UDACITY

‹ Return to "Computer Vision Nanodegree" in the classroom

Facial Keypoint Detection

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

I really like the effort you put into this project! 🎉 I wish others would do similarly. 😅 Nicely done!

Files Submitted

The submission includes models.py and the following Jupyter notebooks, where all questions have been answered and training and visualization cells have been executed:
2. Define the Network Architecture.ipynb, and
3. Facial Keypoint Detection, Complete Pipeline.ipynb.
Other files may be included, but are not necessary for grading purposes. Note that all your files will be zipped and uploaded should you submit via the provided workspace.

`models.py`

Define a convolutional neural network with at least one convolutional layer, i.e. `self.conv1 = nn.Conv2d(1, 32, 5)`. The network should take in a grayscale, square image.

Notebook 2: Define the Network Architecture

Define a `data_transform` and apply it whenever you instantiate a DataLoader. The composed transform should include: rescaling/cropping, normalization, and turning input images into torch Tensors. The transform should turn any input image into a normalized, square, grayscale image and then a Tensor for your model to take it as input.

Select a loss function and optimizer for training the model. The loss and optimization functions should be appropriate for keypoint detection, which is a regression problem.

Train your CNN after defining its loss and optimization functions. You are encouraged, but not required, to visualize the loss over time/epochs by printing it out occasionally and/or plotting the loss over time. Save your best trained model.

After training, all 3 questions about model architecture, choice of loss function, and choice of batch_size and epoch parameters are answered.

1. Good discussion on your choice of `optimizer`. Also on your choice of `criterion`! 👍
2. Excellent description of your model development and training! 👍
3. Awesome discovery! 👍 😃 This is definitely a good regime to consider, as models learn the most, early on, but smaller `batch_size`s allow you to further tune to the nuances of the data, as larger `batch_size`s generally "average away" the nuance. 😅

Your CNN "learns" (updates the weights in its convolutional layers) to recognize features and this criteria requires that you extract at least one convolutional filter from your trained model, apply it to an image, and see what effect this filter has on an image.

Thanks for running the cells and output both filter and images. 😃

After visualizing a feature map, answer: what do you think it detects? This answer should be informed by how a filtered image (from the criteria above) looks.

I think your answer could be a bit more specific, but the further work you did in showcasing the results of the filters is also very helpful in discerning what's going on, under-the-hood – however this is generally an acceptable response and result. 😅

Notebook 3: Facial Keypoint Detection

Use a Haar cascade face detector to detect faces in a given image.

You should transform any face into a normalized, square, grayscale image and then a Tensor for your model to take in as input (similar to what the `data_transform` did in Notebook 2).

After face detection with a Haar cascade and face pre-processing, apply your trained model to each detected face, and display the predicted keypoints for each face in the image.

DOWNLOAD PROJECT

RETURN TO PATH