

Bowen Chen

by Bowen Chen

Submission date: 04-May-2023 01:08AM (UTC+0100)

Submission ID: 204612349

File name: 41332_Bowen_Chen_Bowen_Chen_498640_87462362.pdf (578.2K)

Word count: 3226

Character count: 17363

Facial Expression Recognition on Fer2013 Dataset

Bowen Chen

Student number:2200373

Abstract:

This convolutional neural network (CNN) training used the Fer2013 dataset for emotion classification task, which contains 35,887 facial images, choose four subset labeled as "happy", "neutral", "angry", and "surprise" for training and testing. In terms of data preprocessing, each image was converted to grayscale and normalized. The CNN model was then trained and validated, consisting of multiple layers. Add dropout layer to avoid overfitting. During the training process, Adam optimizer and cross-entropy loss function were utilized, batch size and other hyperparameters were adjusted. The experimental results showed that the overall accuracy of the model on the test set was 72%. Specifically, the model performed well in the "happy" and "angry" categories, but poorly in the "neutral" and "surprise" categories. This may be due to the unbalanced distribution of features in the dataset, making it difficult for the model to learn the subtle features of the "neutral" and "surprise" categories. In addition, the model architecture may not be well-suited for detecting these features.

Key words: Neural Networks; CNN; Facial expression recognition; Fer2013;

1.Introduction of CNN

In the early 1980s to early 1990s, the predecessor of CNN - Neocognitron - was proposed. This model was put forward in a paper published by Japanese scholar Kunihiko Fukushima in 1980, and is a type of neural network model that can be used for handwritten digit recognition. Convolutional Neural Network (CNN) is a type of artificial neural network (ANN) commonly used in deep learning applications for processing images and videos. CNNs are particularly effective in computer vision tasks such as image classification, object detection, and facial recognition. The development of CNN can be traced back to the 1980s when it was initially proposed by Yann LeCun and others for handwritten digit recognition. During the development of CNN, some classic network structures were proposed, including:

- (1) LeNet-5 (1998): proposed by Yann LeCun and others, it was the first successful CNN network structure applied to solve practical problems. It used convolutional layers, pooling layers, and fully connected layers, and was trained using backpropagation algorithm^[1].
- (2) AlexNet (2012): proposed by Alex Krizhevsky and others, it was the first network structure that achieved significant results in the large-scale visual recognition competition (ImageNet). AlexNet introduced the ReLU activation function and Dropout technique, and was trained using GPU^[2].
- (3) VGGNet (2014): proposed by Simonyan and Zisserman, it is characterized by using very small convolutional kernels (3x3), which makes the network deeper without significantly increasing the number of parameters. VGGNet also uses fully connected layers for classification^[3].
- (4) GoogLeNet (2014): proposed by Szegedy and others, it was the first network structure that used the Inception module. The Inception module combines multiple convolutional kernel sizes and pooling methods, which improves the performance of the network^[4].
- (5) ResNet (2015): proposed by Kaiming He and others, it is characterized by using residual connections to solve the problems of gradient disappearance and overfitting caused by the increase of network depth. ResNet increases the depth of the network by adding skip connections in residual blocks^[5].

CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layer is the core of the CNN and is responsible for detecting features in the input image. Each convolutional layer consists of multiple filters, which convolve over the input image and generate a feature map. The pooling layer is used to reduce the size of the feature map, and the fully connected layer is used to classify the image based on the features extracted from the previous layers. One of the key advantages of CNNs is their ability to learn hierarchical representations of the input data. The lower layers of the network learn simple features such as edges and corners, while the higher layers learn more complex features such as facial expressions and object shapes. This hierarchical representation allows the network to achieve high accuracy in image classification tasks^[7].

CNNs are trained using a technique called backpropagation, which adjusts the weights of the network to minimize the difference between the predicted output and the actual output. The weights are updated iteratively using an optimization algorithm such as stochastic gradient descent (SGD). In recent years, CNNs have achieved state-of-the-art performance in many computer vision tasks and have been widely used in various applications, including self-driving cars, medical image analysis, and natural language processing^[8].

2. About the dataset

The Fer2013 dataset [6] is a widely used image dataset for facial expression recognition (FER) tasks, which contains 35,887 grayscale images of faces in 48x48 pixel resolution. The images are classified into seven different facial expression categories, including angry, disgust, fear, happy, sad, surprise, and neutral. This dataset was created by the researchers from Google in 2013, and it has become one of the standard benchmark datasets for FER research.

It is publicly available and can be downloaded from various online sources. It was originally collected from the online image sharing website, Flickr, using search queries for the above mentioned facial expressions. The dataset is annotated manually by human labelers, and each image has a corresponding facial expression label. The dataset is divided into three subsets, including a training set, a validation set, and a test set. The training set consists of 28,709 images, while the validation and test sets contain 3,589 images each.

This dataset has been widely used in research on FER, as well as in the development of machine learning models for facial expression recognition. The dataset has been used to train and test various models, including convolutional neural networks (CNNs), support vector machines (SVMs), and deep belief networks (DBNs). Many researchers have achieved state-of-the-art results on the Fer2013 dataset using CNNs, which have been shown to be particularly effective in FER tasks. Despite its popularity, the Fer2013 dataset has some limitations. For example, the dataset contains relatively few images for certain expression categories, such as disgust and fear, which can affect the performance of machine learning models. Additionally, the dataset only includes images of faces in a specific age range, which limits its generalizability to other age groups. Nevertheless, the Fer2013 dataset remains an important resource for FER research and provides a standard benchmark for evaluating new FER models.

3. Load the data

I chose to download this dataset on Kaggle, where the Fer2013 dataset is available in the form of CSV files, making it easier for us to load the data. The Fer2013 dataset contains 35,887 grayscale facial images of size 48x48 pixels. Grayscale refers to the brightness or gray level of an image, indicating the degree to which the brightness value of each pixel falls between black and white. In grayscale images, each pixel is typically represented by an 8-bit integer that indicates its level of grayness, with a range of values usually from 0 to 255, where 0 represents black and 255 represents white, and intermediate values represent different levels of grayness. Grayscale images are a type of image that falls between black and white images and color images, as they only contain gray level information and not color information. Color images, on the other hand, are composed of three

basic colors, red, green, and blue, and are also known as RGB images. The color of each pixel is determined by the intensity values of the three color channels, each of which typically ranges from 0 to 255 and represents the strength or brightness of the color. By combining the values of these three color channels, different colors can be obtained, resulting in a rich and colorful image. Compared to grayscale images, color images contain more information and can better reflect the colors and details of real scenes. It is evident that the input data format of the Fer2013 dataset is 48x48x1. In the Fer2013.csv file, the data is divided into training, validation, and test sets, with a total of seven facial expression categories.

Firstly, load the dataset into a variable of data frame type, read the number of data, and we can get that the entire dataset contains 35887 samples. Display one figure, shown as below.

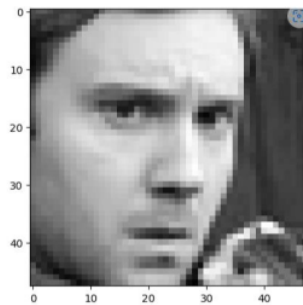


Fig 1. angry emotions

4. choose four subset to train and test

The FER2013 dataset consists of seven basic emotional expressions, namely anger, disgust, fear, happiness, sadness, surprise, and neutral expressions. If we choose to train our model using all seven expressions, it may result in a complex model that finds it difficult to learn the differences between each expression. Next, I need to choose 4 kind of emotions to train and test. As I said above, there are 7 kind of Facial emotions categories in the Fer2013 dataset. So which 4 to choose? I try to train the dataset with all of the seven kind of emotions. Finally get a confusion matrix which shows that my model have better performance on the four categories named 'happy', 'neutral', 'sad', 'surprised'. In order to get a better performance, I choose the 'happy', 'neutral', 'sad', 'surprised' subset to train and test.

Happiness is a common emotion and is easy to recognize. Sadness is another common emotion that can help the model learn a richer range of emotions. Anger is a more intense emotion, and training the model to recognize angry expressions can improve its robustness and usefulness in certain applications. Although neutral expressions are not very common, they still need to be correctly classified if we are to use the trained model for testing in practical applications. Therefore, above are the other reasons we need to include them in the training dataset.

```

for i in tqdm(range(num_of_instances)):
    usages_name = usages[i]
    emotions_Str_Name = emotions_Str[emotions[i]]

    one_hot_label = to_categorical(emotions[i], num_classes) # Select 4 kind of emotions to train and test

    img = list(map(eval, pixels[i].split(' ')))
    np_img = np.asarray(img)
    img = np_img.reshape(48, 48)
    if emotions[i] == 3 or emotions[i] == 4 or emotions[i] == 5 or emotions[i] == 6: #choose: happy sad surprise neutral
        if usages[i] == 'Training':
            x_train.append(img)
            y_train.append(one_hot_label)
        elif usages[i] == 'PrivateTest':
            x_val.append(img)
            y_val.append(one_hot_label)
        else:
            x_test.append(img)
            y_test.append(one_hot_label)

```

Fig 2. python code

5. Data pre-processing

```

x_train = np.array(x_train)
y_train = np.array(y_train)
x_train = x_train.reshape(-1, 48, 48, 1)
x_test = np.array(x_test)
y_test = np.array(y_test)
x_test = x_test.reshape(-1, 48, 48, 1)
x_val = np.array(x_val).reshape(-1, 48, 48, 1)
y_val = np.array(y_val)

```

Fig 3 Data pre-processing python code

Specifically, it performs the following steps:

- (1) Convert the image data and labels of the training and testing sets into NumPy arrays (np.array) so that they can be operated on as arrays.
- (2) Resize the image data of the training, testing, and validation sets to the specified size (48x48 pixels) and convert them from one-dimensional arrays to two-dimensional arrays. This is because we will use a convolutional neural network (CNN) to train our model, which requires a three-dimensional array as input (image width x image height x color channels).

Through these preprocessing steps, we can transform the FER2013 dataset into a format suitable for training a CNN model, and perform more convenient and efficient array operations.

And I did perform normalization on the data. Normalization can improve the training accuracy of the model because it can give different features (or pixels) the same dimensionality during the use of optimization algorithms such as gradient descent, avoiding unstable training caused by differences between different features. At the same time, normalization also helps to accelerate the training process of the model, because normalization can make the optimization algorithm converge

faster, thereby reducing the training time of the model. Therefore, normalization is usually considered an important preprocessing step in machine learning and deep learning, which can improve the training performance and accuracy of the model

More, As shown in Fig2, I have used one-hot encoding for the class labels. Using numerical representations to indicate the size relationship between categories may mislead the learning process of neural networks. For example, numbering categories from 0 to 9 may lead the neural network to believe that there is some quantity-based relationship between categories. Using one-hot encoding can eliminate this bias. One-hot encoding ensures that the distance between each category is equal, which can be useful for some tasks such as clustering. This can also enhance the model's generalization ability, and for categories that have not appeared in the training dataset, one-hot encoding can make it easier for neural networks to classify them into the correct category.

6. My CNN model

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 48, 48, 32)	320
conv2d_19 (Conv2D)	(None, 48, 48, 32)	9248
max_pooling2d_9 (MaxPooling2D)	(None, 24, 24, 32)	0
conv2d_20 (Conv2D)	(None, 24, 24, 64)	18496
conv2d_21 (Conv2D)	(None, 24, 24, 64)	36928
max_pooling2d_10 (MaxPooling2D)	(None, 12, 12, 64)	0
conv2d_22 (Conv2D)	(None, 12, 12, 128)	73856
conv2d_23 (Conv2D)	(None, 12, 12, 128)	147584
max_pooling2d_11 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten_3 (Flatten)	(None, 4608)	0
dense_6 (Dense)	(None, 64)	294976
dropout_3 (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 7)	455

Total params: 581,863
Trainable params: 581,863
Non-trainable params: 0

Fig 4 My CNN model

The above is my convolutional neural network structure. When processing the FER2013 dataset, the model's input is a single -channel gray image of 48x48. The model uses multiple convolutional layers, pooling layers and full connection layers. Specifically, this model includes three volume blocks. Each volume block consists of two convolutional layers and a maximum

pooling layer. The number of convolution cores in the first part is 32, and the second part is 64. The third part is 128. Each convolutional layer uses 3x3 convolution kernel and RELU activation functions. The pooling layer uses the 2x2 pooling core and is set to 2. Next is a full connection layer, connecting 64 neurons, and using the RELU activation function and 50%Dropout. The last full connection layer outputs 7 different emotional categories, and the probability of output the SoftMax activation function outputs each category. During training, the ADAM optimizer and cross -entropy loss function are used to train.

7. Results and analysis

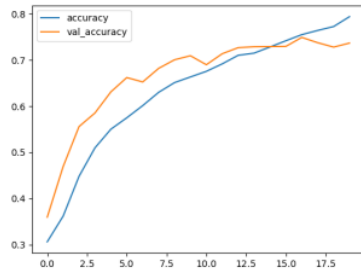


Fig 5 Group 1 training process

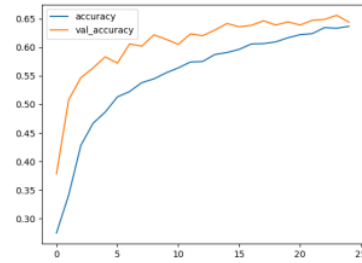


Fig 6 Group 2 training process

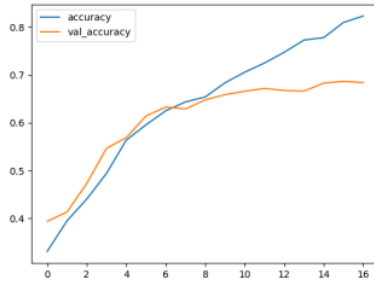


Fig 7 Group 3 training process

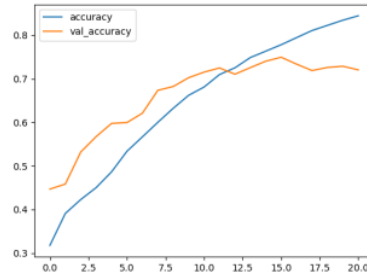


Fig 8 Group 4 training process

Overall classification accuracy on the test set:

Group1: 72%

Group2: 64%

Group3: 67%

Group4: 72%

- Group 1: CNN model as shown in part 6. Batch size is 512, early stopping employed. The others are as default.
- Group 2: CNN model as shown in part 6, and remove two convolutional layers and one pooling layer, also reduce the number of convolutional filters.
- Group 3: CNN model as shown in part 6, and remove the drop out layer
- Group 4: CNN model as shown in part 6, Batch size modified to 128.

The above four figures show the curves of the accuracy of the model on the training and validation sets over epochs during the training process. All groups compile the neural network model with the categorical cross-entropy loss function, Adam optimizer, and accuracy as the evaluation metric. The categorical cross-entropy loss function is commonly used for multi-class classification tasks, while the Adam optimizer is an efficient algorithm for optimizing the model's parameters during training^[9]. The accuracy metric is used to measure how well the model can predict the correct output class.

Firstly, I introduced the early stopping callback function during the training process, which stops the training if the accuracy metric doesn't improve for 5 consecutive epochs. When the model starts to overfit, its performance on the training set continues to improve, but its performance on the validation set starts to decline. Through early stopping, we can stop the training before the model starts to overfit, thus avoiding overfitting.

Based on figures 5 and 6, we can observe that a simple network architecture with fewer layers and convolutional filters can effectively avoid overfitting, but may not achieve high accuracy on the test set. As the network structure becomes more complex and the number of convolutional filters increases, the model's prediction accuracy improves but overfitting may also occur. Additionally, the training speed can become slower with more complex architectures.

In the experimental group of image 5, a dropout layer was added to the network, while in the experimental group of image 7, it was removed. It can be observed that the dropout layer can significantly prevent overfitting. Dropout is a technique used in neural networks to prevent overfitting. During training, some randomly selected neurons are ignored or "dropped out" with a probability, and their contribution to the activation of downstream neurons is temporarily removed. This prevents the network from relying too much on specific neurons and encourages it to learn more robust features that are useful in making predictions on unseen data.

Based on Figure 5 and Figure 8, comparing Group 1 and Group 4, the batch size decreased from 512 to 128. When the batch size is small, fewer samples are used for each iteration of training, so the training time per epoch is longer. However, in the same amount of time, the model can process more iterations, leading to faster convergence. Conversely, when the batch size is large, more samples are used for each iteration, so the training time per epoch is shorter, but the convergence speed of the model may be slower. In addition, a larger batch size can help the model estimate gradients more stably because the gradient estimates for each batch are more accurate. However, a larger batch size consumes more memory and may cause the model to get stuck in a local optimum because it may not explore enough parameter space.

	precision	recall	f1-score	support
3	0.81	0.84	0.82	895
4	0.61	0.64	0.62	653
5	0.82	0.84	0.83	415
6	0.62	0.54	0.57	607
accuracy			0.72	2570
macro avg	0.71	0.71	0.71	2570
weighted avg	0.71	0.72	0.71	2570

Fig 9 The confusion matrix of group 1

Precision refers to the proportion of true positive samples in the samples predicted as positive. Recall refers to the proportion of true positive samples among all positive samples. F1 score is the harmonic mean of precision and recall. Label 3 represents "happy", Label 4 represents "neutral", Label 5 represents "angry", and Label 6 represents "surprised"

According to Fig 9, my model has an overall accuracy of 72% on the test set. It can be seen that the model performs well on the "happy" and "angry" categories, but poorly on the "neutral" and "surprise" categories. Without considering the difference in sample sizes, the possible reason why the CNN performs well in the "happy" and "angry" categories but poorly in the "neutral" and "surprise" categories may be due to the unbalanced distribution of features in the dataset. The CNN model may have learned the dominant features of the "happy" and "angry" categories well, but failed to capture the subtle features of the "neutral" and "surprise" categories. It is also possible that the model architecture is not well-suited for detecting the features of the "neutral" and "surprise" categories.

8. conclusion

The experimental results showed that the overall accuracy of the model on the test set was 72%. Specifically, the model performed well in the "happy" and "angry" categories, but poorly in the "neutral" and "surprise" categories. The possible reason may be due to the unbalanced distribution of features in the dataset. The CNN model may have learned the dominant features of the "happy" and "angry" categories well, but failed to capture the subtle features of the "neutral" and "surprise" categories. It is also possible that the model architecture is not well-suited for detecting the features of the "neutral" and "surprise" categories. Due to the limited computing power of the GPU, I am unable to experiment with more complex network structures.

References

- [1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, November 1998.
- [2] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems 25 (NIPS 2012), pp. 1097-1105, 2012.
- [3] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. Proceedings of the International Conference on Learning Representations (ICLR), 2015.
- [4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1-9, 2015.
- [5] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770-778, 2016.
- [6] <https://www.kaggle.com/datasets/nicolejyt/facialexpressionrecognition>
- [7] https://en.wikipedia.org/wiki/Convolutional_neural_network
- [8] https://en.wikipedia.org/wiki/Convolutional_neural_network#Applications
- [9] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam
- [10] https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy

Appendix

My video is submitted via moodle

```
In [1]: import numpy as np
import pandas as pd

import os
os.environ["OMP_NUM_THREADS"]="1"

data = pd.read_csv('G:/data_csv/FER2013/fer2013.csv')

num_of_instances = len(data) #get the number of the samples
print('The number of samples: ', num_of_instances)

pixels = data['pixels']
emotions = data['emotion']
usages = data['usage']

The number of samples: 35887

In [2]: import cv2
img0 = list(map(eval, pixels[0].split(' ')))
np_img0 = np.asarray(img0)
img0 = np_img0.reshape(48, 48)
import matplotlib.pyplot as plt
plt.imshow(img0, cmap="gray")

Out[2]: <matplotlib.image.AxesImage at 0x22bd35e1520>

In [3]: emotions_Str=['anger','disgust','fear','happy','neutral','sad','surprised']

In [4]: from keras.utils import to_categorical
from PIL import Image
import os

num_classes = 7 # 7 kind of emotions
x_train, y_train, x_val, y_val, x_test, y_test = [], [], [], [], [], []
from tqdm import tqdm

In [5]: for i in tqdm(range(num_of_instances)):
    usages_name = usages[i]
    emotions_Str_Name = emotions_Str[emotions[i]]

    one_hot_label = to_categorical(emotions[i], num_classes) # Select 4 kind of emotions to train and test

    img = list(map(eval, pixels[i].split(' ')))
    np_img = np.asarray(img)
    img = np_img.reshape(48, 48)
    if emotions[i] == 3 or emotions[i] == 4 or emotions[i] == 5 or emotions[i] == 6: #choose: happy sad surprise neutral
        if usages[i] == 'Training':
            x_train.append(img)
            y_train.append(one_hot_label)
        elif usages[i] == 'PrivateTest':
            x_val.append(img)
            y_val.append(one_hot_label)
        else:
            x_test.append(img)
            y_test.append(one_hot_label)

100%|████████████████████| 35887/35887 [07:45<00:00, 77.02it/s]

In [56]: x_train = np.array(x_train)
y_train = np.array(y_train)
x_train = x_train.reshape(-1, 48, 48, 1)
x_test = np.array(x_test)
y_test = np.array(y_test)
x_test = x_test.reshape(-1, 48, 48, 1)
x_val = np.array(x_val).reshape(-1, 48, 48, 1)
y_val = np.array(y_val)

In [57]: len(x_train), len(x_test), len(x_val)

Out[57]: (20181, 2570, 2515)

In [58]: import matplotlib.pyplot as plt
%matplotlib inline

for i in range(4):
    plt.subplot(221+i)
    plt.gray()
    plt.imshow(x_train[i].reshape([48, 48]))
```

```
In [59]: x_train.shape[1:]
x_val.shape

Out[59]: (2515, 48, 48, 1)
```

```
In [65]: from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator

batch_size = 512
epochs = 40

model = Sequential()

model.add(Conv2D(input_shape=(48, 48, 1), filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Conv2D(filters=128, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(filters=128, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Flatten())

# Fully-connected layers
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
```

```
In [66]: model.summary()
```

```
In [67]: from keras.callbacks import EarlyStopping
```

```
In [68]: #Train
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])

# early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_val, y_val), callbacks=[early_stopping])

test_score = model.evaluate(x_test, y_test)
```

```
In [70]: plt.plot(history.epoch, history.history.get('accuracy'), label='accuracy')

plt.plot(history.epoch, history.history.get('val_accuracy'), label='val_accuracy')

plt.legend()

# save this fig
plt.savefig('G:/data_crv/FER2013/accuracy.png')
```

```
In [77]: import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Evaluate function:
def evaluate(estimator, X, y):

    # Use the trained model to predict the classes for the test images
    predictions = estimator.predict(X)
    # Convert the probability distribution for each sample to a single predicted class label
    predicted_labels = np.argmax(predictions, axis=1)

    y_true_labels = np.argmax(y, axis=1)

    # Calculate the classification accuracy of the model on the test set
    accuracy = np.mean(predicted_labels == y_true_labels)

    y_pred = predicted_labels
    print(classification_report(y_true_labels, y_pred))

    # save the classification report
    fig = plt.figure(figsize=(8,6))
    plt.text(0.01, 0.6, classification_report(y_true_labels, y_pred), {'fontsize':12, 'fontfamily':'monospace'})
    plt.axis('off')
    plt.tight_layout()
    plt.savefig('G:/data_csv/FER2013/classification_report.png')

    # Confusion matrix:
    con_matrix = pd.crosstab(pd.Series(y_true_labels, name='Actual'), pd.Series(y_pred, name='Predicted'))
    plt.figure(figsize=(8,6))
    plt.title('Confusion Matrix')
    sns.heatmap(con_matrix, cmap='Blues', annot=True, fnt='g')

    return y_pred
```

```
In [78]: x_test.shape
```

```
Out[78]: (2570, 48, 48, 1)
```

```
In [79]: y_test.shape
```

```
Out[79]: (2570, 7)
```

```
In [80]: evaluate(model, x_test, y_test):
```

```
0.91081 1.0000000000000000 - do d'ne/etun
```


FINAL GRADE

72/100

GENERAL COMMENTS

Instructor

In abstract, good to you including some summary results.

Refs are appropriate and used well in general. Minor formatting issue with in line citation shown as a superscript.

In general well structured and written report.

CSV is fine but using images and tensorflow's library it is possible to argument the data and incorporate other transforms whilst avoiding running out of memory.

Regarding related work, report further on other papers that experimented on the same dataset with similar networks. These could be used to explicitly validate your results.

Some appropriate theory and related concepts identified and discussed and used to guide experimental design.

You spend some time explaining/justifying your class selections and experiment with your choices showing they can affect the the training success - accuracy, etc of your network. Good - why not present a brief summary as a table? Better still use bar plots to compare. Selecting the four classes you finally chose based on a criteria to maximize the accuracy is one idea.. You also chose other criteria and give some justification.

A nice set of experiments adjusting parameters in an informed way. Well done. Comparing your own with other established architectures. would be an

alternative approach to take if you had access to more powerful GPUs.

Discussion of using pre-trained network weights (transfer learning) could be developed included.

Use of further pre-processing and potential of dimensionality reduction could be included.

The relevance of validation data in optimising number of epochs of training provided. Good.

Avoid use of "I" (eg in conclusion) - third person is preferred in formal reports.

Overall a very good effort. Well done.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9



Comment 1

???

PAGE 10

PAGE 11

GRADING FORM: SMU GENERIC_GJ

BOWEN CHEN

0

RESEARCH-INFORMED LITERATURE

KNOWLEDGE AND UNDERSTANDING

ANALYSIS

PRACTICAL APPLICATION AND DEP.

SKILLS FOR PROFESSIONAL PRAC.