Nonverbal Behavior Emotion Detection

- Lestyn Brooks & Amy Janks

Introduction

- Image detection software
  - Detects and categorizes non-verbal behavior and emotions
  - Looks at non-verbal cues to tell how a person is feeling
    - Done by looking at facial expressions and landmarks

Data

- Originally, create our own dataset
- Decided to look at pre-existing dataset
  - Went with the Fer2013 dataset from Kaggle
- Dataset contains
  - Train csv: emotions and pixels
  - Test csv: pixels
  - Fer2013 csv: emotions, pixels, and whether it used for testing or training

Test Dataset

pixels
254 254 254 254 254 249 255 160 2 58 53 70 77 76 75 78 68 18 32 29 0 54 73 75 72 68 75 77 76 76 75 80 51 36 47 40 44 42 37 48 40 64 54 54 86 16 0 161 254 254 254 254 254 248 255 120 2 38 50 47 76 76 83 63 51 142 174 159 88 59 48 67 37 61 46 77 77 75
156 184 198 202 204 207 210 212 213 214 215 214 214 213 216 217 218 217 216 214 213 214 213 214 215 211 207 205 204 202 198 195 193 190 184 180 176 174 172 168 164 168 167 163 161 157 154 150 146 182 199 204 207 210 213 214 214 214 214 212
69 118 61 60 96 121 103 87 103 88 70 90 115 122 123 124 129 132 133 131 131 121 113 110 101 100 99 114 113 105 106 107 120 123 124 130 138 135 136 147 143 137 129 126 125 118 124 144 66 115 57 45 118 213 158 93 95 69 87 108 124 124 125 131 13
205 203 236 157 83 158 120 116 94 86 155 180 205 231 219 217 190 198 208 174 159 167 211 230 215 209 195 210 202 186 187 182 185 221 229 227 218 200 192 156 151 121 160 197 189 127 158 144 186 213 140 43 111 161 90 109 91 130 173 192 221 21
87 79 74 66 74 96 77 80 80 84 83 89 102 91 84 102 108 107 102 89 96 128 152 176 195 207 214 220 222 224 222 220 216 214 205 197 179 147 108 58 96 172 97 52 73 92 94 112 98 73 68 62 79 81 74 75 79 79 76 91 98 85 92 104 106 104 91 87 116 151 174 1
235 233 223 109 34 37 34 31 28 38 56 69 106 136 153 163 145 135 136 127 152 158 152 144 138 121 65 38 56 42 34 31 28 33 35 39 40 34 29 32 39 50 39 25 16 24 28 37 231 233 220 75 22 44 31 39 36 44 70 109 168 186 194 208 207 206 203 208 213 210 201
71 70 104 147 166 170 195 145 156 154 146 129 139 130 117 103 104 107 111 101 90 79 75 110 126 101 79 89 95 113 107 111 105 102 129 126 148 156 171 196 175 148 159 153 154 108 118 116 45 80 83 129 149 176 187 136 153 148 141 133 144 132 113
176 177 170 168 173 171 167 169 166 139 98 107 121 136 138 141 154 155 160 155 153 143 135 121 101 103 101 76 27 13 17 17 17 22 45 59 75 76 59 55 38 67 74 83 92 94 87 108 181 180 180 176 172 174 169 173 169 151 127 125 139 145 149 159 164 167
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 218 217 217 217 217 217 217 217 218 215 212 209
126 126 123 119 116 113 112 111 110 111 93 72 107 109 127 166 190 203 206 209 209 210 211 210 203 199 194 183 173 160 142 121 83 71 56 43 36 41 42 58 64 53 54 59 60 64 61 52 130 128 125 122 120 115 114 113 113 113 87 86 107 125 175 199 206 21
180 175 169 161 157 158 157 154 155 157 162 169 168 165 159 153 150 149 150 151 153 155 163 166 169 170 170 176 179 180 183 188 189 190 192 195 195 198 200 202 206 208 208 209 217 222 224 228 175 169 163 155 149 144 137 126 126 126 132 145
88 46 35 27 22 32 59 59 62 76 136 148 153 126 109 108 92 90 103 118 117 121 130 115 90 88 87 87 85 103 128 110 88 72 54 54 50 34 29 6 101 205 198 200 202 182 180 214 68 34 31 25 28 43 54 68 78 110 147 151 152 124 114 91 79 102 123 124 135 147 13
121 112 64 104 101 87 118 74 91 128 89 109 91 27 127 197 191 186 189 192 194 197 195 192 190 186 178 177 169 161 151 133 105 84 52 29 24 21 22 23 20 16 12 14 18 25 89 80 143 109 76 125 93 111 114 74 109 114 71 115 84 39 188 203 191 191 189 192
165 203 211 204 216 204 202 194 195 191 207 209 196 202 209 214 214 215 193 186 175 128 180 208 160 130 189 215 188 169 184 201 184 190 203 173 193 202 203 203 205 207 204 195 176 163 174 175 167 207 205 205 211 204 195 189 181 183 195 189
22 28 27 28 26 28 31 33 33 30 32 23 19 44 75 110 120 127 138 138 151 155 129 128 127 125 118 117 119 96 109 94 65 54 18 13 22 26 26 27 26 19 22 21 20 18 20 18 25 29 27 24 26 32 31 37 34 27 23 41 93 110 130 152 146 141 151 141 150 151 141 139 136 1
132 154 165 176 182 187 192 199 203 206 208 212 216 221 223 223 221 222 223 222 220 218 217 214 209 207 210 212 206 202 197 192 192 194 195 192 186 177 173 163 151 142 99 70 89 139 141 63 139 153 164 177 185 191 201 207 211 214 216 213 214
255 255 255 254 254 252 174 59 35 34 40 57 84 111 129 139 147 155 161 165 171 174 176 180 183 187 187 186 183 176 169 146 118 84 50 46 85 105 154 189 198 250 252 254 255 254 250 251 255 255 255 252 253 207 110 34 31 34 55 84 109 125 134 141 1
159 161 160 157 148 151 137 143 141 140 147 139 126 159 170 154 179 180 183 193 194 167 175 152 170 164 161 164 166 192 190 203 209 192 203 214 213 215 213 208 213 221 224 226 228 227 226 227 138 134 144 151 136 147 147 137 147 146 147 130
195 246 247 235 206 212 211 193 233 251 247 246 245 245 242 232 209 180 149 124 111 111 112 110 109 107 107 108 108 108 110 114 116 113 128 146 141 159 160 188 190 192 175 227 237 219 228 225 244 244 245 244 220 212 210 232 247 242 244 241
22 9 17 40 41 40 28 19 39 67 84 93 97 104 103 107 108 115 123 129 131 133 135 130 133 133 129 122 114 108 95 86 78 49 13 2 2 4 9 8 5 9 6 4 5 5 5 6 21 10 25 37 43 36 20 29 54 74 90 97 104 106 106 108 112 119 126 130 130 131 131 131 129 131 130 126 1
247 247 247 247 247 247 247 248 245 245 241 245 244 230 217 219 206 136 83 77 70 61 71 57 59 80 50 53 66 53 50 42 47 49 44 53 67 68 60 51 44 44 38 38 50 66 72 65 247 247 247 247 247 247 247 247 246 247 249 249 235 213 197 165 154 114 84 58 43 5
79 74 71 31 0 66 226 11 56 242 19 47 220 0 108 191 0 170 93 11 226 21 71 246 229 235 224 14 27 234 212 9 8 215 44 35 232 0 119 162 0 169 66 1 118 21 25 139 90 65 60 84 12 9 178 3 63 244 29 25 209 0 107 190 0 172 100 8 241 17 80 250 227 233 220 14 2
123 139 123 128 137 117 110 144 150 130 141 147 109 36 51 76 85 97 119 138 153 164 167 170 173 173 167 162 153 144 140 134 129 123 117 106 92 78 69 61 56 55 59 74 85 75 65 60 133 145 119 141 123 113 136 158 131 129 158 141 97 28 72 88 97 116 1

Train Dataset

emotion	pixels
0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 119 115 110 98 91 84 84 90 99 110 126 143 153 158 171 169 172 169 165 129 110 113 107 95 79 66 62 56 57 61 52 43 41 65 61 58 57 56 69 75 70 65 56 54 105 146 154 151 151 155 155 150 147 147
0	151 150 147 155 148 133 111 140 170 174 182 154 153 164 173 178 185 185 189 187 186 193 194 185 183 186 180 173 166 161 147 133 172 151 114 161 161 146 131 104 95 132 163 123 119 129 140 120 151 149 149 153 137 115 129 166 170 18
2	231 212 156 164 174 138 161 173 182 200 106 38 39 74 138 161 164 179 190 201 210 216 220 224 222 218 216 213 217 220 220 218 217 212 174 160 162 160 139 135 137 131 94 56 36 44 27 16 229 175 148 173 154 151 171 172 183 101 23 25 67
4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 19 43 52 13 26 40 59 65 12 20 63 99 98 98 111 75 62 41 73 118 140 192 186 187 188 190 190 187 182 176 173 172 173 25 34 29 35 29 26 20 23 19 31 22 21 20 31 26 17 34 75 37 18 38 80 85 25 38 26 3
6	4 0 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84 115 127 137 142 151 156 155 149 153 152 157 160 162 159 145 121 83 58 48 38 21 17 7 5 25 27 24 25 1 0 0 0 0 0 0 0 0 0 0 0 6 18 26 37 50 62 83 115 134 138 144 147 150 162 163 164 161 165 169 171 17
2	55 55 55 55 55 54 60 68 54 85 151 163 170 179 181 185 188 188 191 196 189 194 198 197 195 194 190 193 195 184 175 172 161 159 158 159 147 136 137 136 146 120 86 93 114 116 99 74 55 55 55 55 55 52 71 86 79 143 156 165 166 171 176 179
4	20 17 19 21 25 38 42 42 46 54 56 62 63 66 82 108 118 130 139 134 132 126 113 97 126 148 157 161 155 154 154 164 189 204 194 168 180 188 214 214 214 216 208 220 205 187 176 162 22 17 17 25 29 32 44 47 46 54 64 67 73 79 96 104 114 134 1
3	77 78 79 79 78 75 60 55 47 48 58 73 77 79 57 50 37 44 56 70 80 82 87 91 86 80 73 66 54 57 68 69 68 68 49 46 75 71 69 70 70 72 72 71 72 74 77 76 83 84 82 81 81 69 60 60 46 57 74 71 70 67 36 40 45 54 65 71 78 77 78 80 84 83 79 76 78 65 52 57 67
3	85 84 90 121 101 102 133 153 153 169 177 189 195 199 205 207 209 216 221 225 221 220 218 222 223 217 220 217 211 196 188 173 170 133 117 131 121 88 73 73 50 27 34 32 34 40 46 63 78 76 101 131 116 117 149 158 166 177 187 195 200 203
2	255 254 255 254 254 179 122 107 95 124 149 150 169 178 179 179 181 181 184 190 191 191 193 190 190 195 194 192 193 196 193 192 188 182 173 162 152 144 129 116 113 106 184 255 252 254 255 255 255 254 254 255 238 146 122 108 126 14
0	30 24 21 23 25 25 49 67 84 103 120 125 130 139 140 139 148 171 178 175 176 174 180 180 178 178 182 185 183 186 186 178 180 172 175 171 155 152 141 136 132 137 131 96 46 37 44 37 31 22 21 22 24 28 44 67 86 106 119 126 124 136 139 135
6	39 75 78 58 58 45 49 48 103 156 81 45 41 38 49 56 60 49 32 31 28 52 83 81 78 75 62 31 18 19 19 20 17 20 16 15 12 10 11 10 23 36 65 59 9 3 5 7 93 69 86 90 84 75 51 129 133 63 46 45 41 41 42 38 33 30 29 29 27 29 39 52 65 43 24 14 14 12 11 11 11
6	219 213 206 202 209 217 216 215 219 218 223 230 227 227 233 235 234 236 237 238 234 226 219 212 208 201 190 183 176 161 74 15 24 22 22 22 21 19 19 20 20 23 7 89 255 252 255 255 215 215 206 205 209 214 215 211 216 216 222 229 225 22
6	148 144 130 129 119 122 129 131 139 153 140 128 139 144 146 143 132 133 134 130 140 142 150 152 150 134 128 149 142 138 156 155 140 136 143 143 139 144 160 170 154 181 185 183 193 193 224 247 149 140 134 132 125 115 114 121 132 1
3	4 2 13 41 56 62 67 87 95 62 65 70 80 107 127 149 153 150 165 168 177 187 176 167 152 128 130 149 149 146 130 139 139 143 134 105 78 56 36 50 69 82 64 35 10 11 13 105 4 4 21 42 53 67 61 70 58 86 107 115 132 145 164 178 178 179 183 192 1
5	107 107 109 109 109 109 110 101 123 140 144 144 149 153 160 161 161 167 168 169 172 172 173 175 176 171 170 166 165 162 162 157 150 149 145 140 136 132 128 111 98 111 109 107 107 107 107 107 107 107 109 109 109 110 107 109 131 14
3	14 14 18 28 27 22 21 30 42 61 77 86 88 95 100 99 101 99 98 99 99 96 101 102 96 95 94 88 78 72 65 55 40 25 20 20 42 64 74 129 133 125 144 151 153 154 154 155 16 14 24 28 27 25 31 43 65 84 97 96 96 101 102 109 104 102 100 101 103 106 108 1
2	255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 2
6	134 124 167 180 197 194 203 210 204 203 209 204 206 211 211 216 219 224 228 230 230 226 222 220 217 217 210 207 213 210 199 191 190 188 177 172 148 142 90 46 81 132 116 81 45 84 162 207 154 107 137 182 201 204 191 201 194 195 195
4	219 192 179 148 208 254 192 98 121 103 145 185 83 58 114 227 225 220 203 202 168 154 157 164 182 211 164 94 122 155 176 238 240 242 192 87 43 39 60 85 34 57 82 130 113 184 225 150 197 179 63 17 37 107 207 133 107 99 132 200 84 87 22
4	1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 7 12 23 45 38 35 14 43 27 31 24 18 20 29 18 6 2 4 2 0 1 1 1 1 0 5 13 16 16 16 15 14 1 1 1 1 1 1 1 1 1 1 1 1 0 0 3 7 9 27 44 43 42 26 28 60 76 61 42 49 51 38 43 31 10 12 17 4 0 1 1 1 1 3 13 16 16 15 15 14 1 1 1 1 1 1 1 1 1 1
2	174 51 37 37 38 41 22 25 22 24 35 51 70 83 98 113 119 127 136 149 149 141 125 107 77 50 30 21 9 38 96 79 72 87 60 23 25 43 29 24 33 51 36 33 26 20 136 255 107 46 43 34 48 35 21 20 16 22 40 60 77 96 113 131 149 162 169 175 177 169 156 142
0	123 125 124 142 209 226 234 236 231 232 235 223 211 196 184 181 182 186 185 193 208 211 208 201 196 192 191 192 194 201 207 216 225 225 223 220 219 220 172 124 124 122 121 121 122 122 122 122 123 125 123 158 221 230 234 236 229 2

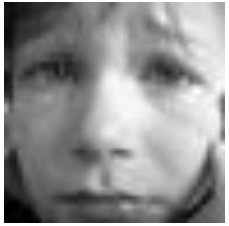
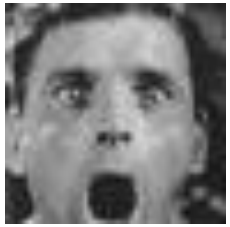
Fer2013

emotion	pixels	Usage
0	70 80 82 72 58 58 60 63 54 58 60 4	Training
0	151 150 147 155 148 133 111 140	Training
2	231 212 156 164 174 138 161 173	Training
4	24 32 36 30 32 23 19 20 30 41 21 2	Training
6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 4	Training
2	55 55 55 55 55 54 60 68 54 85 151	Training
4	20 17 19 21 25 38 42 42 46 54 56 6	Training
3	77 78 79 79 78 75 60 55 47 48 58 7	Training
3	85 84 90 121 101 102 133 153 153	Training
2	255 254 255 254 254 179 122 107	Training
0	30 24 21 23 25 25 49 67 84 103 12	Training
6	39 75 78 58 58 45 49 48 103 156 8	Training
6	219 213 206 202 209 217 216 215	Training
6	148 144 130 129 119 122 129 131	Training
3	4 2 13 41 56 62 67 87 95 62 65 70	Training
5	107 107 109 109 109 109 110 101	Training
3	14 14 18 28 27 22 21 30 42 61 77 8	Training
2	255 255 255 255 255 255 255 255	Training
6	134 124 167 180 197 194 203 210	Training

Script Output Images





Preprocessing, Inspection, and Cleaning

- Turn the pixels into grayscale images
- Resizes the image to 48 by 48
- Rename image to img_00x and places the image into a directory according to the emotion
-

Preprocessing, Inspection, and Cleaning

- Renamed the 3 disgust values to angry
- Graph all of the emotions and how many values are represented those emotions
- Displayed examples of images

Data Preprocessing

- Dlib
  - Contains:
    - face detection functions
      - Create a rectangle around a face or faces
    - Shape predictor functions
      - Plots facial landmarks
        - Like eyes, mouth, eyebrows, nose, and jawline
  -

Data Preprocessing

- Sklearn
  - Functions like scaler transformation and standard scaler
    - Standardizes the data across the board
- The fer2013 contains thousands of values
  - Split the data Train
  - Using sklearn train_test_split function

Libraries

- Dlib
  - Detect faces and maps out facial landmarks
- Sklearn
  - Helped with train the data set
    - fit method
    - Predict
    - train_test_split

Libraries

- Keras and tensorflow
  - Create, build, and save model
  - Load model
- openCV
  - Read images
  - Turn images into grayscale
  - Resizes and reshapes images
  - Displays the images in own window

Training Algorithm

- Splits training data using train_test_split by 10% at random
- CNN algorithm with layers of 7 types presented in keras layers
- Checkpoint function - monitors validation loss
- Early stopping function - when no improvement has been made
- Reduce Learning Rate function - when no improvement is being made reduces the learning rate
- Graph the accuracy and loss

Key Parts of Code - Face Detection and Landmarks

- The get_landmarks method takes an image, detects all of the faces within it, and then collects all of the facial landmarks and stores them.
- The image_landmarks method displays the image, placing a square around all of the faces as well as small circles at all of the identified facial landmarks.

```
def get_landmarks(image_url):
    try:
        url_response = urllib.request.urlopen(image_url)
        img_array = np.array(bytearray(url_response.read()), dtype=np.uint8)
        image = cv2.imdecode(img_array, -1)
    except Exception as e:
        print("Please check the URL and try again.")
        return None,None
    faces = frontalface_detector(image, 1)

    if len(faces):
        landmarks = [(p.x, p.y) for p in landmark_predictions(image, faces[0]).parts()]
    else:
        return None, None
    return image, landmarks

def image_landmarks(image, face_landmarks):
    radius = -1
    circle_thick = 4
    image_copy = image.copy()
    for (x,y) in face_landmarks:
        cv2.circle(image_copy, (x,y), circle_thick, (255,0,0), radius)
        plt.imshow(image_copy, interpolation='nearest')
        plt.axis('off')
        plt.show()
```

Key Parts of Code - Model Creation and Training

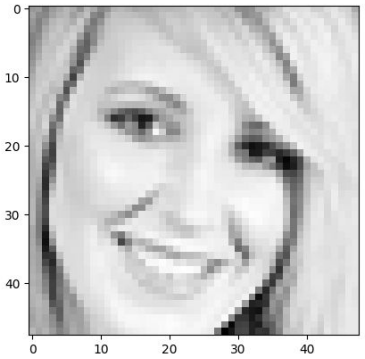
- The emotion_count method preprocesses the images from the dataset, while simplifying the labelling by re-classifying images with disgust to be angry instead.
- The load_data method, well, loads the data from the CSV dataset, reshaping and rescaling it so that it can be fed into the model.
- The two methods essentially work together to preprocess the images and create the model.

```python
def emotion_count(y_train, classes):
    """
    The function re-classify picture with disgust label into angry label
    """
    emo_classcount = {}
    print('Disgust classified as Angry')
    y_train.loc[y_train == 1] = 0
    classes.remove('Disgust')
    for new_num, _class in enumerate(classes):
        y_train.loc[(y_train == emotion[_class])] = new_num
        class_count = sum(y_train == (new_num))
        emo_classcount[_class] = (new_num, class_count)
    return y_train.values, emo_classcount
```

```python
def load_data(sample_split=0.3, usage='Training', classes=['Angry', 'Happy'],
              filepath='C:/Users/Lestyn/Desktop/Programming/NonVerbalBehaviorDetection/fer2013.csv'):
    """
    The function load provided CSV dataset and further reshape, rescale the data for feeding
    """
    df = pd.read_csv(filepath)
    df = df[df.Usage == usage]
    frames = []
    classes.append('Disgust')
    for _class in classes:
        class_df = df[df['emotion'] == emotion[_class]]
        frames.append(class_df)
    data = pd.concat(frames, axis=0)
    rows = random.sample(list(data.index), int(len(data) * sample_split))
    data = data.loc[rows]
    x = list(data["pixels"])
    X = []
    for i in range(len(x)):
        each_pixel = [int(num) for num in x[i].split()]
        X.append(each_pixel)
    ## reshape into 48*48*1 and rescale
    X = np.array(X)
    X = X.reshape(X.shape[0], 48, 48, 1)
    X = X.astype("float32")
    X /= 255

    y_train, new_dict = emotion_count(data.emotion, classes)
    y_train = to_categorical(y_train)
    return X, y_train
```

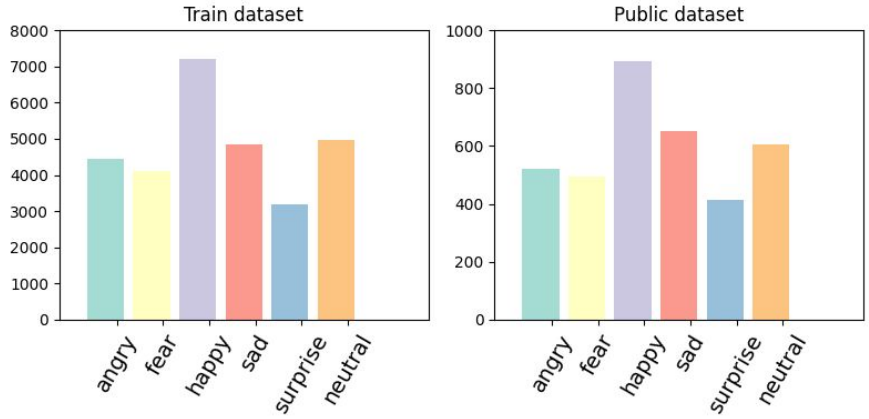
Outputs and Results (Model Creation and Training)

Data Input Overview



Data Visualization

Outputs and Results (Model Creation and Training)



Emotion Counts in Datasets

Outputs and Results (Model Creation and Training)

```
Disgust classified as Angry
Disgust classified as Angry
Disgust classified as Angry
(28709, 48, 48, 1)
(28709, 6)
(3589, 48, 48, 1)
(3589, 6)
(3589, 48, 48, 1)
(3589, 6)
Private test set
<zip object at 0x00000264115EF3C0>
```

Data Preprocessing Results

Key Parts of Code - Model Creation and Training

```
pixels = data['pixels'].tolist()

faces = []
for pixel_sequence in pixels:
    face = [int(pixels) for pixels in pixel_sequence.split(' ')]
    face = np.asarray(face).reshape(width,height)
    #face = face/255.0
    #face = cv2.resize(face.astype('uint8'), (width,height))
    faces.append(face.astype('float32'))
faces = np.asarray(faces)
faces = np.expand_dims(faces, -1)
emotions = pd.get_dummies(data['emotion']).to_numpy()

X_train, X_test, y_train, y_test = train_test_split(faces, emotions, test_size=0.1, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=41)

model = Sequential()

model.add(Conv2D(num_features, kernel_size=(3, 3), activation='relu', input_shape=(width, height, 1), data_format='channels_last', kernel_regularizer=l2(0.01)))
model.add(Conv2D(num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

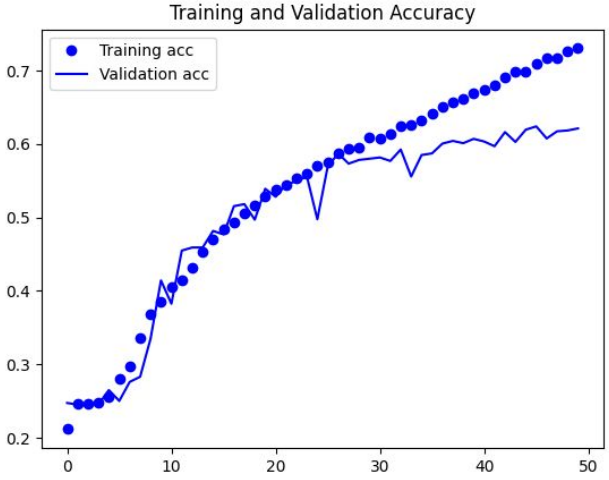
```
model.add(Dropout(0.5))

model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(2*2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*num_features, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(num_labels, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001, beta_1=0.9,
                                                               beta_2=0.99, epsilon=1e-7), metrics=['accuracy'])
lr_reducer=ReduceLROnPlateau(monitor='val_loss', factor=0.9, patience=3, verbose=1)
tensorboard=TensorBoard(log_dir='./logs')
early_stopper=EarlyStopping(monitor='val_loss', min_delta=0, patience=8, verbose=1, mode='auto')

checkpointer=ModelCheckpoint(Model_Path, monitor='val_loss', verbose=1, save_best_only=True)
```

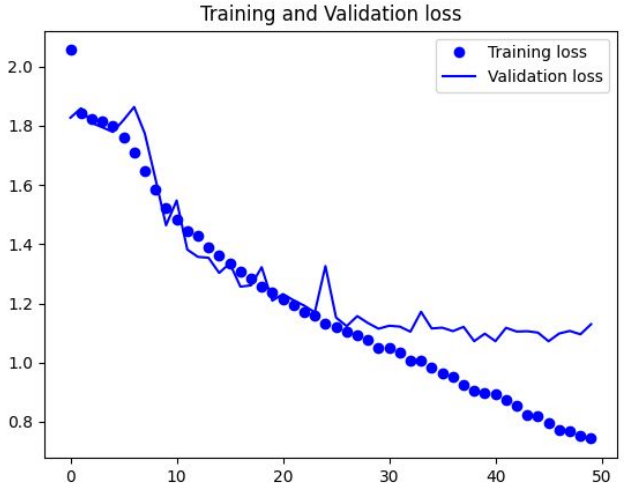
Outputs and Results (Model Creation and Training)

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 64)        640
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 46, 46, 64)        36928
_________________________________________________________________
batch_normalization (BatchNo (None, 46, 46, 64)        256
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 23, 23, 128)       73856
_________________________________________________________________
batch_normalization_1 (Batch (None, 23, 23, 128)       512
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 23, 23, 128)       147584
_________________________________________________________________
batch_normalization_2 (Batch (None, 23, 23, 128)       512
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 11, 11, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 11, 11, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 256)       295168
_________________________________________________________________
batch_normalization_3 (Batch (None, 11, 11, 256)       1024
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 256)       590080
_________________________________________________________________
batch_normalization_4 (Batch (None, 11, 11, 256)       1024
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 256)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 256)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 5, 5, 512)         1180160
_________________________________________________________________
batch_normalization_5 (Batch (None, 5, 5, 512)         2048
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 5, 5, 512)         2359808
_________________________________________________________________
batch_normalization_6 (Batch (None, 5, 5, 512)         2048
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 903
=================================================================
Total params: 5,905,863
Trainable params: 5,902,151
Non-trainable params: 3,712
_________________________________________________________________
```

Layers in the network

Outputs and Results (Model Creation and Training)



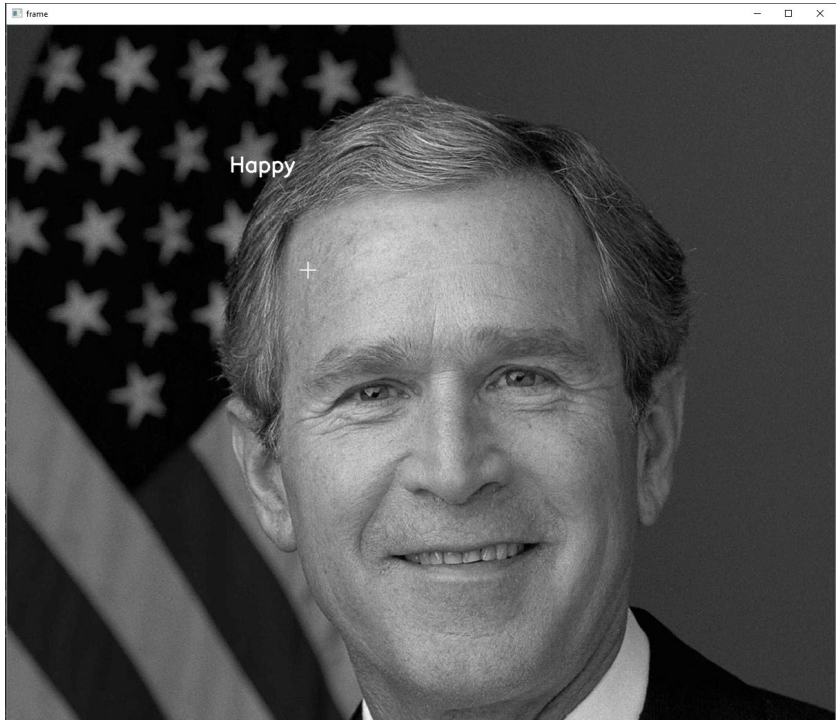


50 epochs

Key Parts of Code - Model Creation and Training

```
history = model.fit(np.array(X_train), np.array(y_train), batch_size=batch_size,
                    epochs=epochs, verbose=1, validation_data=(np.array(X_test),np.array(y_test)),
                    shuffle=True, callbacks=[lr_reducer, tensorboard, early_stopper, checkpointer])

scores = model.evaluate(np.array(X_test), np.array(y_test), batch_size=batch_size)
print("Loss: " + str(scores[0]))
print("Accuracy: " + str(scores[1]))
model.save(Model_Path)

model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights('model_weights.h5')
model.save('model.h5')
print("Saved model to disk")
loaded = load_model("model.h5")
print("loaded model:", loaded)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```

Outputs and Results (Testing Training Algorithm)

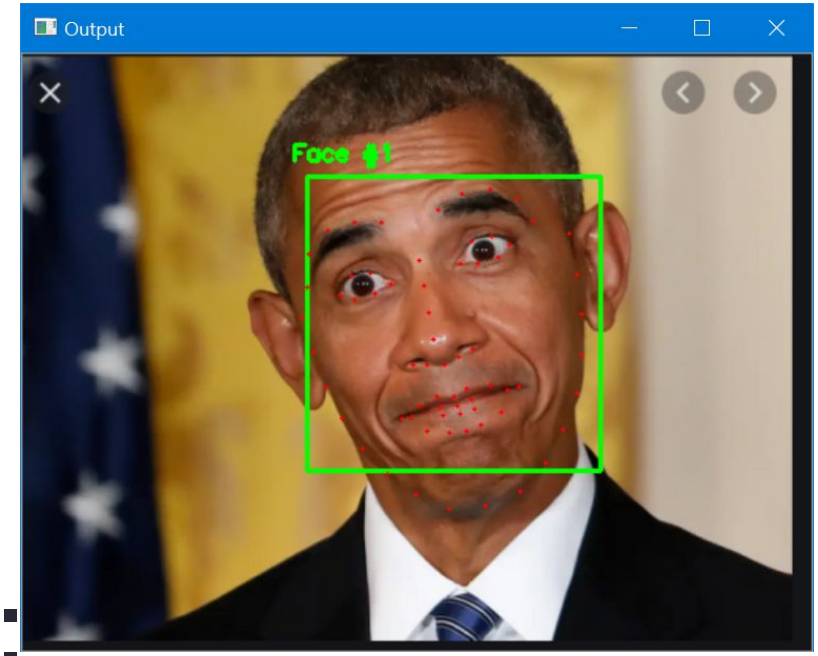



```
['Angry is 4.57%' 'Disgust is 0.00%' 'Fear is 2.54%' 'Happy is 58.94%'
 'Sad is 6.67%' 'Surprise is 0.89%' 'Neutral is 26.39%']
```

Key Parts of Cde – Testing Training Algorithm

```
image = cv2.imread(img)
#image = imutils.resize(image, width=300)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow('gray', gray)
cv2.waitKey(0)
canvas = np.zeros((220,300,3), dtype ="uint8")
frameClone = image.copy()

font = cv2.FONT_HERSHEY_SIMPLEX
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)
def texts(emotions, prod):
    return "{} is {:.2f}%".format(emotions, prod*100)

for(x,y,w,h) in faces:
    cv2.rectangle(image, (x,y),(x+w, y+h), (0,255,0), 2)
    face_crop = gray[y:y+h, x:x+w]
    cropped = np.expand_dims(np.expand_dims(cv2.resize(face_crop, (48, 48)), -1), 0)

    predictions = lmodel.predict(cropped)
    max_index = int(np.argmax(predictions))
    label = emotion_map[predictions.argmax()]
    for(i, (emotion, prob)) in enumerate(zip(emotion_map, predictions)):
        #prob2 = "%.2f" % (prob*100).astype(float)
        text_1=np.vectorize(texts)
        text = text_1(emotions, prob)
        prob = prob*300
        w = prob.astype(int)

        print(text)
        #cv2.rectangle(canvas, (5, (int(i) * 35)), (int(w), (int(i) * 35)), (40, 50, 155), -1)

#        cv2.putText(canvas, text, (10, (i*35)+23),font, 0.45,(55,25,5),2)

    #cv2.putText(frameClone,label,(x,y-10),font, 0.45, (55,25,5),2)
    #cv2.rectangle(frameClone,(x,y),(x+20, y-60), (140,50,155),2)
    cv2.putText(gray, emotion_map[max_index], (x+20, y-60), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
```

Outputs and Results (Face Detection and Landmarks)



Output image with former President Obama's face and landmarks detected

References

https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1643&context=etd_projects

https://poseidon01.ssrn.com/delivery.php?ID=216101117006074024121088094067027110098038084081067053124015102078091112068126002124122053005059029127010082026114084086081117062000046006093031090066003071092125089000066082065068107029004022122017024081119008014005086014086080007109006019110028123069&EXT=pdf

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6514572/pdf/sensors-19-01897.pdf

https://ebookcentral-proquest-com.ezproxy.ltu.edu:9443/lib/lawrencetu-ebooks/reader.action?docID=1889216

Emotion Detection: a Machine Learning Project | by Aarohi Gupta

Challenges in Representation Learning: Facial Expression Recognition Challenge

Facial landmarks with dlib, OpenCV, and Python

Emotion Detection Using OpenCV and Keras | by Karan Sethi | The Startup