Emotion Recognition Based on Facial Expressions

Arturo Llanas Banda
Intelligent Systems Engineering, Generation 2021
202102300405, 344160
Machine Learning, 281601
Advisor: Dr. Juan Carlos Cuevas Tello

December 1, 2024

Engineering Faculty, UASLP

Abstract

Human-computer interaction is a key factor for new artificial intelligences and can be significantly improved by incorporating emotion recognition, being part of non-verbal communication. Recognizing emotions through facial images is a technical and scientific challenge. This work aims to create an emotion classifier using different machine learning algorithms and compare their performances. Models such as Convolutional Neural Networks (CNN), Support Vector Machines (SVM) with feature extraction using ResNet and Deep Neural Networks (DNN) were used.

For the study, the AffectNet dataset was used instead of Fer2013, with initial processing that included normalization, data augmentation, and partitioning into training, validation, and test sets. The algorithms were evaluated using metrics such as precision and confusion matrices. The results highlighted the good performance of the CNN model, which outperformed the other models evaluated. Furthermore, the classifier was tested with new images to test its functionality.

It remains a challenge to improve this problem, due to the subjectivity of the problem. Despite its limitations and complexity, advances in this field are promising. Although the algorithms have shown solid performance, there is still considerable room to improve their accuracy, especially considering that even humans have difficulty identifying emotions consistently.

1 Content

1.1 Problem Statement

Facial emotion recognition (FER) is an essential component in human-computer interaction (HCI), allowing machines to appropriately interpret and respond to non-verbal cues. This has key applications in areas such as healthcare, where monitoring emotional states can improve diagnoses; in education, where it can be adapted to the emotional needs of students; and in commerce, by personalizing user experiences [2, 7].

Despite recent advances, the FER faces significant challenges. Facial expressions can vary considerably due to lighting, head position, partial occlusion, and cultural diversity. Furthermore, emotions are often subjective and contextual, which adds inherent complexity to the classification process. [5].

Deep learning, particularly convolutional neural networks (CNN), has demonstrated superior performance when extracting hierarchical features directly from images. However, it requires large amounts of labeled data and high computational resources. [4]. Alternatively, support vector machines (SVM), combined with feature extraction techniques such as ResNet, offer a more lightweight and effective solution for classification on moderate data sets [1]. Finally, dense neural networks (DNNs) provide flexibility in model building, although they may face overfitting issues in scenarios with limited or unbalanced data. For this project report, I use the AffectNet dataset, a widely recognized resource in the FER field, was used. This dataset includes images labeled with various emotions, providing an ideal context to evaluate and compare the performance of CNN, SVM with ResNet and DNN. This report seeks to identify the most efficient algorithm, considering both accuracy and practical applicability, and contribute to the design of robust recognition systems for improved HCI.

1.2 Methods/Algorithms

The three machine learning algorithms CNN, SVM and DNN were applied as mentioned above, but in order to use them, it was first necessary to understand what these models were, how they differ and also, how the data to be able to use it in the model.

1.3 Dataset Preprocessing

The AffectNet dataset [6] is a large-scale facial expression database designed for emotion recognition tasks. Unlike Fer2013, this one goes to an RGB color scale and not grayscale. It contains tens of thousands of images separated into 8 emotional categories: *neutral*, *happy*, *sad*, *surprise*, *fear*, *disgust*, *anger* and *contempt*. For this work, a subset composed of 29,042 images was used due to computational restrictions. The images were resized to 48×48 pixels to optimize training time while maintaining key facial features. Originally I had planned to use the Fer2013 dataset, however, I had many complications regarding the quality of the dataset because the images were distributed very unevenly. For example, classes like disgust had less than 500 data and others like happy had more than 4000, generating a huge gap for the classifier.

Distribution by Class: The distribution of AffectNet images in each class is as follows.

- **Neutral:** 5,126 images
- **Happy:** 5,044 images
- **Sad:** 3,091 images
- **Surprise:** 4,039 images
- **Fear:** 3,176 images
- **Disgust:** 2,477 images

- **Anger:** 3,218 images
- **Contempt:** 2,871 images

Image Processing: Each image was normalized to have pixel values in the range $[0, 1]$. The dataset was divided into training (80%), validation (10%) and test (10%) subsets. To address class imbalance, data augmentation techniques such as horizontal flipping and random rotation were applied during training in the case of Convolutional Neural Network (CNN) and Deep Neural Network (DNN).

```
# Load images and labels
def load_images_from_folders(base_dir, class_mapping, target_size
   =(48, 48)):# 48 for Computer resources
    X = []
    y = []
    for class_name, class_idx in class_mapping.items():
        folder_path = os.path.join(base_dir, class_name)
        for img_name in os.listdir(folder_path):
            img_path = os.path.join(folder_path, img_name)
            # Load images
            img = cv2.imread(img_path)
            if img is not None:
                # RGB
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                # Redimension and normalization
                img = cv2.resize(img, target_size)
                img = img / 255.0
                X.append(img)
                y.append(class_idx)
    return np.array(X), np.array(y)

# Load images and labels again
X, y = load_images_from_folders(base_dir, class_mapping)
```

Listing 1: Load labels and images CNN and DNN

For the Support Vector Machine (SVM) I had to change the image processing since the SVM has no way to extract features on its own, so I decided to use ResNet50 to extract the features from the data to be able to use them in the SVM, The point is that ResNet50 works best with images at a resolution of 224 x 224 pixels, however computationally it consumes a lot of resources, for this reason I decided to limit the data with those that were worked on the dataset at 1000 per class only and 128 x 128 resolution to balance efficiency and time.

```
# Load images and labels
def load_images_and_labels(base_dir, class_mapping, target_size
   =(128, 128), max_images_per_class=None):
    X = []
```

```
    y = []
    for class_name, class_idx in class_mapping.items():
        folder_path = os.path.join(base_dir, class_name)
        image_count = 0
        for img_name in os.listdir(folder_path):
            if max_images_per_class and image_count >=
               max_images_per_class:
                break
            img_path = os.path.join(folder_path, img_name)
            img = cv2.imread(img_path)
            if img is not None:
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                img = cv2.resize(img, target_size)
                X.append(img)
                y.append(class_idx)
                image_count += 1
    return np.array(X), np.array(y)

# Loads
X, y = load_images_and_labels(base_dir, class_mapping,
   target_size=(128, 128), max_images_per_class=1000)
```

Listing 2: Load labels and images SVM

1.4 Models Architecture

The machine learning algorithms implemented and evaluated for the emotion recognition task: Convolutional Neural Networks (CNN), Support Vector Machines (SVM) with feature extraction using ResNet50, and Deep Neural Networks (DNN) were designed with the computational resources and the dataset you are working with. In this report I focus more on the use of CNNs and the others were used more than anything to compare performances. To begin, it is necessary to understand what each of these models is.

1.4.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks, introduced by LeCun et al. [5], are models specialized in processing data with grid-shaped structures, such as images. They use convolutional layers to automatically extract relevant spatial features from images and pooling layers to reduce dimensionality.
In this project, a custom CNN architecture was implemented with four convolutional and pooling layers, dropout layers (Turn off a number of neurons randomly to maintain randomness) to prevent overfitting and a final dense layer for classification into the eight classes. The final architecture was the following.

```
def create_custom_cnn(classes):
    model = Sequential()

    # Block 1  Convolution
```

```
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
        input_shape=(48, 48, 3)))  # 3 chanels for RGB
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))#Pooling
    model.add(Dropout(0.25))

    # Block 2
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    # Block 3
    model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    # Block 4
    model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    # Flatten the features
    model.add(Flatten())

    # Dense Layers
    model.add(Dense(256, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(512, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    # Output [Neutral, Happy...]
    model.add(Dense(classes, activation='softmax'))

    # Model compilation
    model.compile(optimizer=Adam(learning_rate=0.001), loss='
        sparse_categorical_crossentropy', metrics=['accuracy'])

    return model

# Model creation
num_classes = 8  # Num of classes
model = create_custom_cnn(num_classes)
# Model training
history = model.fit(
    train_generator,
```

```
    validation_data=(X_test, y_test),
    epochs=30,
    class_weight=class_weights_dict
)
```

Listing 3: CNN architecture

In the first layer, the input is defined as (48,48,3) by the dimensions and color channels that the image has. The feature extraction mode in this case is Conv2D with a (3,3) kernel and relu activation that introduces nonlinearity to the model and avoids gradient fading. It starts with 32 output channels in the first layer and grows to 256 for the last.
Next comes the layer that flattens the features to a one-dimensional vector to be processed by the dense layers, both with relu activation.
Finally, in the output layer there is softmax activation that converts the outputs to probabilities per class, in this case for the 8 proposed classes. This architecture is compiled using the adam optimizer which is responsible for balancing weights and calculating the gradient.
As I mentioned, there were finally four convolution layers that I had to use for my model, this after a series of tests with different layers, optimizers and techniques such as data augmentation to generate realistic variations of the original images, such as rotations, flips, or brightness changes. This was only applied for this architecture due to the nature of the model itself, since it reduces overfitting and improves the model's ability to generalize to unseen data and in the other models it had no real justification for use.

```
batch_size = 64
steps_per_epoch = len(X_train) // batch_size

#Data augmentation
train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
#Application
train_generator = train_datagen.flow(X_train, y_train, batch_size
   =batch_size)
```

Listing 4: Data augmentation

CNNs are particularly desirable for AffectNet due to their ability to capture subtle spatial hierarchies in facial expressions.

1.4.2 Support Vector Machines (SVM)

Support Vector Machines, introduced by Cortes and Vapnik [1], are supervised machine learning models designed for classification tasks. They work by searching for the hyperplane that best separates classes in a high-dimensional space. In this project, SVMs were combined with the ResNet50 residual network [3] for feature extraction. ResNet50, a deep

learning model based on residual networks, extracted high-level features from AffectNet images, which were then classified by SVM.

```
# Feature extraction with resnet50
resnet_model = ResNet50(weights='imagenet', include_top=False,
   input_shape=(128, 128, 3))
def extract_features(model, X):
    X_preprocessed = preprocess_input(X)
    features = model.predict(X_preprocessed, batch_size=64,
       verbose=1)
    return features.reshape(features.shape[0], -1)  # Flatten

X_train_features = extract_features(resnet_model, X_train)
X_val_features = extract_features(resnet_model, X_val)
X_test_features = extract_features(resnet_model, X_test)
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_features)
X_val_scaled = scaler.transform(X_val_features)
X_test_scaled = scaler.transform(X_test_features)
# SVM training
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale', class_weight=
   'balanced', random_state=42)
svm_model.fit(X_train_scaled, y_train)
```

Listing 5: SVM architecture

These features are flattened and scaled with StandardScaler to optimize their distribution. The scaled features are then used to train an SVM model with an RBF (Radial Basis Function) kernel that measures the similarity between two points in space using an exponential function, configured to classify emotions with balanced handling of unequal classes.
This combination unites traditional machine learning techniques with advanced feature extraction capabilities.

1.4.3 Deep Neural Networks (DNN)

Deep Neural Networks, which have reemerged thanks to advances in training techniques [4], are multi-layer perceptrons designed to model complex, non-linear relationships in data. Unlike CNNs, DNNs do not perform explicit spatial feature extraction, which limits their ability to process image data without preprocessing. In this project, a DNN with three fully connected layers was implemented to classify emotions.

```
# Model creation
def create_dnn(input_shape, num_classes):
    model = Sequential()
    model.add(Flatten(input_shape=input_shape))  # Flat the
       images
    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(128, activation='relu'))
    model.add(Dropout(0.5))

    model.add(Dense(num_classes, activation='softmax'))  # Output
        layer

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

# Creation
input_shape = (48, 48, 3)  # shape and 3 for rgb chanels
dnn_model = create_dnn(input_shape, num_classes)
dnn_model.summary()
#Training of DNN
history = dnn_model.fit(
    X_train, y_train_onehot,
    validation_data=(X_val, y_val_onehot),
    epochs=30,
    batch_size=64,
    verbose=1
)
```

Listing 6: DNN architecture

In this case, there is not much to explain about the architecture carried out, it is similar to the Convolutional Neural Network, removing the part of the convolution with only the three connected layers. Although flexible, DNNs have limitations compared to CNNs on tasks like AffectNet due to the lack of specialization in the data.

1.5 Evaluation Metrics

To evaluate the performance of the models, standard metrics such as precision, recall, F1-score and accuracy were used. Additionally, confusion matrices were generated to visualize performance on each of the eight emotion classes. Among the evaluated models, the CNN showed the best performance, followed by the SVM+ResNet combination and finally the DNN. These results highlight the importance of selecting an appropriate architecture and dealing with the challenges inherent in imbalanced data and subtle differences in facial expressions, in addition to computational limitations or issues such as timing.

- **Precision**: Proportion of correct predictions for a class over the total predictions made for that class. Is the answers to the question: Of the predictions I made for a class, how many were correct?
- **Recall (Sensibility)**: Proportion of correctly classified examples of a class over the total number of real examples of that class. is the answer to the question: Of all the examples that actually belong to a class, how many did I correctly identify?

- **F1-Score**: Mean between precision and recall; measures the balance between the two.
- **Accuracy (Global precision)**: Proportion of correct predictions over the total number of examples in all classes.

1.6 Results

The results of the confusion matrix generated by the predictions of each model will be seen in the following figures, starting with the figure 1 where we can see the one that corresponds to the CNN.

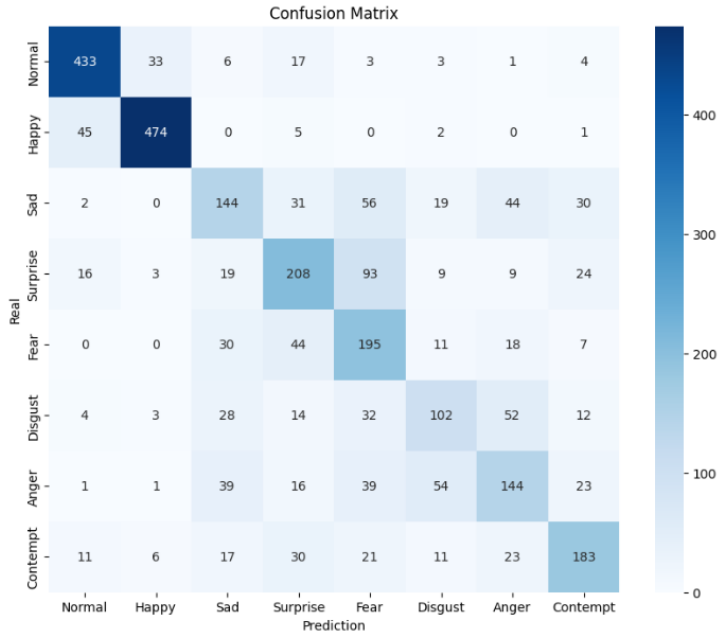

Figure 1: CNN Confusion Matrix.

As can be seen in the table 1, classifying emotions as happy, neutral and contempt showed positive results. Classes like disgust, anger and fear proved very challenging.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Normal	0.85	0.87	0.86	500
Happy	0.91	0.90	0.91	527
Sad	0.51	0.44	0.47	326
Surprise	0.57	0.55	0.56	381
Fear	0.44	0.64	0.52	305
Disgust	0.48	0.41	0.45	247
Anger	0.49	0.45	0.47	317
Contempt	0.64	0.61	0.62	302
Accuracy		0.65		2905
Macro Avg	0.61	0.61	0.61	2905
Weighted Avg	0.65	0.65	0.65	2905

Table 1: Classification metrics for CNN model predictions.

In figure 2 we can see the confusion matrix of the SVM+ResNet50 model with an amount of data reduced by the processing time.

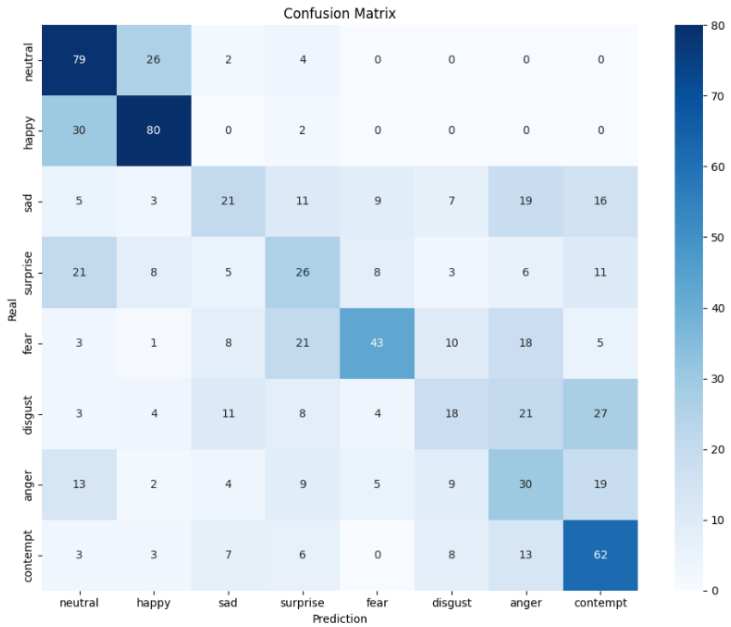

Figure 2: SVM Confusion Matrix.

In the table 2, we can see that the precision for each class dropped considerably compared to the CNN, but the precision trend was maintained in the same classes, the lowest being the same as in the CNN model.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Neutral	0.50	0.71	0.59	111
Happy	0.63	0.71	0.67	112
Sad	0.36	0.23	0.28	91
Surprise	0.30	0.30	0.30	88
Fear	0.62	0.39	0.48	109
Disgust	0.33	0.19	0.24	96
Anger	0.28	0.33	0.30	91
Contempt	0.44	0.61	0.51	102
Accuracy	0.45			800
Macro Avg	0.43	0.43	0.42	800
Weighted Avg	0.44	0.45	0.43	800

Table 2: Classification metrics for SVM model predictions.

Finally, in figure 3 we can observe the confusion matrix of the DNN model without specialized feature extraction.

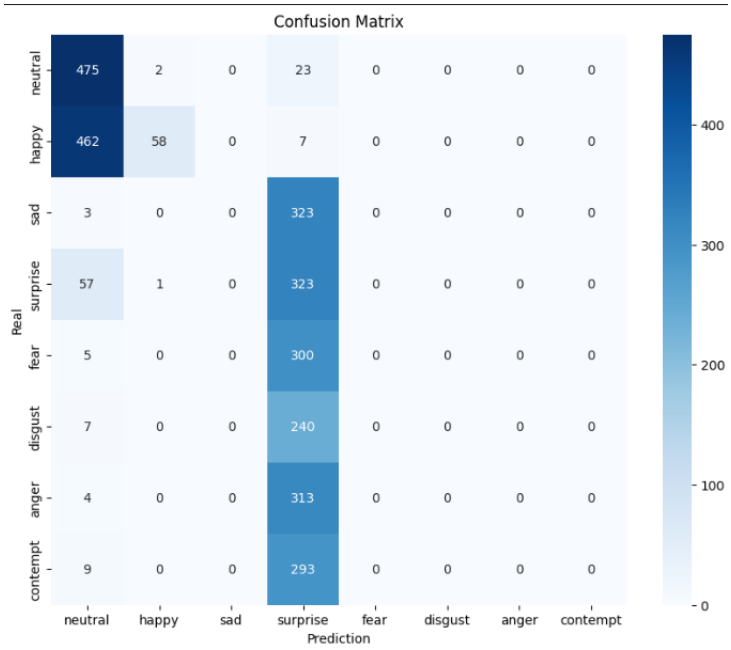

Figure 3: DNN Confusion Matrix.

in the table 3 and in the Confusion Matrix, we can observe that the precision for each class is really low for almost all classes, changing the trend shown in the predictions of the other models, being in this case Surprise the most predicted class. Due to the low capacity of the model, more than half of the classes have a null prediction by the multilayer model.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Neutral	0.46	0.95	0.62	500
Happy	0.95	0.11	0.20	527
Sad	0.00	0.00	0.00	326
Surprise	0.18	0.85	0.29	381
Fear	0.00	0.00	0.00	305
Disgust	0.00	0.00	0.00	247
Anger	0.00	0.00	0.00	317
Contempt	0.00	0.00	0.00	302
Accuracy	0.29			2905
Macro Avg	0.20	0.24	0.14	2905
Weighted Avg	0.28	0.29	0.18	2905

Table 3: Classification metrics for DNN model predictions.

Comparison of the accuracy and performance of each model

Models	CNN	SVM	DNN
Accuracy	0.65	0.45	0.29
Ex.Time	15mins	28mins	13mins

Table 4: Comparison of model performance.

A broad superiority is observed in relation to training time and precision by the Convolutional Neural Network. This is due to the nature of the problem since these are focused on this type of circumstances unlike Support Vector Machines that work with other problems. Since CNN is the best model, I made a program where images are loaded in a 3 x 3 grid and they are classified by the model I saved. We can see in figure 4 that these are new images for the model and also, they are not labeled. This last activity and results left me with a conclusion about this problem that I did not contemplate when proposing this project.

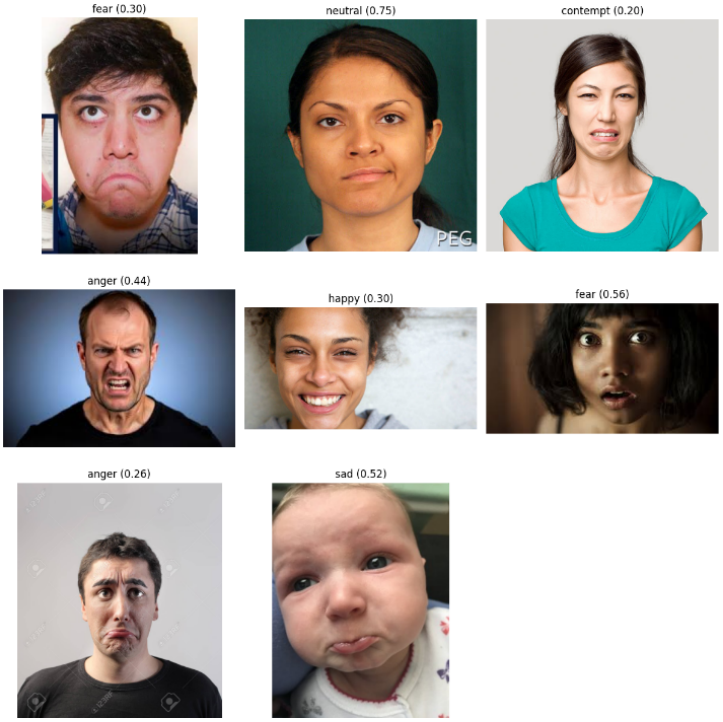

Figure 4: Model predictions on new data.

2 Conclusions

During the execution of this project I encountered several problems such as a resource limitation by services such as Google Colab and the use of GPUS that prevented me from properly using an architecture based on the use of Transfer Learning and ResNet50, VGG16 or MobileNet. I also changed datasets in the last few weeks due to a large imbalance between classes in the previous dataset I was using. These problems were resolved thanks to the information I found in the references about the topic, as well as seeing what people who did something similar before with the same dataset in Kaggle did. I noticed that in most reports, the highest accuaracy achieved was no greater than 0.75, which made me realize that it was not a problem with my implementation but rather a bigger problem that I did not realize at the time. principle. The problem is that, even for a human, it is difficult to recognize the emotions of another human based on their facial expressions, it is really a complex issue. I realized this when testing the model since these new images did not have a label, so when I saw the results of the model, I had a label in mind, but the ones that the model gave me also seemed accurate, especially in classes like dislike and contempt where it is really difficult to appreciate a difference between the

two.
In conclusion, I consider that it is a broader topic than I initially thought and that it involves many disciplines, but doing it also helped me understand that each machine learning model is different and one can be better than another depending on the problem. For image classification, based on what I have seen in my project, I can say that CNNs are the most suitable initially and then you can scale them to a more complex model. To people who would like to do something regarding this topic, I would like to recommend that they investigate more algorithms, more forms of optimizations, parameters, functions, techniques and observe what other people who have already worked with the same problem did.

3 Bibliography

References

[1] Corinna Cortes. Support-vector networks. *Machine Learning*, 1995.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. pages 248–255, 2009.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[6] Ali Mollahosseini, Behzad Hasani, and Mohammad H Mahoor. Affectnet: A database for facial expression, valence, and arousal computing in the wild. *IEEE Transactions on Affective Computing*, 10(1):18–31, 2017.

[7] Wenyi Zhao, Rama Chellappa, P. Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM Computing Surveys*, 35:399–458, 2003.