Stress Image Classification

Name: Tong Luyangyu

Affiliation: Electrical Engineering and Information Systems, Graduate School of Engineering

Student Number: 37245056

E-mail Address: tong-luyangyu5056@g.ecc.u-tokyo.ac.jp

1 Problem description

The problem is that how to classify the images into two categories: stress or non-stress. The reason why I want to do this task is that it is important to care about the mental health. Nowadays, many people have mental issues, but they usually ignore their actual feelings. For example, students may feel much anxiety due to exams or too much homework, if they continue to put themselves in those moods, the situation will become worse and worse. Finally, they may drop out the school or even worse actions.

The dataset is from Kaggle. The name of the dataset is “Stress Non Stress Images”. And the link to the dataset is https://www.kaggle.com/datasets/preritbhagat/stress-non-stress-images.

Figure 1 Five stress levels

2 Problem solution

2.1 Convolutional neural network

First, I tried to import the dataset and visualized the images in the dataset to see what the images are like. The images in the dataset are all labeled and are divided into two classes (stress and non-stress). The samples are shown in Figure 2.

Figure 2 Samples in the dataset

Because convolutional neural network shows good capability in the tasks of image classification. So, then I built up a convolutional neural network (CNN) to classify the images into two classes. The structure of the CNN is shown in the Table 1, as there are only two classes in this task, so the output is set to 2. The batch size is set to 32 so the input is set to 32.

Layer (type)	Output Shape	Param #
resizing (Resizing)	(32, 256, 256, 3)	0
rescaling (Rescaling)	(32, 256, 256, 3)	0
conv2d (Conv2D)	(32, 254, 254, 32)	896
max_pooling2d (MaxPooling2D)	(32, 127, 127, 32)	0
conv2d_1 (Conv2D)	(32, 125, 125, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(32, 62, 62, 64)	0
conv2d_2 (Conv2D)	(32, 60, 60, 64)	36,928
max_pooling2d_2 (MaxPooling2D)	(32, 30, 30, 64)	0
flatten (Flatten)	(32, 57600)	0
dense (Dense)	(32, 64)	3,686,464
dropout (Dropout)	(32, 64)	0
dense_1 (Dense)	(32, 2)	130

Total params: 3,742,914 (14.28 MB)
Trainable params: 3,742,914 (14.28 MB)
Non-trainable params: 0 (0.00 B)

Table 1 Structure of CNN model

Next, I used the images in the “train” file folder to train the model and used the images in the “test” file folder to test the model. **After 10 epochs the accuracy is**

about 98% while loss is about 0.1. The test accuracy is about 90% and the test loss is about 0.33. The training accuracy curve is shown in the Figure 3.

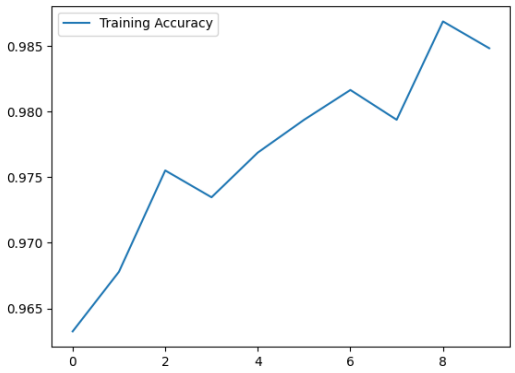

Figure 3 Training accuracy curve (Only CNN)

I also visualized 9 pictures to show the comparison of the predicted results and actual results. The picture is shown in Figure 4:

Figure 4 Predicted results vs actual results (Only CNN)

2.2 Combined CNN with Transformer

Furthermore, as CNN has strong capability to extract the features in the images, and the images in the dataset are taken continuously. Emotions in one person's face are recorded continuously. For example, the person may be firstly told to be angry, which

means stress and then to be happy, which means non-stress. In this case, I think about the RNN and LSTM we have learned in the lecture. Meanwhile I also think about the Transformer. These models show good capability to deal with time series data. Finally, I chose to use Transformer combined with CNN to classify the images. The reason why I do so is that I can firstly use CNN to extract the features in the images and then input these features into Transformer and use the Transformer to deal with the global dependence.

So, first I built up the model combined with CNN and Transformer. The structure of the model is shown in the Table 2.

Layer (type)	Output Shape	Param #
input_layer_19 (InputLayer)	(None, 256, 256, 3)	0
resizing_17 (Resizing)	(None, 256, 256, 3)	0
rescaling_16 (Rescaling)	(None, 256, 256, 3)	0
conv2d_48 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_48 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_49 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_49 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_50 (Conv2D)	(None, 60, 60, 128)	73,856
max_pooling2d_50 (MaxPooling2D)	(None, 30, 30, 128)	0
flatten_16 (Flatten)	(None, 115200)	0
dense_41 (Dense)	(None, 128)	14,745,728
reshape_7 (Reshape)	(None, 1, 128)	0
transformer_block_7 (TransformerBlock)	(None, 1, 128)	396,032
global_average_pooling1d_3 (GlobalAveragePooling1D)	(None, 128)	0
dropout_30 (Dropout)	(None, 128)	0
dense_44 (Dense)	(None, 2)	258

Table 2 Structure of model combined with CNN and Transformer

Next, I used the images in the “train” file folder to train the model and used the images in the “test” file folder to test the model. However, the results seem to be worse than only CNN. **After 10 epochs the accuracy is about 77% while loss is about 0.48. The test accuracy is about 90% and the test loss is about 0.27.** The training accuracy curve is shown in the Figure 5.

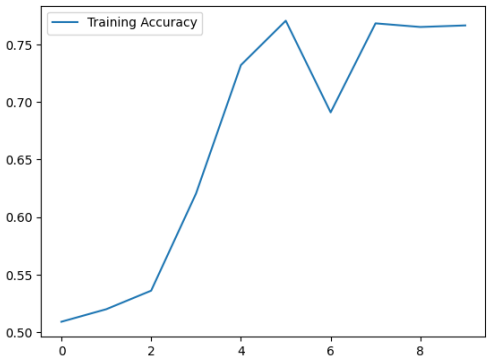

Figure 5 Training accuracy curve (CNN + Transformer)

As Figure 6 shows, I also visualized 9 pictures to show the comparison of the predicted results and actual results.

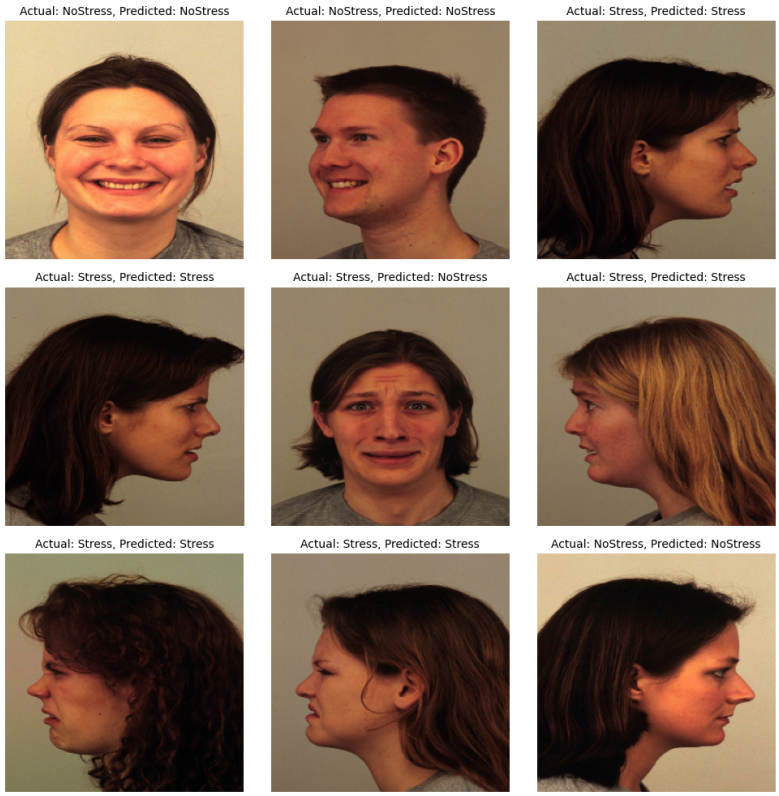

Figure 6 Predicted results vs actual results (CNN + Transformer)

3 Discussion

It seems to be a not very sensible way to try this. Because the structure of the model becomes more complicated while the accuracy is even worse. After analysis, the reason why this situation occurs may include:

1. The complexity of the model is too high, as the dataset is not very huge, so the overfitting problem may occur. The self-attention mechanism of Transformer is especially easy to be influenced.
2. The features of data may not be suitable to use Transformer, the Transformer may even interfere CNN to extract the features of images.
3. The optimizer or the learning rate is not fitted the Transformer very well.

In conclusion, I tried two methods to classify the images into two classes, stress and non-stress. **For CNN, the accuracy can reach up to 98% and for CNN combined with Transformer, the accuracy can reach up to 77%.** The reason why the performance of combined model is worse may be due to small dataset and inappropriate parameters.

Stress_Image_Classificatioin

January 22, 2025

[89]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from tensorflow import keras
```

[ ]:
```
import kagglehub

# Download the dataset
path = kagglehub.dataset_download("preritbhagat/stress-non-stress-images")

print("Path to dataset files:", path)
```

[ ]:
```
import os
current_path = os.getcwd()
print("Current path:", current_path)

# Importing the dataset
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    './KDEF/KDEF/Train',
    shuffle=True,
    batch_size=32,
)

print(len(dataset))
```

[ ]:
```
# Two types of images in the dataset
class_titles = dataset.class_names
class_titles
```

[ ]:
```
# Displaying some images
plt.figure(figsize=(10, 10))

for image, label in dataset.take(1): # take(1) takes the first batch of 32␣
 ↪images
    labels = label.numpy()
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
```

```
        plt.imshow(image[i].numpy().astype("uint8"))
        plt.title(class_titles[labels[i]], fontsize=10)
        plt.axis("off")
```

[ ]:
```
# Build up the CNN model
model = tf.keras.Sequential([
    # Preprocessing layer
    tf.keras.layers.Resizing(256, 256),
    tf.keras.layers.Rescaling(1./255),

    # Convolutional layers and pooling layers
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',␣
 ↪input_shape=(32, 256, 256, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    # Dense layers
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5), # Dropout layer to prevent overfitting
    tf.keras.layers.Dense(2, activation='softmax')
])

model.build((32, 256, 256, 3))

model.summary()
```

[ ]:
```
# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(
    dataset,
    batch_size = 32,
    verbose = 1,
    epochs = 10
)
```

[ ]:
```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.legend()
plt.show()
```

[ ]:
```
# import the test dataset
dataset_test = tf.keras.preprocessing.image_dataset_from_directory(
    './KDEF/KDEF/Test',
    shuffle=True,
    batch_size=32,
)

# Evaluate the model
test_loss, test_acc = model.evaluate(dataset_test)
print(f'The test accuracy is: {test_acc} and the test loss is: {test_loss}')
```

[ ]:
```
# Visualize the predicted types vs actual types
plt.figure(figsize=(10, 10))

prediction_dataset = dataset_test.skip(5).take(1)

for image, label in prediction_dataset:
    labels = label.numpy()
    predictions = model.predict(image)

    batch_size = image.shape[0]
    for i in range(min(batch_size, 9)):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(image[i].numpy().astype("uint8"))
        predicted_label = np.argmax(predictions[i])
        actual_label = labels[i]
        plt.title(f'Actual: {class_titles[actual_label]}, Predicted:␣
 ↪{class_titles[predicted_label]}', fontsize=10)
        plt.axis("off")

plt.tight_layout()
plt.show()
```

[98]:
```
# Try to combine the transformer model with the CNN model
# Build the transformer model
# Transformer Encoder Block
class TransformerBlock(tf.keras.layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = tf.keras.layers.MultiHeadAttention(num_heads=num_heads,␣
 ↪key_dim=embed_dim)
        self.ffn = tf.keras.Sequential([
            tf.keras.layers.Dense(ff_dim, activation="relu"),
            tf.keras.layers.Dense(embed_dim),
        ])
        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
```

```
        self.dropout1 = tf.keras.layers.Dropout(rate)
        self.dropout2 = tf.keras.layers.Dropout(rate)

    def call(self, inputs, training=None):  # Ensure `training` is passed
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)
```

[ ]:
```
# Combine CNN and Transformer
def build_combined_model(input_shape=(256, 256, 3), num_classes=2):
    inputs = tf.keras.Input(shape=input_shape)

    # Step 1: CNN Feature Extraction
    x = tf.keras.layers.Resizing(256, 256)(inputs)
    x = tf.keras.layers.Rescaling(1./255)(x)
    x = tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu')(x)
    x = tf.keras.layers.MaxPooling2D((2, 2))(x)
    x = tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu')(x)
    x = tf.keras.layers.MaxPooling2D((2, 2))(x)
    x = tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu')(x)
    x = tf.keras.layers.MaxPooling2D((2, 2))(x)
    x = tf.keras.layers.Flatten()(x)

    # Step 2: Transformer Encoder
    x = tf.keras.layers.Dense(128, activation='relu')(x)  # Embedding
    x = tf.keras.layers.Reshape((1, 128))(x)  # Ensure shape is [batch_size,␣
 ↪seq_len, embed_dim]
    transformer_block = TransformerBlock(embed_dim=128, num_heads=4, ff_dim=512)
    x = transformer_block(x)

    # Step 3: Classification
    x = tf.keras.layers.GlobalAveragePooling1D()(x)  # Reduce sequence dimension
    x = tf.keras.layers.Dropout(0.5)(x)
    outputs = tf.keras.layers.Dense(num_classes, activation='softmax')(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model


# Build and compile the model
combined_model = build_combined_model()
combined_model.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
```

```
combined_model.summary()
```

[ ]:
```
history = combined_model.fit(
    dataset,
    epochs=10,
    batch_size=32,
    verbose=1
)
```

[ ]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.legend()
plt.show()
```

[ ]:
```
 # import the test dataset
dataset_test = tf.keras.preprocessing.image_dataset_from_directory(
    './KDEF/KDEF/Test',
    shuffle=True,
    batch_size=32,
)

# Evaluate the model
test_loss, test_acc = model.evaluate(dataset_test)
print(f'The test accuracy is: {test_acc} and the test loss is: {test_loss}')
```

[ ]:
```
# Visualize the predicted types vs actual types
plt.figure(figsize=(10, 10))

prediction_dataset = dataset_test.skip(5).take(1)

for image, label in prediction_dataset:
    labels = label.numpy()
    predictions = model.predict(image)

    batch_size = image.shape[0]
    for i in range(min(batch_size, 9)):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(image[i].numpy().astype("uint8"))
        predicted_label = np.argmax(predictions[i])
        actual_label = labels[i]
        plt.title(f'Actual: {class_titles[actual_label]}, Predicted:␣
 ↪{class_titles[predicted_label]}', fontsize=10)
        plt.axis("off")

plt.tight_layout()
```

```
plt.show()
```