Udacity Machine Learning Engineer Nanodegree
Capstone Project

Eric Seidler
2021-03-30

I. Definition

Project Overview

This Person Does Not Exist is a website released in 2018 that displays photorealistic images of people that do not exist. The images are created using a type of unsupervised learning known as generative adversarial networks [1]. This type of network contains two main components, a generator and a discriminator. In the context of this specific application, the generator starts with random data and through training learns how to create more and more realistic photos of humans while the discriminator attempts to tell the difference between photos of real humans and artificially generated ones [2]. The end result is something that (to a human) appears to be a photo of a real person but is not.

I was fascinated by this concept when I first heard about it. Immediately a question came to mind: “Could a machine learning model be trained to outsmart another one?”. In other words, a human would not be able to differentiate between a photo of a real human and one from this website, but could we train a machine learning model to do so?

For this project, I created a machine learning model to tell the difference between a generated photo of a human face and a photo of a real human face. Transfer learning is the process of choosing an existing model that has already been trained on huge datasets, making a slight modification to that model, and then using it for a specific purpose. Within the context of this project, a PyTorch model was used as the starting point, and a custom neural network classifier was attached for the work of determining whether a photo was real or generated. Generated photos were pulled from the aforementioned website using a Python script. Photos of real human faces were taken from two different Kaggle datasets which will be described in more detail below.

Problem Statement

The problem at hand can be described as such. How can we use machine learning, specifically a custom neural network in conjunction with transfer learning, to differentiate between images of real people and artificially generated images of people? And once a model is trained, how accurate is that model?

The steps involved in reaching a solution to this problem include the following:

1. Download and organize real and generated photos of human faces
2. Perform data loading and preprocessing
3. Train the model
4. Evaluate the model's accuracy on a test dataset as well as a few "edge case" images
5. Make changes to the model and/or input data as needed and repeat steps 1 - 4

The hypothesis is that a properly trained model will be able to differentiate between real and generated photos with a reasonable, but not necessarily perfect, level of accuracy. This is based both on the generally accepted notion that machine learning models are great at recognizing patterns in data that are not recognizable by humans. The "reasonable" portion of the hypothesis was included to account for the fact that some of these generated images do look extremely realistic.

Metrics

The metric used in this project was accuracy. Within the problem domain of binary classification, accuracy is defined as the sum of true positives and true negatives divided by the size of the dataset [3].

$$accuracy = \frac{true\ positives + true\ negatives}{total\ size\ of\ dataset}$$

Within the context of this problem, we want a simple, quantitative way to answer the following question. Is the model "right" when making predictions about whether an image is real or generated? Accuracy fits the bill here because it takes into account how many predictions (real or generated) were actually correct. And since accuracy is a commonly used metric, it will be trivial to use this same metric when evaluating other binary classifiers which will serve as a benchmark. The other classifiers used will be discussed in greater detail below.

II. Analysis

Data Exploration and Visualization

Generated photos of human faces were obtained via a Python script from the website www.thispersondoesnotexist.com. Five hundred sixty-seven photos were gathered for a train, validation, and test split of 400, 100, and 67, respectively. Although a larger training dataset would likely produce better results, it was hypothesized that a dataset of this size would be sufficient for determining whether a model would have a decent chance of differentiating between real and generated photos. Images from this dataset have the dimensions of 1024x1024 pixels. Generated images include a decent variety in terms of age, ethnicity, and sex.

Figure 1: Sample Images from Generated Images Dataset

Two different datasets were used for real faces. The first was the Kaggle dataset Labeled Faces in the Wild [4]. The images dimensions are 250x250 pixels. The same 567 images were collected for a parallel 400, 100, and 67 train, validation, and test split. The second dataset for real faces was taken from the Kaggle dataset Real and Fake Face Detection [5]. This set of images contain the dimensions 600x600 pixels, and were set up with the same train, validation, and test split. In both datasets of real images, the entire dataset was downloaded and a series of command-line programs were used to randomize the specific images selected for use in the training data. Just as in the generated photos dataset, real images include a decent variety in terms of age, ethnicity, and sex.

Figure 2: Sample Images from first Real Images Dataset

Figure 3: Sample Images from second Real Images Dataset

There were a few data abnormalities or aspects of the data that needed to be addressed. In most data samples (of images of human faces), there was just one person in the photo.

However, in some of the samples there were additional faces or parts of faces that were present. Images like these were manually removed from the training data for both the real and generated datasets. The rationale was that whether the image was generated or real, we wanted to train the model on images of single faces, not multiple faces. Even if there was just part of another face in a sample image, that image was removed for data consistency.

Figure 4: Real Image with Multiple Faces Present

Figure 5: Generated Image with Multiple Faces Present

In addition to images with multiple faces, some of the images from the generated dataset contained obvious discoloration. These images were removed due to the fact that human faces do not typically naturally contain vivid colors and so it would not be helpful to train the model on such data.

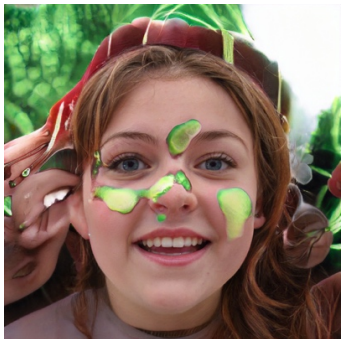

Figure 6: Generated Image with Facial Discoloration

Lastly, there is the question of why two different datasets of real photos were gathered. This will be addressed below in the Refinement portion of the Methodology section.

Algorithms and Techniques

DenseNet161 was chosen as the base model for transfer learning because it is implemented in the PyTorch machine learning framework, and furthermore, this specific model has performed well for me on a previous image classification project. It is a type of Convolutional Neural Network which is a deep learning neural network used for image processing tasks. CNNs are especially suited for use with image data because they reduce the number of network connections which can quickly become infeasible for image processing with traditional fully-connected neural networks [6]. Typically, these types of deep learning neural networks require a vast amount of input data on which to train, but thanks to the technique of transfer learning, a smaller dataset can be used for training. With transfer learning, the final layers of an existing neural network are removed and replaced with a custom neural network that ends up performing the final classification. The conveniently provided DataLoader from PyTorch [7] takes care of handling the batch portion of the Mini-batch gradient descent algorithm [8] used in the training.

Parameters are as follows:

- Training Parameters
    - Number of epochs
    - Batch size
    - Loss criterion
    - Optimizer
- Custom Neural Net Classifier
    - Number of layers
    - Degree of layer connectedness (e.g., full or partial)
    - Dropout rate (to prevent overfitting)

Benchmark

Two simple models were used as a point of comparison. The first was a dummy estimator from sklearn [9]. This type of classifier basically takes a random guess without even looking at the input data. While this at first may seem silly, in terms of a benchmark, it is a good point of comparison because if the trained model does about as well as a random guess, then it is time to reevaluate the model or overall approach. The second benchmark model used was a binary SVM classifier [10]. This type of model is not typically used for image processing and was expected to have a lower accuracy than that of the model created through transfer learning. While the organization of the input data was just slightly different for the benchmark models, both benchmark models can be easily compared using the same aforementioned accuracy metric. The accuracy of the dummy estimator was 52.2% and the accuracy of the SVM classifier was 51%.

III. Methodology

Data Preprocessing

Three different high-level steps were used when performing the data preprocessing. The first step was to take the collection of images and organize them into specific folders that would facilitate simple loading of the images when using the transfer learning approach as well as the simple binary classifiers. When using the transfer learning approach, images were organized into train, validation, and test folders, with each of those folders containing a folder called "real" and "fake" to hold real and generated photos of human faces, respectively. When using the binary classifiers, it was sufficient to organize all generated photos into a folder called "all_fake" and all real photos into a folder called "all_real". This work was done using a combination of Python scripts and shell commands, and randomization was used whenever possible to prevent biased selection of input data samples.

Image preprocessing was performed on all datasets, regardless of the model or approach used to perform classification. For both the transfer learning approach as well as the simple binary classifiers (dummy, SVM classifier), the following transformations were performed on every image before processing:

1. Resize image to 255x255 pixels
2. Crop the resized image to 224x224 pixels
3. Convert the cropped image to a tensor for subsequent processing
4. Normalize pixel values to a range that would work with the DenseNet161 starting model

One additional step was done for the transfer learning approach for only the training dataset. This was to randomly horizontally flip the image before doing training. This was done to prevent overfitting. Horizontal flipping, and not vertical flipping, was chosen because the assumption was made that all input images would be a picture of a face that was "right-side up" since this is the normal orientation of a human face.

Note that images were intentionally not converted to gray scale. This was because it was hypothesized that the transfer learning approach might be able to detect more patterns in RGB color images than it would in grayscale images. In other types of machine learning problems dealing with image data, such as object or word detection, that type of conversation to grayscale would make more sense.

Lastly, as mentioned in the Data Exploration and Visualization section above, all images containing multiple faces or obvious facial discolorations were manually removed from the input datasets.

Implementation

The implementation was slightly different when using the transfer learning model and the other two models used for benchmarking: the dummy classifier and the SVM classifier. For the

transfer learning approach, the implementation steps are as follows and can be more closely examined in the Jupyter notebook called “Capstone”. The same basic steps can also be seen in the Jupyter notebook called “Capstone Round2”, which will be explained in more detail in the Refinement section below.

1. Create a Python dictionary to store the preprocessing transformations described above
2. Create PyTorch DataLoaders for use in the mini-batch gradient descent algorithm
3. Load the DenseNet161 pretrained model
4. Create a new custom neural network to serve as the classifier and assign that classifier to the pretrained model
5. Define the loss function and optimizer to use during the training
6. Train the model using the mini-batch gradient descent technique on the training and validation dataset
7. Evaluate the accuracy of the model on the set of test images
8. Manually send a few images of interest through the trained model to get a more complete sense of model accuracy

For the SVM binary classifier the approach was:

1. Create a function to do the same image preprocessing used in the transfer learning approach
2. Use that function to perform preprocessing on all images, real and generated
3. Shuffle all image, real and generated, and manually divide into train, validation, and test datasets
4. Use SVC from sklearn to fit the model to the training data
5. Adjust one of the hyperparameters after making a prediction on the validation data and repeat Step 4, creating a slightly different trained model
6. Choose one of the trained models and evaluate model accuracy on the test data
7. Manually send a few images of interest through the trained model to get a more complete sense of model accuracy

For the dummy classifier, steps 1 – 4 were the same. Step 5 was evaluating the accuracy of the model on the test data. Since a dummy classifier makes a random guess, there was no need to adjust hyperparameters using the validation dataset.

Refinement

On the first pass of using the transfer learning approach, the accuracy on the test dataset was reported as 100%. This was interpreted as a warning sign that something was off because complete accuracy seems to be fairly unlikely, especially with a first attempt at training a model. Furthermore, even before running the first training, there was some suspicion around the fact that all the real photos of human faces weren’t as closely cropped to the subject’s face as the generated photos were. After seeing 100% accuracy of the trained model, a few specific images were manually selected and passed to the model for prediction. That set of images

confirmed the suspicion: all photos of humans that were closely cropped to the face of the subject were considered generated photos (see the end of the Jupyter notebook called “Capstone”). At this point, it was clear that a new set of data for the real photos of human faces was needed.

The second set of real photos, mentioned in the Data Exploration and Visualization section above, were more closely cropped to the subject’s face and yielded results that were a little more expected and slightly better. This training attempt was executed in the Jupyter Notebook called “Capstone Round 2”. After two epochs of training, the accuracy on the validation dataset was at 83% and the accuracy of the model on the test dataset was 84.56%. This model also correctly identified three out of the four of the hand-selected images as generated or real.

```
Epoch: 1/2
Training loss: 0.615
Validation loss: 0.388
Validation accuracy: 84.5%
----------------------------------------
Epoch: 2/2
Training loss: 0.477
Validation loss: 0.374
Validation accuracy: 83.0%
----------------------------------------

real_murray.jpg
[0.9474542 0.05254574]
['real', 'fake']
----------
real-ryan.jpg
[0.67568403 0.32431594]
['real', 'fake']
----------
close-up-real.jpg
[0.61182207 0.38817793]
['fake', 'real']
----------
fake1.jpg
[0.91382855 0.08617148]
['fake', 'real']
----------
```

Two more attempts were made in the same “Round 2” notebook to train the transfer learning model with a greater number of epochs to see if that would make any significant difference in accuracy on the test datasets. For five epochs, final validation accuracy was 84.5% and test accuracy was 86.76%. Note that accuracy did not change from epoch 2 to 3 and actually went down after epoch 3.

```
Epoch: 1/5
Training loss: 0.607
Validation loss: 0.431
Validation accuracy: 80.0%
----------------------------------------
```

```
Epoch: 2/5
Training loss: 0.474
Validation loss: 0.264
Validation accuracy: 90.5%
----------------------------------------
Epoch: 3/5
Training loss: 0.484
Validation loss: 0.272
Validation accuracy: 90.5%
----------------------------------------
Epoch: 4/5
Training loss: 0.446
Validation loss: 0.303
Validation accuracy: 89.5%
----------------------------------------
Epoch: 5/5
Training loss: 0.438
Validation loss: 0.348
Validation accuracy: 84.5%
----------------------------------------

real_murray.jpg
[0.92451733 0.0754827 ]
['real', 'fake']
-----------
real-ryan.jpg
[0.65155834 0.34844166]
['real', 'fake']
-----------
close-up-real.jpg
[0.6847138  0.31528625]
['fake', 'real']
-----------
fake1.jpg
[0.9086213 0.0913787]
['fake', 'real']
-----------
```

IV. Results

Model Evaluation and Validation

The final model used the same transfer learning approach, but with three training epochs. For this instance of the trained model, the final validation accuracy was 87.5% and test accuracy was 88.97%. Interestingly, for the third hand-selected image, the confidence level (probability) for the image being classified as generated increased, even though the prediction was still wrong.

```
Epoch: 1/3
Training loss: 0.619
Validation loss: 0.395
Validation accuracy: 87.0%
```

```
----------------------------------------
Epoch: 2/3
Training loss: 0.475
Validation loss: 0.41
Validation accuracy: 87.0%
----------------------------------------
Epoch: 3/3
Training loss: 0.459
Validation loss: 0.306
Validation accuracy: 87.5%
----------------------------------------

real_murray.jpg
[0.97550154 0.02449848]
['real', 'fake']
-----------
real-ryan.jpg
[0.85970396 0.14029601]
['real', 'fake']
-----------
close-up-real.jpg
[0.80078644 0.1992136 ]
['fake', 'real']
-----------
fake1.jpg
[0.94383955 0.05616042]
['fake', 'real']
-----------
```

The final model can be described as follows:

- Transfer learning approach using DenseNet161 as a starting point
- Custom neural network classifier attached at the end of the DenseNet161 starting point
    - Two hidden layers: 2208 -> 512 -> 256 -> 2
    - Training dropout rate of 0.2
    - Learning rate of 0.002
    - Loss criterion of negative log likelihood loss
    - Three training epochs

Just like with the other preliminary transfer learning model approaches, a small hand-selected set of test images were fed through the model to determine overall model usefulness. The final model was able to correctly classify, with a fairly high certainty (probability), three out of these four images. The model was reasonably certain about the third image being a generated image, even though it was wrong about that. Even so, overall accuracy of almost 89% seems reasonable, especially for the relatively small size of the input datasets.

Justification

Looking back to the accuracy values from the benchmark section, recall that the accuracy of the dummy estimator was 52.2% while the accuracy of the SVM classifier was 51%. The SVM classifier also labeled all four hand-selected test images as generated, even though only one of

those images were actually generated. The final model accuracy of 88.97% was significantly better. It was also able to correctly identify three of the four hand-selected test images. Although the final model was not a perfect solution, it was definitely enough to (in most cases) detect whether an image was real or generated.

V. Conclusion

Free-Form Visualization

As mentioned in the Results section above, these four hand-selected images served as a good way to verify overall model performance. In the final model, images were classified with over 80% probability, indicating a high level of model confidence. Unfortunately, the third image below, which is a real photo, was classified as a generated image. My hypothesis for this is that the background of this image is playing a non-trivial part in the prediction. See the Improvement section below for more details.

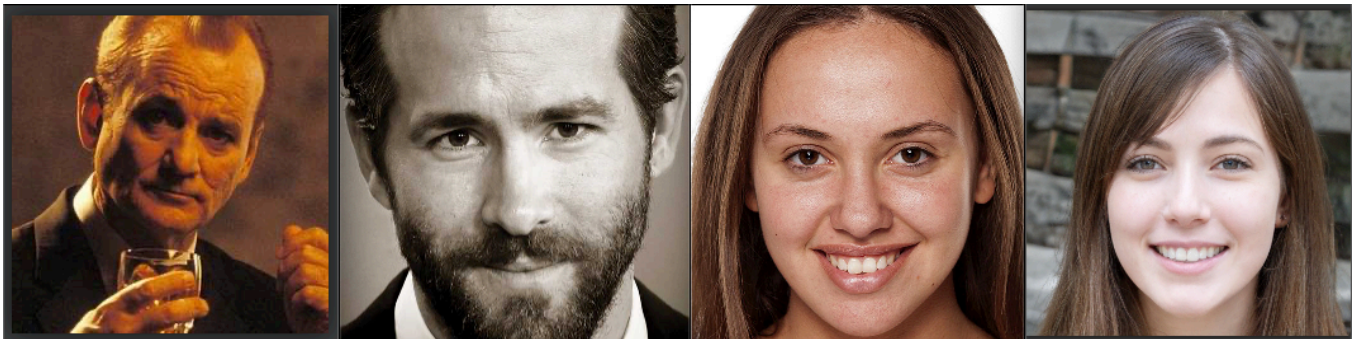

Figure 7: Four hand-selected image samples
`real_murray.jpg`, `real-ryan.jpg`, `close-up-real.jpg` and `fake1.jpg`

Reflection

This project was comprised of the following steps:

1. Initial problem statement and proposal, including identifying sources of data
2. Retrieving and organizing input data
3. Deciding on a starting point for the transfer learning approach
4. Creating a custom neural net classifier to extend the base model
5. Training on the initial datasets
6. Observing that the real photos from the initial input datasets were resulting in inflated accuracy values
7. Going back to gather more suitable real photos to use in the input data
8. Repeating the training, making adjustments to hyperparameters
9. Validating the intermediate and final models using both validation and test datasets as well as hand-selected edge case input images

I found the most interesting aspect of this project to be gathering, organizing, and evaluating the input data. In this particular problem, the first training attempt that yielded 100% accuracy gave me a strong indicator that the input data needed to be revisited before doing any more training or model adjustments. It was also nice that with the input data being image data, it was literally clear to see how the model arrived at some of its initial, flawed conclusions, such as "every image that is closely cropped to the face is a generated image". I would assume that in many cases, finding flaws or assumptions in the input data would be more difficult to do.

The final model does fit my expectations regarding a machine learning model being able to detect generated photos of human faces. That being said, there is definitely some room for improvement. Those ideas will be discussed in the following section.

Improvement

Below are a few ideas for consideration with respect to improving and generalizing the model.

- Train on a much larger input dataset. There are several large datasets available on the internet. Using a larger training dataset would prevent the model from overfitting or focusing on insignificant characteristics of the input images.
- Train on an input dataset that contains more similar backgrounds for real and generated images. For example, consider the third hand-select image from the Free-Form Visualization section above.
- Choose a different base model to use in the transfer learning. There are at least twelve other models from the `torchvision.models` package that could be used.
- Spend more time experimenting with custom neural net classifiers, including different numbers of hidden layers, etc.

VI. Sources

[1] https://papers.nips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf
[2] https://developers.google.com/machine-learning/gan/gan_structure
[3] https://en.wikipedia.org/wiki/Accuracy_and_precision#In_binary_classification
[4] https://www.kaggle.com/atulanandjha/lfwpeople
[5] https://www.kaggle.com/ciplab/real-and-fake-face-detection
[6] https://medium.com/ml-cheat-sheet/convolutional-neural-networks-186870efbf71
[7] https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader
[8] https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size
[9] https://scikit-learn.org/stable/modules/model_evaluation.html#dummy-estimators
[10] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html