Animated Facial Expression Recognition

Tiffany Ho

November 30, 2023

Abstract

This project is research-based. The proposed idea will be to research facial expression and emotion recognition in general and with human faces first to grasp the concept and how it works, then proceed with researching the same concept but with 2D animated cartoon characters and finding a method to implement it. While the idea has been brought up and researched before a few times, it is not widely found in web searches and does not have an established general implementation.

1 Introduction

I choose 2D animated cartoon character facial expression and emotion recognition as the topic of this project because I have always been interested in artificial intelligence, especially artificial emotions and how realistic machines can be in imitating human behavior, such as in realistic humanoid robots.

In this project, I have implemented a convolutional neural network (CNN) model using Python that classifies emotions from images of 2D animated cartoon characters' facial expressions and analyzed the results to gain more insight on how to improve visual emotion interpretation by machines.

1.1 Related Work

In this project, I referenced mainly three related works:

1. Facial Expression Recognition Using Human to Animated-Character Expression Translation by Kamran Ali, Ilkin Sevgi Isler, and Charles E. Hughes.

2. Understanding cartoon emotion using integrated deep neural network on large dataset by Nikita Jain, Vedika Gupta, Shubham Shubham, Agam Madan, Ankit Chaudhary, and K.C. Santosh.

3. Facial Expression Recognition of Animated Characters using Deep Learning by Mohd Ismail Lakhani, James McDermott, Frank G. Glavin, and Sai Priya Nagarajan.

The first related work, Facial Expression Recognition Using Human to Animated-Character Expression Translation, discussed two major problems in facial expression recognition:

1. The presence of inter-subject variations in facial expression recognition dataset.

2. Impure expressions posed by human subjects.

This source presents a solution to combat both of these issues using a novel Human-to-Animation conditional Generative Adversarial Network (HA-GAN). Essentially, this works by mapping multiple human faces with a particular expression and emotion to one cartoon face to allow more accurate facial expression and emotion detection and prediction. Their findings resulted in the HA-GAN framework outperforming the baseline deep neural network. It was also concluded that using this framework produced comparable to even better results than other methods for facial expression detection and emotion recognition from them. Overall, this source provided me with a clear introduction and much good background information to better understand the topic and task at hand and allowed me to start thinking up ideas on how I should begin to implement the main code and structure of the project.

The second related work, Understanding cartoon emotion using integrated deep neural network on large dataset, also works with utilizing human emotion recognition and applies it to 2D animated cartoon characters as well. Through this source, I ruled out one possible method of implementing this process as well as gaining inspiration on how what methods to look into when starting to implement this project. As stated in this source, the OpenCV library is used to identify human faces, but misses anything that is not a real-world entity, in this case 2D animated cartoon characters. Instead, this source proposes a newer approach using an integrated Deep Neural Network (DNN) approach that identifies the emotions of 2D animated cartoon characters. Then, the faces of the 2D animated cartoon characters are segmented into masks using Mask R-CNN (Mask Region-Based Convolutional Neural Network), a popular method to segment objects. The generated masks are then used as input for the actual emotion detection part.

The third related work, Facial Expression Recognition of Animated Characters using Deep Learning, provided me with further insight on how I should conduct my research and gather my training and testing images. This source mainly focuses on successful detection of 2D animated cartoon characters' emotions through facial expression so that children on the autism spectrum are able to learn how to recognize emotions better in real life, since it is often hard for many to do so and is very important in early development. The way they detected occurrences of a character, they used a custom Haar classifier. A Haar classifier is a machine learning object detection program that identifies objects in an image and video. The algorithm has four stages:

1. Calculating Haar Features:

A Haar feature is calculated with adjacent rectangular regions at a specific location in a detection window. It adds together all pixel intensities in each of the regions and calculating the difference between these sums. However, we will need integral images, where the number of calculations is reduced using the integral image, because Haar features can be hard to determine in large images.

2. Creating Integral Images:

As mentioned above, integal images reduce the calculations to get Haar features, which help when dealing with larger images.

3. Using Adaboost:

Adaboost training chooses the best features and trains classifiers to use them. The classifiers consists of a combination of both weak and strong classifiers. These are what the algorithm uses to detect objects. Eventually, the weak classifiers will need to be combined into a strong learner using cascading classifiers. Strong classifiers require a larger set of images to create more accuracy.

4. Implementing Cascading Classifiers:

The last step is to implement cascading classifiers. As stated above, these are a combination of many weak classifiers to form a strong learner. Using the collection of weak classifiers, the cascading classifier sorts an object into either a positive or negative sample, positive being that the object will be used. The goal is for a low false negative rate to indicate overall success.

While researching and going through these related works, the emotions used usually consisted of happiness, sadness, fear, surprise, anger, confusion, and excitement, as well as some more in certain experiments. However, due to the complexity and vast variety of human expression and emotion as well as my inexperience as a student, I decided to focus my project on only three emotions: happiness, sadness, and anger. Using these categories made it easier to collect clear training data and avoid inaccurate results, as these were the most distinguishable emotions to the eye.

1.2 Method

What I did for this project was build a convolutional neural network (CNN) in Python using three emotion classes: happy, sad, and angry.

Python Libraries:

1. cv2

2. numpy

3. tensorflow

4. keras: layers, models

5. os

6. pathlib

7. sklearn.model$_{s}election : train_{t}est_{s}plit$

8. keras.optimizers: Adam

9. keras.regularizers: l2

10. keras.callbacks: EarlyStopping

11. keras.preprocessing.image: ImageDataGenerator

Layers:

Note: The model is sequential.

1. Conv2D, activation = relu

2. Max Pooling

3. Conv2D, activation = relu

4. Max Pooling

5. Flatten

6. Dense, activation = relu

7. Dropout

8. Dense, activation = softmax

Functions:

1. preprocess image:

Resizes image and makes sure it is properly read to prepare for use.

2. predict emotion:

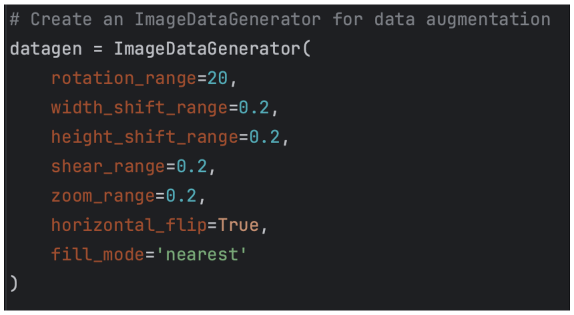
This function is what actually predicts the emotion shown in the image after everything is in the correct format and the model has been trained.

Model:

The model receives training images (all images are in .jpeg format) and is trained to learn from this data. The training dataset is divided into the three emotion classes of happy, sad, and angry, with corresponding images in each (20 images in each). The model goes through each dataset and assigns the corresponding emotion label to each image for training. The model is compiled through categorical cross-entropy. The training process also reshapes the images and converts them into one-hot encoded format for processing. After training, data augmentation is applied to increase accuracy of results.

Early stopping is also implemented for increased accuracy of results. This basically tells the program to stop training when parameter updates no longer yield improvements on test data.

Data augmentation is also used to ensure that the images are processed and learned from as accurately as possible.



```
# Create an ImageDataGenerator for data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

Data augmentation

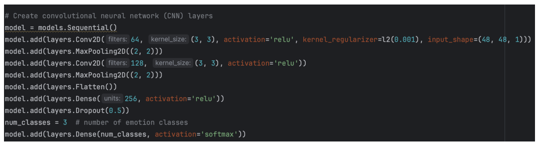
Statistics:

Learning Rate: 0.001

Patience: 5 (when to implement early stopping)

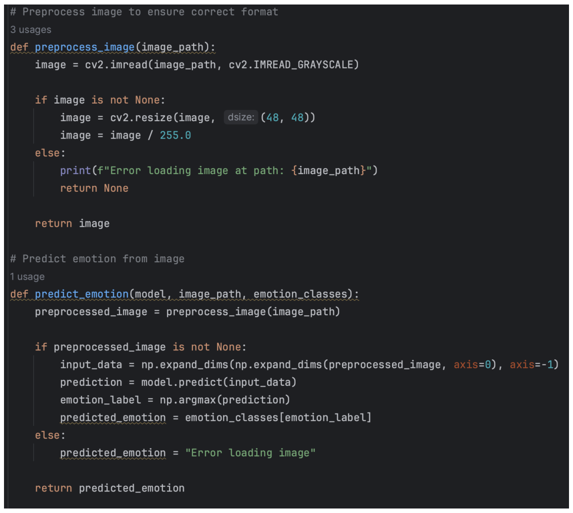
Epochs: compiling - 58, training - 20 (the number of times the learning algorithm will work through the dataset)

Test Size: 0.7 (size of data sets; 70% training data, the other 30% is test data)



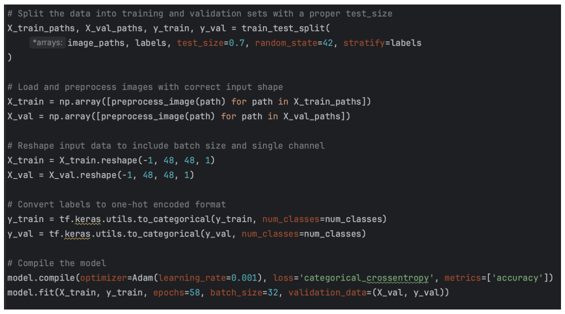
```
# Create convolutional neural network (CNN) layers
model = models.Sequential()
model.add(layers.Conv2D( filters: 64,  kernel_size: (3, 3), activation='relu', kernel_regularizer=l2(0.001), input_shape=(48, 48, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D( filters: 128,  kernel_size: (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense( units: 256, activation='relu'))
model.add(layers.Dropout(0.5))
num_classes = 3  # number of emotion classes
model.add(layers.Dense(num_classes, activation='softmax'))
```

Layers used to implement model



```
# Preprocess image to ensure correct format
3 usages
def preprocess_image(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    if image is not None:
        image = cv2.resize(image,  dsize: (48, 48))
        image = image / 255.0
    else:
        print(f"Error loading image at path: {image_path}")
        return None

    return image

# Predict emotion from image
1 usage
def predict_emotion(model, image_path, emotion_classes):
    preprocessed_image = preprocess_image(image_path)

    if preprocessed_image is not None:
        input_data = np.expand_dims(np.expand_dims(preprocessed_image, axis=0), axis=-1)
        prediction = model.predict(input_data)
        emotion_label = np.argmax(prediction)
        predicted_emotion = emotion_classes[emotion_label]
    else:
        predicted_emotion = "Error loading image"

    return predicted_emotion
```

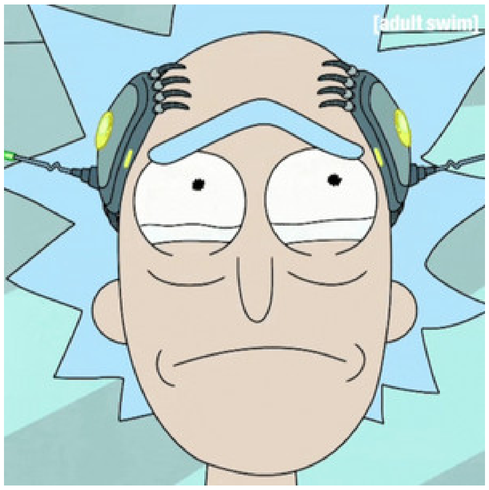
Functions for preprocessing image and predicting emotions

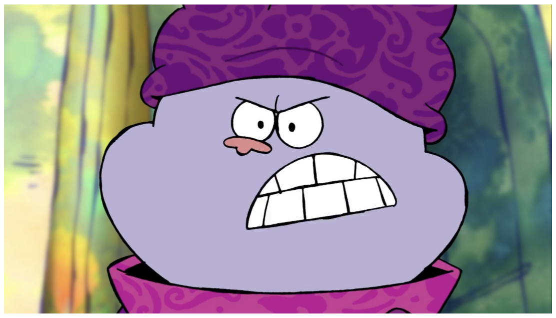


```
# Split the data into training and validation sets with a proper test_size
X_train_paths, X_val_paths, y_train, y_val = train_test_split(
    *arrays: image_paths, labels, test_size=0.7, random_state=42, stratify=labels
)

# Load and preprocess images with correct input shape
X_train = np.array([preprocess_image(path) for path in X_train_paths])
X_val = np.array([preprocess_image(path) for path in X_val_paths])

# Reshape input data to include batch size and single channel
X_train = X_train.reshape(-1, 48, 48, 1)
X_val = X_val.reshape(-1, 48, 48, 1)

# Convert labels to one-hot encoded format
y_train = tf.keras.utils.to_categorical(y_train, num_classes=num_classes)
y_val = tf.keras.utils.to_categorical(y_val, num_classes=num_classes)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=58, batch_size=32, validation_data=(X_val, y_val))
```

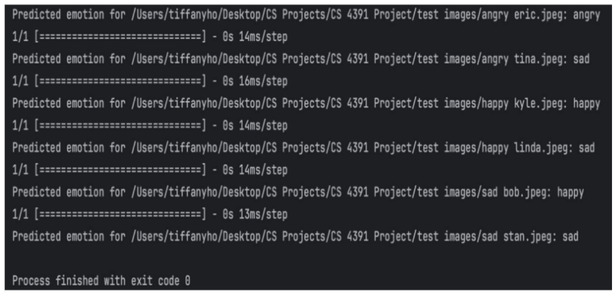
Training the model using training images

Example of a happy cartoon character facial expression


Example of a sad cartoon character facial expression


Example of an angry cartoon character facial expression

1.3 Experiments



```
Predicted emotion for /Users/tiffanyho/Desktop/CS Projects/CS 4391 Project/test images/angry eric.jpeg: angry
1/1 [==============================] - 0s 14ms/step
Predicted emotion for /Users/tiffanyho/Desktop/CS Projects/CS 4391 Project/test images/angry tina.jpeg: sad
1/1 [==============================] - 0s 16ms/step
Predicted emotion for /Users/tiffanyho/Desktop/CS Projects/CS 4391 Project/test images/happy kyle.jpeg: happy
1/1 [==============================] - 0s 14ms/step
Predicted emotion for /Users/tiffanyho/Desktop/CS Projects/CS 4391 Project/test images/happy linda.jpeg: sad
1/1 [==============================] - 0s 14ms/step
Predicted emotion for /Users/tiffanyho/Desktop/CS Projects/CS 4391 Project/test images/sad bob.jpeg: happy
1/1 [==============================] - 0s 13ms/step
Predicted emotion for /Users/tiffanyho/Desktop/CS Projects/CS 4391 Project/test images/sad stan.jpeg: sad


Process finished with exit code 0
```

Results were only able to achieve ~50% accuracy

To test the model, I used six test images, two for each emotion class.

My images for both training and testing data came from Google Images of various 2D animated cartoon characters from a variety of shows.

I was only able to achieve an accuracy of around 50% using this model. As a result, I did some more research and pinpointed three main issues that would improve the accuracy of the model if worked on further.

Issues:

1. Training images data set not large enough:
Because I collected my own training images, I only had 20 for each emotion class, resulting in a very small data set of 60 training images total.
Because of this, the model is less accurate because it does not have enough or images and thus less variety to learn from.

2. Needs more fine tuning of model training statistics:
Variables such as learning rate, epochs, patience, and test size mentioned in the previous section (method) could use further research and testing to determine the best combination of these factors for maximum accuracy.

3. Not enough layers:
Because this model is quite simple, it does not go too much into details. By adding more layers to the CNN, it will allow more in depth training and analysis, thus increasing the accuracy of the results.

1.4 Conclusion

In conclusion, for this project I implemented a CNN to detect emotions through the facial expressions of 2D animated cartoon characters. While it was only able to achieve 50% accuracy, I have pinpointed the main issues that would improve the model if implemented, and this has been a great experience.

1.5 References

A. Mittal, “Haar Cascades, explained,” Medium, https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d: :text=A%20Haar%20classifier%2C%20 or%20a,in%20an%20image%20and%20video.

Ali, Kamran Is ler, Ilkin Hughes, Charles. (2019). Facial Expression Recognition Using Human to Animated-Character Expression Translation. https://www.researchgate.net/publication/336551495

Facial expression recognition of animated characters ... - IEEE xplore, https://ieeexplore.ieee.org/document/9892186.

Jain, N., Gupta, V., Shubham, S. et al. Understanding cartoon emotion using integrated deep neural network on large dataset. Neural Comput Applic 34, 21481–21501 (2022). https://doi.org/10.1007/s00521-021-06003-9