



[< Back to Computer Vision Nanodegree](#)

# Facial Keypoint Detection

## REVIEW

### CODE REVIEW 2

## HISTORY

## Meets Specifications

You've done a **fantastic job** completing the Facial Keypoints Detection Project!

The Facial Keypoints Detection is a well known [machine learning challenge](#). You might want to check out the resources below to further improve your model:

- [Berkeley team's attempt at beating the Facial Keypoints Detection Kaggle competition](#)
- [Facial Keypoints Detection using the Inception model](#)

Keep up the great work 👍

## 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.



Well done submitting all of the required files and answering all of the questions in the notebook!

`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.

Well done defining a convolutional neural network along with the feedforward behavior!

## 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.



Great job defining the `data_transform` to turn an input image into a normalized, square, grayscale image in Tensor format!

You might want to consider augmenting the training data by randomly rotating and/or flipping the dataset. You can read the official documentation on [torchvision.transforms](#) to learn about the available transformations.

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.



Nicely done using the `SmoothL1Loss` loss function along with the `Adam` optimizer! Great job choosing the [Adam algorithm](#) as an optimizer! Here are some of the benefits of the Adam algorithm:

- Straightforward to implement
- Computationally efficient
- Little memory requirements
- Invariant to diagonal rescale of the gradients
- Well suited for problems that are large in terms of data and/or parameters
- Appropriate for non-stationary objectives
- Appropriate for problems with very noisy/or sparse gradients
- Hyper-parameters have intuitive interpretation and typically require little tuning

You've made sure to choose an appropriate loss function for the current regression problem. The chosen loss function does compare outputs value by value instead of looking at probability distribution, which is the case with classification problems.


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.

 Good job training the CNN and saving the trained model in the checkpoint!

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

 The questions have been answered and your reasoning is clear and sound!

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.

 The CNN does "learn" to recognize the features in the image and a convolutional filter has been extracted from the trained model for analysis

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.


Yep, the chosen filters 7, 8 and 13 do seem to be detecting edges

## Notebook 3: Facial Keypoint Detection

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

 Great job using the Haar cascade face detector for detecting frontal faces in the 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).

 Well done transforming the face image into a normalized, grayscale image and passing it into the model as a Tensor

TENSOR:

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 model has been applied and the predicted keypoints are being displayed on each face in the image

 [DOWNLOAD PROJECT](#)

2

[CODE REVIEW COMMENTS](#)[RETURN TO PATH](#)[Rate this review](#)