FACIAL EMOTION RECOGNITION USING CONVOLUTIONAL NEURAL NETWORKS

Nirali Bandaru

Takehome #2

ECE8720 – Dr. Robert J. Schalkoff

Clemson University – Spring 2020

TAKEHOME PROPOSAL #2

Objective:

Training a network for the purpose of "Facial Emotion Recognition using Convolutional Neural Networks"

Primary Reference for Project:

A. Verma, P. Singh and J. S. Rani Alex, "Modified Convolutional Neural Network Architecture Analysis for Facial Emotion Recognition," 2019 International Conference on Systems, Signals and Image Processing (IWSSIP), Osijek, Croatia, 2019, pp. 169-173.

Resources:

Reference paper: https://ieeexplore.ieee.org/document/8787215/authors#authors

Facial Expression Database: https://www.kdef.se/

Other resources may be added along the way. Final bibliography will be included in project report.

Programming environment: Python 3 in Sublime Text

Additional Tools: TensorFlow, Keras

Goals:

1. Train the dataset
2. Use CNN appropriately
3. Use testing data to retrieve outputs
4. Validate testing dataset outputs for accuracy
5. Analyze the system with suitable performance metrics

Deliverables:

1. Accurate categorization of facial emotion images used from the KDEF facial expressions database.
2. Accuracy measures and performance metrics.

DEVELOPMENT EFFORT

Project Workflow:

- Identifying the problem using the reference paper: classification of pictures based on the emotion or expression they represent.
- Understanding Convolutional Neural Networks and a little bit of Deep Neural Networks.
- Understanding the working of Keras and TensorFlow.
- Identifying the parameters specified in the research paper used as reference.
- Building the CNN model with the help of built-in libraries.
- Training the model.
- Testing and validation of model.
- Analyzing results.

Model Parameters:

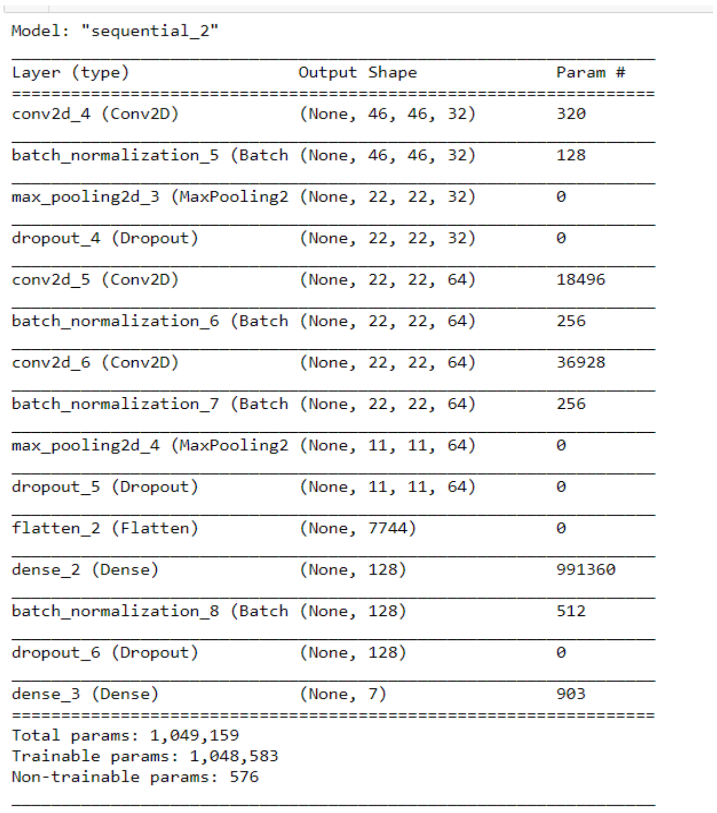
- Dataset used: FER2013 from Kaggle [2] – includes a CSV file with pixels of image and emotion.
- Sample Size: 35,888 images
- CSV file includes, in each column: output number for emotion, pixels of image
- Training-Testing Data Ratio: 80%-20%
- Model Type: Sequential
- Activation function: ReLU
- Algorithm used: Stochastic Gradient Descent also known as Adam
- Loss-type: Categorical cross-entropy
- Metrics: Accuracy
- Layers:
    - Input layer
    - 2DConvolution Layer with batch normalization and pooling followed by a dropout (0.25)
        - Feature size is 32
        - Kernel size is 3x3
        - Pooling size is 4x4
    - 2DConvolution Layer with batch normalization and pooling followed by a dropout (0.25)
        - Feature size is 64
        - Kernel size is 3x3
        - Pooling size is 2x2
    - Flattening layer
    - Fully connected Layer with batch normalization followed by dropout (0.5)
        - Feature size: 128
    - Output layer (7 nodes for each emotion, activation function is SoftMax)
        - 0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral
- Other parameters:
    - Batch size = 100
    - Epochs = 50
    - Learning rate = .001

Problem Encountered:

- Use of the function as_matrix() from the pandas library in Python was troublesome because it was deprecated. Instead, the function to_numpy() was used to convert the data file into Numpy files.
- There was a dimension problem that was resolved by adding another dimension to input dataset.
- General errors while converting data to Numpy.
- Minor mistakes in data type and syntax prolonged debugging efforts.

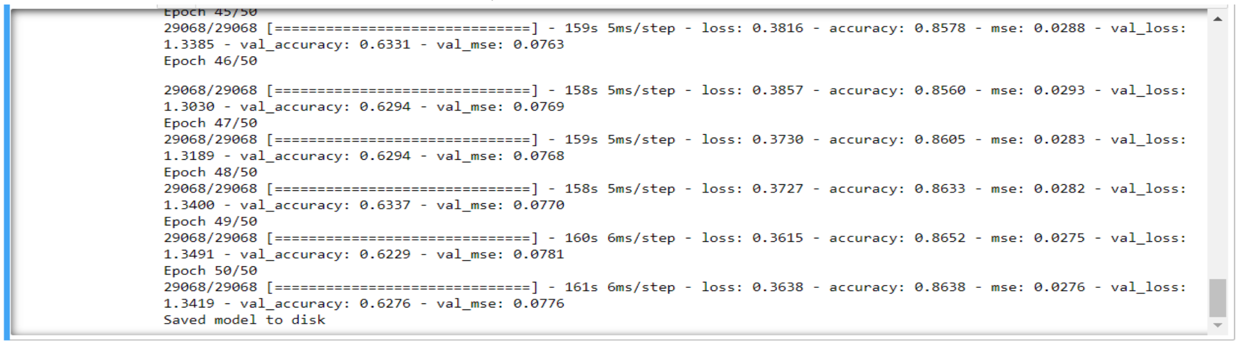
OUTPUTS - Training

Model Parameters are shown below:



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 46, 46, 32)        320
_________________________________________________________________
batch_normalization_5 (Batch (None, 46, 46, 32)        128
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 22, 22, 32)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 22, 22, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 22, 22, 64)        18496
_________________________________________________________________
batch_normalization_6 (Batch (None, 22, 22, 64)        256
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 22, 22, 64)        36928
_________________________________________________________________
batch_normalization_7 (Batch (None, 22, 22, 64)        256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 11, 11, 64)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 11, 11, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 7744)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               991360
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 903
=================================================================
Total params: 1,049,159
Trainable params: 1,048,583
Non-trainable params: 576
_________________________________________________________________
```

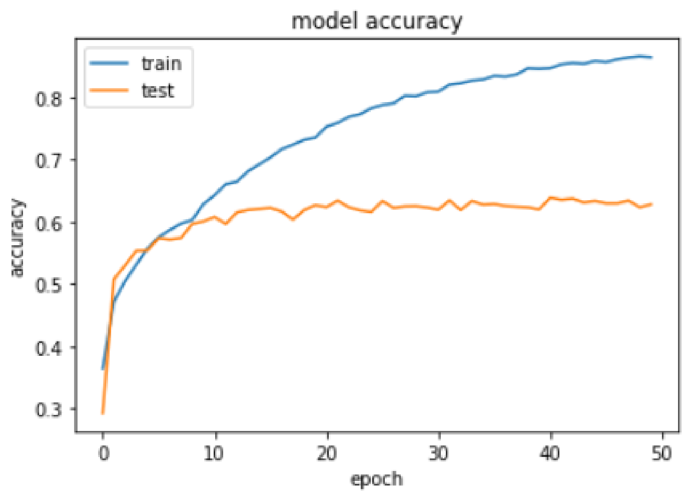
As shown below, the accuracy of the model’s validity testing turned out to be 62.76% after fluctuating between 62% and 63%. MSE is at 0.0776, and Loss is at 0.3638.

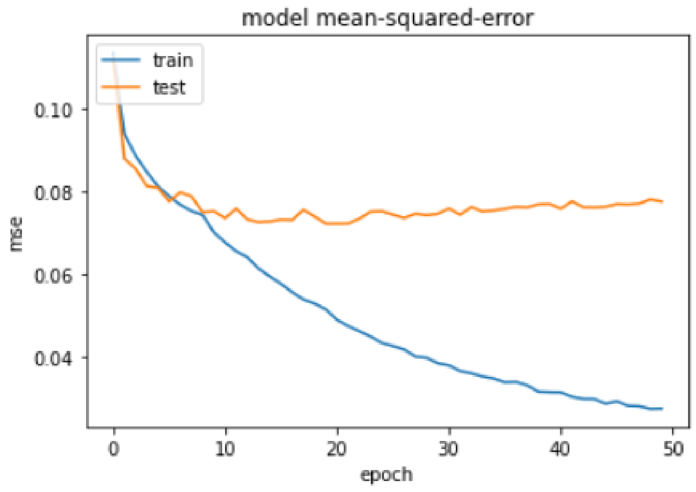


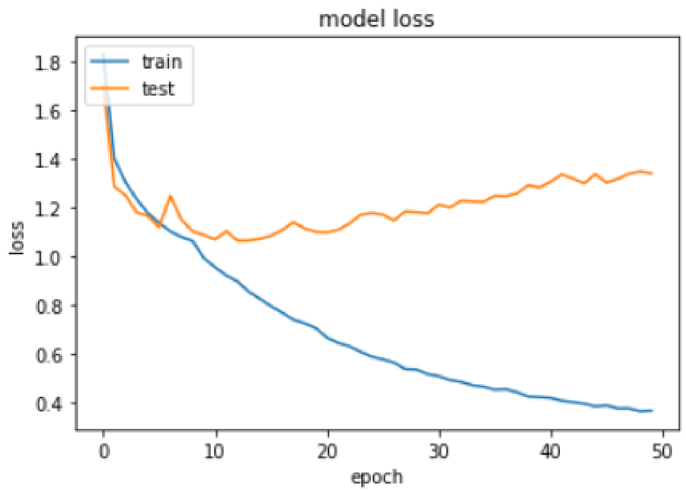
```
Epoch 45/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.3816 - accuracy: 0.8578 - mse: 0.0288 - val_loss:
1.3385 - val_accuracy: 0.6331 - val_mse: 0.0763
Epoch 46/50

29068/29068 [==============================] - 158s 5ms/step - loss: 0.3857 - accuracy: 0.8560 - mse: 0.0293 - val_loss:
1.3030 - val_accuracy: 0.6294 - val_mse: 0.0769
Epoch 47/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.3730 - accuracy: 0.8605 - mse: 0.0283 - val_loss:
1.3189 - val_accuracy: 0.6294 - val_mse: 0.0768
Epoch 48/50
29068/29068 [==============================] - 158s 5ms/step - loss: 0.3727 - accuracy: 0.8633 - mse: 0.0282 - val_loss:
1.3400 - val_accuracy: 0.6337 - val_mse: 0.0770
Epoch 49/50
29068/29068 [==============================] - 160s 6ms/step - loss: 0.3615 - accuracy: 0.8652 - mse: 0.0275 - val_loss:
1.3491 - val_accuracy: 0.6229 - val_mse: 0.0781
Epoch 50/50
29068/29068 [==============================] - 161s 6ms/step - loss: 0.3638 - accuracy: 0.8638 - mse: 0.0276 - val_loss:
1.3419 - val_accuracy: 0.6276 - val_mse: 0.0776
Saved model to disk
```

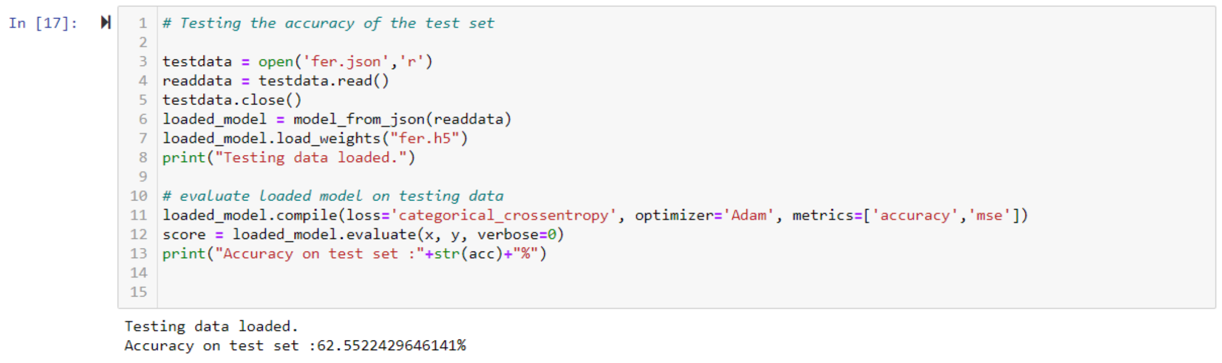
Plots


model accuracy
train
test
accuracy
0.3
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
50
epoch


model mean-squared-error
train
test
mse
0.04
0.06
0.08
0.10
0
10
20
30
40
50
epoch


model loss
train
test
loss
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
0
10
20
30
40
50
epoch

Testing Set



```
In [17]:
# Testing the accuracy of the test set

testdata = open('fer.json','r')
readdata = testdata.read()
testdata.close()
loaded_model = model_from_json(readdata)
loaded_model.load_weights("fer.h5")
print("Testing data loaded.")

# evaluate loaded model on testing data
loaded_model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy','mse'])
score = loaded_model.evaluate(x, y, verbose=0)
print("Accuracy on test set :"+str(acc)+"%")


```

```
Testing data loaded.
Accuracy on test set :62.5522429646141%
```

As shown above, the testing data has an accuracy of 62.55%

CONCLUSION

In this project, I have learned the basics of TensorFlow and how to implement a Convolutional Neural Network using Keras. In the previous takehome, the neural network was implemented from scratch, mathematically. This implementation included multiple hidden layers, hence calling the network a deep neural network, and yet it was simple to do with the help of built-in libraries such as Keras and Numpy. My learning was primarily focused on understanding Convolutional Neural Networks, as the python implementation was simple.

FUTURE WORK

In this implementation, I have used a readily available CSV file where the pixels and the emotion were already entered in the file beforehand, by the makers of the data. In the future, I would like to try to implement code for pre-processing of images and test the data with raw images. As I have limited working knowledge of Python, I could not understand how to do the pre-processing. That would be included in the future, should I pick up this project where I left off. In addition, I would also train the model for longer than 50 epochs. The research paper that I have referred to had used 25 epochs. Although I had doubled it, the accuracy did not turn out to been great. I would also explore the different architectures in my CNN model, as I believe that my model was not effective enough to deliver maximum accuracy.

APPENDIX

I. Full list of training output for 50 Epochs:

```
Epoch 1/50
29068/29068 [==============================] - 169s 6ms/step - loss: 1.8315 -
accuracy: 0.3646 - mse: 0.1134 - val_loss: 1.7048 - val_accuracy: 0.2926 - val
_mse: 0.1121
Epoch 2/50
29068/29068 [==============================] - 162s 6ms/step - loss: 1.4023 -
accuracy: 0.4718 - mse: 0.0938 - val_loss: 1.2874 - val_accuracy: 0.5077 - val
_mse: 0.0879
Epoch 3/50
29068/29068 [==============================] - 162s 6ms/step - loss: 1.3045 -
accuracy: 0.5047 - mse: 0.0886 - val_loss: 1.2509 - val_accuracy: 0.5297 - val
_mse: 0.0855
Epoch 4/50
29068/29068 [==============================] - 163s 6ms/step - loss: 1.2347 -
accuracy: 0.5309 - mse: 0.0847 - val_loss: 1.1802 - val_accuracy: 0.5533 - val
_mse: 0.0813
Epoch 5/50
29068/29068 [==============================] - 163s 6ms/step - loss: 1.1775 -
accuracy: 0.5569 - mse: 0.0813 - val_loss: 1.1660 - val_accuracy: 0.5539 - val
_mse: 0.0809
Epoch 6/50
29068/29068 [==============================] - 171s 6ms/step - loss: 1.1380 -
accuracy: 0.5750 - mse: 0.0789 - val_loss: 1.1188 - val_accuracy: 0.5734 - val
_mse: 0.0776
Epoch 7/50
29068/29068 [==============================] - 162s 6ms/step - loss: 1.1020 -
accuracy: 0.5866 - mse: 0.0768 - val_loss: 1.2485 - val_accuracy: 0.5712 - val
_mse: 0.0798
Epoch 8/50
29068/29068 [==============================] - 161s 6ms/step - loss: 1.0789 -
accuracy: 0.5968 - mse: 0.0753 - val_loss: 1.1499 - val_accuracy: 0.5737 - val
_mse: 0.0788
Epoch 9/50
29068/29068 [==============================] - 160s 6ms/step - loss: 1.0623 -
accuracy: 0.6026 - mse: 0.0743 - val_loss: 1.1023 - val_accuracy: 0.5963 - val
_mse: 0.0750
Epoch 10/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.9924 -
accuracy: 0.6281 - mse: 0.0701 - val_loss: 1.0872 - val_accuracy: 0.6003 - val
_mse: 0.0752
Epoch 11/50
```

```
29068/29068 [==============================] - 160s 5ms/step - loss: 0.9544 -
accuracy: 0.6423 - mse: 0.0677 - val_loss: 1.0690 - val_accuracy: 0.6074 - val
_mse: 0.0735
Epoch 12/50
29068/29068 [==============================] - 161s 6ms/step - loss: 0.9211 -
accuracy: 0.6600 - mse: 0.0655 - val_loss: 1.1029 - val_accuracy: 0.5963 - val
_mse: 0.0758
Epoch 13/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.8963 -
accuracy: 0.6641 - mse: 0.0641 - val_loss: 1.0641 - val_accuracy: 0.6149 - val
_mse: 0.0732
Epoch 14/50
29068/29068 [==============================] - 158s 5ms/step - loss: 0.8539 -
accuracy: 0.6810 - mse: 0.0614 - val_loss: 1.0648 - val_accuracy: 0.6189 - val
_mse: 0.0726
Epoch 15/50
29068/29068 [==============================] - 160s 6ms/step - loss: 0.8249 -
accuracy: 0.6919 - mse: 0.0595 - val_loss: 1.0715 - val_accuracy: 0.6204 - val
_mse: 0.0727
Epoch 16/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.7942 -
accuracy: 0.7031 - mse: 0.0577 - val_loss: 1.0841 - val_accuracy: 0.6220 - val
_mse: 0.0732
Epoch 17/50
29068/29068 [==============================] - 158s 5ms/step - loss: 0.7674 -
accuracy: 0.7162 - mse: 0.0557 - val_loss: 1.1074 - val_accuracy: 0.6164 - val
_mse: 0.0731
Epoch 18/50
29068/29068 [==============================] - 162s 6ms/step - loss: 0.7384 -
accuracy: 0.7235 - mse: 0.0539 - val_loss: 1.1404 - val_accuracy: 0.6031 - val
_mse: 0.0756
Epoch 19/50
29068/29068 [==============================] - 160s 5ms/step - loss: 0.7238 -
accuracy: 0.7312 - mse: 0.0529 - val_loss: 1.1137 - val_accuracy: 0.6186 - val
_mse: 0.0739
Epoch 20/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.7039 -
accuracy: 0.7349 - mse: 0.0515 - val_loss: 1.1001 - val_accuracy: 0.6263 - val
_mse: 0.0722
Epoch 21/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.6646 -
accuracy: 0.7524 - mse: 0.0490 - val_loss: 1.0979 - val_accuracy: 0.6232 - val
_mse: 0.0722
Epoch 22/50
```

```
29068/29068 [==============================] - 159s 5ms/step - loss: 0.6428 -
accuracy: 0.7585 - mse: 0.0475 - val_loss: 1.1080 - val_accuracy: 0.6337 - val
_mse: 0.0722
Epoch 23/50
29068/29068 [==============================] - 161s 6ms/step - loss: 0.6286 -
accuracy: 0.7681 - mse: 0.0463 - val_loss: 1.1344 - val_accuracy: 0.6229 - val
_mse: 0.0734
Epoch 24/50
29068/29068 [==============================] - 160s 5ms/step - loss: 0.6065 -
accuracy: 0.7722 - mse: 0.0450 - val_loss: 1.1704 - val_accuracy: 0.6183 - val
_mse: 0.0751
Epoch 25/50
29068/29068 [==============================] - 161s 6ms/step - loss: 0.5867 -
accuracy: 0.7818 - mse: 0.0434 - val_loss: 1.1782 - val_accuracy: 0.6155 - val
_mse: 0.0752
Epoch 26/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.5750 -
accuracy: 0.7869 - mse: 0.0427 - val_loss: 1.1714 - val_accuracy: 0.6331 - val
_mse: 0.0744
Epoch 27/50
29068/29068 [==============================] - 160s 5ms/step - loss: 0.5619 -
accuracy: 0.7897 - mse: 0.0418 - val_loss: 1.1473 - val_accuracy: 0.6220 - val
_mse: 0.0736
Epoch 28/50
29068/29068 [==============================] - 158s 5ms/step - loss: 0.5351 -
accuracy: 0.8018 - mse: 0.0401 - val_loss: 1.1844 - val_accuracy: 0.6245 - val
_mse: 0.0746
Epoch 29/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.5334 -
accuracy: 0.8011 - mse: 0.0398 - val_loss: 1.1816 - val_accuracy: 0.6248 - val
_mse: 0.0743
Epoch 30/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.5146 -
accuracy: 0.8076 - mse: 0.0385 - val_loss: 1.1768 - val_accuracy: 0.6226 - val
_mse: 0.0746
Epoch 31/50
29068/29068 [==============================] - 160s 5ms/step - loss: 0.5057 -
accuracy: 0.8087 - mse: 0.0380 - val_loss: 1.2126 - val_accuracy: 0.6192 - val
_mse: 0.0759
Epoch 32/50
29068/29068 [==============================] - 158s 5ms/step - loss: 0.4901 -
accuracy: 0.8197 - mse: 0.0366 - val_loss: 1.2010 - val_accuracy: 0.6341 - val
_mse: 0.0744
Epoch 33/50
```

```
29068/29068 [==============================] - 159s 5ms/step - loss: 0.4821 -
accuracy: 0.8220 - mse: 0.0361 - val_loss: 1.2283 - val_accuracy: 0.6186 - val
_mse: 0.0762
Epoch 34/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.4685 -
accuracy: 0.8262 - mse: 0.0353 - val_loss: 1.2261 - val_accuracy: 0.6331 - val
_mse: 0.0752
Epoch 35/50
29068/29068 [==============================] - 160s 6ms/step - loss: 0.4626 -
accuracy: 0.8280 - mse: 0.0348 - val_loss: 1.2238 - val_accuracy: 0.6272 - val
_mse: 0.0754
Epoch 36/50
29068/29068 [==============================] - 158s 5ms/step - loss: 0.4502 -
accuracy: 0.8338 - mse: 0.0339 - val_loss: 1.2490 - val_accuracy: 0.6282 - val
_mse: 0.0758
Epoch 37/50
29068/29068 [==============================] - 158s 5ms/step - loss: 0.4524 -
accuracy: 0.8326 - mse: 0.0340 - val_loss: 1.2468 - val_accuracy: 0.6251 - val
_mse: 0.0763
Epoch 38/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.4395 -
accuracy: 0.8362 - mse: 0.0332 - val_loss: 1.2591 - val_accuracy: 0.6238 - val
_mse: 0.0761
Epoch 39/50
29068/29068 [==============================] - 162s 6ms/step - loss: 0.4217 -
accuracy: 0.8460 - mse: 0.0316 - val_loss: 1.2922 - val_accuracy: 0.6229 - val
_mse: 0.0769
Epoch 40/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.4194 -
accuracy: 0.8453 - mse: 0.0315 - val_loss: 1.2828 - val_accuracy: 0.6195 - val
_mse: 0.0769
Epoch 41/50
29068/29068 [==============================] - 160s 5ms/step - loss: 0.4162 -
accuracy: 0.8462 - mse: 0.0315 - val_loss: 1.3065 - val_accuracy: 0.6384 - val
_mse: 0.0758
Epoch 42/50
29068/29068 [==============================] - 160s 5ms/step - loss: 0.4045 -
accuracy: 0.8523 - mse: 0.0304 - val_loss: 1.3372 - val_accuracy: 0.6350 - val
_mse: 0.0776
Epoch 43/50
29068/29068 [==============================] - 161s 6ms/step - loss: 0.3987 -
accuracy: 0.8544 - mse: 0.0299 - val_loss: 1.3199 - val_accuracy: 0.6368 - val
_mse: 0.0762
Epoch 44/50
```

```
29068/29068 [==============================] - 159s 5ms/step - loss: 0.3924 -
accuracy: 0.8532 - mse: 0.0299 - val_loss: 1.2999 - val_accuracy: 0.6307 - val
_mse: 0.0761
Epoch 45/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.3816 -
accuracy: 0.8578 - mse: 0.0288 - val_loss: 1.3385 - val_accuracy: 0.6331 - val
_mse: 0.0763
Epoch 46/50
29068/29068 [==============================] - 158s 5ms/step - loss: 0.3857 -
accuracy: 0.8560 - mse: 0.0293 - val_loss: 1.3030 - val_accuracy: 0.6294 - val
_mse: 0.0769
Epoch 47/50
29068/29068 [==============================] - 159s 5ms/step - loss: 0.3730 -
accuracy: 0.8605 - mse: 0.0283 - val_loss: 1.3189 - val_accuracy: 0.6294 - val
_mse: 0.0768
Epoch 48/50
29068/29068 [==============================] - 158s 5ms/step - loss: 0.3727 -
accuracy: 0.8633 - mse: 0.0282 - val_loss: 1.3400 - val_accuracy: 0.6337 - val
_mse: 0.0770
Epoch 49/50
29068/29068 [==============================] - 160s 6ms/step - loss: 0.3615 -
accuracy: 0.8652 - mse: 0.0275 - val_loss: 1.3491 - val_accuracy: 0.6229 - val
_mse: 0.0781
Epoch 50/50
29068/29068 [==============================] - 161s 6ms/step - loss: 0.3638 -
accuracy: 0.8638 - mse: 0.0276 - val_loss: 1.3419 - val_accuracy: 0.6276 - val
_mse: 0.0776
```

II. REFERENCES

[1] https://ieeexplore.ieee.org/document/8787215/authors#authors

[2] https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

[3] https://github.com/keras-team/keras/pull/9473

[4] https://keras.io/getting-started/sequential-model-guide/

[5] https://www.tensorflow.org/overview/

[6] https://keras.io/losses/

[7] https://keras.io/metrics/

[8] https://keras.io/getting-started/functional-api-guide/

[9] https://stackoverflow.com/questions/44747343/keras-input-explanation-input-shape-units-batch-size-dim-etc

[10] https://keras.io/layers/core/

[11] https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1

[12] https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/

[13] https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions

[14] https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

[15] https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/

[16] https://www.geeksforgeeks.org/python-pandas-series-as_matrix/

[17] https://stackoverflow.com/questions/41563720/error-when-checking-model-input-expected-convolution2d-input-1-to-have-4-dimens

[18] https://github.com/gitshanks/

[19] https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

[20] https://keras.io/layers/normalization/

[21] https://www.tensorflow.org/tutorials/images/cnn

[22] https://stackabuse.com/reading-and-writing-json-to-a-file-in-python/

[23] https://machinelearningmastery.com/save-load-keras-deep-learning-models/