UDACITY

Return to "Computer Vision Nanodegree" in the classroom

Facial Keypoint Detection

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

I have provided some resources that you might find useful and may further your knowledge:
Detecting facial features using deep learning
Facial recognition using tensorflow
A comparison of different CNN models
Good job on completing the facial keypoint detection 👍 The project meets all specifications! I encourage you to work and experiment on the "Fun with keypoints.ipynb" notebook . Good luck 😀

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.

All the required files models.py, 2. Define the Network Architecture.ipynb, and 3. Facial Keypoint Detection, Complete Pipeline.ipynb have been submitted

`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.

Good job the architecture of CNN is well defined, the network includes a dropout layer to reduce overfitting and multiple (at least 2 convolution/pooling layers to detect complex features).

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.

The data transform is defined correctly, and is composed of rescaling/cropping, normalization, and turning input images into torch Tensors.

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.

The model has been trained correctly, and the best model is saved.
Suggestion - Its a good idea to visualize the losses over epochs

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

All 3 questions have been answered correctly and proper reasoning has been given for choosing model architecture, loss function, batch size, epochs etc

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.

Good job using OpenCV's filter 2D function

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.

Good job on the answer that is exactly what the filter does

Notebook 3: Facial Keypoint Detection

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

Haar cascade has been correctly used for detecting faces 👍

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).

All the following steps have been correctly done:

1. Convert the face from RGB to grayscale,
2. Normalize each grayscale image so that its color range falls in [0,1] instead of [0,255],
3. Rescale the detected face to be the expected square size for your CNN (224x224, suggested), and
4. Reshape the numpy image into a torch image (CxHxW).

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.

The predicted keypoints correctly overlap the image of the face.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review