Pigeon Classification using CNNs to Create Stickers for Messaging Platforms

Leah Kim, Steve Marquez

April 29, 2024

Abstract

My final project aims to use transfer learning on convolutional neural networks for binary classification, specifically on pigeon detection. The overarching goal is to create a mobile-friendly application that, when given a photo with some pigeon, will generate a cartoon-style pigeon before being sent as a sticker on messaging platforms. There are two parts to this project: (A) Identify pigeons in photographs and evaluate the results on challenging datasets. (B) Given photograph inputs of pigeons, the model outputs a cartoon-style image of that very pigeon.

Introduction

To build efficient means of pigeon detection and reimagine pigeons as fun cartoons, I explored various CNNs, GAN architectures, and techniques such as image segmentation.

Previous Work

Older methods for image cartoonization have holistic approaches that attempt to transform the entire image. These were filter-based techniques that emphasized a specific feature such as distinct edges, simpler color palettes, or smoother surfaces that gave a more flat appearance. Prior to choosing this research idea to recreate, the original plan had been to implement CartoonGAN, a method that used deep learning focused on edge-promotion loss. The goal was to generate images that had clear, distinct edges.

Previous works relied on global adjustments, which usually resulted in overstylization or great loss in detail.

Overview

Since the intended use is for cartoonizing pigeons, the classification model was initially designed to recognize whether an input image was a bird or not. To make the project more meaningful however, the model was later trained to identify at least 201 species of birds. This way, if a user wrongly identifies a pigeon, they would receive feedback and also learn to better identify birds themselves.

To implement the image cartoonization, I recreated ideas from a research paper: "Learning to Cartoonize Using White-box Cartoon Representations"[5]:

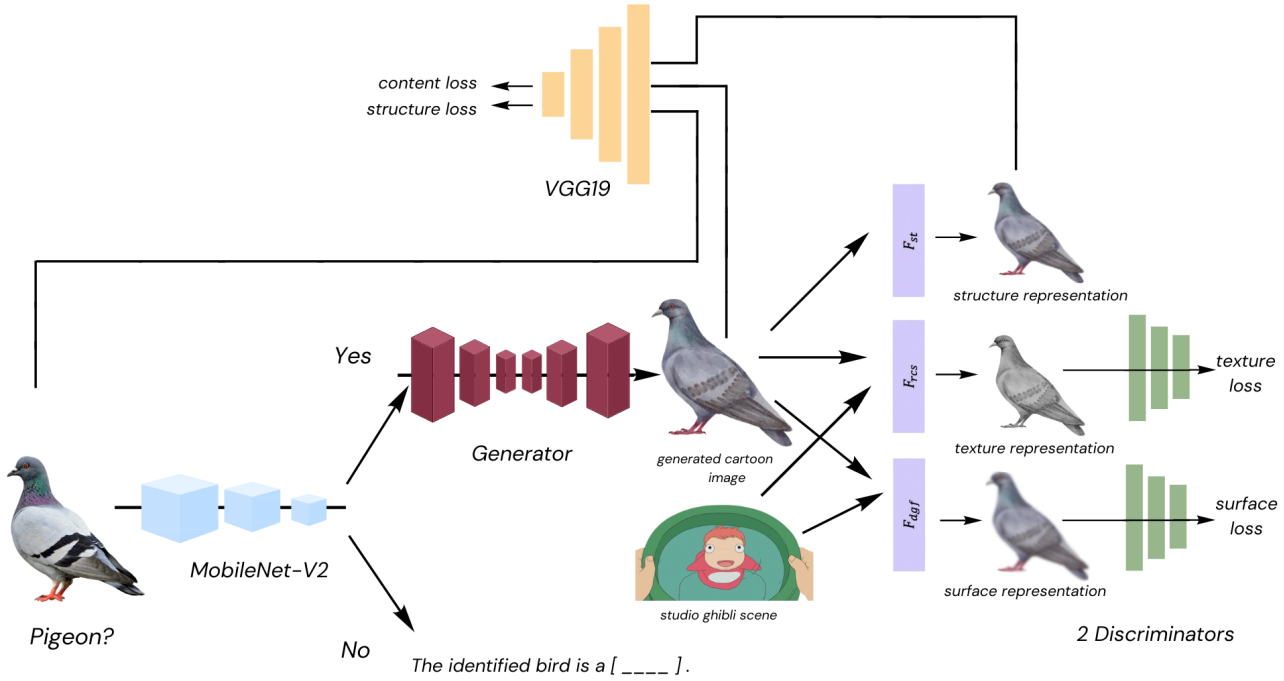
1. Extract three distinct cartoon representations from an image: surface, structure, and texture representation.

2. Use a GAN framework that will learn the three representations and generate cartoonized images.

Not all techniques mentioned in the paper were implemented due to time and resource constraints (also considering difficulty). Parts of the paper were obscure, therefore some parts of our implementation used simpler algorithms. These discrepancies are further addressed in our results and analysis section.

Part A: Pigeon Detection

Transfer learning utilizes a model that has already been pre-trained, and its learnings are used to complete another task. Due to limited pigeon datasets available, this project leverages feature representations from the MobileNetV2 convolutional neural network which has been trained on ImageNet, found in Keras Applications. Although MobileNetV2 may



Full Project Architecture

not be considered as accurate in comparison to larger and more complex models, I decided it was the most efficient model suitable for mobile devices, which is the end goal of this project.

Datasets

In our first approach, rather than using bird classification datasets, I utilized datasets with pigeons vs. no pigeons (positive and negative samples). Initially for the positive samples, I found a dataset[1] with 125 images of pigeons, and I chose the Stanford Background Dataset[3] for our negative samples which contained scenes that pigeons might be found in but weren't present. However, this wasn't good for fine-tuning the model because the model was predicting towards the more common class of negative samples despite seeing positive samples. I tried to augment the positive dataset, but ultimately I extracted images of pigeons from an online gallery[2] which gave us 679 positive samples. Despite this, I realized that when testing on more challenging datasets, such as against other species of birds, the model was not able to generalize– since the only bird it was trained on was pigeons. I explored a second approach: use a different dataset with various species of birds for multi-class classification. My model should be able to identify pigeons even when given photos of other birds. To achieve this, I used the Caltech-UCSD Birds dataset[4] which contains 200 bird species. In this approach, the positive samples were still images with pigeons, and negative samples were any image without pigeons. The model was trained on a total of 201 classes.

Preparing the Data

In the first approach, my positive and negative datasets were split into train, validate, and test folders with using an 80-10-10 ratio. All datasets have been uploaded on Google Drive and a python script was used to label all images into their respective folders.

In this second approach, the data was split into test and train folders using a 90-10 split, and each folder contained all 201 classes of birds.

I utilized the ImageDataGenerator class from the Keras API to preprocess and augment our data. ImageDataGenerator allows the model to receive new variations of the images at each epoch real-time, intended to improve generalization without using extra storage. input.

Model Architecture

MobileNetV2 is the ideal model for my project because its architecture was designed to be efficient for real-time processing required by our mobile application. As the model operates with low latency, the application would be able to perform real-time pigeon detection to deliver a positive user experience on mobile devices. My project leverages weights from pretraining on the ImageNet dataset, which contains over 14 million annotated images for feature detection. This provides an optimal starting point for detecting pigeons and eliminates unnecessary computational costs of training a NN from scratch.

The Process

To make full use of the previously learned features, the layers of the initial model were frozen, and additional layers specific to the pigeon detection task were added.

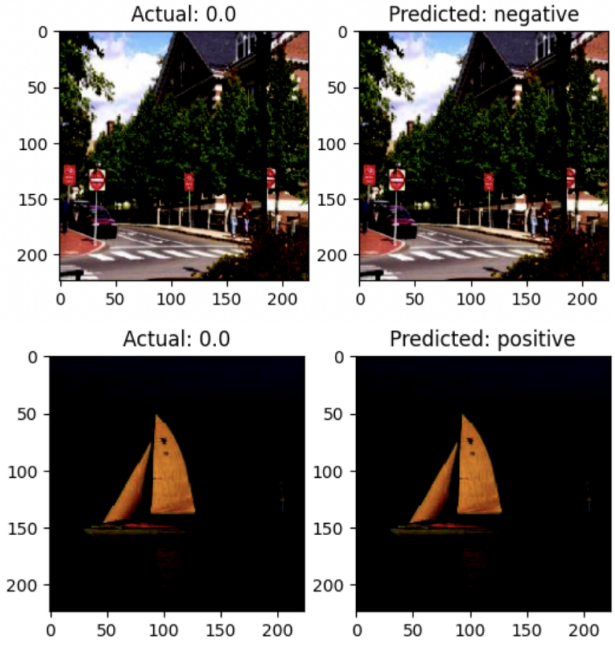
In the first approach, the model was fine-tuned to generate more accurate predictions by unfreezing the last 20 layers. The trained model was really good at binary classification, however I realized that the model should be able to differentiate pigeons from

other classes of birds. I decided that only pigeons would be cartoonized, and any other bird will simply be identified but dismissed. Although a final, binary outcome is desired, the model should be able to identify different species, therefore is a multi-class classification. In this second approach, the MobileNetV2 model was trained on a total of 201 classes, as the Caltech dataset of 200 birds was combined with Cornell lab's images of pigeons.

To do this, I changed my approach: first, initialize the pre-trained MobileNetV2 base model and apply transfer learning by adding additional fully connected layers for multi-class classification. I used the Caltech-UCSD Birds dataset to train the model. Rather than relying on the assumption that users would input photos of a pigeon or no pigeon, this approach ensures that a pigeon can be detected in any photo, in case a user mistakenly takes a photo of a bird that is not a pigeon. For better accuracy, layers starting from Block 13 were unfrozen for finetuning.

Analysis of Results

The model was trained over 10 epochs and returned consistent improvement on the training and validation datasets as shown in Figure 2. Upon evaluating the trained model on the test dataset, the loss was 0.286 and the accuracy was 0.862. These are sample results:

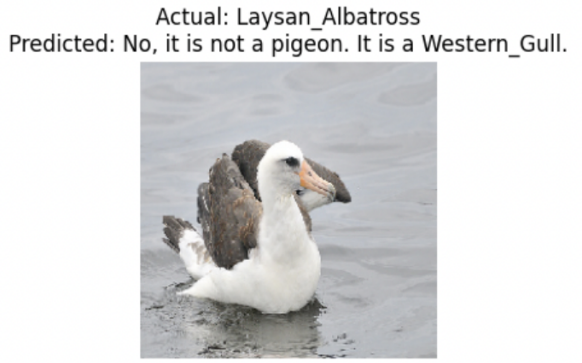


To achieve higher accuracy, the model was fine-tuned by unfreezing the last 20 layers to allow these additional weights to be updated during the next training process. The model was trained for 5 additional epochs and when evaluated on the test data, returned a 0.00024 loss and 1.0 accuracy. This is too accurate:

Epoch	Loss	Accuracy	Val_Loss	Val_Accuracy
1	1.0622	0.3481	0.7138	0.4965
5	0.4049	0.8585	0.4572	0.8298
10	0.2037	0.9520	0.2930	0.9078
Test	0.2859	0.8623	--	--
Finetuning	0.0002413	1.0	--	

To ensure that the model was not overfitting, I ran the model on images of other bird species. As we anticipated, the model was unable to differentiate whether a bird was a pigeon or not. This motivated my second approach: train the model for multi-class classification.

In the second approach, the model was trained with 201 classes, with the 201th class contained images of pigeons. I still used the same MobileNetV2 CNN and datasets, and the only change was the output, which went from binary to multi-class. The model was trained over 50 epochs and reached a 64% accuracy. These are some predictions at this stage of training the model:



To achieve better accuracy, I unfroze previous layers from the original base model. These layers were fine-tuned for 18 epochs and reached 82% accuracy.

Epoch	Loss	Accuracy	Val_Loss	Val_Accuracy
43	0.891	0.7413	1.2972	0.6397
46	0.8366	0.7564	1.3003	0.6364
50	0.8079	0.7672	1.2847	0.6413
Test	1.2847	0.6413	--	--

Part B: Pigeon Cartoonization

The essential features of a cartoon image are:

- smooth surface
- sparse color blocks
- high frequency texture

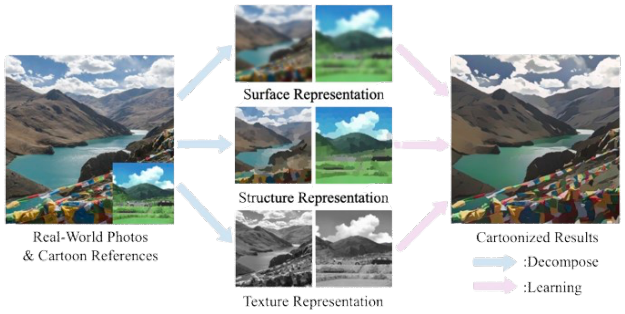


Figure 2 from the research paper

The surface representation reduces high frequency details by smoothing the image. The goal is to reduce texture and noise while preserving the overall geometry of the image. The structure representation aims to preserve the integrity of the image layout so that it is still recognizable. The texture representation, shown in grayscale, focuses on preserving the edges and fine textures to retain depth and detail.

Rather than prioritizing one cartoon feature at the expense of another, the research paper treats each feature independently. Each representation focuses on a different aspect of the input image, which effectively preserves the content integrity of the original images while adding the cartoon effects.

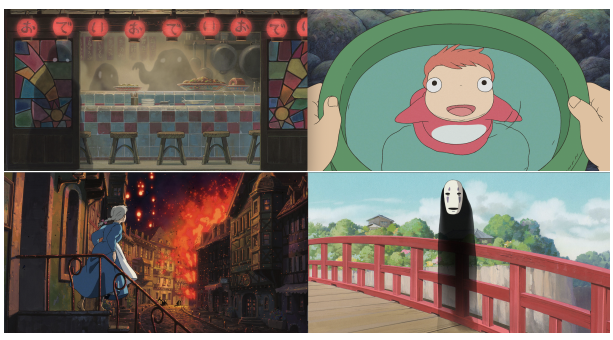
Datasets

To use as input images for the generator, I used the same pigeon gallery and Stanford background dataset used in Part A. We also included landscape images from a gallery[6]

I used scenes from various Studio Ghibli films as the cartoon images. The researchers acquired 20,000 cartoon images but were not able to share their source due to legal reasons. Any cartoon dataset would have been sufficient for this step.

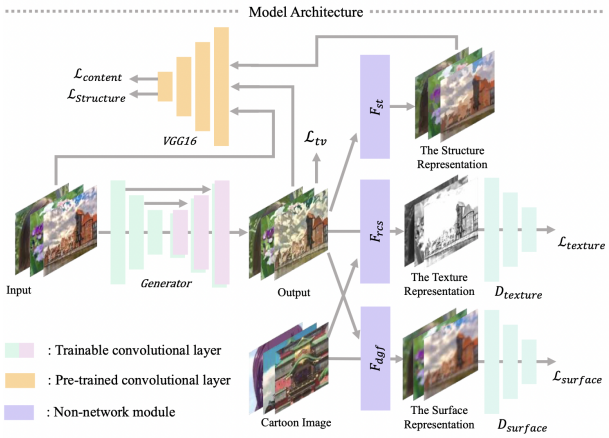
Preparing the Data

The dataset was split between real images and target cartoon images. I gathered 5,718 real-world images and found 878 freely available scenes from the Studio Ghibli webpage. Because there were far more real-world images than cartoon, I repeatedly augmented and shuffled the cartoon images.



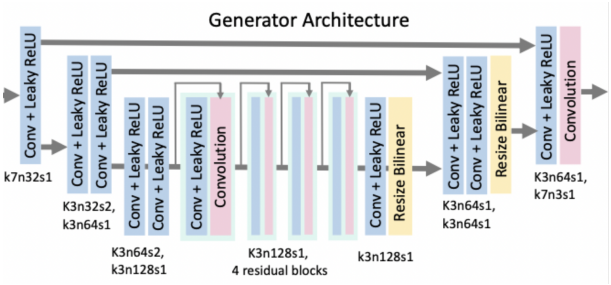
Sample cartoon images from Studio Ghibli

Model Architecture



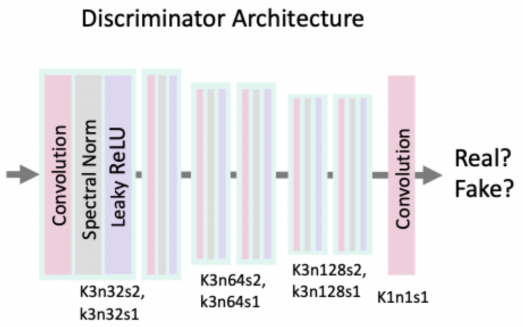
The research paper uses 1 generator and 2 discriminators in their GAN framework, along with a pre-trained VGG network. The VGG model is used for feature extraction to ensure that generated images retain the content of the original image.

The generator transforms real-world images into cartoon style images through training with a surface discriminator and a texture discriminator. The structure and content loss is measured by comparing the features extracted by the VGG model. Specifically, the fourth convolutional layer was chosen for feature extraction. Since the layer is deeper in the network, it captures higher-level features which was important for the texture representation in cartoon images.



The generator was first pre-trained on real-world images to prioritize content integrity. This was the loss function used to calculate the content loss:

$$\mathcal{L}_{content} = \|VGG_n(G(\boldsymbol{I}_p)) - VGG_n(\boldsymbol{I}_p)\|$$



The surface discriminator provides feedback on the smoothness feature of the generated images, using a differentiable guided filter to smooth the images while preserving edges. This was the loss function used to calculate surface loss:

$$\mathcal{L}_{surface}(G,\ D_s) = logD_s(\mathcal{F}_{dgf}(\boldsymbol{I}_c, \boldsymbol{I}_c)) + log(1 - D_s(\mathcal{F}_{dgf}(G(\boldsymbol{I}_p), G(\boldsymbol{I}_p))))$$

Generated cartoon images were converted into grayscale as input to the texture discriminator. To prioritize high-frequency features and edges, a random color shift was applied to better learn texture details instead of colors. This was the loss function:

$$\mathcal{L}_{texture}(G, D_t) = logD_t(\mathcal{F}_{rcs}(\boldsymbol{I}_c)) + log(1 - D_t(\mathcal{F}_{rcs}(G(\boldsymbol{I}_p))))$$

Other loss functions were implemented to measure the structure, total-variation, and total loss– all of which are provided in the source code.

Analysis of Results

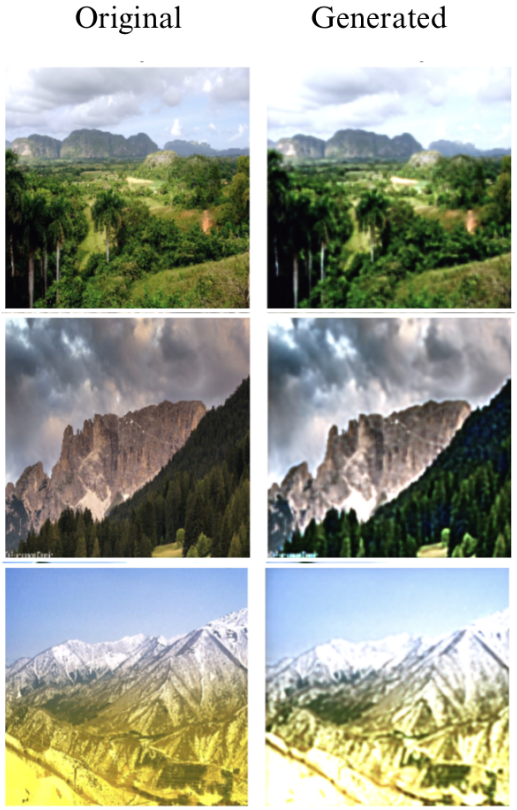
Below are sample results from pre-training the generator before incorporating discriminators:

The model uses a set of lambda values λ as weights in the total loss function to balance the influence of the various features. λ_1, the surface loss, influenced how smooth the generated images should be. λ_2, the texture loss, influences the sharpness of the images. λ_3 structure loss controls the global structure and color blocks. λ_4 is the content loss that influences the semantic similarity between the original and generated image, and λ_5 is the total variation loss that influences the noise in the image.

With a heavy λ_5, the generated images had much emphasis on smoothness and reducing noise, and the content was preserved well. However, the images were too blurry and did not resemble cartoon images:

$\lambda_1 = 0.05,\ \lambda_2 = 100,\ \lambda_3 = 500,\ \lambda_4 = 1000,\ \lambda_5 = 500$

Giving λ_5 a lower weight and increasing λ_4 resulted in more emphasis on texture and edges which resulted in less blurriness:



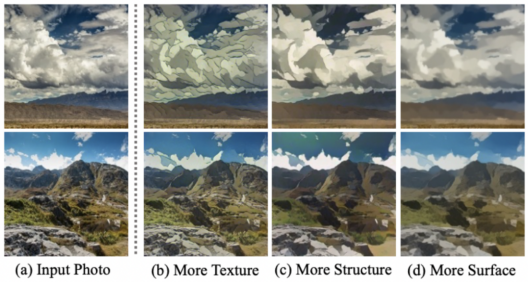
$\lambda_1 = 1,\ \lambda_2 = 100,\ \lambda_3 = 400,\ \lambda_4 = 1600,\ \lambda_5 = 10$

Increasing λtexture led to dominant texture replication at the expense of color accuracy:

When lowering the λtexture weight once again, the structure and content were still preserved but results were still not our target Studio-Ghibli style cartoons. A major challenge seemed to be creating bright and sparse color blocks while preserving edges:

$\lambda_1 = 1,\ \lambda_2 = 20,\ \lambda_3 = 400,\ \lambda_4 = 1600,\ \lambda_5 = 10$

For comparison, these are the results from the research paper:



Discussion

When training the GAN architecture, adjusting the hyperparameter values to find the right balance for our dataset proved to be very difficult. Despite both incremental and large changes to our weights, it was hard to generate the exact cartoon style the project aimed to achieve.

Although there is room for much experimentation and adjustments, one reason for the difficulty might be that I didn't incorporate all of the algorithms that the research paper proposed. Techniques such as using the Felzenszwalb algorithm were used to segment given images, and selective search merged similar regions. A new adaptive coloring algorithm enhanced image colors by considering pixel context and characteristics. Instead of directly implementing these techniques, I explored alternative methods such as the SLIC superpixel segmentation. This method generated superpixels based on color proximity and similarity

Conclusion

In this report, I explain the process of applying what I learned during the semester through implementing classification and GAN models. Although results were not as good as expected, I learned a lot about what to do and how.
Next steps for improvement can be implementing the coloring algorithms for better cartoonization. Running the classification and generator models led to fast results (5-10s) which is ideal for integrating it into a mobile app.

References

[1] Images.CV Pigeon Dataset. (n.d.). [Dataset]. https://images.cv/dataset/pigeon-image-classification-dataset
[2] The Cornell Lab of Ornithology Macaulay Library. (n.d.). Rock Pigeon- Columba Livia [Dataset]. https://search.macaulaylibrary.org/catalog?sort=rating_rank_desc
[3] S. Gould, R. Fulton, D. Koller. Decomposing a Scene into Geometric and Semantically Consistent Regions. Proceedings of International Conference on Computer Vision (ICCV), 2009. [pdf]
[4] Wah, C., Branson, S., Welinder, P., Perona, P., & Belongie, S. (2022). CUB-200-2011 (1.0) [Data set]. CaltechDATA. https://doi.org/10.22002/D1.20098
[5] Xinrui Wang and Jinze Yu "Learning to Cartoonize Using White-box Cartoon Representations." IEEE Conference on Computer Vision and Pattern Recognition, June 2020.
[6] Arnaud Rougetet. (2020). Landscape Pictures, Version 2. Retrieved 04/20/24 from https://www.kaggle.com/datasets/arnaud58/landscapepictures?select=00000002.jpg