SnapCat Filters: Using CNNs to Build Snapchat Filters for Cats

Tiffany Ding (tding5) Soryan Kumar (skumar12) Maggie Wu (mwu27) Sophie Yang (syang55)

CS1430 Spring 2020
Brown University
Github Repo: `https://github.com/maggie1059/SnapCat-Filters`

Abstract

In this project, we created a CNN to identify keypoints on a cat's face and used these predicted keypoints to overlay a filter on the cat, in the fashion of widely-used Snapchat filters. We preprocess our data by converting our images to grayscale, padding, and resizing, then train our model on approximately 7,000 images of cats along with prelabeled keypoints. Our model uses the MobileNetV2 architecture with unfrozen weights, along with a custom series of dense layers. Our model achieved a final testing accuracy of approximately 80% . Our pipeline works on individual images and is also able to superimpose Snapchat-like filters on cats in real time using video capture.

1. Introduction

Snapchat filters have become popular among human users, but cats have had limited opportunities to join in on the fun. In this project, we take on the challenge of adapting the ideas behind human facial feature detection to create Snapchat filters for cats.

2. Related Work

Facial landmark detection is a complex task, and there have been many methods used to tackle this problem. In the past decade, convolutional neural networks (CNNs) have taken off as a way of solving many image related tasks, and facial landmark detection is no exception. Sun et al. (2013) is an example of early work done using CNNs to perform facial landmark detection [3]. Using a facial bounding box outputted by a face detector, they cascade three levels of CNNs, which allows for coarse-to-fine prediction. Each level contains multiple CNNs that look at different areas of the image. For example, at the first level, there are three networks whose input regions are the whole face, eyes and nose, and nose and mouth.

More recently, there has been work done on adapting pre-trained models for facial landmark detection. In 2019, Saxen et al. compared the performance of using the pre-trained networks MobileNetV2 and NasNet-Mobile for human facial landmark detection using a dataset of 202,599 images of celebrities [2].

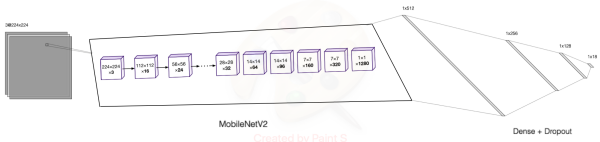
In our project, we contribute to existing literature on using CNNs for landmark detection by confirming that these methodologies that work for human faces can be applied successfully to cats. Additionally, we find that a training set of approximately 7,000 images is sufficient to achieve good results.

3. Methodology

Data. Our dataset was a Kaggle data set containing over 9,000 cat images with labelled facial key points: two for eyes, one for mouth, and six for ears. The data was separated into 7 different folders (of different sizes), with labels named corresponding to their images.

Preprocessing. The images in the data set came in a variety of shapes and sizes, so in order to run it through a CNN, we first had to convert them to the same size. Our first step of preprocessing was to pad each image so that its dimensions were square. Next, we took the padded image and resized it to 224 x 224. Since our training labels (the coordinates of the key points) are dependent on the size of image, we applied the same transformations to the coordinates. We also converted the images from RBG to grayscale and normalized the pixel values by dividing them by 255.

Storing all of our images in memory and passing them between different parts of our pipeline proved to be a recurring difficulty, so we chose to process the images one folder at a time (out of 7 folders total, with an average of 1200 images per folder) and stored these processed images and coordinates by writing them to file as .npy files. Our preprocessing was run separately from the rest of our pipeline, so we did not have to rerun it every time we trained our model. Instead, we could simply unpack and load our already-processed images and coordinates.



Layer (type)	Output Shape	Param #
mobilenetv2_1.00_224 (Model)	(None, 1280)	2257984
dense (Dense)	(None, 512)	655872
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 18)	2322

Total params: 3,080,402
Trainable params: 3,046,290
Non-trainable params: 34,112

Figure 1. Final model architecture using MobileNetV2.

Model Architecture. Our CNN uses Google's MobileNetV2 architecture with four additional dense layers [1]. Between dense layers, we use LeakyReLU (with $\alpha = 0.2$) as the activation function, and apply dropout at rate 0.1.

Training and Accuracy. We sequentially ran each preprocessed folder of train images through the model (one folder per epoch) with a batch size of 32. We used the Adam optimizer (with $\beta_1 = 0.9$ and $\beta_2 = 0.999$) to minimize the mean squared error between the predicted and actual keypoint coordinates. For MobileNetV2, we initialized the weights to the pre-trained weights, but did not freeze any layers. The model was trained for 150 epochs using a learning rate of 10^{-4}, and then for another 100 epochs using a learning rate of 10^{-5}. To obtain the best weights for our model, we used a validation split of 0.1 for each epoch, and saved the weights with the lowest validation loss at the end of training.

To determine the accuracy of our model, we computed the fraction of predicted keypoint coordinates whose distance from the ground-truth coordinates were within a fixed pixel threshold. We computed the test accuracy of our model using thresholds of 16 and 6 pixels. See Table 1 for details.

Filters. The positioning of the filters was determined using predicted keypoints on the cat's face. Given at least two keypoints, filters were resized, rotated, mirrored, and truncated to appropriately fit along the image. Filters were applied to images by overlaying the pixel values of the filter onto the image across all three image channels. Some filters, such as the bow filter and hat filter, required only two keypoints along the ears to fit properly. In these cases, the distance between the ear keypoints was used to determine the size of the filter, while the slope of the line containing both key points was used to adjust the angle of the filter. Additionally, filter truncation was applied when the predicted keypoints were close to the edges of the underlying cat image. Mirroring was also applied for the hat filter based on which predicted ear keypoint was higher. Other filters, such as the dog filter, required seven keypoints (three per ear and one for the nose) to fit properly. In this case, the dog ear and nose filters were overlayed as three separate filters onto the given image.

Live Video. To have our code truly emulate the "Snapchat" style of applying filters, we wrote a script that allows a user to see a filter applied over their cat in real time, using OpenCV's VideoCapture feature. This script reads in frames from a webcam in real time, runs each frame through our CNN to find predicted keypoints, fits a filter over these keypoints, then displays the result back to the user frame-by-frame, in the appearance of a video.

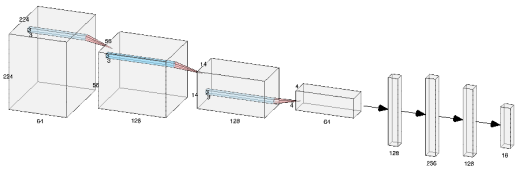
4. Experiments

We experimented with several different combinations of preprocessing techniques and model architectures.

Preprocessing. We tried incorporating different combinations of data augmentation techniques, including grayscale conversion, padding, random translation, mirroring, and resizing. In our final version, we included the grayscale, padding, and resizing. We tried training our model with and without mirroring images, but saw no improvement in model performance with mirroring. Since mirroring randomly between epochs requires re-processing the training data every time, which we did not have the memory to do, we decided to eliminate mirroring in order to train more quickly and to stay within memory limits.

We also originally tried padding all images on all four sides up to the maximum image size in our dataset of 1024x1024, then randomly translating the image within that padding in both x and y directions. However, we realized that this would make smaller images extra small aften we downsized to 224x224, and most pixel information would be lost after resizing. Thus, we changed this function to only pad each image (equally on either side) up until it was square (e.g. a 500x300 image would become 500x500) before downsizing to 224x224.

Model Architecture. We experimented with two main model architectures: one constructed from scratch (inspired by the Medium article "Creating the Snapchat Filter System using Deep Learning"), and one primarily making use of pre-trained MobileNetV2 weights (inspired by the Kaggle notebook "Cat Hipsterizer"). The latter model is described in the Methodology section. Using the first model architecture, we achieved fairly high training/validation accuracies (80-90%), but got a low test accuracy (see Table 1). The model summary and diagram of our original model



Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 112, 112, 64)	640
max_pooling2d_13 (MaxPooling	(None, 56, 56, 64)	0
conv2d_14 (Conv2D)	(None, 28, 28, 128)	73856
max_pooling2d_14 (MaxPooling	(None, 14, 14, 128)	0
conv2d_15 (Conv2D)	(None, 7, 7, 128)	147584
max_pooling2d_15 (MaxPooling	(None, 4, 4, 128)	0
conv2d_16 (Conv2D)	(None, 2, 2, 64)	8256
max_pooling2d_16 (MaxPooling	(None, 1, 1, 64)	0
flatten_4 (Flatten)	(None, 64)	0
dense_13 (Dense)	(None, 128)	8320
dropout_10 (Dropout)	(None, 128)	0
dense_14 (Dense)	(None, 256)	33024
dropout_11 (Dropout)	(None, 256)	0
dense_15 (Dense)	(None, 128)	32896
dropout_12 (Dropout)	(None, 128)	0
dense_16 (Dense)	(None, 18)	2322

Total params: 306,898
Trainable params: 306,898
Non-trainable params: 0

Figure 2. Original model architecture.

are shown in Figure 2.

As seen in Table 1, after switching to using a network based on the MobileNetV2 model, we were able to achieve much more accurate results, even using a smaller threshold of pixel distance for measuring accuracy. See Figure 1 for the final model architecture.

5. Results

Using an 80%-20% train-test split, the test accuracy of our model with different architectures is shown in Table 1. Fine-tuning MobileNetV2 resulted in the highest test accuracy of the two model architectures (i.e. 76% accuracy with a distance threshold of 6 pixels), so we used this model to predict keypoint locations for filter placement.

A visualization of the actual keypoint locations for sample test images (black), along with the keypoint locations predicted by our model (red), are shown in Figure 3.

Sample images of cats with various filters overlaid are included in Figure 4.

We were also able to apply filters over a live video of cat in real-time. Because the speed with which our model predicts does not match the speed of video capture, there is a noticeable lag in the displayed video, although the results are still reasonable (refer to Kitkat_live_final.mp4 in our Github repository).

Model	Threshold	Test Accuracy
Original	6	0.21
Original	16	0.48
MobileNetV2 + Dense	6	0.76
MobileNetV2 + Dense	16	1.0

Table 1. Test Accuracy for our original model vs. fine-tuning MobileNetV2.

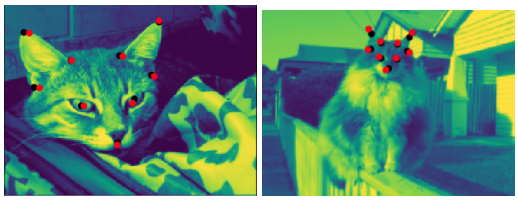


Figure 3. Predicted keypoints overlaid on ground truth keypoints.

Figure 4. Examples of cats with filters overlaid using predicted keypoints.

6. Comparison to Existing Techniques

We compared our result images to pictures taken using Snapchat's actual filters on a cat (courtesy of Soryan's cat Kitkat). Our results are comparable in quality and accuracy, as can be seen in Figure 5.

Figure 5. **Left:** Snapchat. **Right:** Ours.

7. Conclusion

Our greatest challenges during this project involved dealing with large amounts of data in memory and the compute power involved with training models on that data. Additionally, it took a considerable number of hours experimenting with different model architectures to find a successful one, and we saw a huge improvement by using a pre-trained, state-of-the-art model rather than training our own from scratch. We found our results to be more robust than expected, and were generally pleased with the way our filters turned out.

8. Contributions

Much of the project was done as a group effort over Zoom, through partner programming. We wrote our preprocessing functions/files and our original model architecture (preprocess.py, model.py, main.py) together, and ran our preprocessing to produce .npy files storing intermediate images and coordinates over Google Cloud Platform.

Maggie Wu (mwu27). Experimented with and trained different model architectures, including final MobileNetV2 version with dense layers; wrote pipeline for live video capture and filtering (video.py); created Ash hat and toast filters; adapted Soryan's James filter code for toast filter placement (add_toast.py).

Sophie Yang (syang55). Found Kaggle API to efficiently download the dataset onto GCP; experimented with preprocessing (e.g. normalizing the data); trained different model architectures (original model, fine-tuned VGG, and final weights for the MobileNetV2 model that Maggie created); wrote script to place bow filter over cat images (add_bow.py).

Soryan Kumar (skumar12). Proposed project idea and found Kaggle dataset of 9,000 labeled cat images. Created James filter and modified Ash hat in photoshop. Also wrote functions for adding Ash hat and James filters onto the cats in add_filter.py. Also modified video.py for capturing live video and saved output video after live demonstration with Kitkat.

Tiffany Ding (tding5). Experimented with training our original model with additional dropout layers, wrote most of apply_filter.py (with debugging assistance from Sophie and Maggie), adapted Sophie's code for bow filters to work for the dog filter (add_dog_filter.py).

References

[1] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[2] Frerk Saxen, Philipp Werner, Sebastian Handrich, Ehsan Othman, Laslo Dinges, and Ayoub Al-Hamadi. Face attribute detection with mobilenetv2 and nasnet-mobile. In *2019 11th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pages 176–180. IEEE, 2019.

[3] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3476–3483, 2013.