Zheling Zhang: 20727528, Zikang Xiong: 41815789, Jun Yan: 43818924

CS 178 Final Report

Dataset: Emotion Detection

Group Members: Jun Yan, Zheling Zhang, Zikang Xiong

- **Summary**

This project aims to find and fit the most valid model to predict the correct emotions that match the images from the dataset. It is analyzed by Convolution Neural Network (CNN) model and Support Vector Machine.

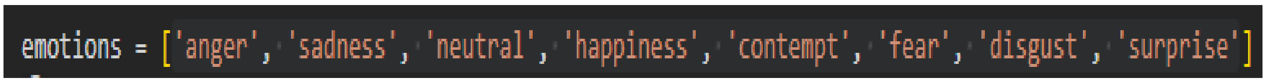
- **Data set exploration**

This data set is a collection of about 60 thousand of images of facial expressions, with each image labeled with one of seven emotions: anger, disgust, fear, happiness, sadness, surprise, or neutral. The images in the dataset are grayscale (examples are below) and have a resolution of 48 x 48 pixels.

- **Exploration with CNN**

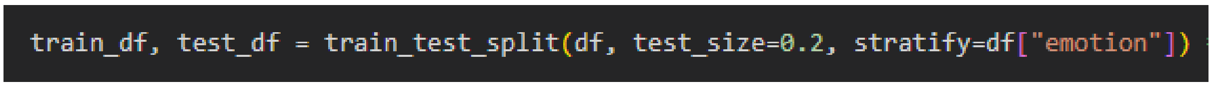
The reason why we chose CNN is that CNN deep learning method is doing well in handling huge amounts of images, and it is able to capture the specific feature of images. The model is evaluated using accuracy, which is a common metric for classification tasks. The ResNet50 model is a pre-trained convolutional neural network (CNN) model. ResNet is a popular CNN architecture that was developed to address the problem of vanishing gradients in very deep neural networks which might help in this exploration.

We accept all images in data set and categorize all emotion into 8 labels: 'anger', 'sadness', 'neutral', 'happiness', 'contempt', 'fear', 'disgust', 'surprise.'



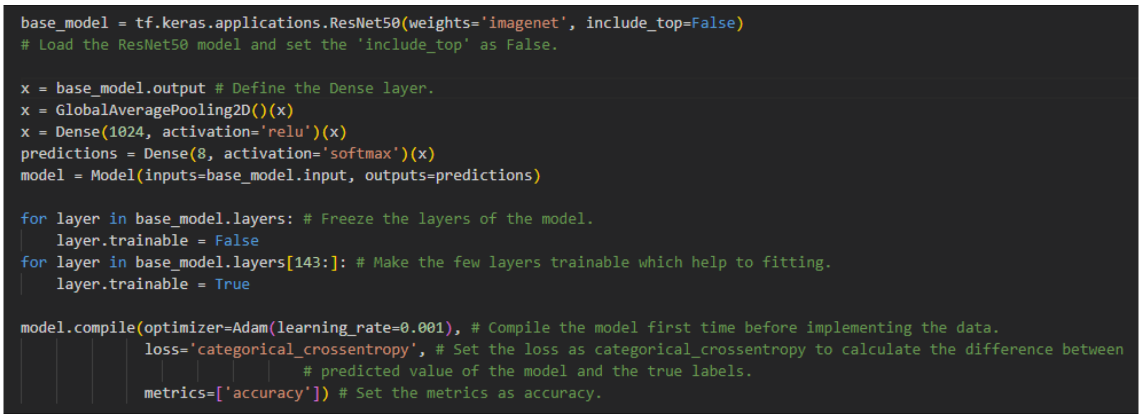
```
emotions = ['anger', 'sadness', 'neutral', 'happiness', 'contempt', 'fear', 'disgust', 'surprise']
```

Then Use the *train_test_split* to construct the training set and testing set.



```
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df["emotion"])
```

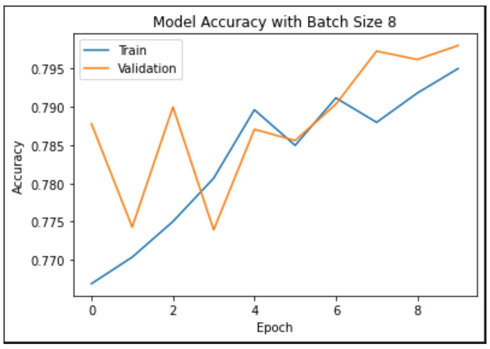
Analyze

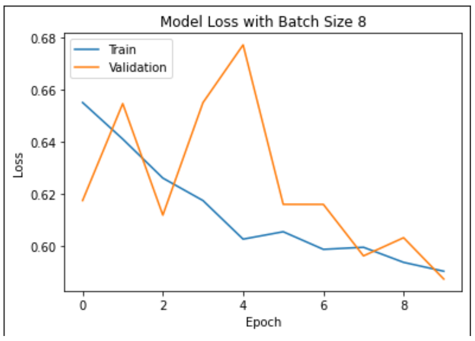


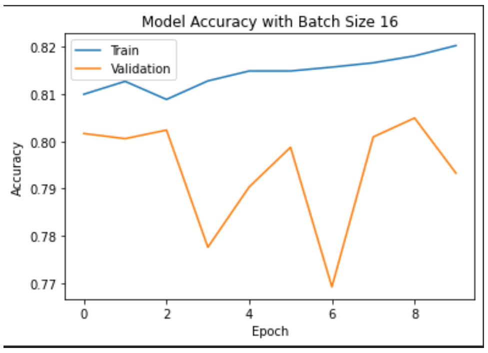
```
base_model = tf.keras.applications.ResNet50(weights='imagenet', include_top=False)
# Load the ResNet50 model and set the 'include_top' as False.

x = base_model.output # Define the Dense layer.
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(8, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

for layer in base_model.layers: # Freeze the layers of the model.
    layer.trainable = False
for layer in base_model.layers[143:]: # Make the few layers trainable which help to fitting.
    layer.trainable = True

model.compile(optimizer=Adam(learning_rate=0.001), # Compile the model first time before implementing the data.
              loss='categorical_crossentropy', # Set the loss as categorical_crossentropy to calculate the difference between
                                                # predicted value of the model and the true labels.
              metrics=['accuracy']) # Set the metrics as accuracy.
```

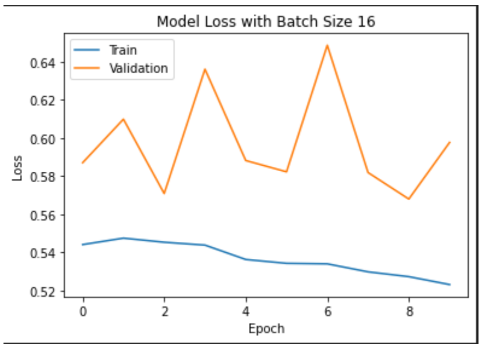
This is the preprocessing model before implementing the data into the model. Before implementing the preprocessing model, the loss and the accuracy score of the model are chaotic and low. After implementing the preprocessing model, we solve the problem of overfitting and lower the loss score and improve the accuracy score significantly.

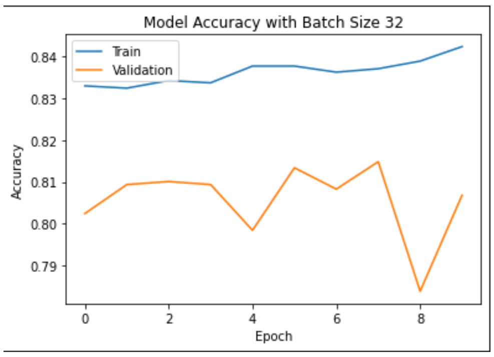
Performance Validation

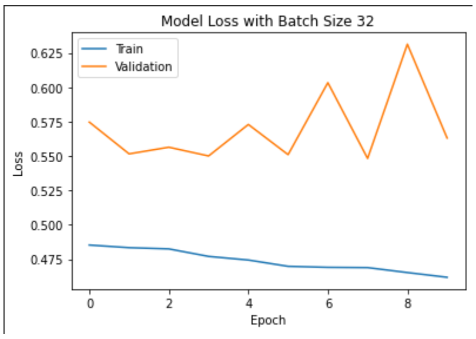


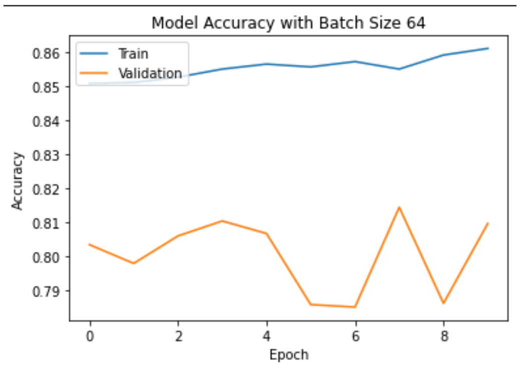


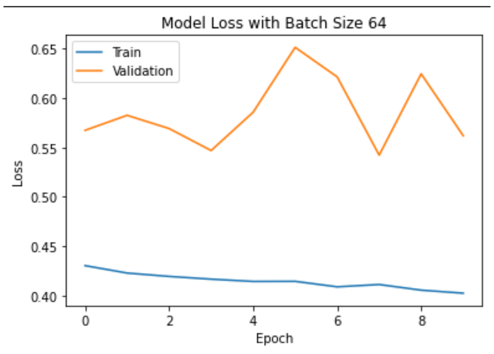


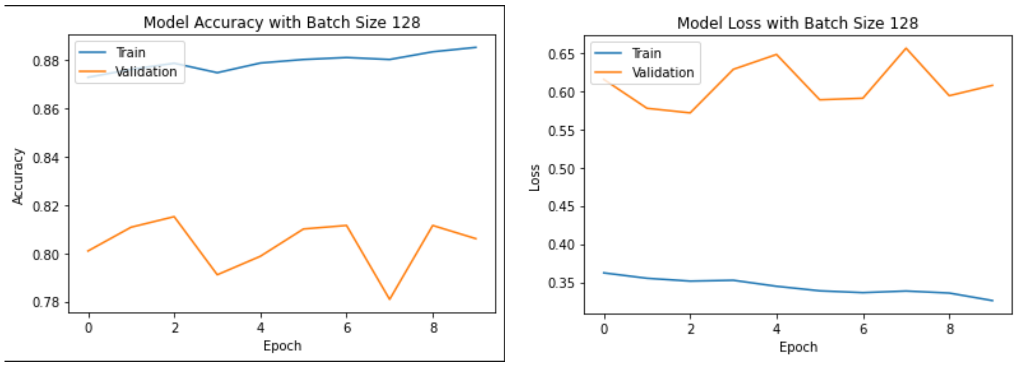




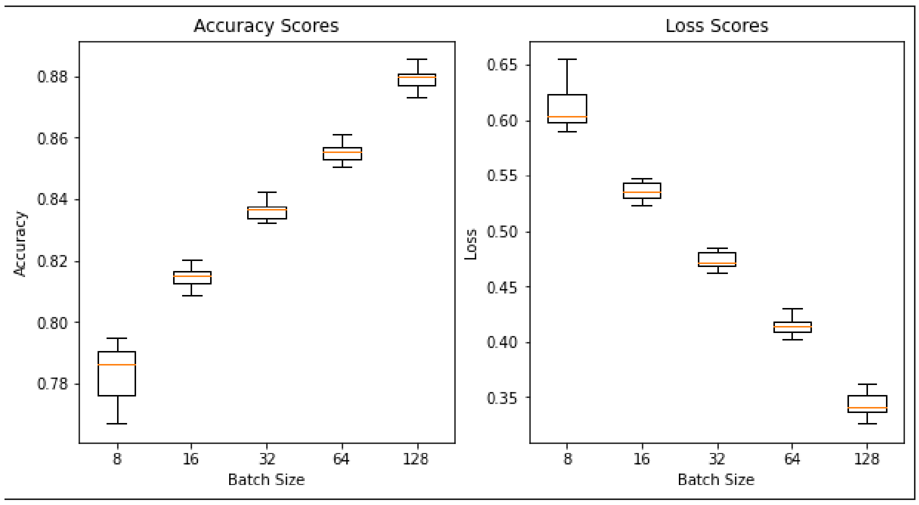








Bar chart of Model Accuracy and Model Loss with different Batch Size are followed.



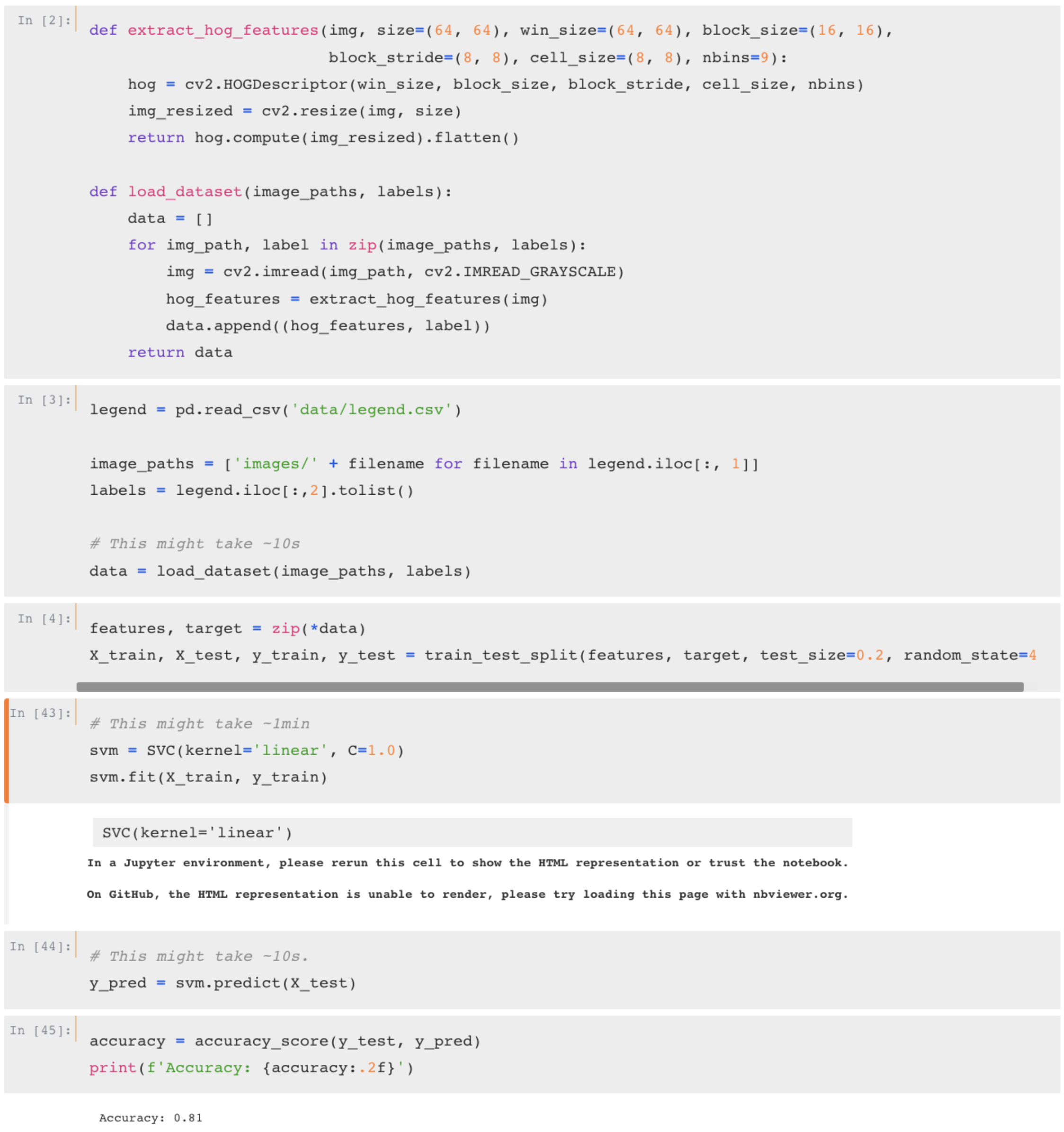
The bar chart shows the accuracy of the model for each of the seven classes. The accuracy is calculated as the ratio of correctly predicted images to the total number of images in the test set for each class. We can easily find that the accuracy is directly proportional to the batch size, and the loss is inversely proportional to the batch size. In terms of distribution, both accuracy and loss scores are highest at size 8. However, in the other sizes, the former has no obvious difference, while the latter shows a trend of gradually expanding the degree of dispersion. The overall accuracy of the model is reported to be approximately 81%, which is a decent performance.

- **Exploration with Support Vector Machine**

We also explored the dataset with approaches other than CNN. We chose the SVM+HOG to be the second approach. HOG (directional gradient histogram) feature is a common image feature extraction method, which can be used to detect face and expression. We first use HOG to preprocess the image data, and then we apply the data to SVM. SVM (Support vector machine) is a commonly used classifier, which can effectively classify images extracted by HOG features.

Zheling Zhang: 20727528, Zikang Xiong: 41815789, Jun Yan: 43818924

Analyze



```
In [2]:
def extract_hog_features(img, size=(64, 64), win_size=(64, 64), block_size=(16, 16),
                         block_stride=(8, 8), cell_size=(8, 8), nbins=9):
    hog = cv2.HOGDescriptor(win_size, block_size, block_stride, cell_size, nbins)
    img_resized = cv2.resize(img, size)
    return hog.compute(img_resized).flatten()

def load_dataset(image_paths, labels):
    data = []
    for img_path, label in zip(image_paths, labels):
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        hog_features = extract_hog_features(img)
        data.append((hog_features, label))
    return data
```

```
In [3]:
legend = pd.read_csv('data/legend.csv')

image_paths = ['images/' + filename for filename in legend.iloc[:, 1]]
labels = legend.iloc[:,2].tolist()

# This might take ~10s
data = load_dataset(image_paths, labels)
```

```
In [4]:
features, target = zip(*data)
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=4
```

```
In [43]:
# This might take ~1min
svm = SVC(kernel='linear', C=1.0)
svm.fit(X_train, y_train)
```

```
SVC(kernel='linear')
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

```
In [44]:
# This might take ~10s.
y_pred = svm.predict(X_test)
```

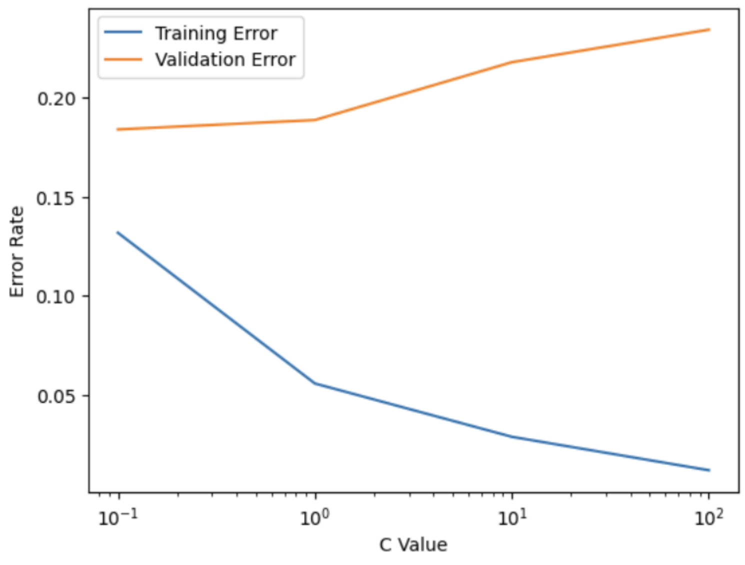
```
In [45]:
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.81
```

This approach also gave us fairly high accuracy on the predicting dataset, as we can see it achieved 81% accuracy.

Performance Validation

We then tried to modify the parameter to improve its performance. In detail, we tried different C values to test the model. The C value is an important parameter used to adjust the penalty parameter of the model. The smaller the value of C, the higher the tolerance of the model to misclassification, which may lead to the underfitting of the model. The larger the value of C, the lower the tolerance of the model to misclassification, which may lead to the overfitting of the model. Under different C values, the performance of the SVM model will be different. We tried C values of [0.1, 1, 10, 100], and we finally found the value of 1 would be the best choice. Since greater C values represent overfitting.



- **Summary**
  By using CNN and SVM HOG models, we successfully classify human expressions and achieve good accuracy (81% for SVM and 88% for CNN), and it turned out CNN has better performance on large-scale image learning. For both models, we adjusted the parameters. In the SVM HOG model, we optimize the performance of the model by adjusting the C value. Different C values will lead to different classification capabilities of the model, so we need to determine the best C value through cross-validation. In the CNN model, batch size was adjusted to optimize the performance of the model. A larger batch size increases the speed of training but introduces problems with memory and computational load, while a smaller batch size improves the accuracy of the model but takes longer to train. Therefore, we need to choose the most appropriate batch size based on the amount of data and computing resources.
- **Contribution**
  Zheling Zhang: Training the CNN model
  Zikang Xiong: Training the SVM model
  Jun Yan: Analyze & Summarize the report