Emotion Recognition Neural Network

Final Project Paper for DATA622: Machine Learning & Big Data

Daniel Dittenhafer

December 18, 2016

1 Overview

As part of the course requirements for Machine Learning & Big Data (DATA622), the problem of developing a machine learning model capable of classifying human face images into one of various emotion classes was presented. Specifically, the goal was to design and train a neural network to recognize the following 8 emotions and emit a probability of each of the emotion classes given the input 2 dimensional (2D) image.

1. anger
2. contempt
3. disgust
4. fear
5. happiness
6. neutral
7. sadness
8. surprise

Although this problem is not new to machine learning and solutions already exist (Microsoft Corporation, 2016), the complexities of the mathematics alone are challenging. This, combined with the application of a variety of neural network techniques, make for many unique solutions. Much of the academic theory for the neural network was derived from the text by Ian Goodfellow, et al, *Deep Learning* (Goodfellow, Bengio, and Courville, 2016).

Finally, the result of the project was a Kaggle competition where the class tested their models against an unlabeled (and less clean) data set (Rowe, 2016).

2 Techniques

The DATA622 class as a whole, under Professor Rowe's guidance, agreed to a general approach using the *Labeled Faces in the Wild* face database as a basis for training data (Huang, Ramesh, Berg, and Learned-Miller, 2007). We then labeled each of the included images using the Microsoft Emotion API. This was considered the "gold standard" for our training purposes and shared amongst the class as well as publicly in the facial_expressions repository on GitHub (Rowe, 2016).

In order to increase the size of our training and test data sets as well as apply regularization via data, each member of the class created ten image transformations using a variety of APIs and shared the transformations with the class as options for each of us when augmenting our data sets. The following code shows an example of a transformation applied during training (Joshi, 2015):

```
def cvBlurMotion1(img):

    size = 15
    kernel_motion_blur = np.zeros((size, size))
    kernel_motion_blur[int((size - 1) / 2), :] = np.ones(size)
    kernel_motion_blur = kernel_motion_blur / size

    img2 = cv2.filter2D(img, -1, kernel_motion_blur)
    return img2
```

As it turns out, Keras, a deep learning library for Theano and TensorFlow, provides an image transformation class, `ImageDataGenerator` (Chollet, 2015). This class takes source images as input and performs randomized transformations (within specified bounds). The following function wraps a call to the `ImageDataGenerator` class's flow function. This function was used to generate twelve additional transformed images per source image during the training process.

```
def imageDataGenTransform(img, y):

    # Using keras ImageDataGenerator to generate random images
    datagen = ImageDataGenerator(
        featurewise_std_normalization=False,
        rotation_range = 20,
        width_shift_range = 0.10,
        height_shift_range = 0.10,
        shear_range = 0.1,
        zoom_range = 0.1,
        horizontal_flip = True)

    x = img.reshape(1, 1, img.shape[0], img.shape[1])
    j = 0
    for imgT, yT in datagen.flow(x, y, batch_size = 1, save_to_dir = None):
        img2 = imgT
        break

    return img2
```

The functions, above, along with all code developed as part of this project can be found in the emotional-faces repository on GitHub.

The initial model was based on a model shared on Kaggle as part of a digit recognition competition (Majumdar, 2016). In general, it used the Keras library with 3 convolutional layers plus 2 dense layers including rectified linear unit activation and a softmax final activation for the class probabilities. This initial model was adjusted in minor ways, with followup training and validation to measure performance.

The final model was very similar to the initial model in many ways. It still relied on convolutional layers at the input and hidden layers with 2 dense layers on the end, but had a forth convolutional hidden layer internally.

```
def emotion_model_jh_v5(outputClasses, input_shape=(1, 150, 150), verbose=False):
    model = Sequential()
    model.add(Convolution2D(32, 8, 8, input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Convolution2D(32, 5, 5))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Convolution2D(64, 3, 3))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Convolution2D(64, 2, 2))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(64))
    model.add(Activation('relu'))
```

```
    model.add(Dense(outputClasses))
    model.add(Activation('softmax'))

    if verbose:
        print (model.summary())

    model.compile(loss='binary_crossentropy',
                  optimizer='rmsprop',
                  metrics=['accuracy'])

    return model
```

As you can see, the structure is approaching "deep" with the 4 hidden layers. Importantly, the convolutional kernel window begins larger (8x8) on the 150x150 input image, and gradually narrows through the remaining 3 convolutional layers. Binary cross entropy was used for the loss function, and Root Mean Square Propagation (RMSProp) was used for the optimization. RMSProp was used for its ability to consistently descend toward the minimum though admittedly not the fastest to arrive (Reid, 2014). Kind of a tortise approach in a tortise vs hare analogy.

Max Pooling was used between convolutional layers to further strengthen the model's resilience to variance in the input images in terms of the exact position of important features (eyes, mouth, etc).

Flattening brings us to a single data dimension for the final hidden layer and output layer's softmax function yielding the model's probability of each of the various emotions.

The eight emotions were coded as integers with the index of the output being the associated integer.

```
def emotionNumerics():
    emoNdx = {}
    emoNdx["anger"] = 0
    emoNdx["disgust"] = 1
    emoNdx["neutral"] = 2
    emoNdx["happiness"] = 3
    emoNdx["surprise"] = 4
    emoNdx["fear"] = 5
    emoNdx["sadness"] = 6
    emoNdx["contempt"] = 7
    return emoNdx
```

3 Performance

The following table shows performance metrics recorded throughout the model development and training process. In general, these training runs were 20 epochs of batch size 200 unless otherwise noted. Strict notes on epoch/batch were note maintained.

Table 1: Model Performance

Name	Val Loss	Val Acc	Training (s)
Model v2	9.107	0.435	4474
Model v3	8.019	0.502	2499
Model v4.1	9.107	0.435	2137
Model v4.2	8.019	0.502	3108
Model v5	6.967	0.195	1682
Model v6	1.712	0.238	3020
Model v7	8.000	0.140	4610
Model v8	1.695	0.195	3313
Model v6 w/ flatten	7.111	0.172	3044

Name	Val Loss	Val Acc	Training (s)
Model v6 (Docker Cloud)	11.154	0.308	3597
Model v3.1	1.599	0.223	687
Model v3.2	1.570	0.360	772
cnn_model_jhamski	0.317	0.346	636
cnn_model_jhamski 150x150	0.517	0.643	NA
emotion_model_jh_v2	0.275	0.556	2027
jh_v3 epoch x40	0.079	0.884	NA
jh_v3 epoch x60	0.071	0.918	NA
jh_v4 Epoch 20	0.235	0.613	NA
jh_v4 Epoch 100	0.182	0.847	NA
jh_v5 Epoch 20	0.132	0.786	NA
jh_v5 Epoch 100	0.098	0.931	NA
jh_v5 Epoch 27 13082 examples (8 transforms)	0.102	0.924	NA
jh_v5 Epoch 27 14536 examples (9 transforms)	0.094	0.926	NA
jh_v5 Epoch 20 29072 examples (19 transforms)	0.255	0.800	NA
jh_v5 Epoch 25 29072 examples (19 transforms)	0.266	0.823	NA

The source training data was discovered to be unbalanced. This likely contributed to early poor performance of the models. An algorithm was created to reduce the over-represented emotion classes inorder to provide a more balanced data set for training. In hind sight, the under-represented emotion classes probably should have been augmented using transformations and this would have maintained a greater diversity in the training data.

The memory requirements for training the model grew as more examples and transformation were included. During the Model v6 - 8 experiment time period, the laptop used for initial development became insuffiencient for further training. The deep learning Docker container was migrated to Docker Cloud integrated with Amazon Web Services. This enabled computing resources independent of the researcher's laptop to be applied to the training, as well as increased scalablility of these computing resources.

A side discussion was held with some classmates and James Hamski reported his model was training quickly and performing reasonably well. Experiments were performed with the Hamski model. Significantly, the researcher's prior experiements were using the full sized 350x350 image data, but scaling the images down to 150x150 significantly improved training time while maintaining accuracy for the given epoch counts.

Experiments on variations of the Hamski model were performed leading up to the `emotion_model_jh_v5` shown above. This, combined with extensions to the transformations for training, became the final neural network used in the Kaggle Competition.

Table 2: Kaggle Submissions

Submission	Model	Transforms	Epochs	Public Score
1	jh_v5	9	27	0.34351
2	jh_v5	19	20	0.57252
3	jh_v5	19	25	0.52672

- Transforms: The number of transforms applied to the training data.
- Epochs: The number of epochs of batch size 200 that the network was trained.
- Public Score: The Kaggle public score during the competition.

Note the increase in transforms used in training from the first submission to the second. Even with fewer training epochs, the network trained with more transforms resulted in better competition performance (generalization). Attempts were made to further increase transforms used for training, but memory limits were reached again after adding just two addition transforms for a total of 21 per raw input image. The training for the 21-transform network is still taking place at the time of this writing and has therefore not (yet) been submitted to the Kaggle competition.

4 Conclusions

Throughout the project, new techniques were learned and in most cases applied to the emotion detection model. Several points are significant in terms of a well performing model for the emotion detection scenario. The input data, 2D images, mean 2D convolutional layers are an excellent fit for the problem. Additionally, full resolution images appear to be less important for the training based on the relative success of the reduced size (150x150) images as compared to the 350x350 original size images. Finally, the use of image transformations (data augmentation) showed a real benefit to the generalizability of the model.

5 References

Chollet, F. Keras. 2015. URL: https://github.com/fchollet/keras.

Goodfellow, I., Y. Bengio and A. Courville. Deep Learning. Sep. 2016. URL: http://deeplearningbook.org.

Huang, G., M. Ramesh, T. Berg and E. Learned-Miller. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. Tech. rep. University of Massachusetts, Amherst, Oct. 2007. URL: http://vis-www.cs.umass.edu/lfw/.

Joshi, P. OpenCV with Python By Example. Sep. 2015. URL: https://www.packtpub.com/mapt/book/application-development/9781785283932/2/ch02lvl1sec23/Embossing.

Majumdar, S. Deep Convolutional Network using Keras. Apr. 2016. URL: https://www.kaggle.com/somshubramajumdar/digit-recognizer/deep-convolutional-network-using-keras.

Microsoft Corporation. Emotion API. Dec. 2016. URL: https://www.microsoft.com/cognitive-services/en-us/emotion-api.

Reid, S. 10 misconceptions about Neural Networks. May. 2014. URL: http://www.turingfinance.com/misconceptions-about-neural-networks/#algo.

Rowe, B. Emotion Detection From Facial Expressions Competition. Dec. 2016. URL: https://inclass.kaggle.com/c/emotion-detection-from-facial-expressions/leaderboard.

— facial_expressions. Dec. 2016. URL: https://github.com/muxspace/facial_expressions.