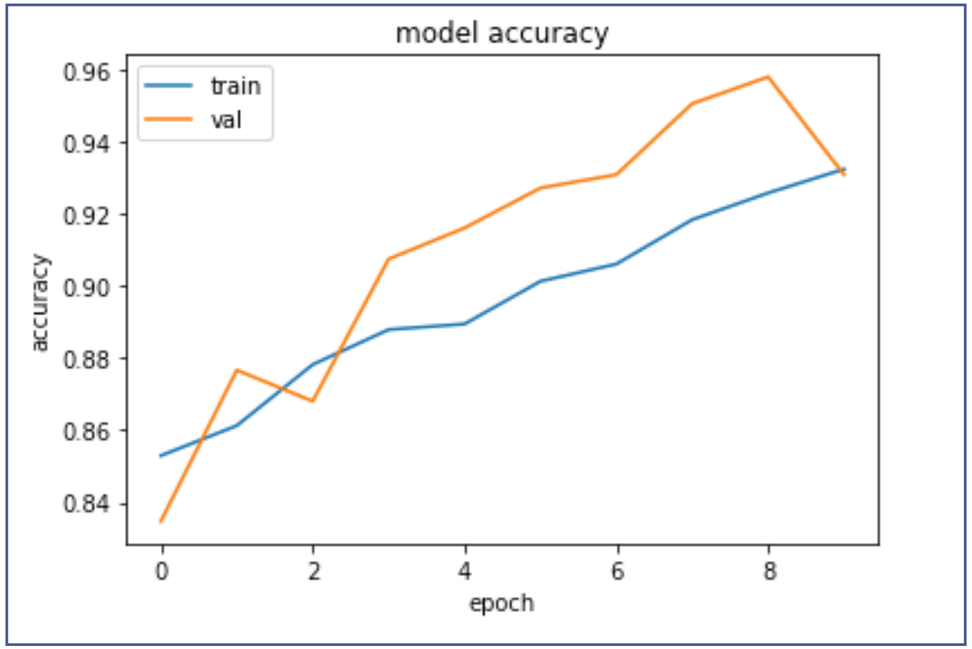


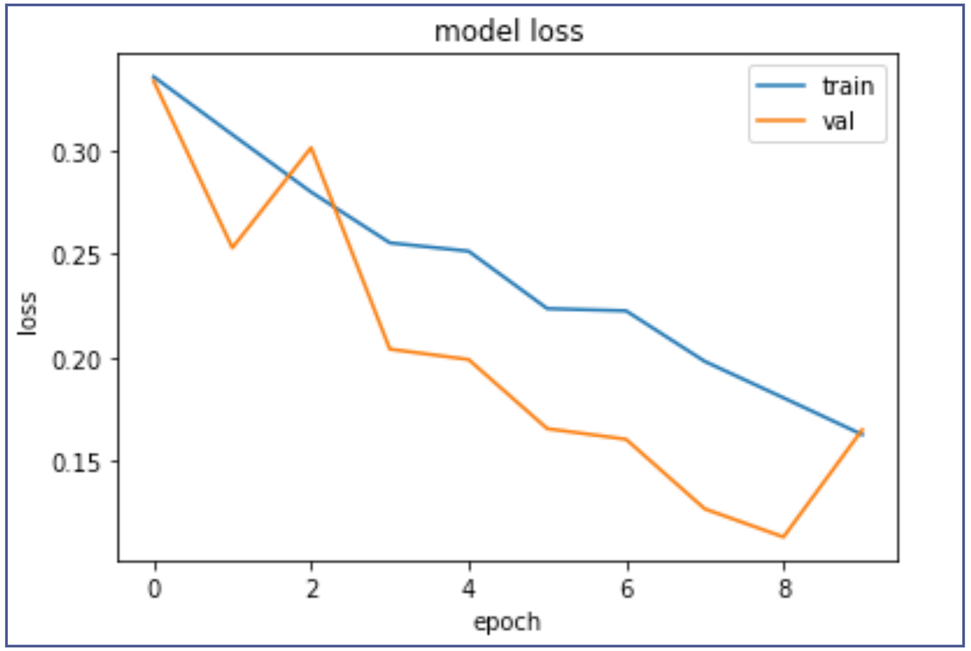
Classification:

- Pokémon that have not evolved typically carry youthful qualities and are generally smaller in size
- Evolved Pokémon are typically mature in appearance and are larger
- **If using a Dummy Classifier, the dominant class (Not Evolved) will be predicted 52% of the time**
- This will serve as our benchmark for assessing performance of the Convolutional Neural Network Models (CNN)

VGG16 Model:

- For the MVP, I used transfer learning to implement the VGG16 architecture and weights
- The CNN model was created by K. Simonyan, A. Zisserman and this version consists of 13 convolutional layers, and 5 max pooling layers
- Total non-trainable parameters: 14,714,688
- I added four additional trainable layers consisting of 1 global average pooling layer and 3 dense layers
- Total trainable parameters: 1,050,625
- Number of epochs: 18

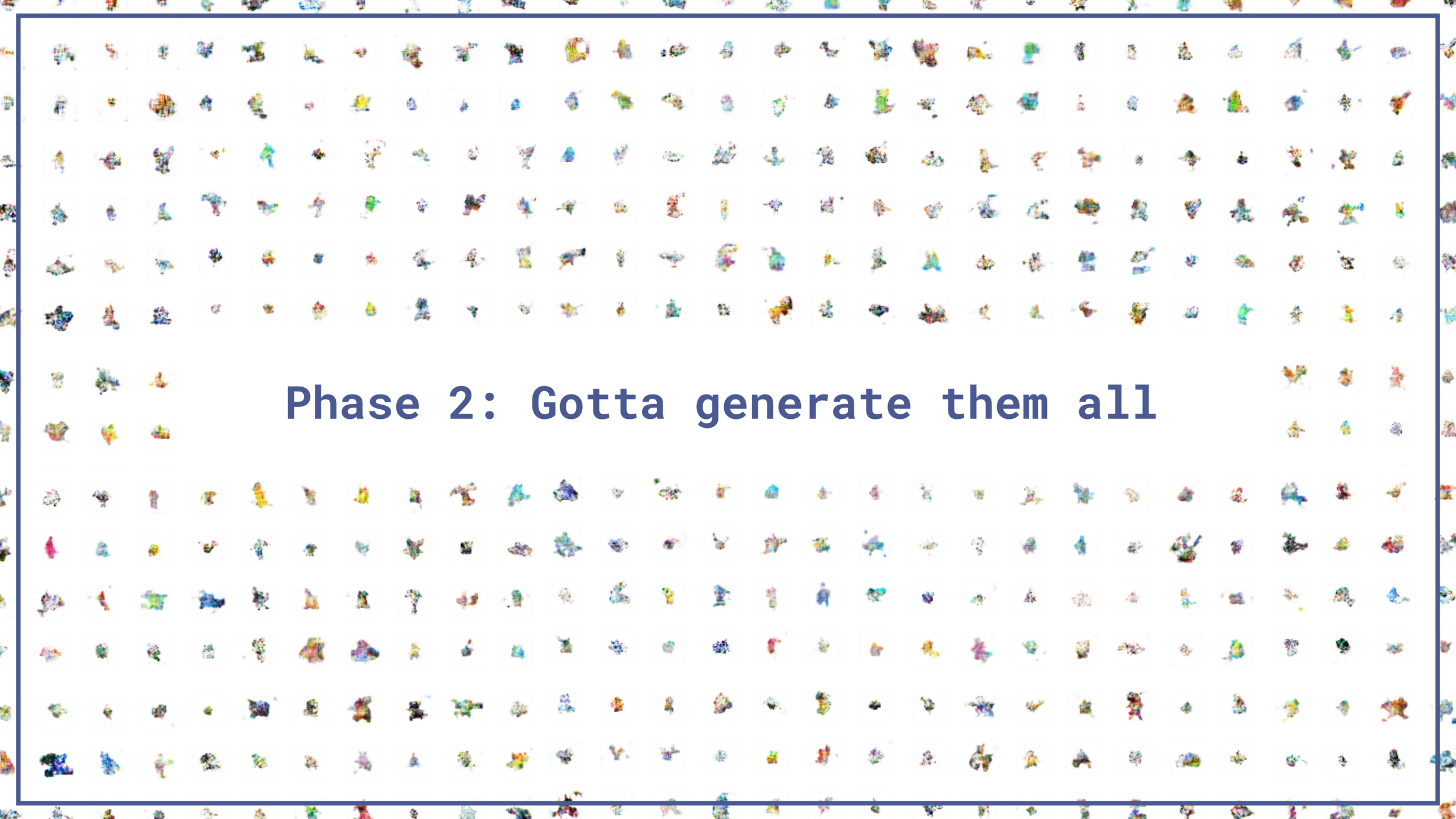




```
Here are the actual classes for each image
['images/TEST/NOT_EVOLVED/snivy.jpg', 'images/TEST/NOT_EVOLVED/delibird.jpg', 'images/TEST/NOT_EVOLVED/goomy.jpg', 'i
mages/TEST/EVOLVED/slaking.jpg', 'images/TEST/NOT_EVOLVED/furfrou.jpg', 'images/TEST/EVOLVED/tentacruel.jpg', 'image
s/TEST/EVOLVED/tranquill.jpg', 'images/TEST/EVOLVED/gloom.jpg', 'images/TEST/NOT_EVOLVED/doduo.jpg']
Below are the predictions
```

MVP:

- Accuracy: 93%
- Loss: 0.1650
- Optimizer: Adam
- Learning Rate: 0.001
- Steps per epoch: 300
- **Significantly better than the baseline metric (52%)**



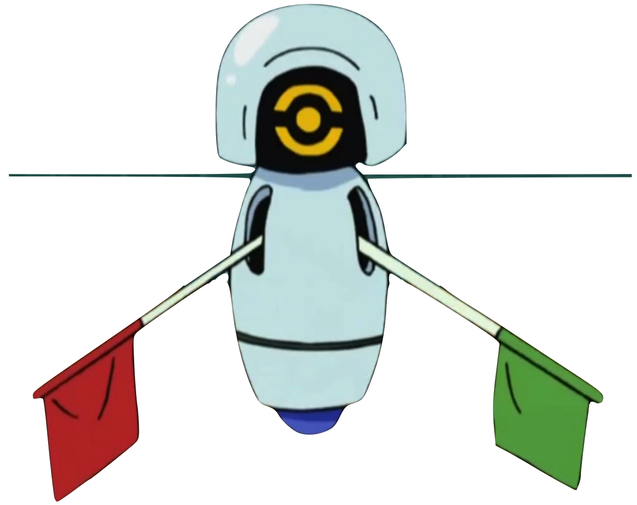
Phase 2: Gotta generate them all

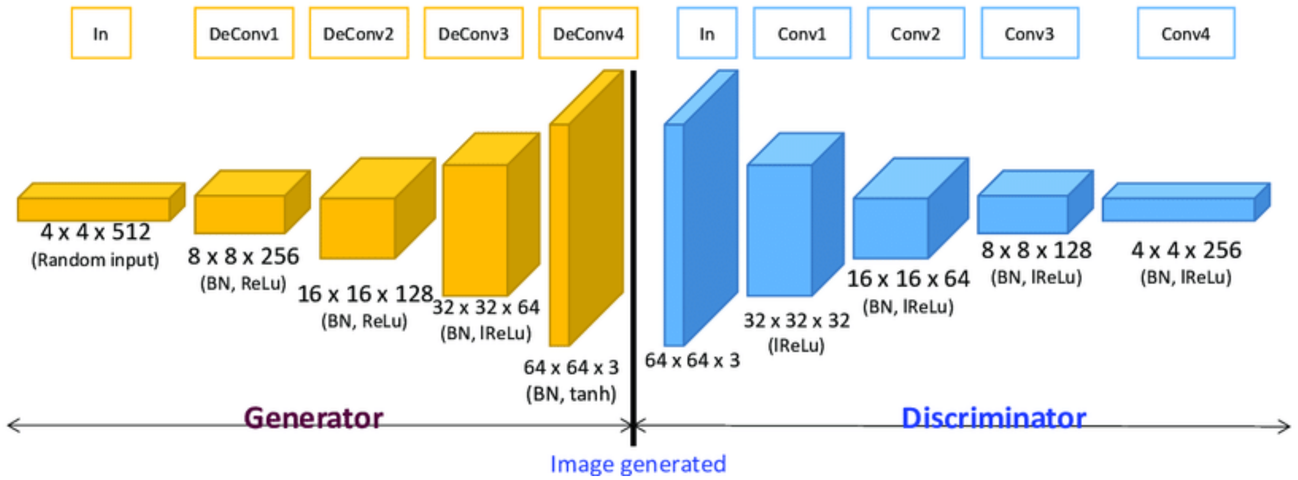
Borrowing the Batmobile:

- I created a DCGAN that was built using the tutorial written by Jeff Heaton
- Deep Convolutional Generative Adversarial Networks are similar to a CNN in that they work with images and use layers
- The difference here is that the two models are adversaries, much like a Pokémon arena
- The Generator (Ash) will create new Pokémon in an attempt to “fool” the Discriminator
- The Discriminator (Battle Judge) will then “judge” the images it receives and determine if they are, in fact, Pokémon
- Dataset: 809 Pokémon

Generator

Discriminator



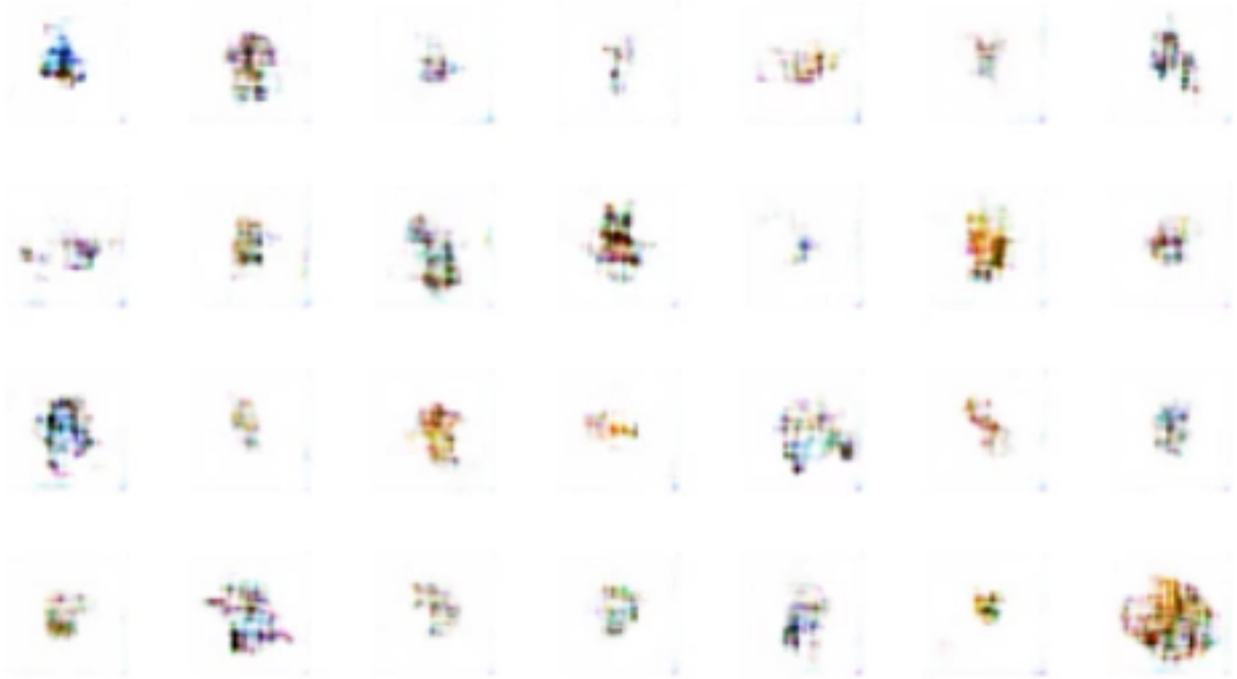


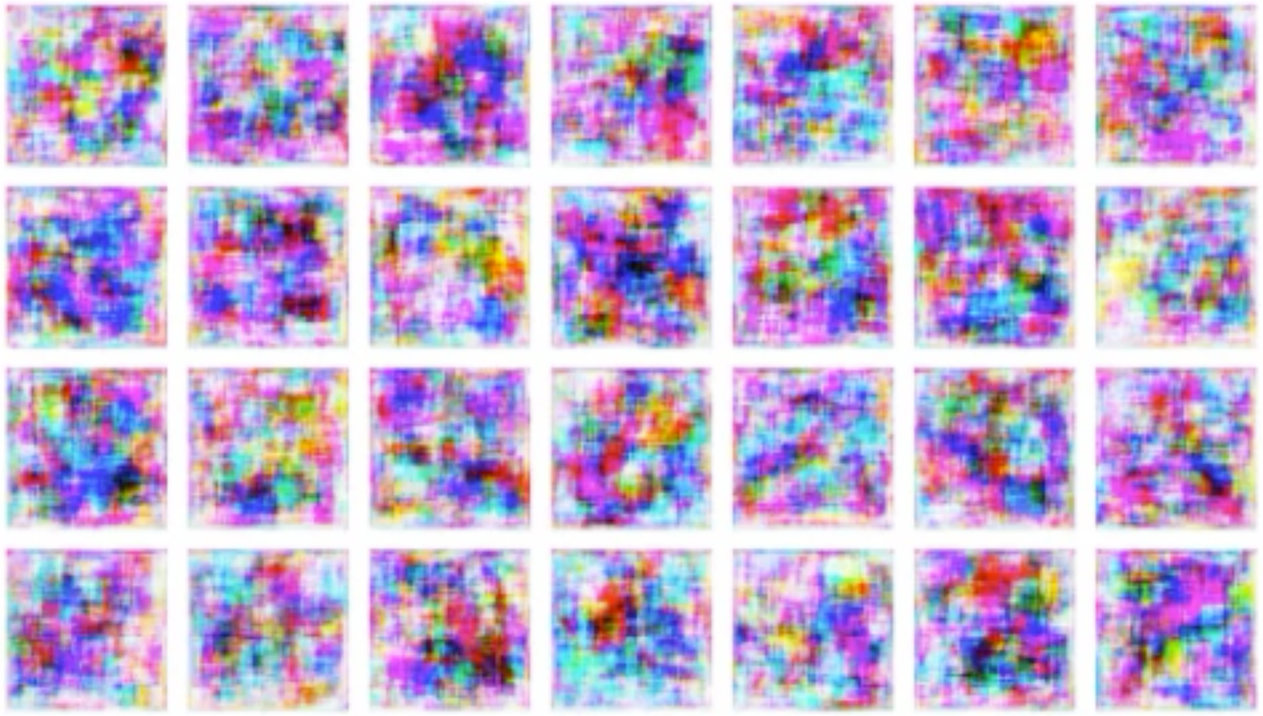
Under the hood:

- All models were trained with mini-batch stochastic gradient descent (SGD) with a mini-batch size of 128.
- All weights were initialized from a zero-centered Normal distribution with standard deviation 0.02.
- The Generator uses ReLu activations as well as tanh for the final layer
- The Discriminator uses lRelu (leaky Relu) for all activation layers
- To implement, I used 3 different notebooks - 2 on Google Colab and 1 on Paperspace Gradient

Stuck in 2nd gear:

- So it turns out building a stable DCGAN is really challenging
- I managed to generate Pokémon-esque blobs of pixels
- Performance of the model was relatively static in that my generator loss and discriminator loss maintained the same scores throughout
- 52,000 epochs
- gen_loss=0.6922744512557983, disc_loss=1.007383942604065, 0:00:00.63



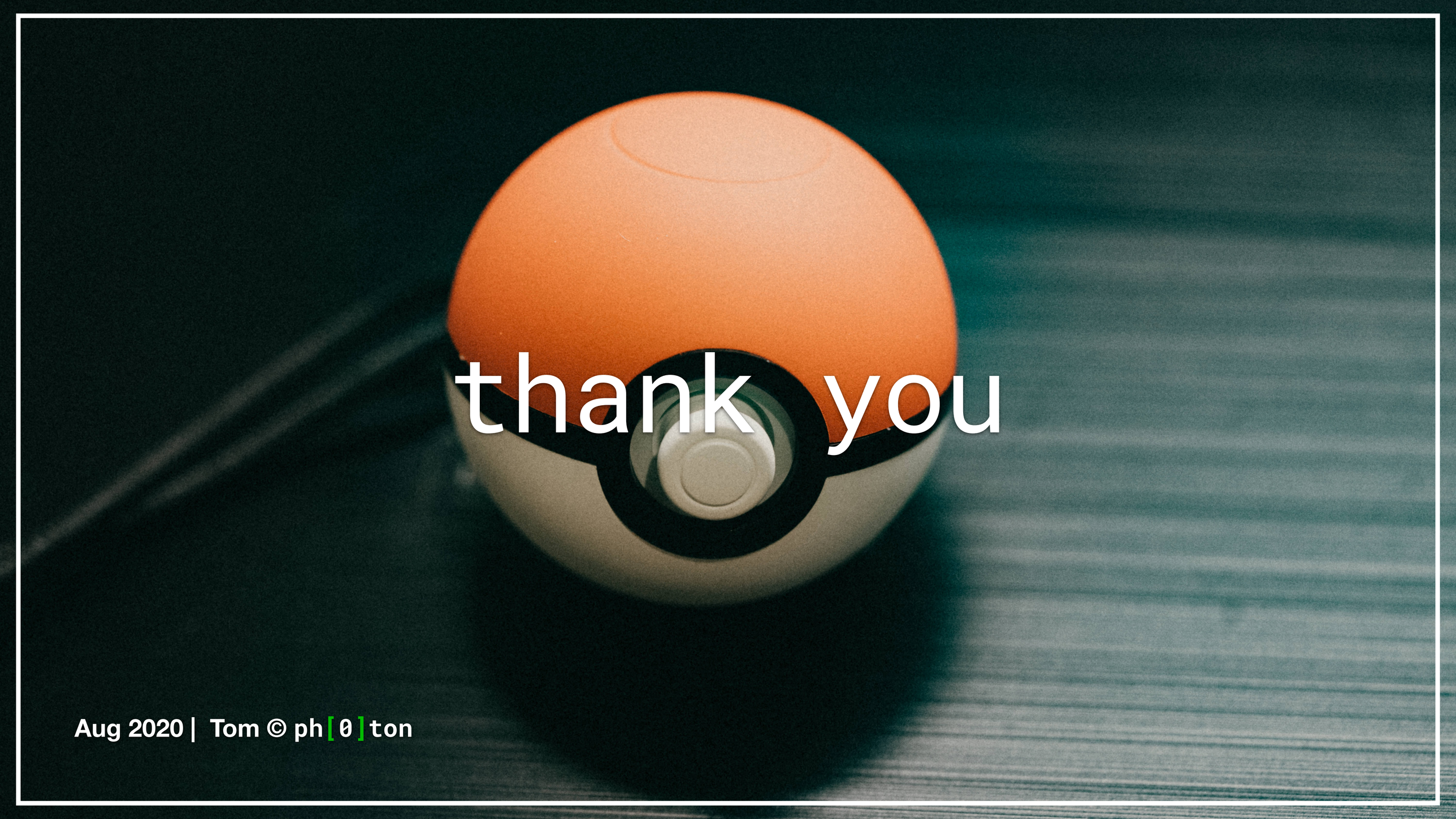


Also the blinker is on:

- **Limitations included:**
  - Cloud computing is complex to set up
  - Free GPU’s are great until they aren’t
  - Limited understanding of how to further tune hyperparameters
  - Stable, High-Res GAN’s are TOUGH

Next Steps:

- LEARN LEARN LEARN
- Successfully set up project to run with Google Cloud TPU’s
- Generate actual Pokémon
- Take generated Pokémon and run them through the CNN to see how they would be classified
- Tell everyone I’ve ever met that I did it



Aug 2020 | Tom © ph[0]ton