Emotion Detection at the Edge

Kathryn Papandrew, Angshuman Paul, Guangyu (Gary) Pei, Kathleen Wang

Abstract

Those with Autism Spectrum Disorder (ASD) have difficulty with social communication and interaction, particularly detecting and classifying an emotion of the person with whom they're interacting and reacting appropriately. Having the ability to receive information about the emotion of others can equip them with the tools to react most effectively. In this project, we built on open source facial emotion classification to bring it to the edge. The architecture of our solution allows inference to be performed at the edge with the images captured by a camera and has those images messaged back to the cloud to continue refining the model and to provide a user web interface.

I. Introduction

A core to diagnosis of ASD is the exhibition of a difficulty in identifying and responding to others' emotions or state of mind indicated by their facial expression. There have been multiple studies about how to equip those with ASD with the ability to identify the emotion of another and act accordingly. These studies have yielded mixed results, seeming to have identified parts of empathy that can be taught and parts that cannot. For studies that are trying to provide information to the user regarding the emotion a person based on their facial expression, they would benefit greatly from having the classification of that emotion automated where the dissemination of that information is flexible.

In this project, we built on open source projects that have used the Facial Expression Recognition (FER) dataset to build a model that can identify a variety of emotions (anger, disgust, fear, happiness, sadness, surprise and neutral) based on a facial expression. Using a camera on an edge device with compute power, we are able to perform inference on facial images at the edge and use a message protocol (MQTT) to transfer data to the cloud for both a web interface for the user and to continue to refine the model. We believe this architecture to be useful in the aforementioned research, and as deemed appropriate by the results of those studies, potentially in everyday life as an aid in everyday interactions.

II . System Architecture and Software Setup

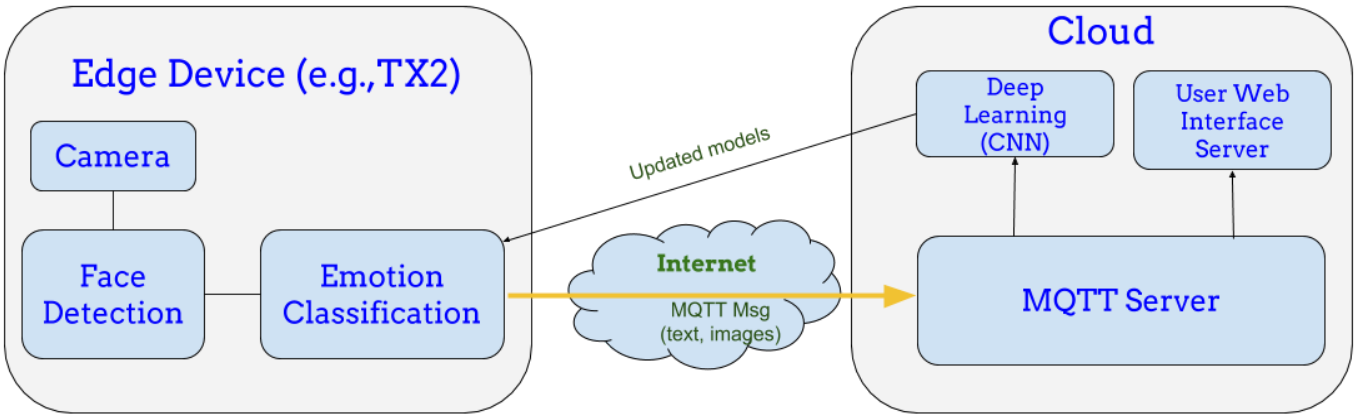

Figure 1: End-to-end emotion detection design

Figure 1 shows our end-to-end design of emotion detection system. The video data source from camera is captured by `OpenCV` and preprocessed before they are applied with face detection and emotion classification. The preprocess steps include reshape, normalization and convert to grayscale. `Keras` framework is used for both deep learning and interference.

Edge Software

The following `python3` packages are required.

- `keras`: Keras framework used to create deep learning (CNN) models.
- `tensorflow`: backend used by keras.
- `Pillow`: Python image library required for image preprocessing.
- `pandas`: supporting functions for image processing.
- `numpy`: supporting functions for image processing.
- `opencv-python (3.4.0)`: providing functions for capturing video stream.
- `statistics`: supporting functions for image processing.
- `h5py`: supporting functions for keras.
- `paho-mqtt`: interface for using MQTT.
- `imageio`: optional interface for reconstructing image data read from MQTT topic.
- `mosquitto & mosquitto-clients`: optional for command line subscription to MQTT

Most of these packages can be installed via `pip3 install` command except `tensorflow`, `opencv-python` and the command line `mosquitto` and `mosquitto-clients`. To install `tensorflow`, instructions for Jetson TX2 from Nvidia for `python3` version should be followed. The installation of opencv-python requires multiple steps and we found that these instructions are adequate. The installation of the command line `mosquitto` will depend on the OS of the machine

being used (e.g. `brew install` for Mac, `apt-get install` for ubuntu and `snap install` for most of the other linux distributions). The list of full `python3` packages is provided in Appendix A.1.

For the emotion classification, we leveraged this CNN implementation. However, we needed to update the implementation to the current `Pillow` interface. More specifically, updates such as the following are needed. All updates and additions can be found in this repository.

```
def load_image(image_path, grayscale=False, color_mode='rgb', target_size=None):
    pil_image = image.load_img(image_path, grayscale, color_mode, target_size,)
    return image.img_to_array(pil_image)
```

To train the emotion classifier, user needs to update training data set path and run the following command.

```
python3 src/train_emotion_classifier.py
```

To demonstrate emotion classification with video feed and stream the results to MQTT server, the following script is used for any pre-recorded video file.

```
python3 src/video_playback_emotion_demo.py -f <path_to_video_file>
```

We also encountered issues of python3-tk on TX2 and program failed with error illustrated below.

```
ImportError: No module named '_tkinter'
```

There are two workarounds. The first is to set backend in `python3` as shown below.

```
import matplotlib
matplotlib.use('agg')
```

Alternatively, user can specify in `matplotlib` configuration file as illustrated below.

```
echo "backend : Agg" > /home/nvidia/.config/matplotlib/matplotlibrc
```

III. Data and Model

For detection on edge, we need two models: one to capture face images from video and two identify the emotion of the captured face image. For the first task, we use the commonly used openCV face detection module as mentioned above. The second task was less straightforward and was the main focus of this project.

In order to train our models to accurately detect emotions, we need a large number of accurately labeled images. Much work has been done in the industry and many open source data are available. For this project, we explored a few state of the art algorithms and data sets and compared two algorithms in depth.

Data

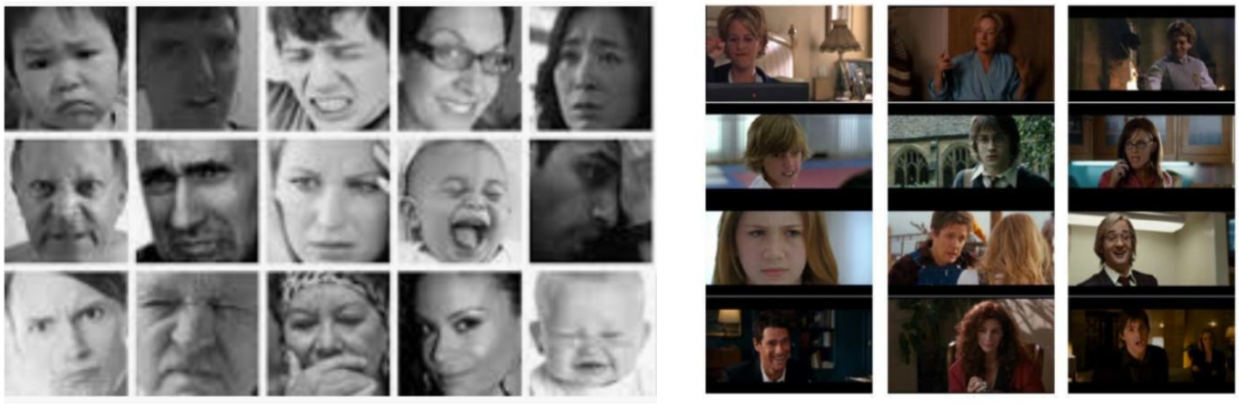

Figure 2: Left FER2013, Right SFEW

Even though images with faces are prevalent, not all of them are available or suitable for training purposes. To further complicate the problem, detecting emotions from still images is quite different from detecting emotions from real life videos. As a result, the state of the art algorithms, and dataset, for emotion detection is separated into two sections: images and videos.

Still Images

Quite a few still image datasets are publicly available. Some of these datasets were created in the lab environment where a person was acting out a particular emotion (E.g. The Extended CohnKanade (CK+) database)[1] (. Some were scraped from the internet and annotated by either real person or even by algorithms (E.g. EmotioNet with one million images).

FER2013

One of the most frequently used dataset is the FER2013 dataset which was launched in 2013[2]. It contains 28,709 training images, 3,589 validation images and 3,589 test images with seven expression labels (anger, disgust, fear, happiness, sadness, surprise and neutral). They images are grayscale and are sourced by Google image search API. Our team chose this dataset to be train our main models because:

1. It has relatively large scale
2. Images are realistic, not acted
3. It has been well studied and many excellent models are available publicly

[1] *Deep Facial Expression Recognition: A Survey, Shan Li and Weihong Deng* https://arxiv.org/abs/1804.08348
[2] https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge

Videos

Training image recognition models based on video clips can be very challenging. In addition to frame by frame facial expressions, we have an additional dimension that captures the changes in each frame. However this change is harder to capture than it seems because facial features are not perfectly aligned in consecutive frames, especially for videos obtained in real life settings. Furthermore, there could be various degrees of emotions in each clip that one label does not accurately capture all of them.

For on-edge detection, we have decided to train our model based still images to make our models light and fast. However, we did use Acted Facial Expressions in the Wild (AFEW) and Static Facial Expressions in the Wild (SFEW) open source data to validate our models (data described below).

AFEW and SFEW

The Acted Facial Expressions in the Wild (AFEW) database contains short video clips from different movies. It contains different environment conditions in both audio and video, great for training generalized models. The Static Facial Expressions in the Wild (SFEW) contains selected static frames from AFEW. Our team used both AFEW and SFEW in testing our models to make sure they are generalizable.

Model

Our baseline model is the out-of-the-box `Inception_V3` model. Our final model is one proposed by Octavio Arriaga, Matias Valdenegro-Toro and Paul Plöger[3]. Compared to Inception, our final model is lightweight but more accurate. When comparing models, we used only the Training set of the FER2013 data and used 20% of the training data for valuation purpose.

Baseline: InceptionV3

We utilize transfer learning as we used `Inception_V3` pre-trained on the Imagenet. The model contains 42 layers and more than 20 millions of parameters. We use the weights as trained on Imagenet and only unfreeze the last classification layer in the first round of training. Unfortunately the pre-training on imagenet does not seem to capture the details in facial emotions as our simple transfer learning model reached training accuracy of 39% and valuation accuracy of 32%..

We further improved the model accuracy by refining more weights. We unfreeze two more layers and retrain our previous model with the new final layer. The accuracy improved significantly with this refinement and reached 71% for training and 51%.

---

[3] *Real-time Convolutional Neural Networks for Emotion and Gender Classification: Octavio Arriaga, Matias Valdenegro-Toro, Paul Plöger Octavio Arriaga, Matias Valdenegro-Toro, Paul Plöger*

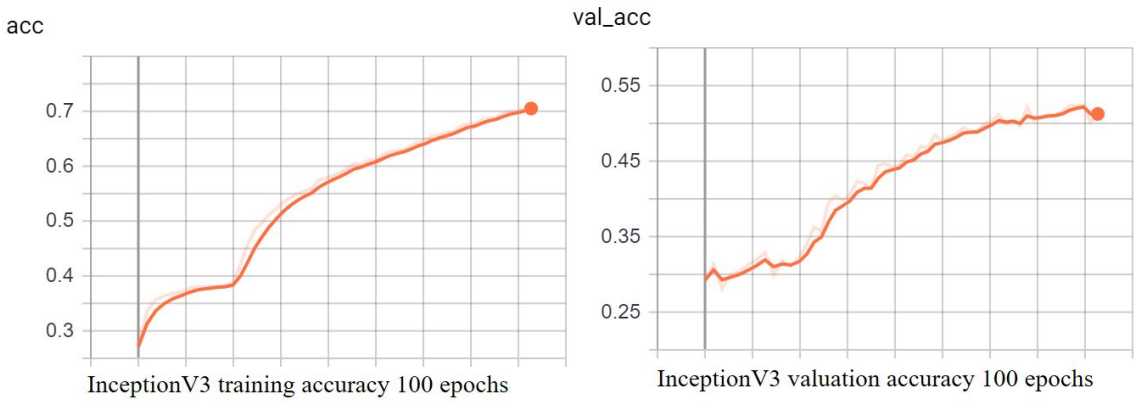

Figure 3: Training and validation accuracy results for InceptionV3

Edge model - *Mini-Xception*

Our final model is the Mini-Xception proposed by Octavio Arriaga, Matias Valdenegro-Toro and Paul Plöger[4]. It contains less than 20 layers and about 60,000 parameters. It is inspired by Xception CNN architecture and contains 4 residual depth-wise separable convolutions. Each convolution is followed by a batch normalization and ReLU activation function.

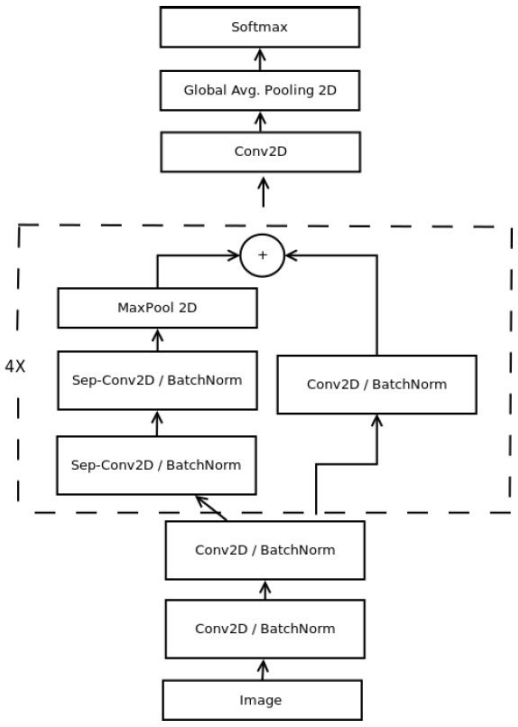

Figure 4: Mini-Xception model structure

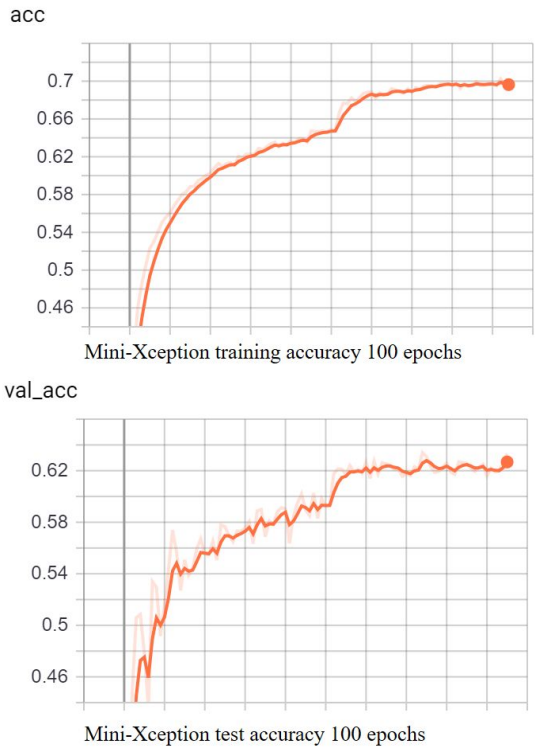

Figure 5: Accuracy for Mini-Xception

[4] *Real-time Convolutional Neural Networks for Emotion and Gender Classification: Octavio Arriaga, Matias Valdenegro-Toro, Paul Plöger Octavio Arriaga, Matias Valdenegro-Toro, Paul Plöger*

Unlike inception net, this model, as pictured on the left, was designed for facial emotion detection on edge. As a result, it is much faster and lighter. The accuracy after running 100 epochs is 68% for training and 62% for valuation. There is much less overfitting.

Our final detection model is trained using the Mini-Xception model but on the entire FER2013 training and valuated on the actual FER2013 valuation dataset.

IV. Detection on Edge/MQTT

We used the Message Queuing Telemetry Transport (MQTT) protocol to publish the output from the classification to the cloud and to subscribe to the message queue to retrieve the messages in the cloud. This was achieved using the [python package from the Eclipse Paho Project](#).

Publish

We followed a twofold approach to publish the messages:

Publish Classified Image - The classified image with bounding boxes was encoded into a memory buffer using the `cv2.imencode` method. This was further base-64 encoded and published to an MQTT topic so that the entire classified image would be available on subscribing to this topic.

```
encode_im = cv2.imencode(".jpg", rgb_image)[1].tostring()
encode64 = base64.b64encode(encode_im)
publish.single(topic="fc_img_mqtt", payload=encode64, hostname="50.23.173.22")
```

Publish Classification JSON - The classified emotion and the associated probability along with additional details including the video file name, frame number and timestamp was also published in the form of a JSON payload so that this data could be accessed in the cloud and used for further training and learning.

```
str_classify = "{video_file: %s, frame_number: %s, emotion_probability: %s,
emotion_text: %s, timestamp: %s}"%(video_file, frame_count, emotion_probability,
emotion_text, str_ts)
publish.single(topic="fc_json_mqtt", payload=encode64, hostname="50.23.173.22")
```

The publish code chunks shown above were added to the `video_playback_emotion_demo.py` script which we used to demonstrate the emotion classification. The messages keep getting published to the topics as the images are classified by the model in the script.

Subscribe

We installed the MQTT server in a virtual machine in IBM Cloud and published the classification messages from the Jetson TX2 to this MQTT server. A separate virtual machine in IBM Cloud was set up with apache web server to act as the web interface for making the classified images accessible in the cloud. A python program running on this virtual machine was used to subscribe to the MQTT topic, reconstruct the classified image along with the bounding boxes and classification and save the reconstructed image to the apache root folder from where it was picked up and displayed in a web page.

```
img = imread(io.BytesIO(base64.b64decode(msg.payload)))
cv2_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
cv2.imwrite("image.jpg", cv2_img)
```

To subscribe to the MQTT topic used for publishing the classified image, the following command is run from the apache root folder:

```
python3 mqtt-subs-img.py
```

The following command can be used to access the JSON message from any cli which has the mosquitto-clients installed:

```
mosquitto_sub -h <50.23.173.22> -t fc_json_mqtt
```

Securing MQTT messages

We used basic cv2 and base-64 encoding for our project but for actual implementations we need to secure the classification data that is published to MQTT. We can do this by enabling SSL encryption for the MQTT messages and making changes to the config files to use the relevant SSL certificates. As an added layer of security, we can also configure the MQTT messages to use passwords. This is very easy to configure in most of the MQTT packages. For example, Mosquitto includes a utility to generate a special password file called `mosquitto_passwd` and we can configure the Mosquitto `default.conf` file to require passwords for publishing and subscribing.

V. Results

We used FER2013 dataset for the training and validation. The training and validation split is 80% vs 20%. FER2013 dataset has total 35,887 labeled images. The number of layers of our deep learning CNN model is 33. The total number of trainable parameters is 641,463. The detailed CNN model is provided in Appendix A.2. The batch size is 32 and number of steps is 897 per epoch. The training program uses data augmentation as following:

```
rotation_range=10,
width_shift_range=0.1,
height_shift_range=0.1,
zoom_range=.1,
horizontal_flip=True
```

The Keras model training framework allows the training to stop automatically when there is no more improvement in terms of accuracy and reduction of loss of the objective function. Figure 2 and Figure 3 show the loss and accuracy versus epochs respectively.

The improvement of validation accuracy levels off around 120 epochs. The final accuracy achieves about 62%.

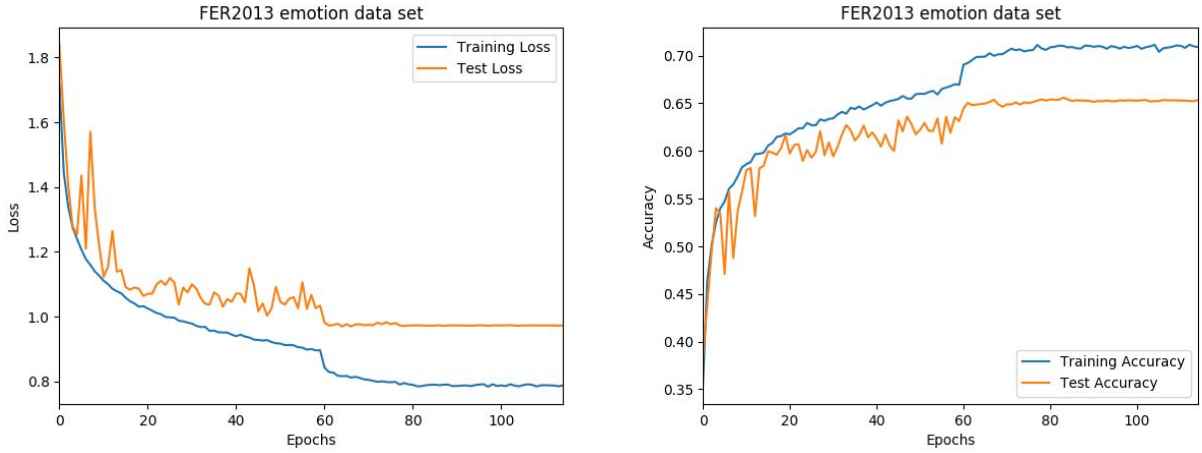

Figure 6: Left: Training and validation loss. Right: Training and validation accuracy

VI. Discussion

Overall, our accuracy was not as high as expected. In continuing this effort, we would refine the model, including adding additional pre-processing and adding additional layers to the model for possibly greater predictive power. Additionally, we would want to do more analysis about the representation in our model of different genders, races, and ethnicities. Our model will only infer as well as we train it, and we would want to make sure we are creating as unbiased a model as possible.

We see the architecture of our solution, particularly the messaging functionality, as incredibly useful. We can continue to gather data with the power of the edge, allow for inference in close to real time for a user. The data engineering allows for flexibility in how the solution is applied and be used easily in the lab for research, and depending on results of research, easily extended to the interactions in broader society. Finally, it can scale well with respect to the publisher-subscriber model.

Appendix

A.1 Reference list of python packages

```
Package             Version
------------------- ------------
absl-py             0.7.0
astor               0.7.1
cycler              0.10.0
cv2                 3.4.0
docutils            0.14
gast                0.2.2
grpcio              1.19.0
h5py                2.9.0
Keras               2.2.4
Keras-Applications  1.0.7
Keras-Preprocessing 1.0.9
kiwisolver          1.0.1
Markdown            3.0.1
matplotlib          3.0.3
numpy               1.16.1
paho-mqtt           1.4.0
pandas              0.24.1
Pillow              5.4.1
pip                 19.0.3
protobuf            3.7.0
pyparsing           2.3.1
python-dateutil     2.8.0
pytz                2018.9
PyYAML              3.13
scipy               1.2.1
setuptools          39.1.0
six                 1.12.0
statistics          1.0.3.5
tensorboard         1.9.0
tensorflow-gpu      1.9.0+nv18.8
termcolor           1.1.0
Werkzeug            0.14.1
```

```
wheel               0.33.1
```

A.2 Emotion classification deep learning CNN model

```
Layer (type)                 Output Shape              Param #
=================================================================
image_array (Conv2D)         (None, 48, 48, 16)        800
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 16)        64
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 48, 48, 16)        12560
_________________________________________________________________
batch_normalization_2 (Batch (None, 48, 48, 16)        64
_________________________________________________________________
activation_1 (Activation)    (None, 48, 48, 16)        0
_________________________________________________________________
average_pooling2d_1 (Average (None, 24, 24, 16)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        12832
_________________________________________________________________
batch_normalization_3 (Batch (None, 24, 24, 32)        128
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 32)        25632
_________________________________________________________________
batch_normalization_4 (Batch (None, 24, 24, 32)        128
_________________________________________________________________
activation_2 (Activation)    (None, 24, 24, 32)        0
_________________________________________________________________
average_pooling2d_2 (Average (None, 12, 12, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
batch_normalization_5 (Batch (None, 12, 12, 64)        256
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
batch_normalization_6 (Batch (None, 12, 12, 64)        256
```

```
_________________________________________________________________
activation_3 (Activation)    (None, 12, 12, 64)        0
_________________________________________________________________
average_pooling2d_3 (Average (None, 6, 6, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 6, 6, 128)         73856
_________________________________________________________________
batch_normalization_7 (Batch (None, 6, 6, 128)         512
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 6, 6, 128)         147584
_________________________________________________________________
batch_normalization_8 (Batch (None, 6, 6, 128)         512
_________________________________________________________________
activation_4 (Activation)    (None, 6, 6, 128)         0
_________________________________________________________________
average_pooling2d_4 (Average (None, 3, 3, 128)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 3, 3, 128)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 3, 3, 256)         295168
_________________________________________________________________
batch_normalization_9 (Batch (None, 3, 3, 256)         1024
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 3, 3, 7)           16135
_________________________________________________________________
global_average_pooling2d_1 ( (None, 7)                 0
_________________________________________________________________
predictions (Activation)     (None, 7)                 0
=================================================================
Total params: 642,935
Trainable params: 641,463
Non-trainable params: 1,472
_________________________________________________________________
```