King Saud University Fall 2024

CSC462 Course Project: Facial Emotion Recognition on FER2013

Lamees Alturki 443200862
Yara Alfouzan 443203087
Section: 72917

1. Facial emotion recognition task

For this project, the chosen image classification task is Emotion Recognition. The goal is to classify images of human faces into discrete emotional categories based on their facial expressions making it a multiclass classification problem. This task is challenging due to the subtle and often ambiguous nature of emotional cues in facial expressions, which can be influenced by unique individual facial characteristics, lighting, and camera angles. Successfully classifying emotions from images has various applications, such as enhancing user experience in virtual settings, aiding in psychological assessments, improving lie detectors, and enhancing the capabilities of social robots. Overall, emotion recognition has the potential to improve interactions across numerous domains, making technology more empathetic and responsive to human needs.

2. FER-2013 Dataset

For this project, we utilized the FER-2013 [1] dataset, which was created by Pierre Luc Carrier and Aaron Courville, and is a significant resource for studying facial expressions in images, developed as part of the Facial Expression Recognition contest. It was generated using the Google image search API to find images of faces corresponding to a set of 184 emotion-related keywords, like "happy" or "angry." They also added words about gender, age, or ethnicity to these keywords. By mixing these terms together, they created almost 600 different search phrases to find a wide variety of facial images online. The dataset contains 35,887 labeled facial images categorized into seven emotional expressions: anger, disgust, fear, happiness, sadness, surprise, and neutral divided into training and testing sets [1].

3. Literature review

N. Mehendale [2] Introduced the FERC (Facial Emotion Recognition using Convolutional Neural Networks) model which employs an effective two-stage CNN structure, separating background removal from facial feature extraction the model works on facial emotion recognition in both images and video frames by extracting the key frames. In the first stage, FERC isolates the face from the background by applying a YCbCr color threshold to detect and isolate skin tones, followed by a Hough transform-based "circles-in-circle" filter. This filter identifies circular facial structures, by detecting point sets that form circles within a specified radius range. This layered filtering isolates the face, thus refining background removal and providing a cleaner input for feature extraction. The second stage focuses on extracting detailed facial features to create an Expressional Vector (EV), capturing essential landmarks like the eyes, mouth, nose, and cheeks. Multiple convolutional layers with 3x3 filters detect these features, measuring the Euclidean distances between key points on the face, which is essential for distinguishing emotional expressions. The resulting EV, a 24-dimensional vector, represents these inter-point distances and serves as the primary input for the model's emotion classifier. A single-layer perceptron then refines the EV by adjusting weights, minimizing classification error and categorizing expressions into one of five target emotions: anger, sadness, happiness, fear, and surprise. FERC was trained on a dataset of 10,000 images, covering the five emotion classes and including well-established datasets like Cohn-Kanade [3], Caltech Faces [4], CMU [5], and NIST [6]. The training process used a 70-30 train-test split, employing 25-fold cross-validation to ensure robust performance across varied data subsets. Hyperparameter tuning demonstrated that an architecture with four CNN layers and four filters per layer achieved an optimal balance between accuracy and computational efficiency. For video analysis, FERC identifies "key frames" with minimal intra-frame changes, selecting frames with the highest edge detail to enhance clarity in feature extraction. The model achieved up to 96% accuracy on the NIST dataset, 85% on Caltech Faces, and 78% on the CMU database. FERC demonstrates a strong potential for real-world applications in emotion detection with notable efficiency. However, some challenges were noted, such as reduced accuracy with images containing shadows, facial hair, or multiple faces, which sometimes led to some misclassifications.

In recent years, several influential CNN architectures have significantly advanced the field of image recognition, including Facial Emotion Recognition (FER). AlexNet, introduced in 2012, was one of the first CNN architectures to demonstrate the power of deep learning in computer vision. It consists of five convolutional layers followed by three fully connected layers and incorporates ReLU (Rectified Linear Unit) activation functions, which improve the model's capacity to learn complex patterns. Additionally, AlexNet includes dropout layers to mitigate overfitting. Building on this foundation, VGGNet introduced a deeper architecture by stacking small 3x3 convolutional filters, allowing for the extraction of more complex features. With variants like VGG16 and VGG19, VGGNet maintains a consistent structure across layers, and it also employs ReLU activations, which contribute to improved feature learning at greater depth. GoogLeNet introduced the innovative Inception modules, which apply multiple filter sizes simultaneously within the same layer. This approach improves feature extraction while maintaining computational efficiency, thanks to the use of global average pooling instead of fully connected layers. ResNet, a transformative architecture in deep learning, introduced a residual learning framework that enables the training of deep networks (up to hundreds of layers). Using skip connections, ResNet preserves information across layers and addresses the vanishing gradient problem, making it foundational in modern deep learning applications [7].

Y. Gan [7] leveraged these advances by using a modified AlexNet architecture for FER on the FER2013 [1] dataset. After modifying AlexNet's final layer to output the seven classes (anger, disgust, fear, happiness, sadness, surprise, neutral), Gan trained the model on a subset of 28,709 images. Experiments included AlexNet, VGGNet, GoogLeNet, and ResNet for comparison, with the modified AlexNet achieving the highest accuracy of 0.64 with 20 passes through the training dataset and a learning rate of 0.001. The analysis noted that while increasing the passes improved accuracy, overfitting remained a concern.

4. Proposed approach

For emotion recognition using the FER-2013 dataset, we will implement a model based on VGGnet a CNN architecture widely used in image processing. The model used will be a variant of this architecture designed to classify facial expressions into distinct emotional categories. We will compare our model to the results Y. Gan [7] achieved. We will experiment with different batch sizes and number of epochs then measure our model's performance by accuracy, recall, precision and F1 score. We will begin with data preprocessing to improve model generalization. Since the dataset is already split into train and test sets, we will use 20% of the training set as a validation set which is used to monitor the model's performance and tune hyperparameters.

5. Model architecture

This project utilizes a custom implementation of the VGGNet, which is a classical convolutional neural network architecture used in large-scale image processing and pattern recognition [8]. The variant of VGGnet used is designed to process grayscale images with dimensions (48 x 48 x 1). The network comprises of 4 convolutional stages and 3 fully connected layers. Each of the convolutional stages contains two convolutional blocks and a max pooling layer at the end of each block to reduce spatial dimensions while retaining critical features. The convolution block consists of 3 convolutional layers, with ReLU activation, and a batch normalization layer [9]. Batch normalization is used to speed up the learning process and prevent gradient vanishing or explosion [10]. The first two fully connected layers are followed by a ReLU activation. The third fully connected layer is for classification. The convolutional stages are responsible for feature extraction, dimension reduction, and non-linearity. The fully connected layers are trained to classify the inputs as described by extracted features [9].

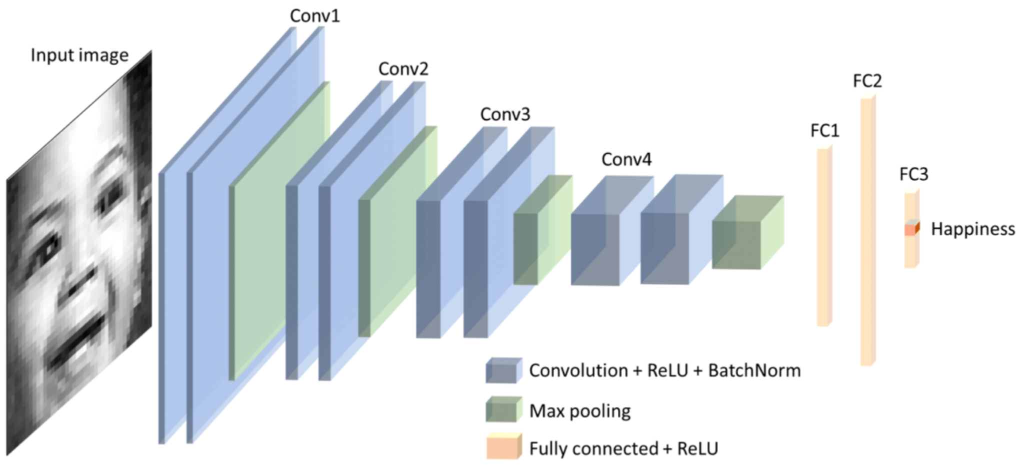


Figure 1: VGGNet variant architecture. The input image is fed into the model, convolutional blocks extract high-level features of the image, and the three fully connected layers classify the emotion of the image.[9]

6. Model development and training

6.1 Data preprocessing

In data preprocessing, the dataset undergoes several preprocessing steps. The grayscale images are resized to (48 x 48) pixels and normalized to have pixel values between 0 and 1 and the training images undergo a variety of transformations to introduce additional variability, enhancing the model's ability to generalize across different conditions. This process encourages the model to learn features that are more invariant to changes in orientation, position, and other visual variations, which is particularly important for handling the diverse facial expressions in the FER2013 dataset. Additionally, specific transformations simulate real-world scenarios where parts of the face may be obscured or only partially visible. The validation and test images, on the other hand, remain unaltered to ensure that the model's performance is evaluated consistently without any added variability.

6.2 Design of Experiment

The batch size and the number of training epochs influenced the performance of the model. To assess their impact, we used 20% of the training set as a validation set which is used to monitor the model's performance and tune hyperparameters then, conducted three experiments with different configurations. In the first experiment, we trained the model for 50 epochs with a batch size of 32. In the second experiment, we increased the training duration to 100 epochs while keeping the batch size at 32 to observe the effect of prolonged training which achieved better results. For the third experiment, we kept the number of epochs as 100 and increased the batch size to 64. In all experiments, we used the SGD optimizer with a learning rate of 0.001 and a momentum of 0.9 to ensure consistent optimization settings. Additionally, we employed a learning rate scheduler, specifically the ReduceLROnPlateau callback, to adjust the learning rate based on validation accuracy. This scheduler reduces the learning rate by a factor of 0.75 after 5 epochs of no improvement, with a lower limit set at 1e-5. These variations allowed us to analyze how different combinations of batch sizes and epochs affect model convergence, learning stability, and overall performance.

6.3 Experimental results

The results from the three experiments on the test set reveal how the combination of batch size and training epochs impacts model performance. In the first experiment (32 batch size, 50 epochs), the model achieved an accuracy of 62%, with notable performance in detecting emotions like happy (f1-score = 0.84) but struggled with disgust and fear, achieving low recall values of 0.13 and 0.22, respectively. In the second experiment (32 batch size, 100 epochs), increasing the number of epochs to 100 improved the accuracy to 64%, with significant improvements in recall for emotions such as disgust 0.33 and fear 0.36, though precision for these classes remained relatively low. The f1-score for happy remained high 0.86, and the model showed better stability overall, particularly for neutral and surprise emotions. The third experiment (64 batch size, 100 epochs) showed a similar accuracy of 64%, but with a slight decrease in the f1-score for angry, disgust, and fear, compared to the second experiment. The model was less sensitive to the smaller classes such as disgust and fear and showed slightly lower precision and recall for those emotions, while the f1-scores for happy 0.85 remained strong. In all experiments, the use of the learning rate scheduler ReduceLROnPlateau helped maintain stable training, particularly by reducing the learning rate after plateauing validation accuracy.

Overall, increasing the number of epochs improved performance across most emotions, but changing the batch size had a less pronounced effect, indicating that training duration plays a more significant role in refining model predictions. Moreover, the results from our experiments indicate that the optimal configuration for the model performance was achieved with 100 epochs and a batch size of 32. This setup led to the highest overall accuracy and f1-score. The results are shown in **Table 1**.

Table 1: Performance of the model on the test set in three experiments with varying hyperparameters (epochs and batch size). Performance is measured by accuracy, and the weighted average of precision, recall, and F1-score across classes.

Experiment	Hyperparameters	Train	Validation	Test

1	Epochs = 50
	Batch size = 32
		Accuracy = 0.6091	Accuracy = 0.6116
			Precision = 0.7990
			Recall = 0.3975
			F1 Score = 0.531
				Accuracy = 0.62
				Precision = 0.62
				Recall = 0.62
				F1 Score = 0.60

2	Epochs = 100
	Batch size = 32
		Accuracy = **0.6794**	Accuracy = **0.6476**
			Precision = **0.8002**
			Recall = **0.4780**
			F1 Score = **0.598**
				Accuracy = 0.64
				Precision = 0.65
				Recall = 0.64
				F1 Score = 0.64

3	Epochs = 100
	Batch size = 64
		Accuracy = 0.6562	Accuracy = 0.6304
			Precision = 0.7743
			Recall = 0.4738
			F1 Score = 0.587
				Accuracy = 0.64
				Precision = 0.63
				Recall = 0.64
				F1 Score = 0.63

6.4 comparing results with existing literature

When comparing our results to existing literature on emotion recognition using deep learning, Many studies have reported similar accuracy ranges for emotion recognition tasks, with values often falling between 60% and 70% which is consistent with our achieved accuracy of 64%. Y. Gan [7] used a modified AlexNet architecture on the FER2013 [1] dataset. achieving the accuracy of 0.64 with 20 passes through the training dataset and a learning rate of 0.001 his analysis noted that while increasing the passes improved accuracy, overfitting remained a concern. our model's performance aligns with their results but increasing passes in our model didn't lead to over fitting since due to serval factors such as reducing the learning rate when a plateau is reached on validation accuracy, batch normalization and data augmentation.

7. Conclusion

Our project focuses on the use of CNNs for Facial Emotion Recognition, specifically a modified VGGnet architecture. We utilized the FER2013 dataset which consisted of approximately 36,000 images, to conduct three experiments with different batch sizes and number of epochs. Experiment 2 with a batch size 32 and 100 epochs showed the best results with an accuracy of 64% which highlights the effectiveness of hyperparameter tuning.

7.1 Model Strengths

The model demonstrates several strengths that contribute to its effectiveness. Its four convolutional blocks excel at extracting both low- level and high-level features, which are crucial for recognizing subtle facial expressions. The inclusion of batch normalization throughout these blocks stabilizes training, accelerates convergence, and mitigates issues like vanishing or exploding gradients, thus enhancing generalization. Additionally, robust data augmentation techniques, such as rotation, shifting, zooming, flipping, and custom preprocessing methods like random erasing and cropping, introduce variability that enables the model to handle diverse facial expressions and occlusions effectively. Furthermore, the integration of a ReduceLROnPlateau learning rate scheduler ensures stable training by dynamically adjusting the learning rate when validation accuracy plateaus, optimizing convergence without the need for manual intervention.

7.2 limitations

One of the key limitations of the model is its sensitivity to imbalanced classes. Despite incorporating data augmentation techniques, the model still struggles with accurately detecting less frequent emotions, such as disgust and fear, as indicated by the lower recall values for these classes. This suggests that the model may not fully learn the distinguishing features of these emotions, leading to suboptimal performance for these categories. Additionally, the model sometimes faces difficulty in handling occlusions, even with the use of random erasing to simulate partial face visibility. Significant occlusions, such as hands or objects obscuring parts of the face, can affect the model's ability to recognize emotions, leading to potential inaccuracies in real-world applications where such occlusions are common.

7.3 future work

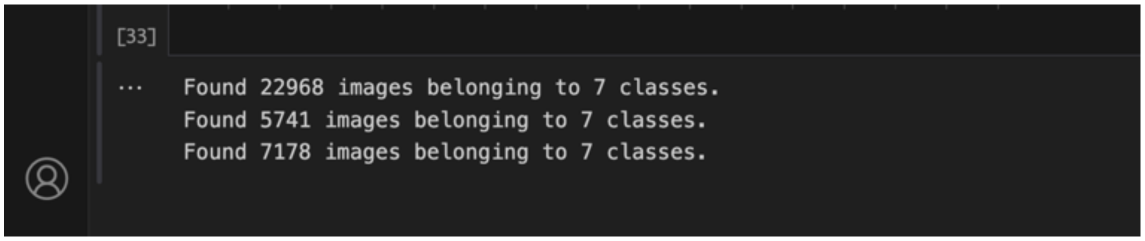
Future work includes addressing the class imbalance through techniques such as oversampling the minority classes such as fear and disgust or employing advanced loss functions which could help improve recall for emotions with lower frequencies, such as disgust and fear. Additionally, Fine-tuning hyperparameters such as learning rate schedulers, dropout rates, and the number of convolutional layers may further refine the model's ability to generalize. Moreover, exploring the use of more complex architectures like pre-trained models (e.g. ResNet) fine-tuned for emotion recognition could also yield significant performance gains. Finally, incorporating a deeper understanding of facial expressions through multi-modal learning (such as combining facial and voice data) could provide a more robust approach to emotion recognition.

8. Screenshots

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import keras
from keras.preprocessing import image
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Flatten,Dense,Dropout,BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import cv2
from tensorflow.keras.applications import VGG16
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint,
EarlyStopping
from keras import regularizers
from tensorflow.keras.optimizers import Adam,RMSprop,SGD,Adamax
from sklearn.metrics import classification_report
from tensorflow.keras.models import load_model



train_dir = "/kaggle/input/fer2013/train" #passing the path with training images
test_dir = "/kaggle/input/fer2013/test" #passing the path with testing images
```

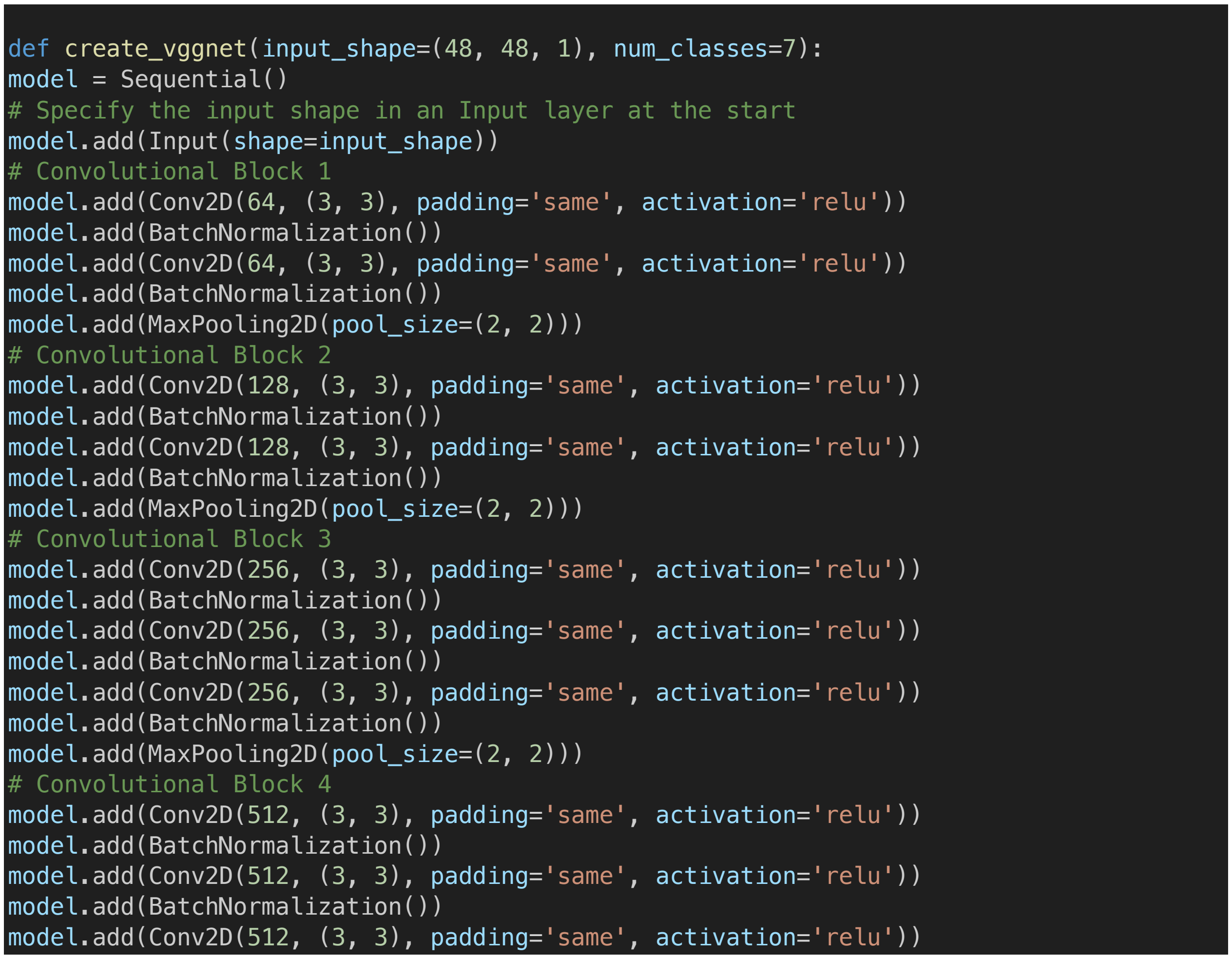
```
train_datagen = ImageDataGenerator(
rescale=1./255,
rotation_range=10,
width_shift_range=0.2,
height_shift_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
vertical_flip=True,
validation_split = 0.2,
preprocessing_function=lambda img: random_erase_and_crop(img)
)


test_datagen = ImageDataGenerator(
rescale=1./255
)

validation_datagen = ImageDataGenerator(rescale = 1./255,
validation_split = 0.2)

def random_erase_and_crop(image, crop_size=(48, 48), erase_prob=0.5 , erase_size=10):
"""Apply ten-crop and random erasing with a 50% probability."""
# Convert image to numpy array (only for processing)
image_np = image.numpy() if isinstance(image, tf.Tensor) else image

# Crop the image to the desired crop size
image_cropped = tf.image.resize(image, crop_size)

# If erase probability condition is met, apply random erase
if np.random.rand() < erase_prob:

# Randomly choose the top-left corner of the erase region
x_erase = np.random.randint(0, crop_size[0] - erase_size)
y_erase = np.random.randint(0, crop_size[1] - erase_size)

# Create a mask for the erased area
mask = np.ones_like(image_cropped)
mask[x_erase:x_erase + erase_size, y_erase:y_erase + erase_size, :] = 0

# Apply the mask to "erase" part of the image
image_cropped = image_cropped * mask

return image_cropped
```

```
train_generator = train_datagen.flow_from_directory(directory = train_dir,
target_size = (48,48),
batch_size = 32,
color_mode='grayscale',
class_mode = "categorical",
subset = "training"
)
validation_generator = validation_datagen.flow_from_directory( directory = train_dir,
target_size = (48,48),
batch_size = 32,
color_mode='grayscale',
class_mode = "categorical",
subset = "validation"
)
test_generator = test_datagen.flow_from_directory( directory = test_dir,
target_size = (48,48),
batch_size = 32,
color_mode='grayscale',
class_mode = "categorical",
shuffle=False
)
```



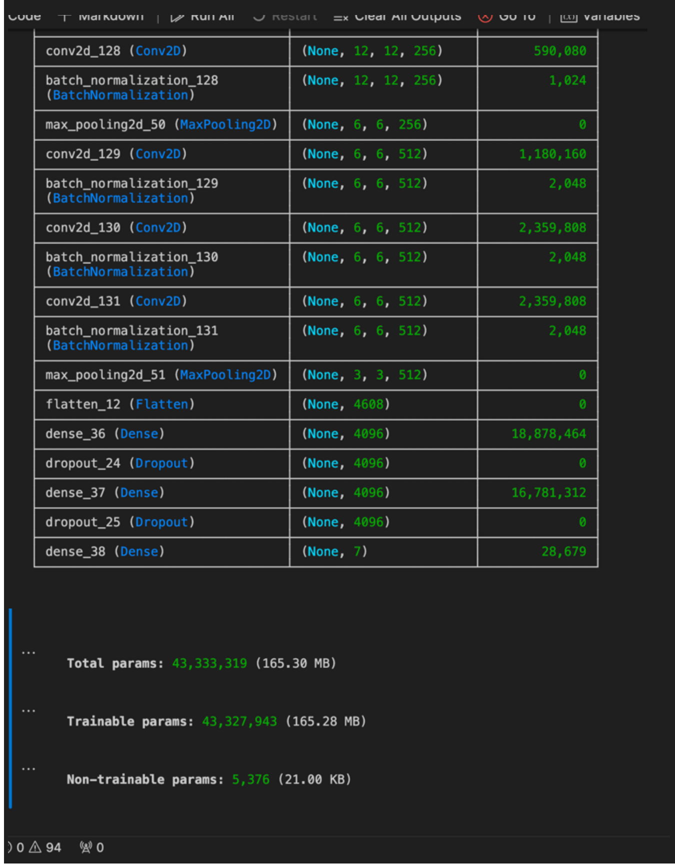
```
[33]
...    Found 22968 images belonging to 7 classes.
       Found 5741 images belonging to 7 classes.
       Found 7178 images belonging to 7 classes.
```

```
# Set up the directory path for images
data_dir = train_dir
labels = os.listdir(data_dir) # Each subfolder is a label, e.g., 'angry', 'happy'

# Parameters
target_size = (48, 48)
num_images = 5 # Number of images to display per label
fig, axes = plt.subplots(len(labels), num_images, figsize=(15, 10))

# Loop through each label and image, load, and display
for label_idx, label in enumerate(labels):
label_dir = os.path.join(data_dir, label)
images = os.listdir(label_dir)[:num_images] # Limit to the first `num_images` files

for img_idx, img_name in enumerate(images):
img_path = os.path.join(label_dir, img_name)
# Load the image
img = image.load_img(img_path, target_size=target_size)
img_array = np.array(img)
# Display the image
ax = axes[label_idx, img_idx]
ax.imshow(img_array, cmap='gray')
ax.axis('off')
# Set label title for the first image in each row
if img_idx == 0:
ax.set_title(label, fontsize=12)

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



```
def create_vggnet(input_shape=(48, 48, 1), num_classes=7):
model = Sequential()
# Specify the input shape in an Input layer at the start
model.add(Input(shape=input_shape))
# Convolutional Block 1
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
# Convolutional Block 2
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
# Convolutional Block 3
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
# Convolutional Block 4
model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
```

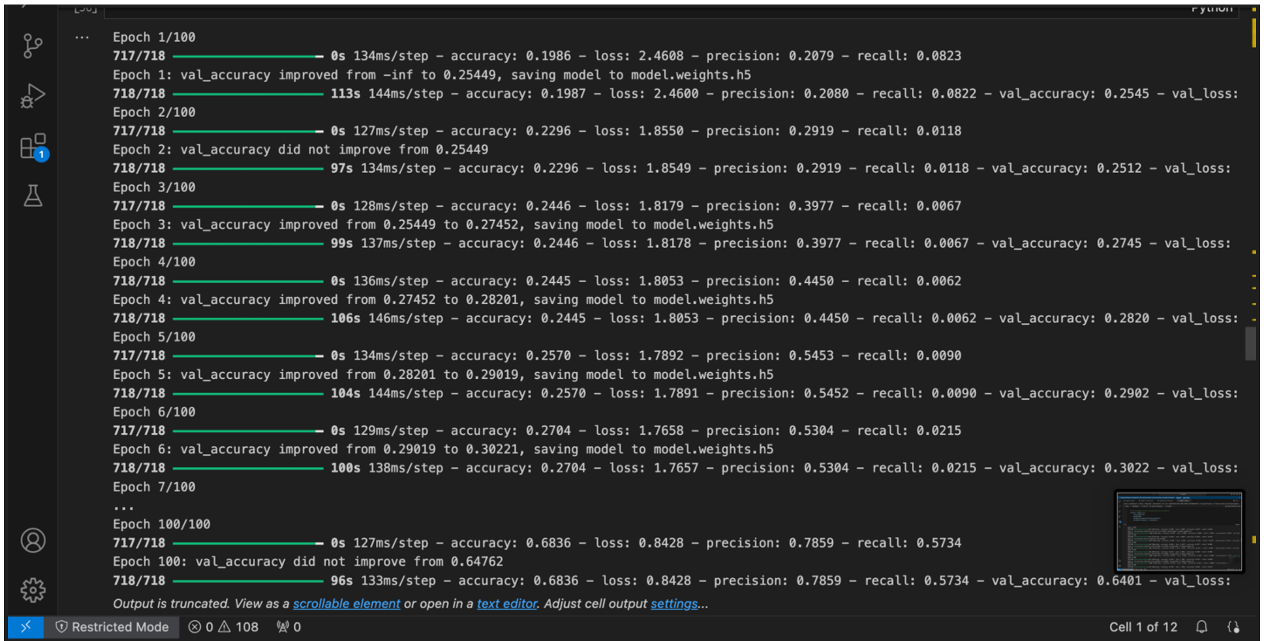
```python
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
# Fully Connected Layers
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
return model

model = create_vggnet()

model.compile(
optimizer=SGD(learning_rate=0.001, momentum=0.9),
loss='categorical_crossentropy',
metrics=['accuracy']
)

# Print the model summary
model.summary()
```



```
│ conv2d_128 (Conv2D)              │ (None, 12, 12, 256)     │       590,080 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ batch_normalization_128          │ (None, 12, 12, 256)     │         1,024 │
│ (BatchNormalization)             │                         │               │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ max_pooling2d_50 (MaxPooling2D)  │ (None, 6, 6, 256)       │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ conv2d_129 (Conv2D)              │ (None, 6, 6, 512)       │     1,180,160 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ batch_normalization_129          │ (None, 6, 6, 512)       │         2,048 │
│ (BatchNormalization)             │                         │               │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ conv2d_130 (Conv2D)              │ (None, 6, 6, 512)       │     2,359,808 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ batch_normalization_130          │ (None, 6, 6, 512)       │         2,048 │
│ (BatchNormalization)             │                         │               │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ conv2d_131 (Conv2D)              │ (None, 6, 6, 512)       │     2,359,808 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ batch_normalization_131          │ (None, 6, 6, 512)       │         2,048 │
│ (BatchNormalization)             │                         │               │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ max_pooling2d_51 (MaxPooling2D)  │ (None, 3, 3, 512)       │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ flatten_12 (Flatten)             │ (None, 4608)            │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dense_36 (Dense)                 │ (None, 4096)            │    18,878,464 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dropout_24 (Dropout)             │ (None, 4096)            │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dense_37 (Dense)                 │ (None, 4096)            │    16,781,312 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dropout_25 (Dropout)             │ (None, 4096)            │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dense_38 (Dense)                 │ (None, 7)               │        28,679 │
└──────────────────────────────────┴─────────────────────────┴───────────────┘

 Total params: 43,333,319 (165.30 MB)

 Trainable params: 43,327,943 (165.28 MB)

 Non-trainable params: 5,376 (21.00 KB)
```

```python
# Use the previously defined custom VGGNet model
```

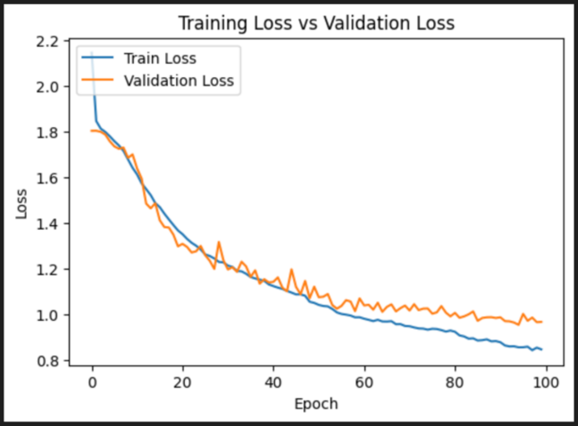
```python
model = create_vggnet(input_shape=(48, 48, 1), num_classes=7)

# Compile the model with SGD optimizer and momentum
model.compile(
optimizer=tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.9),
loss='categorical_crossentropy',
metrics=['accuracy','precision','recall']
)

# Define learning rate scheduler (Reduce LR on Plateau)
reduce_lr = ReduceLROnPlateau(
monitor='val_accuracy', # Monitor validation accuracy
factor=0.75, # Reduce learning rate by 0.75
patience=5, # Wait 5 epochs with no improvement
min_lr=1e-5, # Lower limit on learning rate
verbose=1
)

# Checkpoint to save the best model
checkpoint = ModelCheckpoint(
'model.weights.h5',
monitor='val_accuracy', # Monitor validation accuracy
save_best_only=True,
save_weights_only=True,
mode='max',
verbose=1
)
# Fit the model on the training data with validation
history = model.fit(
train_generator,
epochs=100,
validation_data=validation_generator,
callbacks=[reduce_lr, checkpoint]
)
```

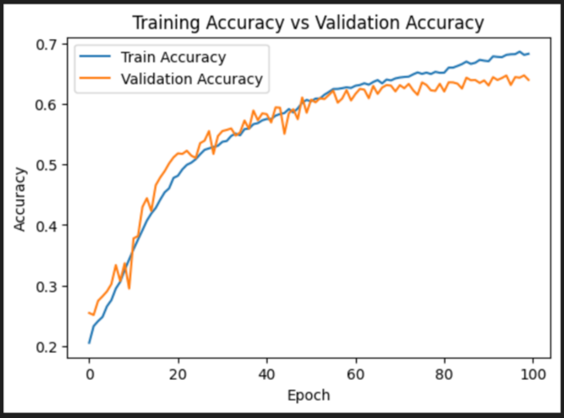


```
Epoch 1/100
717/718 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - accuracy: 0.1986 - loss: 2.4608 - precision: 0.2079 - recall: 0.0823
Epoch 1: val_accuracy improved from -inf to 0.25449, saving model to model.weights.h5
718/718 ━━━━━━━━━━━━━━━━━━━━ 113s 144ms/step - accuracy: 0.1987 - loss: 2.4600 - precision: 0.2080 - recall: 0.0822 - val_accuracy: 0.2545 - val_loss:
Epoch 2/100
717/718 ━━━━━━━━━━━━━━━━━━━━ 0s 127ms/step - accuracy: 0.2296 - loss: 1.8550 - precision: 0.2919 - recall: 0.0118
Epoch 2: val_accuracy did not improve from 0.25449
718/718 ━━━━━━━━━━━━━━━━━━━━ 97s 134ms/step - accuracy: 0.2296 - loss: 1.8549 - precision: 0.2919 - recall: 0.0118 - val_accuracy: 0.2512 - val_loss:
Epoch 3/100
717/718 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step - accuracy: 0.2446 - loss: 1.8179 - precision: 0.3977 - recall: 0.0067
Epoch 3: val_accuracy improved from 0.25449 to 0.27452, saving model to model.weights.h5
718/718 ━━━━━━━━━━━━━━━━━━━━ 99s 137ms/step - accuracy: 0.2446 - loss: 1.8178 - precision: 0.3977 - recall: 0.0067 - val_accuracy: 0.2745 - val_loss:
Epoch 4/100
718/718 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step - accuracy: 0.2445 - loss: 1.8053 - precision: 0.4450 - recall: 0.0062
Epoch 4: val_accuracy improved from 0.27452 to 0.28201, saving model to model.weights.h5
718/718 ━━━━━━━━━━━━━━━━━━━━ 106s 146ms/step - accuracy: 0.2445 - loss: 1.8053 - precision: 0.4450 - recall: 0.0062 - val_accuracy: 0.2820 - val_loss:
Epoch 5/100
717/718 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - accuracy: 0.2570 - loss: 1.7892 - precision: 0.5453 - recall: 0.0090
Epoch 5: val_accuracy improved from 0.28201 to 0.29019, saving model to model.weights.h5
718/718 ━━━━━━━━━━━━━━━━━━━━ 104s 144ms/step - accuracy: 0.2570 - loss: 1.7891 - precision: 0.5452 - recall: 0.0090 - val_accuracy: 0.2902 - val_loss:
Epoch 6/100
717/718 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step - accuracy: 0.2704 - loss: 1.7658 - precision: 0.5304 - recall: 0.0215
Epoch 6: val_accuracy improved from 0.29019 to 0.30221, saving model to model.weights.h5
718/718 ━━━━━━━━━━━━━━━━━━━━ 100s 138ms/step - accuracy: 0.2704 - loss: 1.7657 - precision: 0.5304 - recall: 0.0215 - val_accuracy: 0.3022 - val_loss:
Epoch 7/100
...
Epoch 100/100
717/718 ━━━━━━━━━━━━━━━━━━━━ 0s 127ms/step - accuracy: 0.6836 - loss: 0.8428 - precision: 0.7859 - recall: 0.5734
Epoch 100: val_accuracy did not improve from 0.64762
718/718 ━━━━━━━━━━━━━━━━━━━━ 96s 133ms/step - accuracy: 0.6836 - loss: 0.8428 - precision: 0.7859 - recall: 0.5734 - val_accuracy: 0.6401 - val_loss:
```

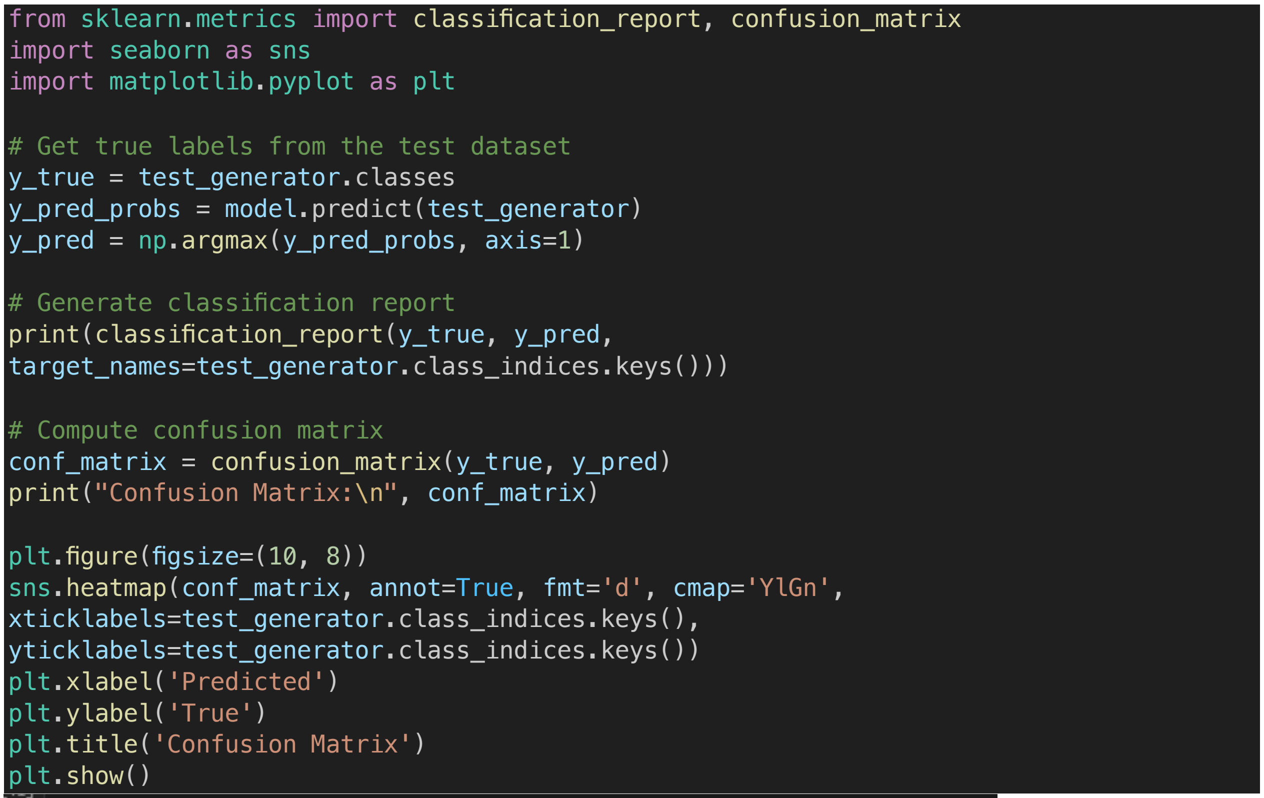
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
# Plot Training and Validation Accuracy
plt.figure(figsize=(6, 4))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training Accuracy vs Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='upper left')
plt.show()

# Plot Training and Validation Loss
plt.figure(figsize=(6, 4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training Loss vs Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper left')
plt.show()
```

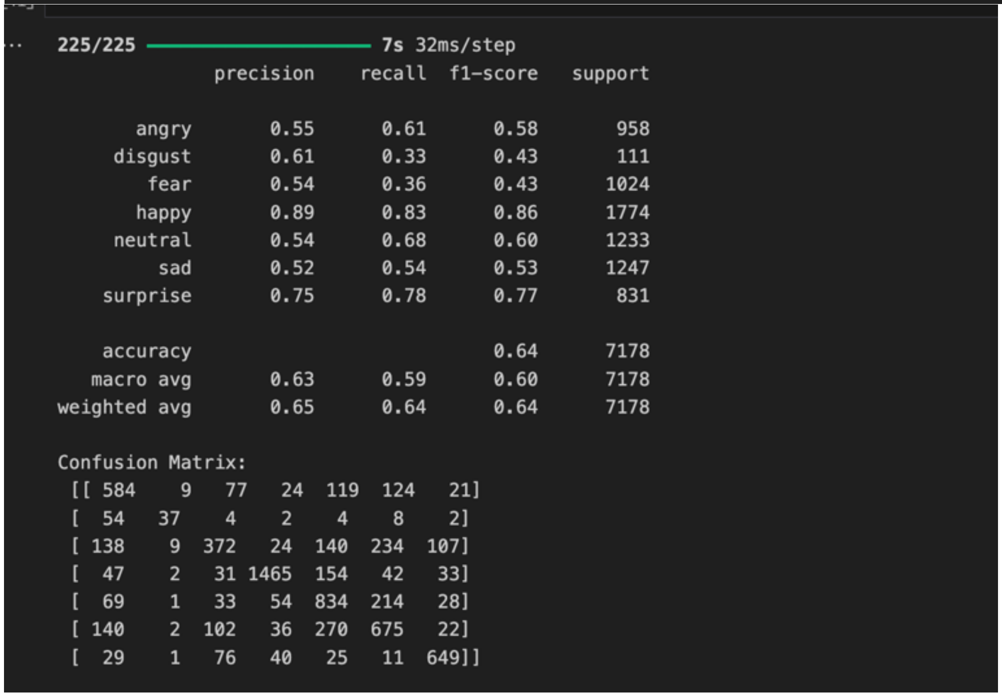




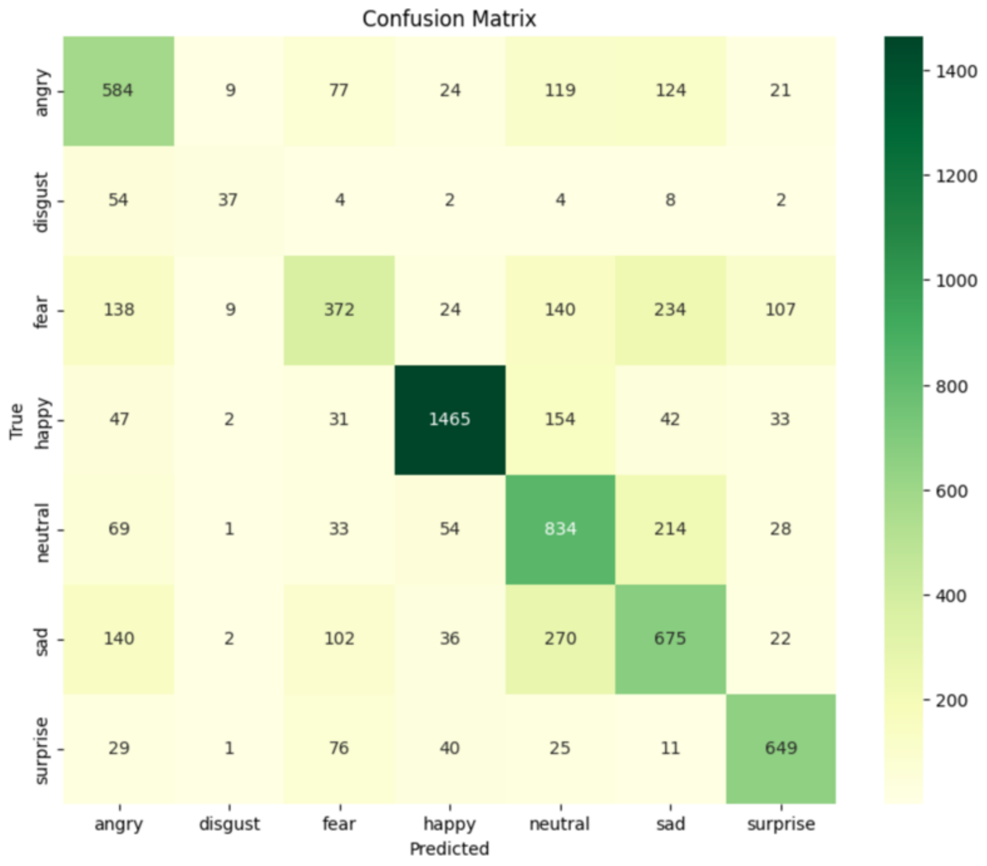
```python
import numpy as np
```



```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Get true labels from the test dataset
y_true = test_generator.classes
y_pred_probs = model.predict(test_generator)
y_pred = np.argmax(y_pred_probs, axis=1)

# Generate classification report
print(classification_report(y_true, y_pred,
target_names=test_generator.class_indices.keys()))

# Compute confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", conf_matrix)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='YlGn',
xticklabels=test_generator.class_indices.keys(),
yticklabels=test_generator.class_indices.keys())
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



```
...  225/225 ━━━━━━━━━━━━━━━━━━━━ 7s 32ms/step
              precision    recall  f1-score   support

       angry       0.55      0.61      0.58       958
     disgust       0.61      0.33      0.43       111
        fear       0.54      0.36      0.43      1024
       happy       0.89      0.83      0.86      1774
     neutral       0.54      0.68      0.60      1233
         sad       0.52      0.54      0.53      1247
    surprise       0.75      0.78      0.77       831

    accuracy                           0.64      7178
   macro avg       0.63      0.59      0.60      7178
weighted avg       0.65      0.64      0.64      7178

Confusion Matrix:
 [[ 584    9   77   24  119  124   21]
 [  54   37    4    2    4    8    2]
 [ 138    9  372   24  140  234  107]
 [  47    2   31 1465  154   42   33]
 [  69    1   33   54  834  214   28]
 [ 140    2  102   36  270  675   22]
 [  29    1   76   40   25   11  649]]
```



True \ Predicted	angry	disgust	fear	happy	neutral	sad	surprise
angry	584	9	77	24	119	124	21
disgust	54	37	4	2	4	8	2
fear	138	9	372	24	140	234	107
happy	47	2	31	1465	154	42	33
neutral	69	1	33	54	834	214	28
sad	140	2	102	36	270	675	22
surprise	29	1	76	40	25	11	649

10. Work Distribution

Lamees Alturki	Contributed to all aspects of the project
Yara Alfouzan	Contributed to all aspects of the project

References

[1] I. J. Goodfellow *et al.*, “Challenges in Representation Learning: A report on three machine learning contests,” *arXiv:1307.0414 [cs, stat]*, Jul. 2013, Available: https://arxiv.org/abs/1307.0414

[2] N. Mehendale, “Facial emotion recognition using convolutional neural networks (FERC),” *SN Applied Sciences*, vol. 2, no. 3, Feb. 2020, doi: https://doi.org/10.1007/s42452-020-2234-1.

[3] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, “The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression,” *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, Jun. 2010, doi: https://doi.org/10.1109/cvprw.2010.5543262.

[4] “Perona Lab - Caltech 10k Web Faces,” *Caltech.edu*, 2024. https://www.vision.caltech.edu/datasets/caltech_10k_webfaces/ (accessed Oct. 25, 2024).

[5] “The CMU Multi-PIE Face Database,” *www.cs.cmu.edu*. https://www.cs.cmu.edu/afs/cs/project/PIE/MultiPie/Multi-Pie/Home.html

[6] P. A. Flanagan, “Biometric Special Databases and Software,” *NIST*, Sep. 17, 2018. https://www.nist.gov/itl/iad/image-group/resources/biometric-special-databases-and-software

[7] Y. Gan, “Facial Expression Recognition Using Convolutional Neural Network,” Proceedings of the 2nd International Conference on Vision, Image and Signal Processing - ICVISP 2018, 2018, doi: https://doi.org/10.1145/3271553.3271584.

[8] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv.org, Apr. 10, 2015. https://arxiv.org/abs/1409.1556

[9] Y. Khaireddin and Z. Chen, “Facial Emotion Recognition: State of the Art Performance on FER2013,” arXiv:2105.03588 [cs], May 2021, Available: https://arxiv.org/abs/2105.03588

[10] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” arXiv.org, 2015. https://arxiv.org/abs/1502.03167