Starter Project Writeup

Project: genClassBezier2D

By: ***Jacob Frausto***

Project Idea

Recognizing shapes is an essential task in computer vision. An amalgamation of different application areas rely heavily on shape recognition, including robotics, healthcare, security systems, and assistance for the impaired.

My goal with this project is to first provide a procedure for generating a few datasets of abstract 2D shapes (formed using Bezier curves). I then prepare the architecture for a simple convolutional neural network (CNN) to classify said shapes based on complexity.

Data

The data for this project is entirely generated by the procedure provided. This procedure forms three classes (high, medium, and low complexity shape classifications). Of these shapes, there is an 80/10/10 split between training, validation, and test sets used by the model.

The complexity of a shape correlates directly with the number of Bezier curves used to form it, as well as an increasing number of sporadically-placed intermediate sampling points on each curve. A more in-depth look at the random shape generation used here can be found in subsection *3.1* of the paper titled, "A supervised neural network for drag prediction of arbitrary 2D shapes in laminar flows at low Reynolds number" [3].

In total, 45,000 2D shapes were generated for three datasets (15,000 images in each), taking a total of approximately 13 hours to render.

Figure 1: From left to right: generated shapes with low, medium, and high complexities respectively.

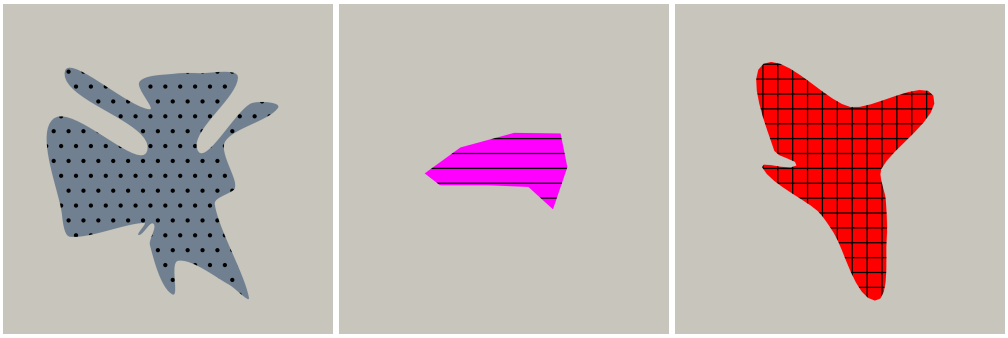

Figure 2: Examples of shapes generated for 'colors' dataset. The shapes in the 'textures' dataset are the same but with one consistent color.

Hardware Used

Training took approximately 46.5 hours total to complete running on a virtual machine with a singular NVIDIA Tesla K80 GPU.

Related Work

Typically, the description of a shape in 2D shape recognition is usually divided into three parts: its point set, contour, and skeleton [5]. Recent improvements in 2D shape classification rely on these conventional methods while introducing deep learning approaches. Prior to this, the bag of words method has been the dominant method for summarising many local features to discriminate objects and scenes. In [1] many of these techniques and algorithms are discussed in-depth, and their performance is reviewed.

In this project, I focus more on feature extraction and edge detection. One way to do so is by using a Canny edge detector. This operator uses a multi-stage algorithm to detect a

wide range of edges in images. It is a reasonably practical form of edge detection [2], and can be used for a multitude of tasks.

However, in [4] we are provided with an alternative and new edge detection algorithm that tackles two critical issues in this long-standing vision problem: (1) holistic image training and prediction; and (2) multi-scale and multi-level feature learning. The proposed method, holistically-nested edge detection (HED), performs "image-to-image prediction by means of a deep learning model that leverages fully convolutional neural networks and deeply-supervised nets." HED also "automatically learns rich hierarchical representations that are important in order to approach the human ability resolve the challenging ambiguity in edge and object boundary detection."

Method

In short, the method for this project included performing the following in order:

1. Generate shapes
2. For each of 6 experiments:
   (a) Image preprocessing
   (b) Training model
   (c) Testing model
3. Record results

The experiments I ran consisted of training and testing on three different datasets with two different preprocessing techniques (enhanced Canny edge detection and HED). I have chosen these because I want to see the effect that colors, textures, and utilization of HED have on the accuracy of 2D shape classification.

Each of the three datasets is different. The first 'shapes' dataset includes all shapes having the same color and texture. The second 'textures' dataset includes all shapes having the same color but random textures. The last 'colors' dataset includes all shapes having random color and texture.

In general, I expect the model to perform best on the 'shapes' dataset because of its simplicity. I also expect the model to perform better on the experiments that use the HED preprocessing technique.

Model

The model proposed was built using the Sequential model in Keras with four convolutional layers in its stack. A convolution/pooling pattern is repeated several times, with various filters for each layer (ranging from 32 to 256). It has a total of 1,143,043 trainable parameters. It uses a categorical cross-entropy loss function and an Adam optimizer with a learning rate of $1e-4$. The model's architecture is illustrated in Figure 4.

Image Preprocessing

I applied two different image preprocessing techniques for each of the three datasets.

For the first technique, we convert each image to grayscale, flip it horizontally and vertically, apply a contrasting color jitter, add Gaussian blur and Gaussian noise, then apply Canny edge detection. Finally, we dilate the edges procured for better detection.

In the second technique, we apply holistically-nested edge detection as described in [4].

Both of these techniques are illustrated in 3

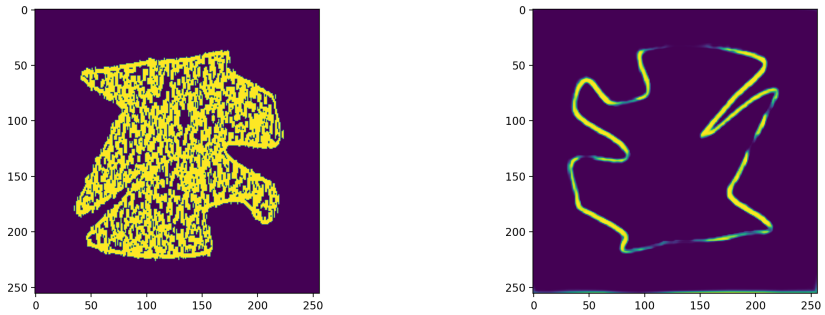

Figure 3: Left: An generated shape with the Canny edge detection preprocessing technique applied. Right: The same shape with the HED preprocessing technique applied.

Training

We train the model for 50 epochs using Keras *fit* function. During this, we provide a held back validation set of data in order to evaluate how well the model makes predictions based on new data.

Testing

We test the model using Keras *evaluate* function.

Overfitting

When first evaluating the model, I originally planned to use a mere 15,000 images (5000 for each of the three datasets). Through this, I discovered a vicious amount of overfitting to the point where accuracy was not improving on held-back validation sets. It became clear that adjusting hyperparameters or tweaking the image preprocessing techniques would not be enough. I decided to narrow down to two possible approaches to solving this issue:

- Generate more shapes
- Data augmentation

Thanks to the scope and goal of this project, I decided to go with the first option, thereby upping the overall number of generated shapes to 45,000. However, given a scenario in which I could not simply generate more data, I would likely utilize some data augmentation to provide the model with more training data.

Technical Challenges

The main technical challenge that I faced when working on this project came from utilizing the *HED* model to extract edge detection features. Using the pre-trained model turned out to be much more complicated than expected because of the OpenCV DNN module, which I found complicated to use.

I also ran into a pretty significant issue working with the *tf.data.Dataset* class and the way it utilizes tensors. It became challenging to reshape datasets to the input shape desired after applying either preprocessing technique. I eventually determined that I would have to deconstruct and reconstruct each dataset before and after preprocessing, respectively.

There was also the aforementioned overfitting issue, which was promptly taken care of.

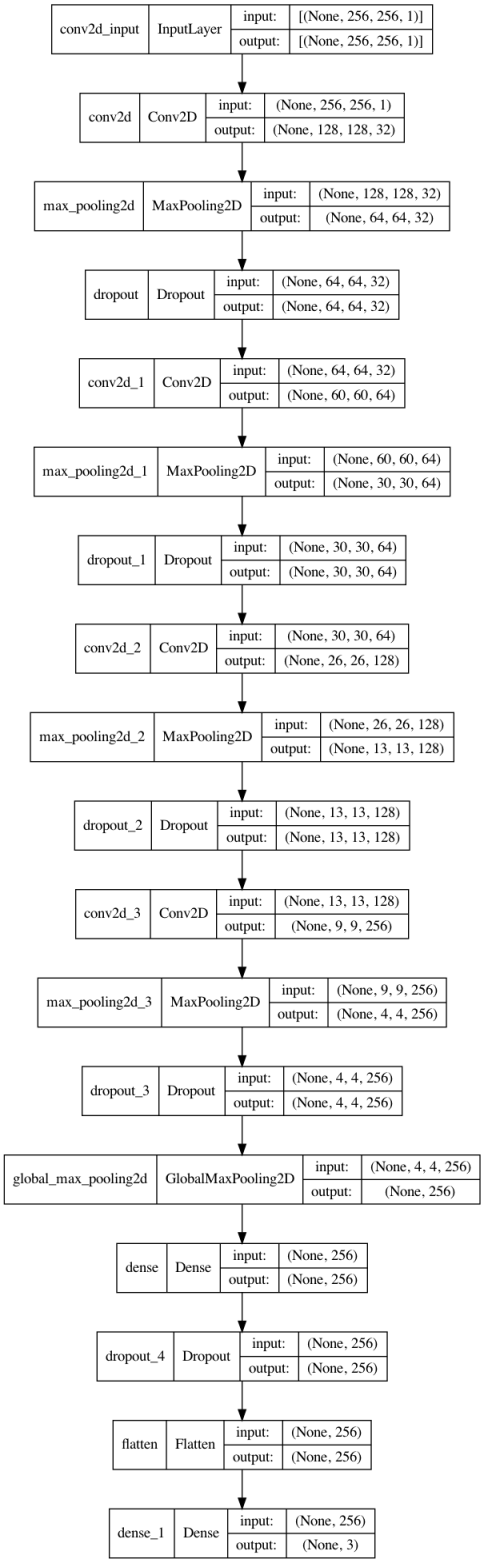

Figure 4: The BezierModel architecture.

Results

Dataset	Final Training Accuracy (%)	Final Training Loss	Final Testing Accuracy (%)	Final Testing Loss
Shapes w/o HED	99.07%	0.026681	96.27%	0.1019
Shapes w/ HED	99.49%	0.014500	97.53%	0.059090
Textures w/o HED	99.08%	0.023191	93.80%	0.181462
Textures w/ HED	**98.98%**	**0.027738**	96.47%	0.102275
Colors w/o HED	97.69%	0.063695	91.13%	0.227862
Colors w/ HED	97.73%	0.058948	**90.66%**	**0.255062**

Table 1: Model Accuracy/Loss for the different datasets and preprocessing techniques used.

Overall, the results of each experiment were satisfying and made sense. As we can see in Table 1, the final training and testing accuracy for each experiment (after 50 epochs of training) was above 90%. What's more, the final losses of each experiment were extremely low, indicating that little error was made in the training of our model.

One interesting thing to point out about Table 1 are the bolded numbers. These numbers are for the experiments that used HED during preprocessing that performed worse when compared to their counterpart experiments that did not use HED in preprocessing. The difference in performance here is minimal, however.

It seems that between our datasets, the experiments on 'shapes' and 'textures' performed similarly, whereas the 'colors' experiments came out with worse performance. This indicates that our model performs better when shapes are rendered the same color and that applying textures onto shapes themselves has a very nominal effect.

The categorical accuracy and loss performed well and behaved as expected, shown by Figure 5.

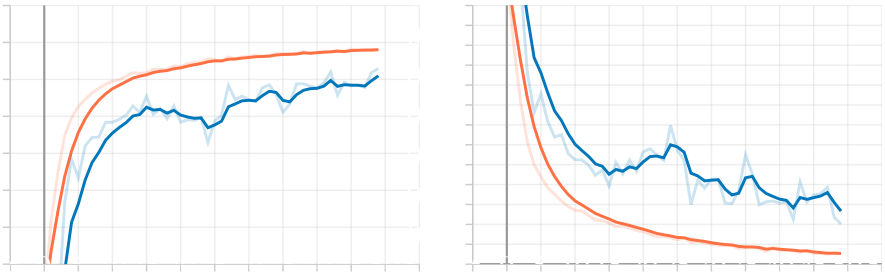

Figure 5: Left: Categorical accuracy by epoch of 'shapes' dataset during training. Right: Loss by epoch of 'shapes' dataset during training. The behavior for each was typical for all experiments ran.

Conclusion

All in all, I am quite satisfied with the outcome of this project and the experiments within. The results show the efficacy of the model I designed and the robust nature of the shape generation procedure.

After obtaining the results from all six experiments, I have concluded on two points:

- The randomization of textures and colors negatively impacts accuracy.
- The use of HED in image preprocessing improves accuracy, but only marginally compared to Canny edge detection.

Both of these conclusions were expected.

References

[1] Habibollah Agh Atabay. “Binary shape classification using Convolutional Neural Networks”. In: *IIOAB Journal* 7 (Oct. 2016), pp. 332–336.

[2] John Canny. “A Computational Approach to Edge Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[3] Jonathan Viquerat and Elie Hachem. “A supervised neural network for drag prediction of arbitrary 2D shapes in laminar flows at low Reynolds number”. In: *Computers Fluids* 210 (2020), p. 104645. ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2020.104645. URL: https://www.sciencedirect.com/science/article/pii/S0045793020302164.

[4] Saining Xie and Zhuowen Tu. “Holistically-nested edge detection”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1395–1403.

[5] Chaoyan Zhang et al. “SCN: A Novel Shape Classification Algorithm Based on Convolutional Neural Network”. In: *Symmetry* 13.3 (2021). ISSN: 2073-8994. DOI: 10.3390/sym13030499. URL: https://www.mdpi.com/2073-8994/13/3/499.