Model Training

June 4, 2020

1 Prerequisites

[15]:
```
import numpy as np
import pandas as pd
from tensorflow import keras
from keras.models import Sequential, model_from_json
from keras.layers import Conv2D, BatchNormalization, MaxPooling2D, Flatten,␣
 ↪Dense, Dropout
```

2 Preprocessing

2.1 Read data file and convert to dataframe:

[27]:
```
data = pd.read_csv('train.csv')
df_data = pd.DataFrame(data.values, columns=['Emotion', 'Pixels'])
df_data
```

[27]:
```
     Emotion                                             Pixels
0          3  221 240 251 254 255 255 255 255 255 255 255 25…
1          6  100 107 108 104 103 113 117 115 120 130 138 14…
2          4  35 50 56 57 63 76 74 79 85 86 105 133 145 152 …
3          6  119 124 129 135 136 140 142 149 159 156 163 16…
4          2  160 173 186 194 188 185 175 162 153 143 135 12…
…        …                                                …
4173       5  62 76 93 86 69 73 71 70 82 90 93 95 93 102 107…
4174       6  129 129 131 134 143 151 156 158 151 155 163 17…
4175       3  86 89 97 108 111 110 105 103 101 103 92 87 95 …
4176       5  119 120 119 124 132 128 118 109 104 108 112 12…
4177       3  151 138 129 137 150 153 134 127 135 137 134 14…

[4178 rows x 2 columns]
```

2.2 Convert 'Emotion' column to array:

[17]:
```
labels = np.array(df_data['Emotion'], dtype=np.float32)
```

2.3 Convert 'Pixels' column to array:

[18]:

```python
# Convert column to list
pixels = list(df_data['Pixels'])

for i in range(len(pixels)):
    # Split single string into many strings
    pixels[i] = pixels[i].split()

    for j in range(len(pixels[i])):
        # Convert strings to float objects
        pixels[i][j] = float(pixels[i][j])



for i in range(1, len(pixels)):
    # Create list of all pixel values
    ls_pixels = pixels[0]
    ls_pixels.extend(pixels[i])

# Reshape and normalise pixel list
pixels = np.array(ls_pixels,
                  dtype=np.float32).reshape(len(df_data.index),
                                            48,48,1) / 255.0
```

3 Model

3.1 Create the model:

[19]:

```python
model = Sequential()

# First convolutional layer
model.add(Conv2D(input_shape=(48,48,1),
                 filters=64, kernel_size=2, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())

#Second convolutional layer
model.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())

# Third convolutional layer
model.add(Conv2D(filters=128, kernel_size=2, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
```

```python
# Flatten from 4D to 2D
model.add(Flatten())

# Dense layer
model.add(Dense(units=100, activation='relu'))

# Apply 20% dropout rate
model.add(Dropout(rate=0.2))

# Dense output layer
model.add(Dense(units=7, activation='softmax'))
```

[29]:

```python
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 47, 47, 64)        320
_________________________________________________________________
batch_normalization_1 (Batch (None, 47, 47, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 22, 22, 64)        16448
_________________________________________________________________
batch_normalization_2 (Batch (None, 22, 22, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 11, 11, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 128)       32896
_________________________________________________________________
batch_normalization_3 (Batch (None, 10, 10, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               320100
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 707
=================================================================
Total params: 371,495
Trainable params: 370,983
Non-trainable params: 512
```

```
-----------------------------------------------------------------
```

3.2 Compile the model:

[20]:

```
optimizer = 'adam'
loss = 'sparse_categorical_crossentropy'
metrics = ['accuracy']
```

[21]:

```
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

3.3 Train the model:

[22]:

```
# Analyse data in batches of this size
batch_size = 20

# Run model for this many epochs
epochs = 60
```

[23]:

```
model.fit(pixels, labels, epochs=epochs,
          batch_size=batch_size, validation_split=0.2)
```

```
Train on 3342 samples, validate on 836 samples
Epoch 1/60
3342/3342 [==============================] - 28s 8ms/step - loss: 1.5177 -
accuracy: 0.4865 - val_loss: 4.0278 - val_accuracy: 0.0969
Epoch 2/60
3342/3342 [==============================] - 27s 8ms/step - loss: 0.9767 -
accuracy: 0.6335 - val_loss: 2.3351 - val_accuracy: 0.2739
Epoch 3/60
3342/3342 [==============================] - 29s 9ms/step - loss: 0.8357 -
accuracy: 0.6834 - val_loss: 1.0914 - val_accuracy: 0.6268
Epoch 4/60
3342/3342 [==============================] - 32s 10ms/step - loss: 0.6536 -
accuracy: 0.7606 - val_loss: 0.6596 - val_accuracy: 0.7691
Epoch 5/60
3342/3342 [==============================] - 34s 10ms/step - loss: 0.5824 -
accuracy: 0.7822 - val_loss: 0.5718 - val_accuracy: 0.7967
Epoch 6/60
3342/3342 [==============================] - 31s 9ms/step - loss: 0.4943 -
accuracy: 0.8130 - val_loss: 0.9112 - val_accuracy: 0.6974
Epoch 7/60
3342/3342 [==============================] - 31s 9ms/step - loss: 0.4289 -
accuracy: 0.8396 - val_loss: 0.7655 - val_accuracy: 0.7141
Epoch 8/60
3342/3342 [==============================] - 29s 9ms/step - loss: 0.3759 -
accuracy: 0.8570 - val_loss: 0.7224 - val_accuracy: 0.7620
Epoch 9/60
```

```
3342/3342 [==============================] - 29s 9ms/step - loss: 0.3272 -
accuracy: 0.8797 - val_loss: 0.6853 - val_accuracy: 0.7512
Epoch 10/60
3342/3342 [==============================] - 30s 9ms/step - loss: 0.2568 -
accuracy: 0.9042 - val_loss: 0.5373 - val_accuracy: 0.8445
Epoch 11/60
3342/3342 [==============================] - 28s 8ms/step - loss: 0.2576 -
accuracy: 0.9031 - val_loss: 0.6203 - val_accuracy: 0.8397
Epoch 12/60
3342/3342 [==============================] - 28s 8ms/step - loss: 0.1930 -
accuracy: 0.9318 - val_loss: 0.6101 - val_accuracy: 0.8170
Epoch 13/60
3342/3342 [==============================] - 28s 8ms/step - loss: 0.1532 -
accuracy: 0.9434 - val_loss: 0.5706 - val_accuracy: 0.8325
Epoch 14/60
3342/3342 [==============================] - 29s 9ms/step - loss: 0.1693 -
accuracy: 0.9372 - val_loss: 0.7342 - val_accuracy: 0.8038
Epoch 15/60
3342/3342 [==============================] - 31s 9ms/step - loss: 0.1438 -
accuracy: 0.9467 - val_loss: 0.6378 - val_accuracy: 0.8409
Epoch 16/60
3342/3342 [==============================] - 31s 9ms/step - loss: 0.1143 -
accuracy: 0.9602 - val_loss: 0.5743 - val_accuracy: 0.8529
Epoch 17/60
3342/3342 [==============================] - 29s 9ms/step - loss: 0.1001 -
accuracy: 0.9641 - val_loss: 0.6811 - val_accuracy: 0.8289
Epoch 18/60
3342/3342 [==============================] - 28s 8ms/step - loss: 0.1190 -
accuracy: 0.9551 - val_loss: 0.7008 - val_accuracy: 0.8170
Epoch 19/60
3342/3342 [==============================] - 27s 8ms/step - loss: 0.1196 -
accuracy: 0.9554 - val_loss: 0.9731 - val_accuracy: 0.7404
Epoch 20/60
3342/3342 [==============================] - 28s 8ms/step - loss: 0.1321 -
accuracy: 0.9515 - val_loss: 0.6815 - val_accuracy: 0.8373
Epoch 21/60
3342/3342 [==============================] - 27s 8ms/step - loss: 0.1198 -
accuracy: 0.9545 - val_loss: 0.8603 - val_accuracy: 0.7907
Epoch 22/60
3342/3342 [==============================] - 46s 14ms/step - loss: 0.1224 -
accuracy: 0.9560 - val_loss: 0.7472 - val_accuracy: 0.8301
Epoch 23/60
3342/3342 [==============================] - 64s 19ms/step - loss: 0.1644 -
accuracy: 0.9470 - val_loss: 0.7849 - val_accuracy: 0.7787
Epoch 24/60
3342/3342 [==============================] - 64s 19ms/step - loss: 0.0927 -
accuracy: 0.9677 - val_loss: 0.7543 - val_accuracy: 0.8481
Epoch 25/60
```

```
3342/3342 [==============================] - 53s 16ms/step - loss: 0.0654 -
accuracy: 0.9797 - val_loss: 0.6754 - val_accuracy: 0.8553
Epoch 26/60
3342/3342 [==============================] - 47s 14ms/step - loss: 0.0485 -
accuracy: 0.9853 - val_loss: 0.8130 - val_accuracy: 0.7990
Epoch 27/60
3342/3342 [==============================] - 54s 16ms/step - loss: 0.0955 -
accuracy: 0.9677 - val_loss: 0.7319 - val_accuracy: 0.8505
Epoch 28/60
3342/3342 [==============================] - 53s 16ms/step - loss: 0.0709 -
accuracy: 0.9755 - val_loss: 0.7244 - val_accuracy: 0.8337
Epoch 29/60
3342/3342 [==============================] - 71s 21ms/step - loss: 0.0585 -
accuracy: 0.9785 - val_loss: 0.8941 - val_accuracy: 0.8170
Epoch 30/60
3342/3342 [==============================] - 61s 18ms/step - loss: 0.0461 -
accuracy: 0.9832 - val_loss: 0.8274 - val_accuracy: 0.8313
Epoch 31/60
3342/3342 [==============================] - 56s 17ms/step - loss: 0.0640 -
accuracy: 0.9794 - val_loss: 1.1306 - val_accuracy: 0.8062
Epoch 32/60
3342/3342 [==============================] - 53s 16ms/step - loss: 0.1211 -
accuracy: 0.9599 - val_loss: 0.8530 - val_accuracy: 0.8301
Epoch 33/60
3342/3342 [==============================] - 54s 16ms/step - loss: 0.1032 -
accuracy: 0.9689 - val_loss: 0.9137 - val_accuracy: 0.8074
Epoch 34/60
3342/3342 [==============================] - 60s 18ms/step - loss: 0.0692 -
accuracy: 0.9788 - val_loss: 0.7277 - val_accuracy: 0.8672
Epoch 35/60
3342/3342 [==============================] - 63s 19ms/step - loss: 0.0688 -
accuracy: 0.9767 - val_loss: 0.8382 - val_accuracy: 0.8457
Epoch 36/60
3342/3342 [==============================] - 33s 10ms/step - loss: 0.0695 -
accuracy: 0.9737 - val_loss: 0.8772 - val_accuracy: 0.8397
Epoch 37/60
3342/3342 [==============================] - 30s 9ms/step - loss: 0.0595 -
accuracy: 0.9785 - val_loss: 0.9349 - val_accuracy: 0.8242
Epoch 38/60
3342/3342 [==============================] - 27s 8ms/step - loss: 0.0846 -
accuracy: 0.9719 - val_loss: 0.9523 - val_accuracy: 0.7967
Epoch 39/60
3342/3342 [==============================] - 30s 9ms/step - loss: 0.0621 -
accuracy: 0.9823 - val_loss: 0.8200 - val_accuracy: 0.8242
Epoch 40/60
3342/3342 [==============================] - 26s 8ms/step - loss: 0.0694 -
accuracy: 0.9779 - val_loss: 1.1086 - val_accuracy: 0.8278
Epoch 41/60
```

```
3342/3342 [==============================] - 27s 8ms/step - loss: 0.0547 -
accuracy: 0.9817 - val_loss: 0.9102 - val_accuracy: 0.8505
Epoch 42/60
3342/3342 [==============================] - 27s 8ms/step - loss: 0.0411 -
accuracy: 0.9838 - val_loss: 0.9667 - val_accuracy: 0.8194
Epoch 43/60
3342/3342 [==============================] - 29s 9ms/step - loss: 0.0519 -
accuracy: 0.9826 - val_loss: 0.9230 - val_accuracy: 0.8278
Epoch 44/60
3342/3342 [==============================] - 29s 9ms/step - loss: 0.0461 -
accuracy: 0.9850 - val_loss: 0.9908 - val_accuracy: 0.8397
Epoch 45/60
3342/3342 [==============================] - 29s 9ms/step - loss: 0.1197 -
accuracy: 0.9638 - val_loss: 1.1920 - val_accuracy: 0.8110
Epoch 46/60
3342/3342 [==============================] - 30s 9ms/step - loss: 0.1207 -
accuracy: 0.9596 - val_loss: 0.9956 - val_accuracy: 0.8050
Epoch 47/60
3342/3342 [==============================] - 29s 9ms/step - loss: 0.0621 -
accuracy: 0.9803 - val_loss: 0.9800 - val_accuracy: 0.8337
Epoch 48/60
3342/3342 [==============================] - 28s 8ms/step - loss: 0.0387 -
accuracy: 0.9865 - val_loss: 0.9581 - val_accuracy: 0.8433
Epoch 49/60
3342/3342 [==============================] - 29s 9ms/step - loss: 0.0345 -
accuracy: 0.9871 - val_loss: 1.1804 - val_accuracy: 0.8301
Epoch 50/60
3342/3342 [==============================] - 29s 9ms/step - loss: 0.0514 -
accuracy: 0.9829 - val_loss: 1.0350 - val_accuracy: 0.8254
Epoch 51/60
3342/3342 [==============================] - 30s 9ms/step - loss: 0.0472 -
accuracy: 0.9850 - val_loss: 1.0631 - val_accuracy: 0.8433
Epoch 52/60
3342/3342 [==============================] - 31s 9ms/step - loss: 0.0176 -
accuracy: 0.9928 - val_loss: 0.9804 - val_accuracy: 0.8481
Epoch 53/60
3342/3342 [==============================] - 32s 9ms/step - loss: 0.0350 -
accuracy: 0.9892 - val_loss: 1.1719 - val_accuracy: 0.8409
Epoch 54/60
3342/3342 [==============================] - 26s 8ms/step - loss: 0.0346 -
accuracy: 0.9877 - val_loss: 0.9682 - val_accuracy: 0.8397
Epoch 55/60
3342/3342 [==============================] - 26s 8ms/step - loss: 0.0519 -
accuracy: 0.9856 - val_loss: 0.9546 - val_accuracy: 0.8409
Epoch 56/60
3342/3342 [==============================] - 28s 9ms/step - loss: 0.0530 -
accuracy: 0.9803 - val_loss: 1.0716 - val_accuracy: 0.8062
Epoch 57/60
```

```
3342/3342 [==============================] - 28s 8ms/step - loss: 0.0713 -
accuracy: 0.9782 - val_loss: 1.3906 - val_accuracy: 0.7656
Epoch 58/60
3342/3342 [==============================] - 28s 8ms/step - loss: 0.0460 -
accuracy: 0.9832 - val_loss: 1.2043 - val_accuracy: 0.8337
Epoch 59/60
3342/3342 [==============================] - 28s 8ms/step - loss: 0.0514 -
accuracy: 0.9829 - val_loss: 1.3013 - val_accuracy: 0.8242
Epoch 60/60
3342/3342 [==============================] - 28s 8ms/step - loss: 0.0453 -
accuracy: 0.9850 - val_loss: 1.0828 - val_accuracy: 0.8409
```

[23]:

```
<keras.callbacks.callbacks.History at 0x1c71c827fd0>
```

3.4 Save the model:

[24]:

```python
# Serialise model to JSON
model_json = model.to_json()
with open('model.json', 'w') as json_file:
    json_file.write(model_json)
```

[25]:

```python
# Serialise weights to HDF5
model.save_weights('model.h5')
print('Saved trained model to disk')
```

```
Saved trained model to disk
```