SOC

FACIAL RECOGNITION AND EMOTION ANALYSIS

Divya Sharma

24B0702

MODEL 1:

I began with a simple CNN architecture which comprised of 3 Convolution layers each of which included features like BatchNormalization to normalize the inputs for each convolution layer and speedup the training process; and MaxPooling to reduce spatial dimensions. I used ReLU activation function. I kept increasing the number of filters in each successive layer as the deeper layers will have to capture more abstract features whereas the initial layers are only focused on the superficial features. Further, I added a Flatten layer along with dense layers of 128units and the final one of 7 units with softmax activation function as the output layer for the 7 emotion classes. . For training the model, I used Adam optimizer and set the epochs to 30 with a batch size of 64. Since I ran it over a subset of only 5000 images from the dataset, this led to an accuracy of 43%.

[5]:
```
import tensorflow as tf
import os
import numpy as np
import random
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator, array_to_img
import matplotlib.pyplot as plt

emotion_labels = { 0: 'Angry' ,
                   1: 'Disgust' ,
                   2: 'Fear' ,
                   3: 'Happy' ,
                   4: 'Sad' ,
                   5: 'Surprise' ,
                   6: 'Neutral' }

images=[]
labels=[]

for i in range(5000):
    img = ds[i]['images'].numpy().astype('float32')/255.0
    img = img.reshape(48,48,1)
    label = ds[i]['labels'].numpy()
    images.append(img)
    labels.append(label)

images = np.array(images)
labels = np.array(labels)

X_temp, X_test, y_temp, y_test = train_test_split(images, labels, test_size=0.1, stratify=labels, random_state=42)
train_images, val_images, train_labels, val_labels = train_test_split(X_temp, y_temp, test_size=0.1111, stratify=y_temp, random_state=42)

data_aug_gen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True
)
```

[6]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization , GlobalAveragePooling2D
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(7, activation='softmax'))

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_images, train_labels,
    validation_data=(val_images, val_labels),
    epochs=30,
    batch_size=64
)

test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_acc)
```

MODEL 2:

In this model, I added dropout layers to reduce overfitting. I began with a dropout of 0.2 in the first convolution block and progressively increased it to 0.5 in the dense layer of 128 units. I kept the rest of the things same and the accuracy obtained this time was 53.3%.

```
for i in range(len(ds)):
    img = ds[i]['images'].numpy().astype('float32')/255.0
    img = img.reshape(48,48,1)
    label = ds[i]['labels'].numpy()
    images.append(img)
    labels.append(label)

images = np.array(images)
labels = np.array(labels)

X_temp, X_test, y_temp, y_test = train_test_split(images, labels, test_size=0.1, stratify=labels, random_state=42)
train_images, val_images, train_labels, val_labels = train_test_split(X_temp, y_temp, test_size=0.1111, stratify=y_temp, random_state=42)

data_aug_gen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True
)
```

[7]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization , GlobalAveragePooling2D
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))


model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    data_aug_gen.flow(train_images, train_labels, batch_size=64),
    validation_data=(val_images, val_labels),
    epochs=30
)


test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_acc)
```

MODEL 3:

I changed the Flatten layer to GlobalAveragePooling so that the model could generalize better and the parameters could be reduced. After this, I added a dense layer of 128 units and the final output layer of 7 units. The accuracy reached 53.5%.

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))

model.add(GlobalAveragePooling2D())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    data_aug_gen.flow(train_images, train_labels, batch_size=64),
    validation_data=(val_images, val_labels),
    epochs=30
)
```

MODEL 4:

After the initial three convolution layers which comprised of MaxPooling layers, I added two more convolution layers having 256 and 512 filters respectively. Each of these was followed by BatchNormalization and dropout(0.4). I kept the GlobalAveragePooling layer untouched. To improve training, I added the EarlyStopping feature with a patience of 5 and also decided to increase training for upto 40 epochs (after once training for 30 epochs). This time I received an accuracy of 59.2%.

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))

model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(GlobalAveragePooling2D())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))


model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = model.fit(
    data_aug_gen.flow(train_images, train_labels, batch_size=64),
    validation_data=(val_images, val_labels),
    epochs=40,
    callbacks=[early_stop]
)

test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_acc)
```

Since a lot of time was being taken per epoch on CPU, I tried to shift to GoogleCollab in order to use GPU but I faced some technical issues over there so I came back to Jupyter Notebook.

MODEL 5:

This time I tried to improve the fully connected part. I added an additional dense layer of 256 units before the one with 128 units after GlobalAveragePooling to get an accuracy of 60%. I first tried it using early stopping with patience set to 5 but I got lower accuracy as the training stopped early (only about 20 epochs) so I decided to increase the patience to 10.

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))

model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(GlobalAveragePooling2D())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))


model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience= 10, restore_best_weights=True)

history = model.fit(
    data_aug_gen.flow(train_images, train_labels, batch_size=64),
    validation_data=(val_images, val_labels),
    epochs=40,
    callbacks=[early_stop]
)
```

I also experimented with increasing the dense layers further to 512 and 256 units in place of 256 and 128 as in Model5, but the accuracy slightly dropped to 59.2%.

```
model.add(GlobalAveragePooling2D())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))


model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model.fit(
    data_aug_gen.flow(train_images, train_labels, batch_size=64),
    validation_data=(val_images, val_labels),
    epochs=40,
    callbacks=[early_stop]
)

test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_acc)
```

Therefore I removed this one and believe that model 5 was the most appropriate one with an accuracy of 60%.