Artificial Intelligence Nanodegree

Computer Vision Capstone

Project: Facial Keypoint Detection

Welcome to the final Computer Vision project in the Artificial Intelligence Nanodegree program!

In this project, you'll combine your knowledge of computer vision techniques and deep learning to build and end-to-end facial keypoint recognition system! Facial keypoints include points around the eyes, nose, and mouth on any face and are used in many applications, from facial tracking to emotion recognition.

There are three main parts to this project:

Part 1 : Investigating OpenCV, pre-processing, and face detection

Part 2 : Training a Convolutional Neural Network (CNN) to detect facial keypoints

Part 3 : Putting parts 1 and 2 together to identify facial keypoints on any image!

**Here's what you need to know to complete the project:*

1. In this notebook, some template code has already been provided for you, and you will need to implement additional functionality to successfully complete this project. You will not need to modify the included code beyond what is requested.

   a. Sections that begin with **'(IMPLEMENTATION)'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section, and the specifics of the implementation are marked in the code block with a 'TODO' statement. Please be sure to read the instructions carefully!

1. In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation.

   a. Each section where you will answer a question is preceded by a **'Question X'** header.

   b. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. Markdown cells can be edited by double-clicking the cell to enter edit mode.

The rubric contains **optional** suggestions for enhancing the project beyond the minimum requirements. If you decide to pursue the "(Optional)" sections, you should include the code in this IPython notebook.

Your project submission will be evaluated based on your answers to *each* of the questions and the code implementations you provide.

Steps to Complete the Project

Each part of the notebook is further broken down into separate steps. Feel free to use the links below to navigate the notebook.

In this project you will get to explore a few of the many computer vision algorithms built into the OpenCV library. This expansive computer vision library is now almost 20 years old (https://en.wikipedia.org/wiki/OpenCV#History) and still growing!

The project itself is broken down into three large parts, then even further into separate steps. Make sure to read through each step, and complete any sections that begin with **'(IMPLEMENTATION)'** in the header; these implementation sections may contain multiple TODOs that will be marked in code. For convenience, we provide links to each of these steps below.

Part 1 : Investigating OpenCV, pre-processing, and face detection

- Step 0: Detect Faces Using a Haar Cascade Classifier
- Step 1: Add Eye Detection
- Step 2: De-noise an Image for Better Face Detection
- Step 3: Blur an Image and Perform Edge Detection
- Step 4: Automatically Hide the Identity of an Individual

Part 2 : Training a Convolutional Neural Network (CNN) to detect facial keypoints

- Step 5: Create a CNN to Recognize Facial Keypoints
- Step 6: Compile and Train the Model
- Step 7: Visualize the Loss and Answer Questions

Part 3 : Putting parts 1 and 2 together to identify facial keypoints on any image!

- Step 8: Build a Robust Facial Keypoints Detector (Complete the CV Pipeline)

Step 0: Detect Faces Using a Haar Cascade Classifier

Have you ever wondered how Facebook automatically tags images with your friends' faces? Or how high-end cameras automatically find and focus on a certain person's face? Applications like these depend heavily on the machine learning task known as *face detection* - which is the task of automatically finding faces in images containing people.

At its root face detection is a classification problem - that is a problem of distinguishing between distinct classes of things. With face detection these distinct classes are 1) images of human faces and 2) everything else.

We use OpenCV's implementation of Haar feature-based cascade classifiers (http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html) to detect human faces in images. OpenCV provides many pre-trained face detectors, stored as XML files on github (https://github.com/opencv/opencv/tree/master/data/haarcascades). We have downloaded one of these detectors and stored it in the `detector_architectures` directory.

Import Resources

In the next python cell, we load in the required libraries for this section of the project.

In [1]:

```
# Import required libraries for this section

%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import math
import cv2                               # OpenCV library for computer vision
from PIL import Image
import time
```

Next, we load in and display a test image for performing face detection.

Note: by default OpenCV assumes the ordering of our image's color channels are Blue, then Green, then Red. This is slightly out of order with most image types we'll use in these experiments, whose color channels are ordered Red, then Green, then Blue. In order to switch the Blue and Red channels of our test image around we will use OpenCV's `cvtColor` function, which you can read more about by checking out some of its documentation located here (http://docs.opencv.org/3.2.0/df/d9d/tutorial_py_colorspaces.html). This is a general utility function that can do other transformations too like converting a color image to grayscale, and transforming a standard color image to HSV color space.

In [2]:

```
# Load in color image for face detection
image = cv2.imread('images/test_image_1.jpg')

# Convert the image to RGB colorspace
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Plot our image using subplots to specify a size and title
fig = plt.figure(figsize = (8,8))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Original Image')
ax1.imshow(image)
```

Out[2]:

```
<matplotlib.image.AxesImage at 0x7f194c3abef0>
```

There are a lot of people - and faces - in this picture. 13 faces to be exact! In the next code cell, we demonstrate how to use a Haar Cascade classifier to detect all the faces in this test image.

This face detector uses information about patterns of intensity in an image to reliably detect faces under varying light conditions. So, to use this face detector, we'll first convert the image from color to grayscale.

Then, we load in the fully trained architecture of the face detector -- found in the file *haarcascade_frontalface_default.xml* - and use it on our image to find faces!

To learn more about the parameters of the detector see this post (https://stackoverflow.com/questions/20801015/recommended-values-for-opencv-detectmultiscale-parameters).

In [3]:

```
# Convert the RGB  image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# Extract the pre-trained face detector from an xml file
face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_front
alface_default.xml')

# Detect the faces in image
faces = face_cascade.detectMultiScale(gray, 4, 6)

# Print the number of faces detected in the image
print('Number of faces detected:', len(faces))

# Make a copy of the orginal image to draw face detections on
image_with_detections = np.copy(image)

# Get the bounding box for each detected face
for (x,y,w,h) in faces:
    # Add a red bounding box to the detections image
    cv2.rectangle(image_with_detections, (x,y), (x+w,y+h), (255,0,0), 3)


# Display the image with the detections
fig = plt.figure(figsize = (8,8))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Image with Face Detections')
ax1.imshow(image_with_detections)
```

```
Number of faces detected: 13

Out[3]:

<matplotlib.image.AxesImage at 0x7f194baee438>
```

In the above code, `faces` is a numpy array of detected faces, where each row corresponds to a detected face. Each detected face is a 1D array with four entries that specifies the bounding box of the detected face. The first two entries in the array (extracted in the above code as `x` and `y`) specify the horizontal and vertical positions of the top left corner of the bounding box. The last two entries in the array (extracted here as `w` and `h`) specify the width and height of the box.

Step 1: Add Eye Detections

There are other pre-trained detectors available that use a Haar Cascade Classifier - including full human body detectors, license plate detectors, and more. A full list of the pre-trained architectures can be found here (https://github.com/opencv/opencv/tree/master/data/haarcascades).

To test your eye detector, we'll first read in a new test image with just a single face.

In [4]:

```
# Load in color image for face detection
image = cv2.imread('images/james.jpg')

# Convert the image to RGB colorspace
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Plot the RGB image
fig = plt.figure(figsize = (6,6))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Original Image')
ax1.imshow(image)
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x7f194bac94a8>
```

Notice that even though the image is a black and white image, we have read it in as a color image and so it will still need to be converted to grayscale in order to perform the most accurate face detection.

So, the next steps will be to convert this image to grayscale, then load OpenCV's face detector and run it with parameters that detect this face accurately.

In [5]:

```
# Convert the RGB  image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# Extract the pre-trained face detector from an xml file
face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_front
alface_default.xml')

# Detect the faces in image
faces = face_cascade.detectMultiScale(gray, 1.25, 6)

# Print the number of faces detected in the image
print('Number of faces detected:', len(faces))

# Make a copy of the orginal image to draw face detections on
image_with_detections = np.copy(image)

# Get the bounding box for each detected face
for (x,y,w,h) in faces:
    # Add a red bounding box to the detections image
    cv2.rectangle(image_with_detections, (x,y), (x+w,y+h), (255,0,0), 3)


# Display the image with the detections
fig = plt.figure(figsize = (6,6))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Image with Face Detection')
ax1.imshow(image_with_detections)
```

```
Number of faces detected: 1

Out[5]:

<matplotlib.image.AxesImage at 0x7f194ba9ed30>
```

(IMPLEMENTATION) Add an eye detector to the current face detection setup.

A Haar-cascade eye detector can be included in the same way that the face detector was and, in this first task, it will be your job to do just this.

To set up an eye detector, use the stored parameters of the eye cascade detector, called `haarcascade_eye.xml`, located in the `detector_architectures` subdirectory. In the next code cell, create your eye detector and store its detections.

A few notes before you get started:

First, make sure to give your loaded eye detector the variable name

`eye_cascade`

and give the list of eye regions you detect the variable name

`eyes`

Second, since we've already run the face detector over this image, you should only search for eyes *within the rectangular face regions detected in* `faces`. This will minimize false detections.

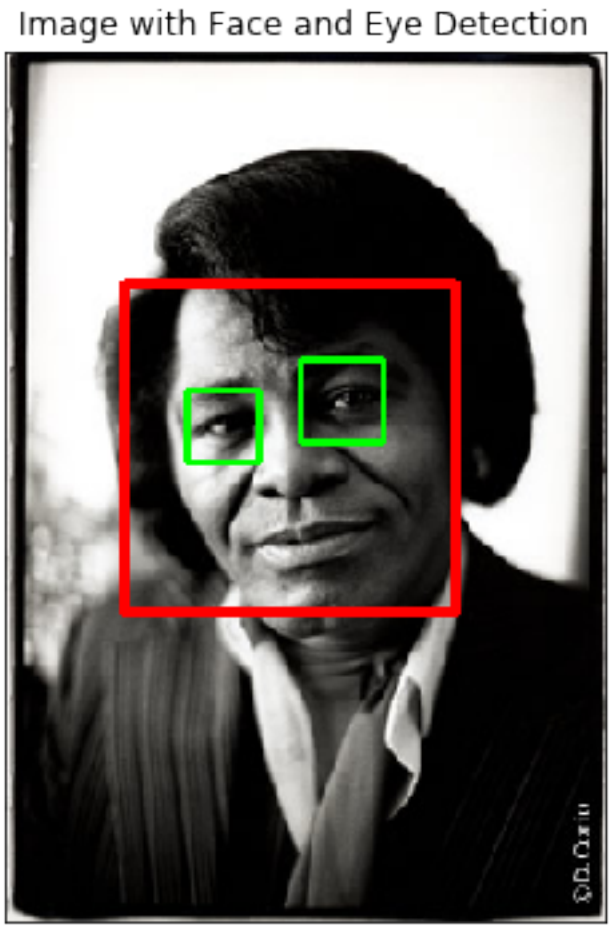
Lastly, once you've run your eye detector over the facial detection region, you should display the RGB image with both the face detection boxes (in red) and your eye detections (in green) to verify that everything works as expected.

In [6]:

```
# Make a copy of the original image to plot rectangle detections
image_with_detections = np.copy(image)

# Loop over the detections and draw their corresponding face detection boxes
for (x,y,w,h) in faces:
    cv2.rectangle(image_with_detections, (x,y), (x+w,y+h),(255,0,0), 3)

# Do not change the code above this comment!


## TODO: Add eye detection, using haarcascade_eye.xml, to the current face det
ector algorithm
# Extract the pre-trained eye detector from an xml file
eye_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_eye.xm
l')

# Detect the eyes in image
eyes = []
for (x,y,w,h) in faces:
    eyes_detected = eye_cascade.detectMultiScale(gray[y:y + h, x:x + w])
    for (ex, ey, ew, eh) in eyes_detected:
        eyes.append((x + ex, y + ey, ew, eh))

## TODO: Loop over the eye detections and draw their corresponding boxes in gr
een on image_with_detections
for (ex, ey, ew, eh) in eyes:
    cv2.rectangle(image_with_detections,(ex, ey),(ex + ew, ey + eh),(0,255,0),
2)

# Plot the image with both faces and eyes detected
fig = plt.figure(figsize = (6,6))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Image with Face and Eye Detection')
ax1.imshow(image_with_detections)
```

Out[6]:

<matplotlib.image.AxesImage at 0x7f194b9fee48>



(Optional) Add face and eye detection to your laptop camera

It's time to kick it up a notch, and add face and eye detection to your laptop's camera! Afterwards, you'll be able to show off your creation like in the gif shown below - made with a completed version of the code!

Notice that not all of the detections here are perfect - and your result need not be perfect either. You should spend a small amount of time tuning the parameters of your detectors to get reasonable results, but don't hold out for perfection. If we wanted perfection we'd need to spend a ton of time tuning the parameters of each detector, cleaning up the input image frames, etc. You can think of this as more of a rapid prototype.

The next cell contains code for a wrapper function called `laptop_camera_face_eye_detector` that, when called, will activate your laptop's camera. You will place the relevant face and eye detection code in this wrapper function to implement face/eye detection and mark those detections on each image frame that your camera captures.

Before adding anything to the function, you can run it to get an idea of how it works - a small window should pop up showing you the live feed from your camera; you can press any key to close this window.

Note: Mac users may find that activating this function kills the kernel of their notebook every once in a while. If this happens to you, just restart your notebook's kernel, activate cell(s) containing any crucial import statements, and you'll be good to go!

In [7]:

```
### Add face and eye detection to this laptop camera function
# Make sure to draw out all faces/eyes found in each frame on the shown video
feed

import cv2
import time

def detect_features(frame):
    face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_f
rontalface_default.xml')
    eye_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_ey
e.xml')
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect the face
    faces = face_cascade.detectMultiScale(gray, 1.25, 6)
    # For each face ...
    for (x,y,w,h) in faces:
        # Draw a red rectangle around the detected face (frame uses BGR color
code)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
        face_gray = gray[y:y+h, x:x+w]
        face_color = frame[y:y+h, x:x+w]
        # Detect the eyes in the face
        eyes = eye_cascade.detectMultiScale(face_gray)
        # For each eye ...
        for (ex, ey, ew, eh) in eyes:
        # Draw a green rectangle around the detected eye
            cv2.rectangle(face_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0
), 2)
    return frame


# wrapper function for face/eye detection with your laptop camera
def laptop_camera_go():
    # Create instance of video capturer
    cv2.namedWindow("face detection activated")
    vc = cv2.VideoCapture(0)

    # Try to get the first frame
    if vc.isOpened():
        rval, frame = vc.read()
    else:
        rval = False

    # Keep the video stream open
    while rval:
        # Plot the image from camera with all the face and eye detections mark
ed
        frame = detect_features(frame)
        cv2.imshow("face detection activated", frame)

        # Exit functionality - press any key to exit laptop video
        key = cv2.waitKey(20)
        if key > 0 and key < 255: # Exit by pressing any key (Needed to add <
255 on my PC)
            # Destroy windows
```

```
            cv2.destroyAllWindows()

            # Make sure window closes on OSx
            for i in range (1,5):
                cv2.waitKey(1)
            return

        # Read next frame
        time.sleep(0.05)             # control framerate for computation - def
ault 20 frames per sec
        rval, frame = vc.read()
```

In [8]:

```
# Call the laptop camera face/eye detector function above
laptop_camera_go()
```

Step 2: De-noise an Image for Better Face Detection

Image quality is an important aspect of any computer vision task. Typically, when creating a set of images to train a deep learning network, significant care is taken to ensure that training images are free of visual noise or artifacts that hinder object detection. While computer vision algorithms - like a face detector - are typically trained on 'nice' data such as this, new test data doesn't always look so nice!

When applying a trained computer vision algorithm to a new piece of test data one often cleans it up first before feeding it in. This sort of cleaning - referred to as *pre-processing* - can include a number of cleaning phases like blurring, de-noising, color transformations, etc., and many of these tasks can be accomplished using OpenCV.

In this short subsection we explore OpenCV's noise-removal functionality to see how we can clean up a noisy image, which we then feed into our trained face detector.

Create a noisy image to work with

In the next cell, we create an artificial noisy version of the previous multi-face image. This is a little exaggerated - we don't typically get images that are this noisy - but image noise (https://digital-photography-school.com/how-to-avoid-and-reduce-noise-in-your-images/), or 'grainy-ness' in a digitial image - is a fairly common phenomenon.

In [9]:

```
# Load in the multi-face test image again
image = cv2.imread('images/test_image_1.jpg')

# Convert the image copy to RGB colorspace
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Make an array copy of this image
image_with_noise = np.asarray(image)

# Create noise - here we add noise sampled randomly from a Gaussian distributi
on: a common model for noise
noise_level = 40
noise = np.random.randn(image.shape[0],image.shape[1],image.shape[2])*noise_le
vel

# Add this noise to the array image copy
image_with_noise = image_with_noise + noise

# Convert back to uint8 format
image_with_noise = np.asarray([np.uint8(np.clip(i,0,255)) for i in image_with_
noise])

# Plot our noisy image!
fig = plt.figure(figsize = (8,8))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Noisy Image')
ax1.imshow(image_with_noise)
```

Out[9]:

<matplotlib.image.AxesImage at 0x7f194b9e5e48>

In the context of face detection, the problem with an image like this is that - due to noise - we may miss some faces or get false detections.

In the next cell we apply the same trained OpenCV detector with the same settings as before, to see what sort of detections we get.

In [10]:

```
# Convert the RGB  image to grayscale
gray_noise = cv2.cvtColor(image_with_noise, cv2.COLOR_RGB2GRAY)

# Extract the pre-trained face detector from an xml file
face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_front
alface_default.xml')

# Detect the faces in image
faces = face_cascade.detectMultiScale(gray_noise, 4, 6)

# Print the number of faces detected in the image
print('Number of faces detected:', len(faces))

# Make a copy of the orginal image to draw face detections on
image_with_detections = np.copy(image_with_noise)

# Get the bounding box for each detected face
for (x,y,w,h) in faces:
    # Add a red bounding box to the detections image
    cv2.rectangle(image_with_detections, (x,y), (x+w,y+h), (255,0,0), 3)


# Display the image with the detections
fig = plt.figure(figsize = (8,8))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Noisy Image with Face Detections')
ax1.imshow(image_with_detections)
```

```
Number of faces detected: 12

Out[10]:

<matplotlib.image.AxesImage at 0x7f194b93a4a8>
```

With this added noise we now miss one of the faces!

(IMPLEMENTATION) De-noise this image for better face detection

Time to get your hands dirty: using OpenCV's built in color image de-noising functionality called `fastNlMeansDenoisingColored` - de-noise this image enough so that all the faces in the image are properly detected. Once you have cleaned the image in the next cell, use the cell that follows to run our trained face detector over the cleaned image to check out its detections.

You can find its official documentation here ([documentation for denoising] (http://docs.opencv.org/trunk/d1/d79/group__photo__denoise.html#ga21abc1c8b0e15f78cd3eff672cb6c4
and a useful example here (http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_photo/py_non_local_means/py_non_local_means.html).

Note: you can keep all parameters *except* `photo_render` fixed as shown in the second link above. Play around with the value of this parameter - see how it affects the resulting cleaned image.

In [11]:

```
## TODO: Use OpenCV's built in color image de-noising function to clean up our
noisy image!

denoised_image = cv2.fastNlMeansDenoisingColored(image_with_noise, None, h=15,
hColor=15,
                                                 templateWindowSize=7, searchW
indowSize=21)

# Plot our denoised image
fig = plt.figure(figsize = (8,8))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Denoised Image')
ax1.imshow(denoised_image)
```

Out[11]:

```
<matplotlib.image.AxesImage at 0x7f194b90fb38>
```

In [12]:

```
## TODO: Run the face detector on the de-noised image to improve your detectio
ns and display the result
# Convert the RGB  image to grayscale
gray_noise = cv2.cvtColor(denoised_image, cv2.COLOR_RGB2GRAY)

# Extract the pre-trained face detector from an xml file
face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_front
alface_default.xml')

# Detect the faces in image
faces = face_cascade.detectMultiScale(gray_noise, 4, 6)

# Print the number of faces detected in the image
print('Number of faces detected:', len(faces))

# Make a copy of the orginal image to draw face detections on
image_with_detections = np.copy(denoised_image)

# Get the bounding box for each detected face
for (x,y,w,h) in faces:
    # Add a red bounding box to the detections image
    cv2.rectangle(image_with_detections, (x,y), (x+w,y+h), (255,0,0), 3)


# Display the image with the detections
fig = plt.figure(figsize = (8,8))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Noisy Image with Face Detections')
ax1.imshow(image_with_detections)
```

```
Number of faces detected: 13

Out[12]:

<matplotlib.image.AxesImage at 0x7f194b86b470>
```

Step 3: Blur an Image and Perform Edge Detection

Now that we have developed a simple pipeline for detecting faces using OpenCV - let's start playing around with a few fun things we can do with all those detected faces!

Importance of Blur in Edge Detection

Edge detection is a concept that pops up almost everywhere in computer vision applications, as edge-based features (as well as features built on top of edges) are often some of the best features for e.g., object detection and recognition problems.

Edge detection is a dimension reduction technique - by keeping only the edges of an image we get to throw away a lot of non-discriminating information. And typically the most useful kind of edge-detection is one that preserves only the important, global structures (ignoring local structures that aren't very discriminative). So removing local structures / retaining global structures is a crucial pre-processing step to performing edge detection in an image, and blurring can do just that.

Below is an animated gif showing the result of an edge-detected cat taken from Wikipedia (https://en.wikipedia.org/wiki/Gaussian_blur#Common_uses), where the image is gradually blurred more and more prior to edge detection. When the animation begins you can't quite make out what it's a picture of, but as the animation evolves and local structures are removed via blurring the cat becomes visible in the edge-detected image.

Edge detection is a **convolution** performed on the image itself, and you can read about Canny edge detection on this OpenCV documentation page (http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html).

Canny edge detection

In the cell below we load in a test image, then apply *Canny edge detection* on it. The original image is shown on the left panel of the figure, while the edge-detected version of the image is shown on the right. Notice how the result looks very busy - there are too many little details preserved in the image before it is sent to the edge detector. When applied in computer vision applications, edge detection should preserve *global* structure; doing away with local structures that don't help describe what objects are in the image.

In [13]:

```
# Load in the image
image = cv2.imread('images/fawzia.jpg')

# Convert to RGB colorspace
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# Perform Canny edge detection
edges = cv2.Canny(gray,100,200)

# Dilate the image to amplify edges
edges = cv2.dilate(edges, None)

# Plot the RGB and edge-detected image
fig = plt.figure(figsize = (15,15))
ax1 = fig.add_subplot(121)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Original Image')
ax1.imshow(image)

ax2 = fig.add_subplot(122)
ax2.set_xticks([])
ax2.set_yticks([])

ax2.set_title('Canny Edges')
ax2.imshow(edges, cmap='gray')
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x7f1941725f60>
```

Without first blurring the image, and removing small, local structures, a lot of irrelevant edge content gets picked up and amplified by the detector (as shown in the right panel above).

(IMPLEMENTATION) Blur the image *then* perform edge detection

In the next cell, you will repeat this experiment - blurring the image first to remove these local structures, so that only the important boudnary details remain in the edge-detected image.

Blur the image by using OpenCV's `filter2d` functionality - which is discussed in this documentation page (http://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html) - and use an *averaging kernel* of width equal to 4.

In [14]:

```
### TODO: Blur the test image using OpenCV's filter2d functionality,
# Use an averaging kernel, and a kernel width equal to 4

# Create the averaging kernel
kernel_size = 4
kernel = np.ones((kernel_size, kernel_size), dtype=np.float32) / (kernel_size
* kernel_size)

# Blur the image using our kernel
gray_blurred = cv2.filter2D(gray, -1, kernel)

## TODO: Then perform Canny edge detection and display the output
# Perform Canny edge detection
edges = cv2.Canny(gray_blurred,100,200)

# Dilate the image to amplify edges
edges = cv2.dilate(edges, None)

# Plot the RGB and edge-detected image
fig = plt.figure(figsize = (15,15))
ax1 = fig.add_subplot(121)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Original Image')
ax1.imshow(image)

ax2 = fig.add_subplot(122)
ax2.set_xticks([])
ax2.set_yticks([])

ax2.set_title('Canny Edges')
ax2.imshow(edges, cmap='gray')
```

```
Out[14]:

<matplotlib.image.AxesImage at 0x7f1941655080>
```

Step 4: Automatically Hide the Identity of an Individual

If you film something like a documentary or reality TV, you must get permission from every individual shown on film before you can show their face, otherwise you need to blur it out - by blurring the face a lot (so much so that even the global structures are obscured)! This is also true for projects like Google's StreetView maps (https://www.google.com/streetview/) - an enormous collection of mapping images taken from a fleet of Google vehicles. Because it would be impossible for Google to get the permission of every single person accidentally captured in one of these images they blur out everyone's faces, the detected images must automatically blur the identity of detected people. Here's a few examples of folks caught in the camera of a Google street view vehicle.

Read in an image to perform identity detection

Let's try this out for ourselves. Use the face detection pipeline built above and what you know about using the `filter2D` to blur and image, and use these in tandem to hide the identity of the person in the following image - loaded in and printed in the next cell.

In [15]:

```
# Load in the image
image = cv2.imread('images/gus.jpg')

# Convert the image to RGB colorspace
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the image
fig = plt.figure(figsize = (6,6))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])
ax1.set_title('Original Image')
ax1.imshow(image)
```

Out[15]:

```
<matplotlib.image.AxesImage at 0x7f194053ea90>
```

(IMPLEMENTATION) Use blurring to hide the identity of an individual in an image

The idea here is to 1) automatically detect the face in this image, and then 2) blur it out! Make sure to adjust the parameters of the *averaging* blur filter to completely obscure this person's identity.

In [16]:

```
## TODO: Implement face detection
# Convert the RGB  image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# Extract the pre-trained face detector from an xml file
face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_front
alface_default.xml')

# Detect the faces in image
faces = face_cascade.detectMultiScale(gray, 1.4, 6)

# Print the number of faces detected in the image
print('Number of faces detected:', len(faces))

# Make a copy of the orginal image to draw face detections on
censored_image = np.copy(image)

# Create the averaging kernel
kernel_size = 75
kernel = np.ones((kernel_size, kernel_size), dtype=np.float32) / (kernel_size
* kernel_size)

## TODO: Blur the bounding box around each detected face using an averaging fi
lter and display the result
for (x,y,w,h) in faces:
    censored_image[y:y+h, x:x+w] = cv2.blur(censored_image[y:y+h, x:x+w], (80,
80))
#     censored_image = cv2.blur(censored_image, (16, 16))
#     censored_image = cv2.filter2D(censored_image, -1, kernel)

# Plot the RGB and edge-detected image
fig = plt.figure(figsize = (15,15))
ax1 = fig.add_subplot(121)
ax1.set_xticks([])
ax1.set_yticks([])

ax1.set_title('Original Image')
ax1.imshow(image)

ax2 = fig.add_subplot(122)
ax2.set_xticks([])
ax2.set_yticks([])

ax2.set_title('censored Image')
ax2.imshow(censored_image)
```

```
Number of faces detected: 1

Out[16]:

<matplotlib.image.AxesImage at 0x7f19404e8390>
```

(Optional) Build identity protection into your laptop camera

In this optional task you can add identity protection to your laptop camera, using the previously completed code where you added face detection to your laptop camera - and the task above. You should be able to get reasonable results with little parameter tuning - like the one shown in the gif below.

As with the previous video task, to make this perfect would require significant effort - so don't strive for perfection here, strive for reasonable quality.

The next cell contains code a wrapper function called `laptop_camera_identity_hider` that - when called - will activate your laptop's camera. You need to place the relevant face detection and blurring code developed above in this function in order to blur faces entering your laptop camera's field of view.

Before adding anything to the function you can call it to get a hang of how it works - a small window will pop up showing you the live feed from your camera, you can press any key to close this window.

Note: Mac users may find that activating this function kills the kernel of their notebook every once in a while. If this happens to you, just restart your notebook's kernel, activate cell(s) containing any crucial import statements, and you'll be good to go!

In [17]:

```
### Insert face detection and blurring code into the wrapper below to create a
n identity protector on your laptop!
import cv2
import time

def hide_identity(frame):
    face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_f
rontalface_default.xml')
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, 1.25, 6)
    for (x,y,w,h) in faces:
        frame[y:y+h, x:x+w] = cv2.blur(frame[y:y+h, x:x+w], (30, 30))

    return frame


def laptop_camera_go():
    # Create instance of video capturer
    cv2.namedWindow("face detection activated")
    vc = cv2.VideoCapture(0)

    # Try to get the first frame
    if vc.isOpened():
        rval, frame = vc.read()
    else:
        rval = False

    # Keep video stream open
    while rval:
        frame = hide_identity(frame)
        # Plot image from camera with detections marked
        cv2.imshow("face detection activated", frame)

        # Exit functionality - press any key to exit laptop video
        key = cv2.waitKey(20)
        if key > 0 and key < 255: # Exit by pressing any key
            # Destroy windows
            cv2.destroyAllWindows()

            for i in range (1,5):
                cv2.waitKey(1)
            return

        # Read next frame
        time.sleep(0.05)                # control framerate for computation - def
ault 20 frames per sec
        rval, frame = vc.read()
```

In [18]:

```
# Run laptop identity hider
laptop_camera_go()
```

Step 5: Create a CNN to Recognize Facial Keypoints

OpenCV is often used in practice with other machine learning and deep learning libraries to produce interesting results. In this stage of the project you will create your own end-to-end pipeline - employing convolutional networks in keras along with OpenCV - to apply a "selfie" filter to streaming video and images.

You will start by creating and then training a convolutional network that can detect facial keypoints in a small dataset of cropped images of human faces. We then guide you towards OpenCV to expanding your detection algorithm to more general images. What are facial keypoints? Let's take a look at some examples.

Facial keypoints (also called facial landmarks) are the small blue-green dots shown on each of the faces in the image above - there are 15 keypoints marked in each image. They mark important areas of the face - the eyes, corners of the mouth, the nose, etc. Facial keypoints can be used in a variety of machine learning applications from face and emotion recognition to commercial applications like the image filters popularized by Snapchat.

Below we illustrate a filter that, using the results of this section, automatically places sunglasses on people in images (using the facial keypoints to place the glasses correctly on each face). Here, the facial keypoints have been colored lime green for visualization purposes.

Make a facial keypoint detector

But first things first: how can we make a facial keypoint detector? Well, at a high level, notice that facial keypoint detection is a *regression problem*. A single face corresponds to a set of 15 facial keypoints (a set of 15 corresponding (x, y) coordinates, i.e., an output point). Because our input data are images, we can employ a *convolutional neural network* to recognize patterns in our images and learn how to identify these keypoint given sets of labeled data.

In order to train a regressor, we need a training set - a set of facial image / facial keypoint pairs to train on. For this we will be using this dataset from Kaggle (https://www.kaggle.com/c/facial-keypoints-detection/data). We've already downloaded this data and placed it in the `data` directory. Make sure that you have both the *training* and *test* data files. The training dataset contains several thousand 96×96 grayscale images of cropped human faces, along with each face's 15 corresponding facial keypoints (also called landmarks) that have been placed by hand, and recorded in (x, y) coordinates. This wonderful resource also has a substantial testing set, which we will use in tinkering with our convolutional network.

To load in this data, run the Python cell below - notice we will load in both the training and testing sets.

The `load_data` function is in the included `utils.py` file.

In [19]:

```
from utils import *

# Load training set
X_train, y_train = load_data()
print("X_train.shape == {}".format(X_train.shape))
print("y_train.shape == {}; y_train.min == {:.3f}; y_train.max == {:.3f}".form
at(
    y_train.shape, y_train.min(), y_train.max()))

# Load testing set
X_test, _ = load_data(test=True)
print("X_test.shape == {}".format(X_test.shape))
```

```
Using TensorFlow backend.

X_train.shape == (2140, 96, 96, 1)
y_train.shape == (2140, 30); y_train.min == -0.920; y_train.max ==
0.996
X_test.shape == (1783, 96, 96, 1)
```

The `load_data` function in `utils.py` originates from this excellent blog post (http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/), which you are *strongly* encouraged to read. Please take the time now to review this function. Note how the output values - that is, the coordinates of each set of facial landmarks - have been normalized to take on values in the range $[-1, 1]$, while the pixel values of each input point (a facial image) have been normalized to the range $[0, 1]$.

Note: the original Kaggle dataset contains some images with several missing keypoints. For simplicity, the `load_data` function removes those images with missing labels from the dataset. As an ***optional*** extension, you are welcome to amend the `load_data` function to include the incomplete data points.

Visualize the Training Data

Execute the code cell below to visualize a subset of the training data.

In [20]:

```
import matplotlib.pyplot as plt
%matplotlib inline

fig = plt.figure(figsize=(20,20))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05
)
for i in range(9):
    ax = fig.add_subplot(3, 3, i + 1, xticks=[], yticks=[])
    plot_data(X_train[i], y_train[i], ax)
```

For each training image, there are two landmarks per eyebrow (**four** total), three per eye (**six** total), **four** for the mouth, and **one** for the tip of the nose.

Review the `plot_data` function in `utils.py` to understand how the 30-dimensional training labels in `y_train` are mapped to facial locations, as this function will prove useful for your pipeline.

(IMPLEMENTATION) Specify the CNN Architecture

In this section, you will specify a neural network for predicting the locations of facial keypoints. Use the code cell below to specify the architecture of your neural network. We have imported some layers that you may find useful for this task, but if you need to use more Keras layers, feel free to import them in the cell.

Your network should accept a 96×96 grayscale image as input, and it should output a vector with 30 entries, corresponding to the predicted (horizontal and vertical) locations of 15 facial keypoints. If you are not sure where to start, you can find some useful starting architectures in this blog (http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/), but you are not permitted to copy any of the architectures that you find online.

In [21]:

```
# Import deep learning resources from Keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dropout, Flatten, Dense
from keras.layers.advanced_activations import ELU

## TODO: Specify a CNN architecture
# Your model should accept 96x96 pixel graysale images in
# It should have a fully-connected output layer with 30 values (2 for each fac
ial keypoint)

model = Sequential([
    Conv2D(filters=32, kernel_size=2, strides=2, padding='same', input_shape=X
_train.shape[1:]),
    Dropout(0.45),
    ELU(),
    MaxPool2D(pool_size=2),
    Flatten(),
    Dense(500),
    ELU(),
    Dense(125),
    ELU(),
    Dense(30),
])

# Summarize the model
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 32)        160
_________________________________________________________________
dropout_1 (Dropout)          (None, 48, 48, 32)        0
_________________________________________________________________
elu_1 (ELU)                  (None, 48, 48, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 18432)             0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               9216500
_________________________________________________________________
elu_2 (ELU)                  (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 125)               62625
_________________________________________________________________
elu_3 (ELU)                  (None, 125)               0
_________________________________________________________________
dense_3 (Dense)              (None, 30)                3780
=================================================================
Total params: 9,283,065
Trainable params: 9,283,065
Non-trainable params: 0
_________________________________________________________________
```

Step 6: Compile and Train the Model

After specifying your architecture, you'll need to compile and train the model to detect facial keypoints'

(IMPLEMENTATION) Compile and Train the Model

Use the `compile` method (https://keras.io/models/sequential/#sequential-model-methods) to configure the learning process. Experiment with your choice of optimizer (https://keras.io/optimizers/); you may have some ideas about which will work best (`SGD` vs. `RMSprop`, etc), but take the time to empirically verify your theories.

Use the `fit` method (https://keras.io/models/sequential/#sequential-model-methods) to train the model. Break off a validation set by setting `validation_split=0.2`. Save the returned `History` object in the `history` variable.

Your model is required to attain a validation loss (measured as mean squared error) of at least **XYZ**. When you have finished training, save your model (https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model) as an HDF5 file with file path `my_model.h5`.

In [22]:

```
from keras.optimizers import Nadam
from keras.losses import mean_absolute_error

optimizer = Nadam(lr=0.001, epsilon=1e-4)
max_epochs = 2000
batch_size = 128
validation_split = 0.2

## Compile the model
model.compile(loss=mean_absolute_error, optimizer=optimizer, metrics=['accurac
y'])

# Save the model as model.h5
from keras.callbacks import ModelCheckpoint, EarlyStopping
checkpointer = ModelCheckpoint(filepath='my_model.h5', verbose=1, save_weights
_only=False, save_best_only=True)
early_stopping = EarlyStopping(monitor='val_loss', min_delta=1e-6, patience=20
0, verbose=1, mode='auto')

# train the model
history = model.fit(X_train, y_train, validation_split=validation_split, batch
_size=batch_size,
            epochs = max_epochs, callbacks = [checkpointer, early_stopping], v
erbose = 0, shuffle = True)

## TODO: Save the model as model.h5
# Checkpointer above does this
```

```
Epoch 00000: val_loss improved from inf to 0.63529, saving model t
o my_model.h5
```

```
Epoch 00001: val_loss improved from 0.63529 to 0.42529, saving mod
el to my_model.h5
Epoch 00002: val_loss improved from 0.42529 to 0.30100, saving mod
el to my_model.h5
Epoch 00003: val_loss improved from 0.30100 to 0.24508, saving mod
el to my_model.h5
Epoch 00004: val_loss improved from 0.24508 to 0.20903, saving mod
el to my_model.h5
Epoch 00005: val_loss improved from 0.20903 to 0.14801, saving mod
el to my_model.h5
Epoch 00006: val_loss did not improve
Epoch 00007: val_loss did not improve
Epoch 00008: val_loss did not improve
Epoch 00009: val_loss did not improve
Epoch 00010: val_loss improved from 0.14801 to 0.11649, saving mod
el to my_model.h5
Epoch 00011: val_loss improved from 0.11649 to 0.11023, saving mod
el to my_model.h5
Epoch 00012: val_loss improved from 0.11023 to 0.09260, saving mod
el to my_model.h5
Epoch 00013: val_loss improved from 0.09260 to 0.09005, saving mod
el to my_model.h5
Epoch 00014: val_loss did not improve
Epoch 00015: val_loss did not improve
Epoch 00016: val_loss improved from 0.09005 to 0.08605, saving mod
el to my_model.h5
Epoch 00017: val_loss improved from 0.08605 to 0.07903, saving mod
el to my_model.h5
Epoch 00018: val_loss did not improve
Epoch 00019: val_loss did not improve
Epoch 00020: val_loss did not improve
Epoch 00021: val_loss did not improve
Epoch 00022: val_loss did not improve
Epoch 00023: val_loss did not improve
Epoch 00024: val_loss did not improve
Epoch 00025: val_loss did not improve
Epoch 00026: val_loss did not improve
Epoch 00027: val_loss did not improve
Epoch 00028: val_loss did not improve
Epoch 00029: val_loss did not improve
Epoch 00030: val_loss did not improve
Epoch 00031: val_loss did not improve
Epoch 00032: val_loss did not improve
Epoch 00033: val_loss did not improve
Epoch 00034: val_loss did not improve
Epoch 00035: val_loss did not improve
Epoch 00036: val_loss did not improve
Epoch 00037: val_loss did not improve
Epoch 00038: val_loss did not improve
Epoch 00039: val_loss did not improve
Epoch 00040: val_loss did not improve
Epoch 00041: val_loss did not improve
Epoch 00042: val_loss did not improve
Epoch 00043: val_loss did not improve
Epoch 00044: val_loss did not improve
Epoch 00045: val_loss did not improve
Epoch 00046: val_loss did not improve
```

```
Epoch 00047: val_loss did not improve
Epoch 00048: val_loss did not improve
Epoch 00049: val_loss did not improve
Epoch 00050: val_loss did not improve
Epoch 00051: val_loss did not improve
Epoch 00052: val_loss did not improve
Epoch 00053: val_loss did not improve
Epoch 00054: val_loss did not improve
Epoch 00055: val_loss did not improve
Epoch 00056: val_loss did not improve
Epoch 00057: val_loss did not improve
Epoch 00058: val_loss did not improve
Epoch 00059: val_loss did not improve
Epoch 00060: val_loss did not improve
Epoch 00061: val_loss did not improve
Epoch 00062: val_loss did not improve
Epoch 00063: val_loss did not improve
Epoch 00064: val_loss did not improve
Epoch 00065: val_loss did not improve
Epoch 00066: val_loss did not improve
Epoch 00067: val_loss did not improve
Epoch 00068: val_loss did not improve
Epoch 00069: val_loss did not improve
Epoch 00070: val_loss did not improve
Epoch 00071: val_loss did not improve
Epoch 00072: val_loss did not improve
Epoch 00073: val_loss did not improve
Epoch 00074: val_loss did not improve
Epoch 00075: val_loss did not improve
Epoch 00076: val_loss did not improve
Epoch 00077: val_loss did not improve
Epoch 00078: val_loss did not improve
Epoch 00079: val_loss did not improve
Epoch 00080: val_loss did not improve
Epoch 00081: val_loss did not improve
Epoch 00082: val_loss did not improve
Epoch 00083: val_loss did not improve
Epoch 00084: val_loss did not improve
Epoch 00085: val_loss did not improve
Epoch 00086: val_loss did not improve
Epoch 00087: val_loss did not improve
Epoch 00088: val_loss did not improve
Epoch 00089: val_loss did not improve
Epoch 00090: val_loss did not improve
Epoch 00091: val_loss did not improve
Epoch 00092: val_loss did not improve
Epoch 00093: val_loss did not improve
Epoch 00094: val_loss did not improve
Epoch 00095: val_loss did not improve
Epoch 00096: val_loss did not improve
Epoch 00097: val_loss did not improve
Epoch 00098: val_loss did not improve
Epoch 00099: val_loss did not improve
Epoch 00100: val_loss did not improve
Epoch 00101: val_loss did not improve
Epoch 00102: val_loss did not improve
Epoch 00103: val_loss did not improve
```

```
Epoch 00104: val_loss did not improve
Epoch 00105: val_loss did not improve
Epoch 00106: val_loss did not improve
Epoch 00107: val_loss did not improve
Epoch 00108: val_loss did not improve
Epoch 00109: val_loss did not improve
Epoch 00110: val_loss did not improve
Epoch 00111: val_loss did not improve
Epoch 00112: val_loss did not improve
Epoch 00113: val_loss did not improve
Epoch 00114: val_loss did not improve
Epoch 00115: val_loss did not improve
Epoch 00116: val_loss did not improve
Epoch 00117: val_loss did not improve
Epoch 00118: val_loss did not improve
Epoch 00119: val_loss did not improve
Epoch 00120: val_loss did not improve
Epoch 00121: val_loss did not improve
Epoch 00122: val_loss did not improve
Epoch 00123: val_loss did not improve
Epoch 00124: val_loss did not improve
Epoch 00125: val_loss did not improve
Epoch 00126: val_loss did not improve
Epoch 00127: val_loss did not improve
Epoch 00128: val_loss did not improve
Epoch 00129: val_loss did not improve
Epoch 00130: val_loss did not improve
Epoch 00131: val_loss did not improve
Epoch 00132: val_loss did not improve
Epoch 00133: val_loss did not improve
Epoch 00134: val_loss did not improve
Epoch 00135: val_loss did not improve
Epoch 00136: val_loss did not improve
Epoch 00137: val_loss did not improve
Epoch 00138: val_loss did not improve
Epoch 00139: val_loss did not improve
Epoch 00140: val_loss did not improve
Epoch 00141: val_loss did not improve
Epoch 00142: val_loss did not improve
Epoch 00143: val_loss did not improve
Epoch 00144: val_loss did not improve
Epoch 00145: val_loss did not improve
Epoch 00146: val_loss did not improve
Epoch 00147: val_loss did not improve
Epoch 00148: val_loss did not improve
Epoch 00149: val_loss did not improve
Epoch 00150: val_loss did not improve
Epoch 00151: val_loss did not improve
Epoch 00152: val_loss did not improve
Epoch 00153: val_loss did not improve
Epoch 00154: val_loss did not improve
Epoch 00155: val_loss did not improve
Epoch 00156: val_loss did not improve
Epoch 00157: val_loss did not improve
Epoch 00158: val_loss did not improve
Epoch 00159: val_loss did not improve
Epoch 00160: val_loss did not improve
```

```
Epoch 00161: val_loss did not improve
Epoch 00162: val_loss did not improve
Epoch 00163: val_loss did not improve
Epoch 00164: val_loss did not improve
Epoch 00165: val_loss did not improve
Epoch 00166: val_loss did not improve
Epoch 00167: val_loss did not improve
Epoch 00168: val_loss did not improve
Epoch 00169: val_loss did not improve
Epoch 00170: val_loss did not improve
Epoch 00171: val_loss did not improve
Epoch 00172: val_loss did not improve
Epoch 00173: val_loss did not improve
Epoch 00174: val_loss did not improve
Epoch 00175: val_loss did not improve
Epoch 00176: val_loss did not improve
Epoch 00177: val_loss did not improve
Epoch 00178: val_loss did not improve
Epoch 00179: val_loss did not improve
Epoch 00180: val_loss improved from 0.07903 to 0.07623, saving mod
el to my_model.h5
Epoch 00181: val_loss did not improve
Epoch 00182: val_loss did not improve
Epoch 00183: val_loss did not improve
Epoch 00184: val_loss improved from 0.07623 to 0.06985, saving mod
el to my_model.h5
Epoch 00185: val_loss did not improve
Epoch 00186: val_loss did not improve
Epoch 00187: val_loss did not improve
Epoch 00188: val_loss did not improve
Epoch 00189: val_loss did not improve
Epoch 00190: val_loss did not improve
Epoch 00191: val_loss did not improve
Epoch 00192: val_loss did not improve
Epoch 00193: val_loss did not improve
Epoch 00194: val_loss did not improve
Epoch 00195: val_loss did not improve
Epoch 00196: val_loss improved from 0.06985 to 0.06270, saving mod
el to my_model.h5
Epoch 00197: val_loss did not improve
Epoch 00198: val_loss did not improve
Epoch 00199: val_loss did not improve
Epoch 00200: val_loss did not improve
Epoch 00201: val_loss did not improve
Epoch 00202: val_loss did not improve
Epoch 00203: val_loss did not improve
Epoch 00204: val_loss did not improve
Epoch 00205: val_loss did not improve
Epoch 00206: val_loss did not improve
Epoch 00207: val_loss did not improve
Epoch 00208: val_loss did not improve
Epoch 00209: val_loss did not improve
Epoch 00210: val_loss did not improve
Epoch 00211: val_loss did not improve
Epoch 00212: val_loss did not improve
Epoch 00213: val_loss did not improve
Epoch 00214: val_loss did not improve
```

```
Epoch 00215: val_loss did not improve
Epoch 00216: val_loss did not improve
Epoch 00217: val_loss did not improve
Epoch 00218: val_loss did not improve
Epoch 00219: val_loss did not improve
Epoch 00220: val_loss did not improve
Epoch 00221: val_loss did not improve
Epoch 00222: val_loss did not improve
Epoch 00223: val_loss did not improve
Epoch 00224: val_loss did not improve
Epoch 00225: val_loss did not improve
Epoch 00226: val_loss did not improve
Epoch 00227: val_loss did not improve
Epoch 00228: val_loss did not improve
Epoch 00229: val_loss did not improve
Epoch 00230: val_loss did not improve
Epoch 00231: val_loss did not improve
Epoch 00232: val_loss did not improve
Epoch 00233: val_loss did not improve
Epoch 00234: val_loss did not improve
Epoch 00235: val_loss did not improve
Epoch 00236: val_loss did not improve
Epoch 00237: val_loss did not improve
Epoch 00238: val_loss did not improve
Epoch 00239: val_loss did not improve
Epoch 00240: val_loss did not improve
Epoch 00241: val_loss did not improve
Epoch 00242: val_loss did not improve
Epoch 00243: val_loss did not improve
Epoch 00244: val_loss did not improve
Epoch 00245: val_loss did not improve
Epoch 00246: val_loss did not improve
Epoch 00247: val_loss did not improve
Epoch 00248: val_loss did not improve
Epoch 00249: val_loss improved from 0.06270 to 0.05728, saving mod
el to my_model.h5
Epoch 00250: val_loss did not improve
Epoch 00251: val_loss improved from 0.05728 to 0.05177, saving mod
el to my_model.h5
Epoch 00252: val_loss did not improve
Epoch 00253: val_loss did not improve
Epoch 00254: val_loss did not improve
Epoch 00255: val_loss did not improve
Epoch 00256: val_loss did not improve
Epoch 00257: val_loss did not improve
Epoch 00258: val_loss did not improve
Epoch 00259: val_loss did not improve
Epoch 00260: val_loss did not improve
Epoch 00261: val_loss did not improve
Epoch 00262: val_loss did not improve
Epoch 00263: val_loss did not improve
Epoch 00264: val_loss did not improve
Epoch 00265: val_loss did not improve
Epoch 00266: val_loss did not improve
Epoch 00267: val_loss did not improve
Epoch 00268: val_loss did not improve
Epoch 00269: val_loss did not improve
```

```
Epoch 00270: val_loss did not improve
Epoch 00271: val_loss did not improve
Epoch 00272: val_loss did not improve
Epoch 00273: val_loss improved from 0.05177 to 0.04400, saving mod
el to my_model.h5
Epoch 00274: val_loss did not improve
Epoch 00275: val_loss did not improve
Epoch 00276: val_loss did not improve
Epoch 00277: val_loss did not improve
Epoch 00278: val_loss did not improve
Epoch 00279: val_loss did not improve
Epoch 00280: val_loss did not improve
Epoch 00281: val_loss did not improve
Epoch 00282: val_loss did not improve
Epoch 00283: val_loss did not improve
Epoch 00284: val_loss did not improve
Epoch 00285: val_loss did not improve
Epoch 00286: val_loss did not improve
Epoch 00287: val_loss did not improve
Epoch 00288: val_loss did not improve
Epoch 00289: val_loss did not improve
Epoch 00290: val_loss did not improve
Epoch 00291: val_loss did not improve
Epoch 00292: val_loss did not improve
Epoch 00293: val_loss did not improve
Epoch 00294: val_loss did not improve
Epoch 00295: val_loss did not improve
Epoch 00296: val_loss did not improve
Epoch 00297: val_loss did not improve
Epoch 00298: val_loss did not improve
Epoch 00299: val_loss did not improve
Epoch 00300: val_loss did not improve
Epoch 00301: val_loss did not improve
Epoch 00302: val_loss did not improve
Epoch 00303: val_loss did not improve
Epoch 00304: val_loss improved from 0.04400 to 0.04223, saving mod
el to my_model.h5
Epoch 00305: val_loss did not improve
Epoch 00306: val_loss did not improve
Epoch 00307: val_loss did not improve
Epoch 00308: val_loss did not improve
Epoch 00309: val_loss did not improve
Epoch 00310: val_loss did not improve
Epoch 00311: val_loss did not improve
Epoch 00312: val_loss did not improve
Epoch 00313: val_loss did not improve
Epoch 00314: val_loss did not improve
Epoch 00315: val_loss improved from 0.04223 to 0.03897, saving mod
el to my_model.h5
Epoch 00316: val_loss did not improve
Epoch 00317: val_loss did not improve
Epoch 00318: val_loss did not improve
Epoch 00319: val_loss did not improve
Epoch 00320: val_loss did not improve
Epoch 00321: val_loss did not improve
Epoch 00322: val_loss did not improve
Epoch 00323: val_loss did not improve
```

```
Epoch 00324: val_loss did not improve
Epoch 00325: val_loss did not improve
Epoch 00326: val_loss did not improve
Epoch 00327: val_loss did not improve
Epoch 00328: val_loss did not improve
Epoch 00329: val_loss did not improve
Epoch 00330: val_loss did not improve
Epoch 00331: val_loss did not improve
Epoch 00332: val_loss did not improve
Epoch 00333: val_loss did not improve
Epoch 00334: val_loss did not improve
Epoch 00335: val_loss did not improve
Epoch 00336: val_loss did not improve
Epoch 00337: val_loss did not improve
Epoch 00338: val_loss did not improve
Epoch 00339: val_loss did not improve
Epoch 00340: val_loss did not improve
Epoch 00341: val_loss did not improve
Epoch 00342: val_loss did not improve
Epoch 00343: val_loss did not improve
Epoch 00344: val_loss did not improve
Epoch 00345: val_loss did not improve
Epoch 00346: val_loss did not improve
Epoch 00347: val_loss did not improve
Epoch 00348: val_loss did not improve
Epoch 00349: val_loss did not improve
Epoch 00350: val_loss did not improve
Epoch 00351: val_loss did not improve
Epoch 00352: val_loss did not improve
Epoch 00353: val_loss improved from 0.03897 to 0.03879, saving mod
el to my_model.h5
Epoch 00354: val_loss did not improve
Epoch 00355: val_loss did not improve
Epoch 00356: val_loss did not improve
Epoch 00357: val_loss did not improve
Epoch 00358: val_loss did not improve
Epoch 00359: val_loss did not improve
Epoch 00360: val_loss did not improve
Epoch 00361: val_loss did not improve
Epoch 00362: val_loss did not improve
Epoch 00363: val_loss improved from 0.03879 to 0.03869, saving mod
el to my_model.h5
Epoch 00364: val_loss did not improve
Epoch 00365: val_loss did not improve
Epoch 00366: val_loss did not improve
Epoch 00367: val_loss did not improve
Epoch 00368: val_loss did not improve
Epoch 00369: val_loss did not improve
Epoch 00370: val_loss did not improve
Epoch 00371: val_loss did not improve
Epoch 00372: val_loss improved from 0.03869 to 0.03727, saving mod
el to my_model.h5
Epoch 00373: val_loss did not improve
Epoch 00374: val_loss did not improve
Epoch 00375: val_loss did not improve
Epoch 00376: val_loss did not improve
Epoch 00377: val_loss did not improve
```

```
Epoch 00378: val_loss did not improve
Epoch 00379: val_loss did not improve
Epoch 00380: val_loss did not improve
Epoch 00381: val_loss did not improve
Epoch 00382: val_loss did not improve
Epoch 00383: val_loss did not improve
Epoch 00384: val_loss did not improve
Epoch 00385: val_loss did not improve
Epoch 00386: val_loss did not improve
Epoch 00387: val_loss did not improve
Epoch 00388: val_loss did not improve
Epoch 00389: val_loss did not improve
Epoch 00390: val_loss improved from 0.03727 to 0.03689, saving mod
el to my_model.h5
Epoch 00391: val_loss did not improve
Epoch 00392: val_loss did not improve
Epoch 00393: val_loss did not improve
Epoch 00394: val_loss did not improve
Epoch 00395: val_loss did not improve
Epoch 00396: val_loss improved from 0.03689 to 0.03665, saving mod
el to my_model.h5
Epoch 00397: val_loss improved from 0.03665 to 0.03310, saving mod
el to my_model.h5
Epoch 00398: val_loss did not improve
Epoch 00399: val_loss did not improve
Epoch 00400: val_loss did not improve
Epoch 00401: val_loss did not improve
Epoch 00402: val_loss did not improve
Epoch 00403: val_loss did not improve
Epoch 00404: val_loss did not improve
Epoch 00405: val_loss did not improve
Epoch 00406: val_loss did not improve
Epoch 00407: val_loss did not improve
Epoch 00408: val_loss did not improve
Epoch 00409: val_loss did not improve
Epoch 00410: val_loss did not improve
Epoch 00411: val_loss did not improve
Epoch 00412: val_loss did not improve
Epoch 00413: val_loss did not improve
Epoch 00414: val_loss did not improve
Epoch 00415: val_loss did not improve
Epoch 00416: val_loss did not improve
Epoch 00417: val_loss did not improve
Epoch 00418: val_loss did not improve
Epoch 00419: val_loss did not improve
Epoch 00420: val_loss improved from 0.03310 to 0.03087, saving mod
el to my_model.h5
Epoch 00421: val_loss did not improve
Epoch 00422: val_loss did not improve
Epoch 00423: val_loss did not improve
Epoch 00424: val_loss did not improve
Epoch 00425: val_loss did not improve
Epoch 00426: val_loss did not improve
Epoch 00427: val_loss did not improve
Epoch 00428: val_loss did not improve
Epoch 00429: val_loss did not improve
Epoch 00430: val_loss did not improve
```

```
Epoch 00431: val_loss did not improve
Epoch 00432: val_loss did not improve
Epoch 00433: val_loss did not improve
Epoch 00434: val_loss did not improve
Epoch 00435: val_loss did not improve
Epoch 00436: val_loss did not improve
Epoch 00437: val_loss did not improve
Epoch 00438: val_loss did not improve
Epoch 00439: val_loss did not improve
Epoch 00440: val_loss did not improve
Epoch 00441: val_loss did not improve
Epoch 00442: val_loss did not improve
Epoch 00443: val_loss did not improve
Epoch 00444: val_loss did not improve
Epoch 00445: val_loss did not improve
Epoch 00446: val_loss did not improve
Epoch 00447: val_loss did not improve
Epoch 00448: val_loss did not improve
Epoch 00449: val_loss did not improve
Epoch 00450: val_loss did not improve
Epoch 00451: val_loss did not improve
Epoch 00452: val_loss improved from 0.03087 to 0.03074, saving mod
el to my_model.h5
Epoch 00453: val_loss did not improve
Epoch 00454: val_loss did not improve
Epoch 00455: val_loss did not improve
Epoch 00456: val_loss did not improve
Epoch 00457: val_loss did not improve
Epoch 00458: val_loss did not improve
Epoch 00459: val_loss did not improve
Epoch 00460: val_loss did not improve
Epoch 00461: val_loss did not improve
Epoch 00462: val_loss did not improve
Epoch 00463: val_loss did not improve
Epoch 00464: val_loss did not improve
Epoch 00465: val_loss did not improve
Epoch 00466: val_loss did not improve
Epoch 00467: val_loss did not improve
Epoch 00468: val_loss did not improve
Epoch 00469: val_loss did not improve
Epoch 00470: val_loss did not improve
Epoch 00471: val_loss did not improve
Epoch 00472: val_loss did not improve
Epoch 00473: val_loss did not improve
Epoch 00474: val_loss did not improve
Epoch 00475: val_loss did not improve
Epoch 00476: val_loss did not improve
Epoch 00477: val_loss did not improve
Epoch 00478: val_loss did not improve
Epoch 00479: val_loss did not improve
Epoch 00480: val_loss did not improve
Epoch 00481: val_loss did not improve
Epoch 00482: val_loss did not improve
Epoch 00483: val_loss did not improve
Epoch 00484: val_loss did not improve
Epoch 00485: val_loss did not improve
Epoch 00486: val_loss did not improve
```

```
Epoch 00487: val_loss did not improve
Epoch 00488: val_loss did not improve
Epoch 00489: val_loss did not improve
Epoch 00490: val_loss did not improve
Epoch 00491: val_loss did not improve
Epoch 00492: val_loss did not improve
Epoch 00493: val_loss did not improve
Epoch 00494: val_loss did not improve
Epoch 00495: val_loss did not improve
Epoch 00496: val_loss did not improve
Epoch 00497: val_loss improved from 0.03074 to 0.03020, saving mod
el to my_model.h5
Epoch 00498: val_loss did not improve
Epoch 00499: val_loss did not improve
Epoch 00500: val_loss did not improve
Epoch 00501: val_loss did not improve
Epoch 00502: val_loss did not improve
Epoch 00503: val_loss did not improve
Epoch 00504: val_loss improved from 0.03020 to 0.02954, saving mod
el to my_model.h5
Epoch 00505: val_loss did not improve
Epoch 00506: val_loss did not improve
Epoch 00507: val_loss did not improve
Epoch 00508: val_loss did not improve
Epoch 00509: val_loss did not improve
Epoch 00510: val_loss did not improve
Epoch 00511: val_loss did not improve
Epoch 00512: val_loss did not improve
Epoch 00513: val_loss did not improve
Epoch 00514: val_loss did not improve
Epoch 00515: val_loss did not improve
Epoch 00516: val_loss did not improve
Epoch 00517: val_loss did not improve
Epoch 00518: val_loss did not improve
Epoch 00519: val_loss did not improve
Epoch 00520: val_loss did not improve
Epoch 00521: val_loss did not improve
Epoch 00522: val_loss did not improve
Epoch 00523: val_loss did not improve
Epoch 00524: val_loss did not improve
Epoch 00525: val_loss did not improve
Epoch 00526: val_loss did not improve
Epoch 00527: val_loss did not improve
Epoch 00528: val_loss did not improve
Epoch 00529: val_loss did not improve
Epoch 00530: val_loss did not improve
Epoch 00531: val_loss did not improve
Epoch 00532: val_loss did not improve
Epoch 00533: val_loss did not improve
Epoch 00534: val_loss did not improve
Epoch 00535: val_loss did not improve
Epoch 00536: val_loss did not improve
Epoch 00537: val_loss did not improve
Epoch 00538: val_loss did not improve
Epoch 00539: val_loss did not improve
Epoch 00540: val_loss did not improve
Epoch 00541: val_loss did not improve
```

```
Epoch 00542: val_loss did not improve
Epoch 00543: val_loss did not improve
Epoch 00544: val_loss did not improve
Epoch 00545: val_loss did not improve
Epoch 00546: val_loss did not improve
Epoch 00547: val_loss did not improve
Epoch 00548: val_loss did not improve
Epoch 00549: val_loss did not improve
Epoch 00550: val_loss did not improve
Epoch 00551: val_loss did not improve
Epoch 00552: val_loss did not improve
Epoch 00553: val_loss did not improve
Epoch 00554: val_loss did not improve
Epoch 00555: val_loss did not improve
Epoch 00556: val_loss did not improve
Epoch 00557: val_loss did not improve
Epoch 00558: val_loss did not improve
Epoch 00559: val_loss did not improve
Epoch 00560: val_loss did not improve
Epoch 00561: val_loss did not improve
Epoch 00562: val_loss did not improve
Epoch 00563: val_loss did not improve
Epoch 00564: val_loss did not improve
Epoch 00565: val_loss did not improve
Epoch 00566: val_loss did not improve
Epoch 00567: val_loss did not improve
Epoch 00568: val_loss did not improve
Epoch 00569: val_loss improved from 0.02954 to 0.02909, saving mod
el to my_model.h5
Epoch 00570: val_loss did not improve
Epoch 00571: val_loss did not improve
Epoch 00572: val_loss did not improve
Epoch 00573: val_loss did not improve
Epoch 00574: val_loss improved from 0.02909 to 0.02868, saving mod
el to my_model.h5
Epoch 00575: val_loss did not improve
Epoch 00576: val_loss did not improve
Epoch 00577: val_loss did not improve
Epoch 00578: val_loss did not improve
Epoch 00579: val_loss did not improve
Epoch 00580: val_loss did not improve
Epoch 00581: val_loss did not improve
Epoch 00582: val_loss did not improve
Epoch 00583: val_loss did not improve
Epoch 00584: val_loss did not improve
Epoch 00585: val_loss did not improve
Epoch 00586: val_loss did not improve
Epoch 00587: val_loss did not improve
Epoch 00588: val_loss did not improve
Epoch 00589: val_loss did not improve
Epoch 00590: val_loss did not improve
Epoch 00591: val_loss did not improve
Epoch 00592: val_loss improved from 0.02868 to 0.02831, saving mod
el to my_model.h5
Epoch 00593: val_loss did not improve
Epoch 00594: val_loss did not improve
Epoch 00595: val_loss did not improve
```

```
Epoch 00596: val_loss did not improve
Epoch 00597: val_loss did not improve
Epoch 00598: val_loss did not improve
Epoch 00599: val_loss did not improve
Epoch 00600: val_loss did not improve
Epoch 00601: val_loss did not improve
Epoch 00602: val_loss did not improve
Epoch 00603: val_loss did not improve
Epoch 00604: val_loss did not improve
Epoch 00605: val_loss did not improve
Epoch 00606: val_loss did not improve
Epoch 00607: val_loss did not improve
Epoch 00608: val_loss did not improve
Epoch 00609: val_loss did not improve
Epoch 00610: val_loss did not improve
Epoch 00611: val_loss did not improve
Epoch 00612: val_loss did not improve
Epoch 00613: val_loss did not improve
Epoch 00614: val_loss did not improve
Epoch 00615: val_loss did not improve
Epoch 00616: val_loss did not improve
Epoch 00617: val_loss did not improve
Epoch 00618: val_loss did not improve
Epoch 00619: val_loss did not improve
Epoch 00620: val_loss did not improve
Epoch 00621: val_loss did not improve
Epoch 00622: val_loss did not improve
Epoch 00623: val_loss did not improve
Epoch 00624: val_loss did not improve
Epoch 00625: val_loss did not improve
Epoch 00626: val_loss did not improve
Epoch 00627: val_loss did not improve
Epoch 00628: val_loss did not improve
Epoch 00629: val_loss did not improve
Epoch 00630: val_loss improved from 0.02831 to 0.02830, saving mod
el to my_model.h5
Epoch 00631: val_loss did not improve
Epoch 00632: val_loss did not improve
Epoch 00633: val_loss did not improve
Epoch 00634: val_loss did not improve
Epoch 00635: val_loss did not improve
Epoch 00636: val_loss did not improve
Epoch 00637: val_loss did not improve
Epoch 00638: val_loss did not improve
Epoch 00639: val_loss did not improve
Epoch 00640: val_loss did not improve
Epoch 00641: val_loss did not improve
Epoch 00642: val_loss did not improve
Epoch 00643: val_loss did not improve
Epoch 00644: val_loss did not improve
Epoch 00645: val_loss did not improve
Epoch 00646: val_loss did not improve
Epoch 00647: val_loss did not improve
Epoch 00648: val_loss did not improve
Epoch 00649: val_loss improved from 0.02830 to 0.02808, saving mod
el to my_model.h5
Epoch 00650: val_loss did not improve
```

```
Epoch 00651: val_loss did not improve
Epoch 00652: val_loss did not improve
Epoch 00653: val_loss did not improve
Epoch 00654: val_loss did not improve
Epoch 00655: val_loss did not improve
Epoch 00656: val_loss improved from 0.02808 to 0.02770, saving mod
el to my_model.h5
Epoch 00657: val_loss did not improve
Epoch 00658: val_loss did not improve
Epoch 00659: val_loss did not improve
Epoch 00660: val_loss did not improve
Epoch 00661: val_loss did not improve
Epoch 00662: val_loss did not improve
Epoch 00663: val_loss did not improve
Epoch 00664: val_loss did not improve
Epoch 00665: val_loss did not improve
Epoch 00666: val_loss did not improve
Epoch 00667: val_loss did not improve
Epoch 00668: val_loss did not improve
Epoch 00669: val_loss did not improve
Epoch 00670: val_loss did not improve
Epoch 00671: val_loss did not improve
Epoch 00672: val_loss did not improve
Epoch 00673: val_loss did not improve
Epoch 00674: val_loss did not improve
Epoch 00675: val_loss did not improve
Epoch 00676: val_loss did not improve
Epoch 00677: val_loss did not improve
Epoch 00678: val_loss did not improve
Epoch 00679: val_loss did not improve
Epoch 00680: val_loss improved from 0.02770 to 0.02696, saving mod
el to my_model.h5
Epoch 00681: val_loss did not improve
Epoch 00682: val_loss did not improve
Epoch 00683: val_loss did not improve
Epoch 00684: val_loss did not improve
Epoch 00685: val_loss did not improve
Epoch 00686: val_loss did not improve
Epoch 00687: val_loss did not improve
Epoch 00688: val_loss did not improve
Epoch 00689: val_loss did not improve
Epoch 00690: val_loss did not improve
Epoch 00691: val_loss did not improve
Epoch 00692: val_loss did not improve
Epoch 00693: val_loss did not improve
Epoch 00694: val_loss did not improve
Epoch 00695: val_loss did not improve
Epoch 00696: val_loss did not improve
Epoch 00697: val_loss did not improve
Epoch 00698: val_loss did not improve
Epoch 00699: val_loss did not improve
Epoch 00700: val_loss did not improve
Epoch 00701: val_loss did not improve
Epoch 00702: val_loss did not improve
Epoch 00703: val_loss did not improve
Epoch 00704: val_loss did not improve
Epoch 00705: val_loss did not improve
```

```
Epoch 00706: val_loss did not improve
Epoch 00707: val_loss did not improve
Epoch 00708: val_loss did not improve
Epoch 00709: val_loss did not improve
Epoch 00710: val_loss did not improve
Epoch 00711: val_loss did not improve
Epoch 00712: val_loss did not improve
Epoch 00713: val_loss did not improve
Epoch 00714: val_loss did not improve
Epoch 00715: val_loss did not improve
Epoch 00716: val_loss did not improve
Epoch 00717: val_loss did not improve
Epoch 00718: val_loss did not improve
Epoch 00719: val_loss did not improve
Epoch 00720: val_loss did not improve
Epoch 00721: val_loss did not improve
Epoch 00722: val_loss did not improve
Epoch 00723: val_loss did not improve
Epoch 00724: val_loss did not improve
Epoch 00725: val_loss did not improve
Epoch 00726: val_loss did not improve
Epoch 00727: val_loss did not improve
Epoch 00728: val_loss did not improve
Epoch 00729: val_loss did not improve
Epoch 00730: val_loss did not improve
Epoch 00731: val_loss did not improve
Epoch 00732: val_loss did not improve
Epoch 00733: val_loss did not improve
Epoch 00734: val_loss did not improve
Epoch 00735: val_loss did not improve
Epoch 00736: val_loss did not improve
Epoch 00737: val_loss did not improve
Epoch 00738: val_loss did not improve
Epoch 00739: val_loss did not improve
Epoch 00740: val_loss did not improve
Epoch 00741: val_loss did not improve
Epoch 00742: val_loss did not improve
Epoch 00743: val_loss did not improve
Epoch 00744: val_loss did not improve
Epoch 00745: val_loss did not improve
Epoch 00746: val_loss did not improve
Epoch 00747: val_loss did not improve
Epoch 00748: val_loss did not improve
Epoch 00749: val_loss did not improve
Epoch 00750: val_loss did not improve
Epoch 00751: val_loss did not improve
Epoch 00752: val_loss did not improve
Epoch 00753: val_loss did not improve
Epoch 00754: val_loss did not improve
Epoch 00755: val_loss did not improve
Epoch 00756: val_loss did not improve
Epoch 00757: val_loss did not improve
Epoch 00758: val_loss did not improve
Epoch 00759: val_loss did not improve
Epoch 00760: val_loss did not improve
Epoch 00761: val_loss did not improve
Epoch 00762: val_loss did not improve
```

```
Epoch 00763: val_loss did not improve
Epoch 00764: val_loss did not improve
Epoch 00765: val_loss did not improve
Epoch 00766: val_loss did not improve
Epoch 00767: val_loss did not improve
Epoch 00768: val_loss did not improve
Epoch 00769: val_loss did not improve
Epoch 00770: val_loss improved from 0.02696 to 0.02625, saving mod
el to my_model.h5
Epoch 00771: val_loss did not improve
Epoch 00772: val_loss did not improve
Epoch 00773: val_loss did not improve
Epoch 00774: val_loss did not improve
Epoch 00775: val_loss did not improve
Epoch 00776: val_loss did not improve
Epoch 00777: val_loss did not improve
Epoch 00778: val_loss did not improve
Epoch 00779: val_loss did not improve
Epoch 00780: val_loss did not improve
Epoch 00781: val_loss did not improve
Epoch 00782: val_loss did not improve
Epoch 00783: val_loss did not improve
Epoch 00784: val_loss did not improve
Epoch 00785: val_loss did not improve
Epoch 00786: val_loss did not improve
Epoch 00787: val_loss did not improve
Epoch 00788: val_loss did not improve
Epoch 00789: val_loss did not improve
Epoch 00790: val_loss did not improve
Epoch 00791: val_loss did not improve
Epoch 00792: val_loss did not improve
Epoch 00793: val_loss did not improve
Epoch 00794: val_loss did not improve
Epoch 00795: val_loss did not improve
Epoch 00796: val_loss did not improve
Epoch 00797: val_loss did not improve
Epoch 00798: val_loss did not improve
Epoch 00799: val_loss did not improve
Epoch 00800: val_loss did not improve
Epoch 00801: val_loss did not improve
Epoch 00802: val_loss did not improve
Epoch 00803: val_loss did not improve
Epoch 00804: val_loss did not improve
Epoch 00805: val_loss did not improve
Epoch 00806: val_loss did not improve
Epoch 00807: val_loss did not improve
Epoch 00808: val_loss did not improve
Epoch 00809: val_loss did not improve
Epoch 00810: val_loss did not improve
Epoch 00811: val_loss did not improve
Epoch 00812: val_loss did not improve
Epoch 00813: val_loss did not improve
Epoch 00814: val_loss did not improve
Epoch 00815: val_loss did not improve
Epoch 00816: val_loss did not improve
Epoch 00817: val_loss did not improve
Epoch 00818: val_loss did not improve
```

```
Epoch 00819: val_loss did not improve
Epoch 00820: val_loss did not improve
Epoch 00821: val_loss did not improve
Epoch 00822: val_loss did not improve
Epoch 00823: val_loss did not improve
Epoch 00824: val_loss did not improve
Epoch 00825: val_loss did not improve
Epoch 00826: val_loss did not improve
Epoch 00827: val_loss did not improve
Epoch 00828: val_loss did not improve
Epoch 00829: val_loss did not improve
Epoch 00830: val_loss did not improve
Epoch 00831: val_loss did not improve
Epoch 00832: val_loss did not improve
Epoch 00833: val_loss did not improve
Epoch 00834: val_loss did not improve
Epoch 00835: val_loss did not improve
Epoch 00836: val_loss did not improve
Epoch 00837: val_loss did not improve
Epoch 00838: val_loss did not improve
Epoch 00839: val_loss did not improve
Epoch 00840: val_loss did not improve
Epoch 00841: val_loss did not improve
Epoch 00842: val_loss did not improve
Epoch 00843: val_loss did not improve
Epoch 00844: val_loss did not improve
Epoch 00845: val_loss did not improve
Epoch 00846: val_loss did not improve
Epoch 00847: val_loss did not improve
Epoch 00848: val_loss did not improve
Epoch 00849: val_loss did not improve
Epoch 00850: val_loss did not improve
Epoch 00851: val_loss did not improve
Epoch 00852: val_loss did not improve
Epoch 00853: val_loss did not improve
Epoch 00854: val_loss did not improve
Epoch 00855: val_loss did not improve
Epoch 00856: val_loss did not improve
Epoch 00857: val_loss did not improve
Epoch 00858: val_loss did not improve
Epoch 00859: val_loss did not improve
Epoch 00860: val_loss did not improve
Epoch 00861: val_loss did not improve
Epoch 00862: val_loss did not improve
Epoch 00863: val_loss did not improve
Epoch 00864: val_loss did not improve
Epoch 00865: val_loss did not improve
Epoch 00866: val_loss did not improve
Epoch 00867: val_loss did not improve
Epoch 00868: val_loss did not improve
Epoch 00869: val_loss did not improve
Epoch 00870: val_loss did not improve
Epoch 00871: val_loss did not improve
Epoch 00872: val_loss did not improve
Epoch 00873: val_loss did not improve
Epoch 00874: val_loss did not improve
Epoch 00875: val_loss did not improve
```

```
Epoch 00876: val_loss did not improve
Epoch 00877: val_loss did not improve
Epoch 00878: val_loss did not improve
Epoch 00879: val_loss did not improve
Epoch 00880: val_loss did not improve
Epoch 00881: val_loss did not improve
Epoch 00882: val_loss did not improve
Epoch 00883: val_loss did not improve
Epoch 00884: val_loss did not improve
Epoch 00885: val_loss did not improve
Epoch 00886: val_loss did not improve
Epoch 00887: val_loss did not improve
Epoch 00888: val_loss did not improve
Epoch 00889: val_loss did not improve
Epoch 00890: val_loss did not improve
Epoch 00891: val_loss did not improve
Epoch 00892: val_loss did not improve
Epoch 00893: val_loss did not improve
Epoch 00894: val_loss did not improve
Epoch 00895: val_loss did not improve
Epoch 00896: val_loss did not improve
Epoch 00897: val_loss did not improve
Epoch 00898: val_loss did not improve
Epoch 00899: val_loss did not improve
Epoch 00900: val_loss did not improve
Epoch 00901: val_loss did not improve
Epoch 00902: val_loss did not improve
Epoch 00903: val_loss did not improve
Epoch 00904: val_loss did not improve
Epoch 00905: val_loss did not improve
Epoch 00906: val_loss did not improve
Epoch 00907: val_loss did not improve
Epoch 00908: val_loss did not improve
Epoch 00909: val_loss did not improve
Epoch 00910: val_loss did not improve
Epoch 00911: val_loss did not improve
Epoch 00912: val_loss did not improve
Epoch 00913: val_loss did not improve
Epoch 00914: val_loss did not improve
Epoch 00915: val_loss did not improve
Epoch 00916: val_loss did not improve
Epoch 00917: val_loss did not improve
Epoch 00918: val_loss did not improve
Epoch 00919: val_loss did not improve
Epoch 00920: val_loss did not improve
Epoch 00921: val_loss did not improve
Epoch 00922: val_loss did not improve
Epoch 00923: val_loss did not improve
Epoch 00924: val_loss did not improve
Epoch 00925: val_loss did not improve
Epoch 00926: val_loss did not improve
Epoch 00927: val_loss did not improve
Epoch 00928: val_loss did not improve
Epoch 00929: val_loss did not improve
Epoch 00930: val_loss did not improve
Epoch 00931: val_loss did not improve
Epoch 00932: val_loss did not improve
```

```
Epoch 00933: val_loss did not improve
Epoch 00934: val_loss did not improve
Epoch 00935: val_loss did not improve
Epoch 00936: val_loss did not improve
Epoch 00937: val_loss did not improve
Epoch 00938: val_loss did not improve
Epoch 00939: val_loss did not improve
Epoch 00940: val_loss did not improve
Epoch 00941: val_loss did not improve
Epoch 00942: val_loss did not improve
Epoch 00943: val_loss did not improve
Epoch 00944: val_loss did not improve
Epoch 00945: val_loss did not improve
Epoch 00946: val_loss did not improve
Epoch 00947: val_loss did not improve
Epoch 00948: val_loss did not improve
Epoch 00949: val_loss did not improve
Epoch 00950: val_loss did not improve
Epoch 00951: val_loss did not improve
Epoch 00952: val_loss did not improve
Epoch 00953: val_loss did not improve
Epoch 00954: val_loss did not improve
Epoch 00955: val_loss did not improve
Epoch 00956: val_loss did not improve
Epoch 00957: val_loss did not improve
Epoch 00958: val_loss did not improve
Epoch 00959: val_loss did not improve
Epoch 00960: val_loss did not improve
Epoch 00961: val_loss did not improve
Epoch 00962: val_loss did not improve
Epoch 00963: val_loss did not improve
Epoch 00964: val_loss did not improve
Epoch 00965: val_loss did not improve
Epoch 00966: val_loss did not improve
Epoch 00967: val_loss did not improve
Epoch 00968: val_loss did not improve
Epoch 00969: val_loss did not improve
Epoch 00970: val_loss improved from 0.02625 to 0.02600, saving mod
el to my_model.h5
Epoch 00971: val_loss did not improve
Epoch 00972: val_loss did not improve
Epoch 00973: val_loss did not improve
Epoch 00974: val_loss did not improve
Epoch 00975: val_loss did not improve
Epoch 00976: val_loss did not improve
Epoch 00977: val_loss did not improve
Epoch 00978: val_loss did not improve
Epoch 00979: val_loss did not improve
Epoch 00980: val_loss did not improve
Epoch 00981: val_loss did not improve
Epoch 00982: val_loss did not improve
Epoch 00983: val_loss did not improve
Epoch 00984: val_loss did not improve
Epoch 00985: val_loss did not improve
Epoch 00986: val_loss did not improve
Epoch 00987: val_loss did not improve
Epoch 00988: val_loss did not improve
```

```
Epoch 00989: val_loss did not improve
Epoch 00990: val_loss did not improve
Epoch 00991: val_loss did not improve
Epoch 00992: val_loss did not improve
Epoch 00993: val_loss did not improve
Epoch 00994: val_loss did not improve
Epoch 00995: val_loss did not improve
Epoch 00996: val_loss improved from 0.02600 to 0.02599, saving mod
el to my_model.h5
Epoch 00997: val_loss did not improve
Epoch 00998: val_loss did not improve
Epoch 00999: val_loss did not improve
Epoch 01000: val_loss did not improve
Epoch 01001: val_loss did not improve
Epoch 01002: val_loss did not improve
Epoch 01003: val_loss did not improve
Epoch 01004: val_loss did not improve
Epoch 01005: val_loss did not improve
Epoch 01006: val_loss did not improve
Epoch 01007: val_loss did not improve
Epoch 01008: val_loss did not improve
Epoch 01009: val_loss did not improve
Epoch 01010: val_loss did not improve
Epoch 01011: val_loss did not improve
Epoch 01012: val_loss did not improve
Epoch 01013: val_loss did not improve
Epoch 01014: val_loss did not improve
Epoch 01015: val_loss did not improve
Epoch 01016: val_loss did not improve
Epoch 01017: val_loss did not improve
Epoch 01018: val_loss did not improve
Epoch 01019: val_loss did not improve
Epoch 01020: val_loss did not improve
Epoch 01021: val_loss did not improve
Epoch 01022: val_loss did not improve
Epoch 01023: val_loss did not improve
Epoch 01024: val_loss improved from 0.02599 to 0.02598, saving mod
el to my_model.h5
Epoch 01025: val_loss did not improve
Epoch 01026: val_loss did not improve
Epoch 01027: val_loss did not improve
Epoch 01028: val_loss did not improve
Epoch 01029: val_loss did not improve
Epoch 01030: val_loss did not improve
Epoch 01031: val_loss did not improve
Epoch 01032: val_loss improved from 0.02598 to 0.02596, saving mod
el to my_model.h5
Epoch 01033: val_loss did not improve
Epoch 01034: val_loss did not improve
Epoch 01035: val_loss did not improve
Epoch 01036: val_loss did not improve
Epoch 01037: val_loss did not improve
Epoch 01038: val_loss did not improve
Epoch 01039: val_loss did not improve
Epoch 01040: val_loss did not improve
Epoch 01041: val_loss did not improve
Epoch 01042: val_loss did not improve
```

```
Epoch 01043: val_loss did not improve
Epoch 01044: val_loss did not improve
Epoch 01045: val_loss did not improve
Epoch 01046: val_loss did not improve
Epoch 01047: val_loss did not improve
Epoch 01048: val_loss did not improve
Epoch 01049: val_loss did not improve
Epoch 01050: val_loss did not improve
Epoch 01051: val_loss did not improve
Epoch 01052: val_loss did not improve
Epoch 01053: val_loss did not improve
Epoch 01054: val_loss did not improve
Epoch 01055: val_loss did not improve
Epoch 01056: val_loss did not improve
Epoch 01057: val_loss did not improve
Epoch 01058: val_loss did not improve
Epoch 01059: val_loss did not improve
Epoch 01060: val_loss did not improve
Epoch 01061: val_loss did not improve
Epoch 01062: val_loss did not improve
Epoch 01063: val_loss did not improve
Epoch 01064: val_loss did not improve
Epoch 01065: val_loss did not improve
Epoch 01066: val_loss improved from 0.02596 to 0.02564, saving mod
el to my_model.h5
Epoch 01067: val_loss did not improve
Epoch 01068: val_loss did not improve
Epoch 01069: val_loss did not improve
Epoch 01070: val_loss did not improve
Epoch 01071: val_loss did not improve
Epoch 01072: val_loss did not improve
Epoch 01073: val_loss did not improve
Epoch 01074: val_loss did not improve
Epoch 01075: val_loss did not improve
Epoch 01076: val_loss did not improve
Epoch 01077: val_loss did not improve
Epoch 01078: val_loss did not improve
Epoch 01079: val_loss did not improve
Epoch 01080: val_loss did not improve
Epoch 01081: val_loss did not improve
Epoch 01082: val_loss did not improve
Epoch 01083: val_loss did not improve
Epoch 01084: val_loss did not improve
Epoch 01085: val_loss did not improve
Epoch 01086: val_loss did not improve
Epoch 01087: val_loss did not improve
Epoch 01088: val_loss did not improve
Epoch 01089: val_loss did not improve
Epoch 01090: val_loss did not improve
Epoch 01091: val_loss did not improve
Epoch 01092: val_loss did not improve
Epoch 01093: val_loss did not improve
Epoch 01094: val_loss did not improve
Epoch 01095: val_loss did not improve
Epoch 01096: val_loss did not improve
Epoch 01097: val_loss did not improve
Epoch 01098: val_loss did not improve
```

```
Epoch 01099: val_loss did not improve
Epoch 01100: val_loss did not improve
Epoch 01101: val_loss did not improve
Epoch 01102: val_loss did not improve
Epoch 01103: val_loss did not improve
Epoch 01104: val_loss did not improve
Epoch 01105: val_loss did not improve
Epoch 01106: val_loss did not improve
Epoch 01107: val_loss did not improve
Epoch 01108: val_loss did not improve
Epoch 01109: val_loss did not improve
Epoch 01110: val_loss did not improve
Epoch 01111: val_loss did not improve
Epoch 01112: val_loss did not improve
Epoch 01113: val_loss did not improve
Epoch 01114: val_loss did not improve
Epoch 01115: val_loss did not improve
Epoch 01116: val_loss did not improve
Epoch 01117: val_loss did not improve
Epoch 01118: val_loss did not improve
Epoch 01119: val_loss did not improve
Epoch 01120: val_loss did not improve
Epoch 01121: val_loss did not improve
Epoch 01122: val_loss did not improve
Epoch 01123: val_loss did not improve
Epoch 01124: val_loss did not improve
Epoch 01125: val_loss did not improve
Epoch 01126: val_loss did not improve
Epoch 01127: val_loss did not improve
Epoch 01128: val_loss did not improve
Epoch 01129: val_loss did not improve
Epoch 01130: val_loss did not improve
Epoch 01131: val_loss did not improve
Epoch 01132: val_loss did not improve
Epoch 01133: val_loss did not improve
Epoch 01134: val_loss did not improve
Epoch 01135: val_loss did not improve
Epoch 01136: val_loss did not improve
Epoch 01137: val_loss did not improve
Epoch 01138: val_loss did not improve
Epoch 01139: val_loss did not improve
Epoch 01140: val_loss did not improve
Epoch 01141: val_loss did not improve
Epoch 01142: val_loss did not improve
Epoch 01143: val_loss did not improve
Epoch 01144: val_loss did not improve
Epoch 01145: val_loss did not improve
Epoch 01146: val_loss did not improve
Epoch 01147: val_loss did not improve
Epoch 01148: val_loss did not improve
Epoch 01149: val_loss did not improve
Epoch 01150: val_loss did not improve
Epoch 01151: val_loss did not improve
Epoch 01152: val_loss did not improve
Epoch 01153: val_loss did not improve
Epoch 01154: val_loss did not improve
Epoch 01155: val_loss did not improve
```

```
Epoch 01156: val_loss did not improve
Epoch 01157: val_loss did not improve
Epoch 01158: val_loss did not improve
Epoch 01159: val_loss did not improve
Epoch 01160: val_loss did not improve
Epoch 01161: val_loss did not improve
Epoch 01162: val_loss did not improve
Epoch 01163: val_loss did not improve
Epoch 01164: val_loss did not improve
Epoch 01165: val_loss did not improve
Epoch 01166: val_loss did not improve
Epoch 01167: val_loss did not improve
Epoch 01168: val_loss did not improve
Epoch 01169: val_loss did not improve
Epoch 01170: val_loss did not improve
Epoch 01171: val_loss did not improve
Epoch 01172: val_loss did not improve
Epoch 01173: val_loss improved from 0.02564 to 0.02536, saving mod
el to my_model.h5
Epoch 01174: val_loss did not improve
Epoch 01175: val_loss did not improve
Epoch 01176: val_loss did not improve
Epoch 01177: val_loss did not improve
Epoch 01178: val_loss did not improve
Epoch 01179: val_loss did not improve
Epoch 01180: val_loss did not improve
Epoch 01181: val_loss did not improve
Epoch 01182: val_loss did not improve
Epoch 01183: val_loss did not improve
Epoch 01184: val_loss did not improve
Epoch 01185: val_loss did not improve
Epoch 01186: val_loss did not improve
Epoch 01187: val_loss did not improve
Epoch 01188: val_loss did not improve
Epoch 01189: val_loss did not improve
Epoch 01190: val_loss did not improve
Epoch 01191: val_loss did not improve
Epoch 01192: val_loss did not improve
Epoch 01193: val_loss did not improve
Epoch 01194: val_loss did not improve
Epoch 01195: val_loss did not improve
Epoch 01196: val_loss did not improve
Epoch 01197: val_loss did not improve
Epoch 01198: val_loss did not improve
Epoch 01199: val_loss did not improve
Epoch 01200: val_loss did not improve
Epoch 01201: val_loss did not improve
Epoch 01202: val_loss did not improve
Epoch 01203: val_loss did not improve
Epoch 01204: val_loss did not improve
Epoch 01205: val_loss did not improve
Epoch 01206: val_loss did not improve
Epoch 01207: val_loss did not improve
Epoch 01208: val_loss did not improve
Epoch 01209: val_loss did not improve
Epoch 01210: val_loss did not improve
Epoch 01211: val_loss did not improve
```

```
Epoch 01212: val_loss did not improve
Epoch 01213: val_loss did not improve
Epoch 01214: val_loss did not improve
Epoch 01215: val_loss did not improve
Epoch 01216: val_loss did not improve
Epoch 01217: val_loss did not improve
Epoch 01218: val_loss did not improve
Epoch 01219: val_loss did not improve
Epoch 01220: val_loss did not improve
Epoch 01221: val_loss did not improve
Epoch 01222: val_loss did not improve
Epoch 01223: val_loss did not improve
Epoch 01224: val_loss did not improve
Epoch 01225: val_loss did not improve
Epoch 01226: val_loss did not improve
Epoch 01227: val_loss did not improve
Epoch 01228: val_loss did not improve
Epoch 01229: val_loss did not improve
Epoch 01230: val_loss did not improve
Epoch 01231: val_loss did not improve
Epoch 01232: val_loss did not improve
Epoch 01233: val_loss did not improve
Epoch 01234: val_loss did not improve
Epoch 01235: val_loss did not improve
Epoch 01236: val_loss did not improve
Epoch 01237: val_loss did not improve
Epoch 01238: val_loss did not improve
Epoch 01239: val_loss did not improve
Epoch 01240: val_loss did not improve
Epoch 01241: val_loss did not improve
Epoch 01242: val_loss did not improve
Epoch 01243: val_loss did not improve
Epoch 01244: val_loss did not improve
Epoch 01245: val_loss did not improve
Epoch 01246: val_loss did not improve
Epoch 01247: val_loss did not improve
Epoch 01248: val_loss did not improve
Epoch 01249: val_loss did not improve
Epoch 01250: val_loss did not improve
Epoch 01251: val_loss did not improve
Epoch 01252: val_loss did not improve
Epoch 01253: val_loss did not improve
Epoch 01254: val_loss did not improve
Epoch 01255: val_loss did not improve
Epoch 01256: val_loss did not improve
Epoch 01257: val_loss did not improve
Epoch 01258: val_loss did not improve
Epoch 01259: val_loss did not improve
Epoch 01260: val_loss did not improve
Epoch 01261: val_loss did not improve
Epoch 01262: val_loss did not improve
Epoch 01263: val_loss did not improve
Epoch 01264: val_loss did not improve
Epoch 01265: val_loss did not improve
Epoch 01266: val_loss did not improve
Epoch 01267: val_loss did not improve
Epoch 01268: val_loss did not improve
```

```
Epoch 01269: val_loss did not improve
Epoch 01270: val_loss did not improve
Epoch 01271: val_loss did not improve
Epoch 01272: val_loss did not improve
Epoch 01273: val_loss did not improve
Epoch 01274: val_loss did not improve
Epoch 01275: val_loss did not improve
Epoch 01276: val_loss did not improve
Epoch 01277: val_loss did not improve
Epoch 01278: val_loss did not improve
Epoch 01279: val_loss did not improve
Epoch 01280: val_loss did not improve
Epoch 01281: val_loss did not improve
Epoch 01282: val_loss did not improve
Epoch 01283: val_loss did not improve
Epoch 01284: val_loss did not improve
Epoch 01285: val_loss did not improve
Epoch 01286: val_loss did not improve
Epoch 01287: val_loss did not improve
Epoch 01288: val_loss did not improve
Epoch 01289: val_loss did not improve
Epoch 01290: val_loss did not improve
Epoch 01291: val_loss did not improve
Epoch 01292: val_loss did not improve
Epoch 01293: val_loss did not improve
Epoch 01294: val_loss did not improve
Epoch 01295: val_loss did not improve
Epoch 01296: val_loss did not improve
Epoch 01297: val_loss did not improve
Epoch 01298: val_loss did not improve
Epoch 01299: val_loss did not improve
Epoch 01300: val_loss did not improve
Epoch 01301: val_loss did not improve
Epoch 01302: val_loss did not improve
Epoch 01303: val_loss did not improve
Epoch 01304: val_loss did not improve
Epoch 01305: val_loss did not improve
Epoch 01306: val_loss did not improve
Epoch 01307: val_loss did not improve
Epoch 01308: val_loss did not improve
Epoch 01309: val_loss did not improve
Epoch 01310: val_loss did not improve
Epoch 01311: val_loss did not improve
Epoch 01312: val_loss did not improve
Epoch 01313: val_loss did not improve
Epoch 01314: val_loss did not improve
Epoch 01315: val_loss did not improve
Epoch 01316: val_loss did not improve
Epoch 01317: val_loss did not improve
Epoch 01318: val_loss did not improve
Epoch 01319: val_loss did not improve
Epoch 01320: val_loss did not improve
Epoch 01321: val_loss did not improve
Epoch 01322: val_loss did not improve
Epoch 01323: val_loss did not improve
Epoch 01324: val_loss did not improve
Epoch 01325: val_loss did not improve
```

```
Epoch 01326: val_loss did not improve
Epoch 01327: val_loss did not improve
Epoch 01328: val_loss did not improve
Epoch 01329: val_loss did not improve
Epoch 01330: val_loss did not improve
Epoch 01331: val_loss did not improve
Epoch 01332: val_loss did not improve
Epoch 01333: val_loss did not improve
Epoch 01334: val_loss did not improve
Epoch 01335: val_loss did not improve
Epoch 01336: val_loss did not improve
Epoch 01337: val_loss did not improve
Epoch 01338: val_loss did not improve
Epoch 01339: val_loss did not improve
Epoch 01340: val_loss did not improve
Epoch 01341: val_loss did not improve
Epoch 01342: val_loss did not improve
Epoch 01343: val_loss did not improve
Epoch 01344: val_loss did not improve
Epoch 01345: val_loss did not improve
Epoch 01346: val_loss did not improve
Epoch 01347: val_loss did not improve
Epoch 01348: val_loss did not improve
Epoch 01349: val_loss did not improve
Epoch 01350: val_loss did not improve
Epoch 01351: val_loss did not improve
Epoch 01352: val_loss did not improve
Epoch 01353: val_loss did not improve
Epoch 01354: val_loss did not improve
Epoch 01355: val_loss did not improve
Epoch 01356: val_loss did not improve
Epoch 01357: val_loss did not improve
Epoch 01358: val_loss did not improve
Epoch 01359: val_loss did not improve
Epoch 01360: val_loss did not improve
Epoch 01361: val_loss did not improve
Epoch 01362: val_loss did not improve
Epoch 01363: val_loss did not improve
Epoch 01364: val_loss did not improve
Epoch 01365: val_loss did not improve
Epoch 01366: val_loss did not improve
Epoch 01367: val_loss did not improve
Epoch 01368: val_loss did not improve
Epoch 01369: val_loss did not improve
Epoch 01370: val_loss did not improve
Epoch 01371: val_loss did not improve
Epoch 01372: val_loss did not improve
Epoch 01373: val_loss did not improve
Epoch 01374: val_loss did not improve
Epoch 01374: early stopping
```

Step 7: Visualize the Loss and Test Predictions

(IMPLEMENTATION) Answer a few questions and visualize the loss

Question 1: Outline the steps you took to get to your final neural network architecture and your reasoning at each step.

Answer: Started with a convolution hidden layer (16, 2, 2), Max Pooling and a dense hidden layer (100) with ReLU activations. I experimented with adding more convolutional layers (up to three with max pooling) but couldn't get much better performance. Increasing the number of filters on the convolution helped but only up to 32. Adding a second fully-connected layer also helped. I tried batch normalization but performance was poor and the model tended to be unstable with that. I tried L2 regularizion of activity, bias and kernel. Kernel performed better than the other two but not as well as using dropout. I tuned Dropout to get good consistent outputs. I switched from ReLU to ELU and found ELU provided better numbers. I tried MaxPooling size 3 but 2 worked better. I added checkpointing to select the best performing model and early stopping to get good performance without waiting too long.

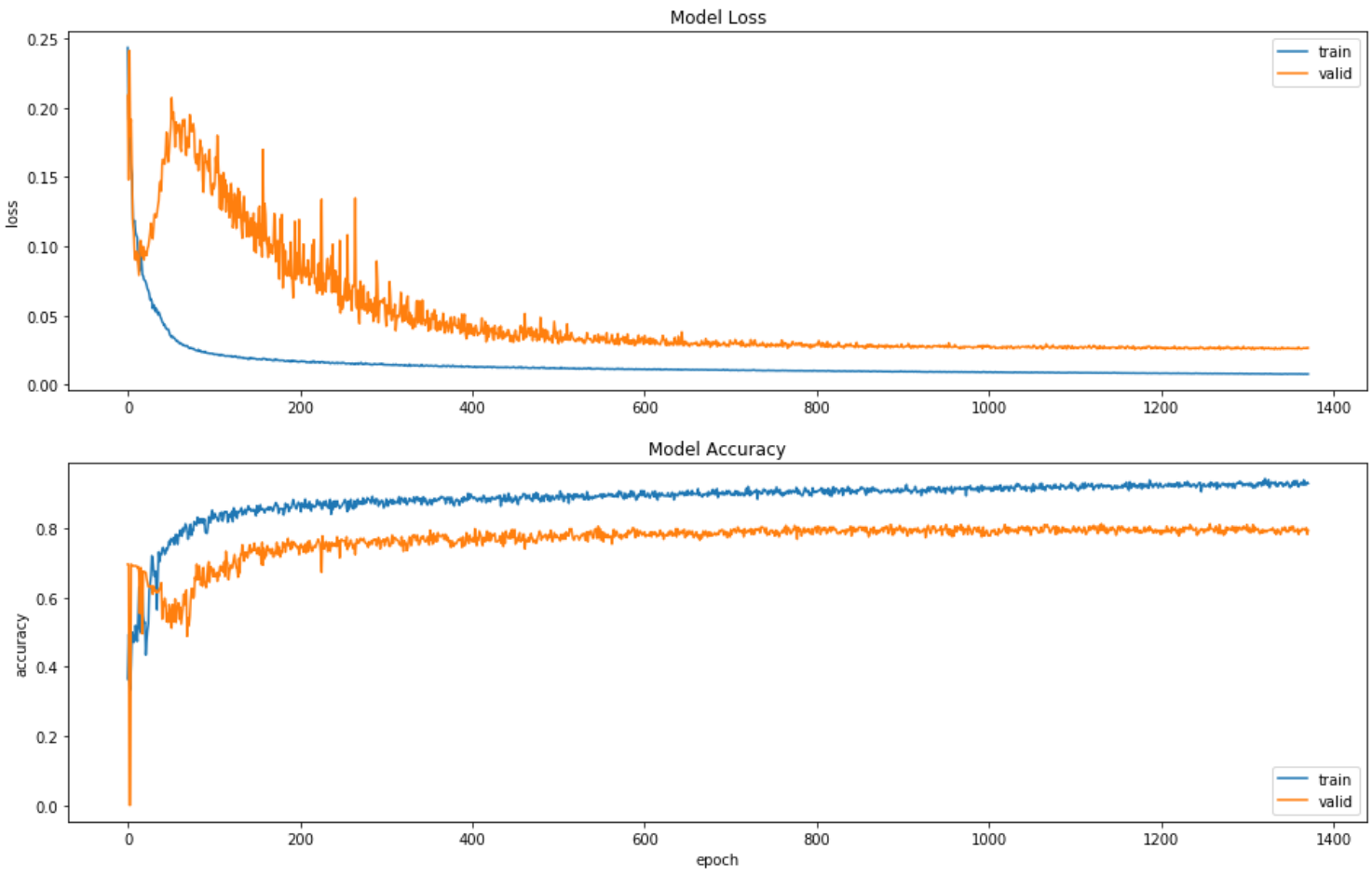
Question 2: Defend your choice of optimizer. Which optimizers did you test, and how did you determine which worked best?

Answer: I initially tuned SGD, trying different momentum and decay and the performance was OK. I then tried all the other optimizers with their default params against this. Most were comparable though some were unstable (adamax) with this model. I found Nadam provided the best combination of stability and performance. It was consistently producing better accuracy. I tried a few values of epsilon (high and low) and setled on 1e-4 as the best for this model.

Use the code cell below to plot the training and validation loss of your neural network. You may find this resource (http://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/) useful.

In [23]:

```
## TODO: Visualize the training and validation loss of your neural network
# summarize history for loss
# fig = plt.figure(figsize=(20,10))
# plt.plot(history.history['loss'])
# plt.plot(history.history['val_loss'])
# plt.title('model loss')
# plt.ylabel('loss')
# plt.xlabel('epoch')
# plt.legend(['train', 'validation'], loc='upper right')

# Plot the loss and accuracy
fig = plt.figure(figsize = (16,10))
ax1 = fig.add_subplot(211)
ax1.plot(history.history['loss'][4:])
ax1.plot(history.history['val_loss'][4:])
ax1.set_title('Model Loss')
plt.ylabel('loss')
plt.legend(['train', 'valid'], loc='upper right')

ax2 = fig.add_subplot(212)
ax2.plot(history.history['acc'][4:])
ax2.plot(history.history['val_acc'][4:])
ax2.set_title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='lower right')

plt.show()
```



Question 3: Do you notice any evidence of overfitting or underfitting in the above plot? If so, what steps have you taken to improve your model? Note that slight overfitting or underfitting will not hurt your chances of a successful submission, as long as you have attempted some solutions towards improving your model (such as *regularization, dropout, increased/decreased number of layers, etc*).

Answer: I saw significant overfitting in models with many layers and many units. I worked through several cycles of increasing complexity and trying different approaches to regularization. I found dropout worked best and keeping the network relatively simple also helped. Training loss suffers greatly if too much regularization is applied. The current model is slightly overfitting but accuracy is pretty good compared to other iterations in the design.

In [24]:

```
# Some stats that helped with the analysis
i = history.history['val_loss'].index(min(history.history['val_loss']))
j = history.history['val_acc'].index(max(history.history['val_acc']))
k = len(history.history['val_acc']) - 1
print("\n\nHighest Validation Accuracy\n==========================\nVal accur
acy: {}\nTrain accuracy: {}\nVal loss: {}\nTrain loss: {}\nepoch: {}".format(
    history.history['val_acc'][j], history.history['acc'][j], history.history[
'val_loss'][j],
    history.history['loss'][j], j))
print("\n\nLowest Validation Loss\n=====================\nVal accuracy: {}\nT
rain accuracy: {}\nVal loss: {}\nTrain loss: {}\nepoch: {}".format(
    history.history['val_acc'][i], history.history['acc'][i], history.history[
'val_loss'][i],
    history.history['loss'][i], i))
print("\n\nLast Record Processed\n====================\nVal accuracy: {}\nTra
in accuracy: {}\nVal loss: {}\nTrain loss: {}\nepoch: {}".format(
    history.history['val_acc'][k], history.history['acc'][k], history.history[
'val_loss'][k],
    history.history['loss'][k], k))
```

```
Highest Validation Accuracy
===========================
Val accuracy: 0.8130841154918492
Train accuracy: 0.9281542067215821
Val loss: 0.027190862414992866
Train loss: 0.008419174648751722
epoch: 1134


Lowest Validation Loss
======================
Val accuracy: 0.7897196272823298
Train accuracy: 0.9199766360710715
Val loss: 0.025360380029566934
Train loss: 0.008152678243328477
epoch: 1173


Last Record Processed
=====================
Val accuracy: 0.7943925267068025
Train accuracy: 0.929906543727233
Val loss: 0.026545740785860577
Train loss: 0.007615202694052012
epoch: 1374
```

Visualize a Subset of the Test Predictions

Execute the code cell below to visualize your model's predicted keypoints on a subset of the testing images.

In [25]:

```
y_test = model.predict(X_test)
fig = plt.figure(figsize=(20,20))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05
)
for i in range(9):
    ax = fig.add_subplot(3, 3, i + 1, xticks=[], yticks=[])
    plot_data(X_test[i], y_test[i], ax)
```

Step 8: Complete the pipeline

With the work you did in Sections 1 and 2 of this notebook, along with your freshly trained facial keypoint detector, you can now complete the full pipeline. That is given a color image containing a person or persons you can now

- Detect the faces in this image automatically using OpenCV
- Predict the facial keypoints in each face detected in the image
- Paint predicted keypoints on each face detected

In this Subsection you will do just this!

(IMPLEMENTATION) Facial Keypoints Detector

Use the OpenCV face detection functionality you built in previous Sections to expand the functionality of your keypoints detector to color images with arbitrary size. Your function should perform the following steps

1. Accept a color image.
2. Convert the image to grayscale.
3. Detect and crop the face contained in the image.
4. Locate the facial keypoints in the cropped image.
5. Overlay the facial keypoints in the original (color, uncropped) image.

Note: step 4 can be the trickiest because remember your convolutional network is only trained to detect facial keypoints in 96×96 grayscale images where each pixel was normalized to lie in the interval $[0, 1]$, and remember that each facial keypoint was normalized during training to the interval $[-1, 1]$. This means - practically speaking - to paint detected keypoints onto a test face you need to perform this same pre-processing to your candidate face - that is after detecting it you should resize it to 96×96 and normalize its values before feeding it into your facial keypoint detector. To be shown correctly on the original image the output keypoints from your detector then need to be shifted and re-normalized from the interval $[-1, 1]$ to the width and height of your detected face.

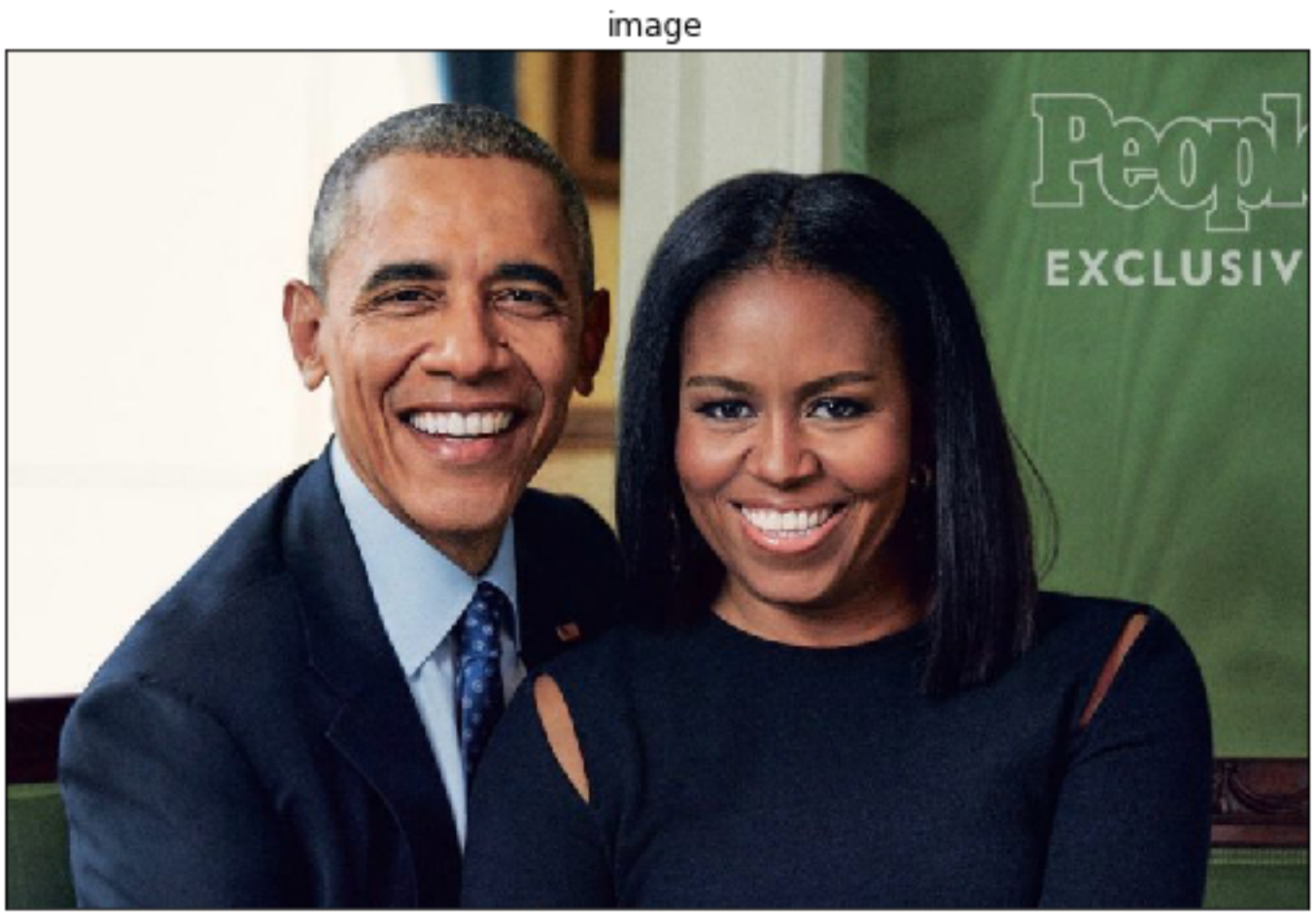
When complete you should be able to produce example images like the one below

In [26]:

```
# Load in color image for face detection
image = cv2.imread('images/obamas4.jpg')


# Convert the image to RGB colorspace
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


# plot our image
fig = plt.figure(figsize = (9,9))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])
ax1.set_title('image')
ax1.imshow(image)
```

Out[26]:

```
<matplotlib.image.AxesImage at 0x7f18b6c7dc18>
```



In [27]:

```
### TODO: Use the face detection code we saw in Section 1 with your trained co
nv-net
# Convert the RGB  image to grayscale
from utils import *
from keras.models import load_model

# Load the learned model for keypoint detection
model = load_model("my_model.h5")

gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# Extract the pre-trained face detector from an xml file
face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_front
alface_default.xml')

# Detect the faces in image
faces = face_cascade.detectMultiScale(gray, 1.25, 6)

# Make a copy of the orginal image to draw face keypoints on
image_with_keypoints = np.copy(image)

## TODO : Paint the predicted keypoints on the test image
face_images = []
for (x,y,w,h) in faces:
    face_image = cv2.resize(gray[y:y+h, x:x+w], (96, 96), interpolation = cv2.
INTER_CUBIC)
    face_images.append(face_image)

# Reshape to match shape to our training data
face_images = np.array(face_images).reshape(-1, 96, 96, 1) / 255

keypoints_list = model.predict(face_images)
for i, (x,y,w,h) in enumerate(faces):
    xy = np.tile(np.array([x , y]), 15)
    hw = np.tile(np.array([h / 2, w / 2]), 15)
    # Translate and scale out keypoints to match the photo
    keypoints_list[i] = np.round((keypoints_list[i] + 1) * hw) + xy

# Scatterplot keypoints over the photo (matplotlib > opencv for circles)
fig = plt.figure(figsize = (9,9))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])
ax1.set_title('image with keypoints')
ax1.imshow(image_with_keypoints)
for keypoints in keypoints_list:
    ax1.scatter(keypoints[0::2],
            keypoints[1::2],
            marker='o',
            c='greenyellow',
            s=5)
```

(Optional) Further Directions - add a filter using facial keypoints to your laptop camera

Now you can add facial keypoint detection to your laptop camera - as illustrated in the gif below.

The next Python cell contains the basic laptop video camera function used in the previous optional video exercises. Combine it with the functionality you developed for keypoint detection and marking in the previous exercise and you should be good to go!

In [28]:

```
import cv2
import time
from keras.models import load_model

# Load the learned model for keypoint detection
model = load_model("my_model.h5")
# load the face cascade
face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_front
alface_default.xml')

# Mostly lifted and shifted from previous section
def detect_features(frame):
    # Using gray for detection (as usual)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Needed for matplotlib ops below
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    faces = face_cascade.detectMultiScale(gray, 1.25, 6)

    face_images = []
    for (x,y,w,h) in faces:
```

```
        face_image = cv2.resize(gray[y:y+h, x:x+w], (96, 96), interpolation =
cv2.INTER_CUBIC)
        face_images.append(face_image)

    face_images = np.array(face_images).reshape(-1, 96, 96, 1) / 255

    keypoints_list = model.predict(face_images)
    for i, (x,y,w,h) in enumerate(faces):
        xy = np.tile(np.array([x , y]), 15)
        hw = np.tile(np.array([h / 2, w / 2]), 15)
        keypoints_list[i] = np.round((keypoints_list[i] + 1) * hw) + xy

    # Overlay the scatter plot on the picture
    # matplotlib makes nicer circles :)
    fig = plt.figure(figsize = (9,9))
    ax1 = fig.add_subplot(111)
    ax1.set_xticks([])
    ax1.set_yticks([])
    image_with_keypoints = ax1.imshow(rgb)
    for keypoints in keypoints_list:
        ax1.scatter(keypoints[0::2],
                keypoints[1::2],
                marker='o',
                c='greenyellow',
                s=5)

    # The rest of the code here converts from matplotlib format back to opencv
    # redraw the canvas
    fig.canvas.draw()

    # convert canvas to image
    img = np.fromstring(fig.canvas.tostring_rgb(), dtype=np.uint8, sep='')
    img  = img.reshape(fig.canvas.get_width_height()[::-1] + (3,))

    # img is rgb, convert to opencv's default bgr
    frame = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    plt.close()                                    # don't want matplotlib to pl
ot

    return frame

def laptop_camera_go():
    # Create instance of video capturer
    cv2.namedWindow("face detection activated")
    vc = cv2.VideoCapture(0)

    # Try to get the first frame
    if vc.isOpened():
        rval, frame = vc.read()
    else:
        rval = False

    # keep video stream open
    while rval:
        # plot image from camera with detections marked
        frame = detect_features(frame)
        cv2.imshow("face detection activated", frame)
```

```
        # exit functionality - press any key to exit laptop video
        key = cv2.waitKey(20)
        if key > 0 and key < 255: # exit by pressing any key
            # destroy windows
            cv2.destroyAllWindows()

            # hack from stack overflow for making sure window closes on osx --
> https://stackoverflow.com/questions/6116564/destroywindow-does-not-close-win
dow-on-mac-using-python-and-opencv
            for i in range (1,5):
                cv2.waitKey(1)
            return

        # read next frame
        time.sleep(0.05)             # control framerate for computation - def
ault 20 frames per sec
        rval, frame = vc.read()
```

In [29]:

```
# Run your keypoint face painter
laptop_camera_go()
```

(Optional) Further Directions - add a filter using facial keypoints

Using your freshly minted facial keypoint detector pipeline you can now do things like add fun filters to a person's face automatically. In this optional exercise you can play around with adding sunglasses automatically to each individual's face in an image as shown in a demonstration image below.

To produce this effect an image of a pair of sunglasses shown in the Python cell below.

In [30]:

```
# Load in sunglasses image - note the usage of the special option
# cv2.IMREAD_UNCHANGED, this option is used because the sunglasses
# image has a 4th channel that allows us to control how transparent each pixel
in the image is
sunglasses = cv2.imread("images/sunglasses_4.png", cv2.IMREAD_UNCHANGED)

# Plot the image
fig = plt.figure(figsize = (6,6))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])
ax1.imshow(sunglasses)
ax1.axis('off');
```

This image is placed over each individual's face using the detected eye points to determine the location of the sunglasses, and eyebrow points to determine the size that the sunglasses should be for each person (one could also use the nose point to determine this).

Notice that this image actually has *4 channels*, not just 3.

In [31]:

```
# Print out the shape of the sunglasses image
print ('The sunglasses image has shape: ' + str(np.shape(sunglasses)))
```

```
The sunglasses image has shape: (1123, 3064, 4)
```

It has the usual red, blue, and green channels any color image has, with the 4th channel representing the transparency level of each pixel in the image. Here's how the transparency channel works: the lower the value, the more transparent the pixel will become. The lower bound (completely transparent) is zero here, so any pixels set to 0 will not be seen.

This is how we can place this image of sunglasses on someone's face and still see the area around of their face where the sunglasses lie - because these pixels in the sunglasses image have been made completely transparent.

Lets check out the alpha channel of our sunglasses image in the next Python cell. Note because many of the pixels near the boundary are transparent we'll need to explicitly print out non-zero values if we want to see them.

In [32]:

```
# Print out the sunglasses transparency (alpha) channel
alpha_channel = sunglasses[:,:,3]
print ('the alpha channel here looks like')
print (alpha_channel)

# Just to double check that there are indeed non-zero values
# Let's find and print out every value greater than zero
values = np.where(alpha_channel != 0)
print ('\n the non-zero values of the alpha channel look like')
print (values)
```

```
the alpha channel here looks like
[[0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 ...,
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]]

 the non-zero values of the alpha channel look like
(array([  17,   17,   17, ..., 1109, 1109, 1109]), array([ 687,  6
88,  689, ..., 2376, 2377, 2378]))
```

This means that when we place this sunglasses image on top of another image, we can use the transparency channel as a filter to tell us which pixels to overlay on a new image (only the non-transparent ones with values greater than zero).

One last thing: it's helpful to understand which keypoint belongs to the eyes, mouth, etc. So, in the image below, we also display the index of each facial keypoint directly on the image so that you can tell which keypoints are for the eyes, eyebrows, etc.

With this information, you're well on your way to completing this filtering task! See if you can place the sunglasses automatically on the individuals in the image loaded in / shown in the next Python cell.

In [33]:

```
# Load in color image for face detection
image_bgr = cv2.imread('images/obamas4.jpg')

# Convert the image to RGB colorspace
image = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)


# Plot the image
fig = plt.figure(figsize = (8,8))
ax1 = fig.add_subplot(111)
ax1.set_xticks([])
ax1.set_yticks([])
ax1.set_title('Original Image')
ax1.imshow(image)
```

Out[33]:

```
<matplotlib.image.AxesImage at 0x7f193ba785f8>
```

In [34]:

```
## (Optional) TODO: Use the face detection code we saw in Section 1 with your
trained conv-net to put
## sunglasses on the individuals in our test image

# Blend transparent from Anthony Budd:
# https://stackoverflow.com/questions/40895785/using-opencv-to-overlay-transpa
rent-image-onto-another-image
def blend_transparent(face_img, overlay_t_img):
    # Split out the transparency mask from the colour info
    overlay_img = overlay_t_img[:,:,:3] # Grab the BRG planes
    overlay_mask = overlay_t_img[:,:,3:]  # And the alpha plane

    # Again calculate the inverse mask
    background_mask = 255 - overlay_mask
```

```
    # Turn the masks into three channel, so we can use them as weights
    overlay_mask = cv2.cvtColor(overlay_mask, cv2.COLOR_GRAY2BGR)
    background_mask = cv2.cvtColor(background_mask, cv2.COLOR_GRAY2BGR)

    # Create a masked out face image, and masked out overlay
    # We convert the images to floating point in range 0.0 - 1.0
    face_part = (face_img * (1 / 255.0)) * (background_mask * (1 / 255.0))
    overlay_part = (overlay_img * (1 / 255.0)) * (overlay_mask * (1 / 255.0))

    # And finally just add them together, and rescale it back to an 8bit integ
er image
    return np.uint8(cv2.addWeighted(face_part, 255.0, overlay_part, 255.0, 0.0
))

# Convert the RGB  image to grayscale
from utils import *
from keras.models import load_model

# Load the learned model for keypoint detection
model = load_model("my_model.h5")

def wear_glasses(image_bgr, glasses):

    gray = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2GRAY)

    # Extract the pre-trained face detector from an xml file
    face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_f
rontalface_default.xml')

    # Detect the faces in image
    faces = face_cascade.detectMultiScale(gray, 1.25, 6)

    face_images = []
    xy = []
    wh = []
    for (x,y,w,h) in faces:
        face_image = cv2.resize(gray[y:y+h, x:x+w], (96, 96), interpolation =
cv2.INTER_CUBIC)
        face_images.append(face_image)
        # We'll use xy later to translate the eye (and brow) points
        xy.append([[x, y]] * 4)
        # We'll use wh later to scale distances between the eye (and brow) poi
nts
        wh.append([[w, h]] * 4)
    xy = np.array(xy)
    wh = np.array(wh)

    # Usa a similar shape to what we trained with
    face_images = np.array(face_images).reshape(-1, 96, 96, 1) / 255

    # Predict keypoints
    keypoints_by_face = model.predict(face_images)

    # Get unscaled eye points from all keypoints
    eyepoints_by_face = [[[keypoints[i], keypoints[i + 1]]
                          for i in [18, 2, 0, 14]]              # our eyepoi
nts are keypoints 9, 1, 0, 7
```

```
                            for keypoints in keypoints_by_face]     # two coordi
nates per point

    # Scale and translate the eyepoints to the full photo
    eyepoints_by_face = (np.array(eyepoints_by_face) + 1) / 2 * wh + xy

    # Define bounding box as sunglasses starting x, y and width and height
    bbox = []
    for eyepoints in eyepoints_by_face:
        sx = int(eyepoints[0][0])
        sy = int(eyepoints[0][1])
        sw = int(eyepoints[3][0] - eyepoints[0][0])
        # Maintain the aspect ration of the sunglasses
        # Could widen and narrow with head turns but it'll probably jitter lik
e the perspective did
        sh = int(sw * glasses.shape[0] / glasses.shape[1])
        bbox.append([sx, sy, sw, sh])

    image_with_glasses = image_bgr.copy()

    # Overlay sunglasses
    for sx, sy, sw, sh in bbox:
        image_with_glasses[sy:sy+sh, sx:sx+sw] = blend_transparent(image_with_
glasses[sy:sy+sh, sx:sx+sw], cv2.resize(glasses, (sw, sh)))

    return image_with_glasses

## Tried a perspective transform but it was too jittery for video
## Combining optical flow with perspective would probably produce very good re
sults
## ... a project for after the course ;)
#         image_with_glasses[y:y+h, x:x+w] = blend_transparent(image_with_glas
ses[y:y+h, x:x+w], cv2.resize(sunglasses, (w, h)))
#     # refpoints = np.float32([[75, 161], [155, 201], [352, 201], [434, 166]]
)
#     refpoints = np.float32([[98, 171], [170, 214], [362, 214], [445, 170]])

#     sg_512 = cv2.resize(sunglasses, (400,147))
#     sg_ref = np.zeros((512,512,4), np.uint8)
#     sg_ref[145:292, 56:456] += sg_512


#     image_with_glasses = image_bgr.copy()
#     for i, (x,y,w,h) in enumerate(faces):
#         M = cv2.getPerspectiveTransform(refpoints, eyepoints_by_face[i])
#         sg_dest = cv2.warpPerspective(sg_ref, M, (512, 512))
#         image_with_glasses[y:y+h, x:x+w] = blend_transparent(image_with_glas
ses[y:y+h, x:x+w], cv2.resize(sg_dest, (w, h)))


image_with_glasses = wear_glasses(image_bgr, sunglasses)
# Convert the image to RGB colorspace
# Won't need this for the video
image_with_glasses = cv2.cvtColor(image_with_glasses, cv2.COLOR_BGR2RGB)

# Display our image
fig = plt.figure(figsize = (9,9))
```

```
ax1 = fig.add_subplot(111)

ax1.set_xticks([])
ax1.set_yticks([])
ax1.set_title('image with glasses')
ax1.imshow(image_with_glasses)
```

Out[34]:

```
<matplotlib.image.AxesImage at 0x7f17897c7630>
```

(Optional) Further Directions - add a filter using facial keypoints to your laptop camera

Now you can add the sunglasses filter to your laptop camera - as illustrated in the gif below.

The next Python cell contains the basic laptop video camera function used in the previous optional video exercises. Combine it with the functionality you developed for adding sunglasses to someone's face in the previous optional exercise and you should be good to go!

In [35]:

```
import cv2
import time
from keras.models import load_model
import numpy as np

def laptop_camera_go():
    # Create instance of video capturer
    cv2.namedWindow("face detection activated")
    vc = cv2.VideoCapture(0)

    # try to get the first frame
    if vc.isOpened():
        rval, frame = vc.read()
    else:
        rval = False

    # Keep video stream open
    while rval:
        # Plot image from camera with detections marked
        ### Put glasses on the video image ###
        frame = wear_glasses(frame, sunglasses)
        cv2.imshow("face detection activated", frame)

        # Exit functionality - press any key to exit laptop video
        key = cv2.waitKey(20)
        if key > 0 and key < 255: # exit by pressing any key
            # Destroy windows
            cv2.destroyAllWindows()

            for i in range (1,5):
                cv2.waitKey(1)
            return

        # Read next frame
        time.sleep(0.05)             # control framerate for computation - def
ault 20 frames per sec
        rval, frame = vc.read()
```

In [36]:

```
# Load facial landmark detector model
model = load_model('my_model.h5')

# Run sunglasses painter
laptop_camera_go()
```