CS767 Advanced Machine Learning
Surya Pranav Sukumaran

Recognition of Emotion from Images - Using Deep Learning

BOSTON UNIVERSITY

Abstract

This case study for CS767 dives into emotion recognition from images using deep learning. It examines CNN architectures, focusing on a baseline CNN, a modified VGGNet, and a pre-trained ResNet-50 with transfer learning. The study employs the FER 2013 dataset, containing grayscale facial expression images, to train and evaluate these models. The report outlines the methodology, experiments, and results, emphasizing deep learning's role in emotion detection. It concludes with ResNet-50 showing superior performance due to its architecture, highlighting deep learning's potential in human-computer interaction which can be beneficial in terms of mental health, it has been observed by Turcian D and Stoicu-Tivadar V. that by recognizing facial expressions, depression, schizophrenia, or other similar conditions can be detected. Additionally, the report covers practical applications, including a Flask API app for emotion detection, and discusses limitations and future work, suggesting exploration of more pre-trained models and advanced techniques for enhancing accuracy and applicability in healthcare and in other fields.

Introduction

Human emotion recognition stands as one of the most fascinating and challenging frontiers in the realm of artificial intelligence. The ability to discern emotional states from images not only pushes the envelope of computer vision but also bridges the gap between human-computer interaction. This case study goes into the intricate process of detecting and classifying human emotions from images using deep learning.

Emotions are conveyed through many different facial expressions, often subtle and complex. Capturing this nuance requires a system that can interpret a high degree of variability within visual data. Traditional machine learning techniques have laid the groundwork, but they fall short in grasping the depth of human expressions. Deep learning, with its hierarchical feature learning, offers a profound leap in this domain. It mimics the layered processing of the human brain, yielding an unparalleled proficiency in pattern recognition.

This case study explores the application of Convolutional Neural Networks (CNNs), an architecture inspired by the organization of the visual cortex. Starting with a baseline CNN model, and a modified VGGNet architecture model, finally ending with the implementation of a pre-trained ResNet-50 model equipped with transfer learning. The progression captures the evolution from a basic understanding of emotional cues to a more complex recognition capability, adept at distinguishing even the most closely related emotions like ‘fear’ and ‘surprised’ or ‘neutral’ and ‘sad’.

Utilizing the FER 2013 dataset, composed of 48x48 pixel grayscale images of facial expressions, we have systematically assessed and compared the performance of three distinct models. Each model's architecture is a testament to the advancements in deep learning, and their performance metrics serve as a quantifiable measure of these advancements in emotion recognition tasks. This report goes through the methodology, experiments, and critical analyses that led to the development of a deep learning solution capable of interpreting human emotions.

Installations

Here are the installations required for this machine learning project.

1. **TensorFlow:** A comprehensive machine learning library.

```
!pip install tensorflow
```

2. **Keras Applications:** Provides pre-trained models for Keras.

```
!pip install keras_applications
```

3. **Keras:** A high-level neural networks API.

```
!pip install keras
```

4. **VGGFace:** A deep learning project for facial recognition.

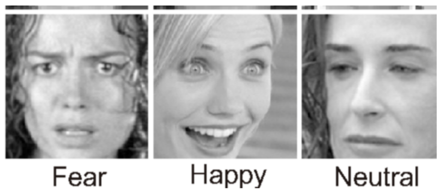
```
!pip install git+https://github.com/yaledhlab/vggface.git
```

Dataset

FER-2013 stands for **Facial Expression Recognition 2013**. This dataset is widely used in the field of computer vision, particularly for training models to recognize human emotions from facial images.

- **Content:** The dataset consists of 35,887 grayscale, 48x48 pixel images of faces.

- **Emotions:** Images are categorized into seven emotion classes: anger, disgust, fear, happiness, sadness, surprise, and neutral.
- **Usage:** Primarily used for training and evaluating models for facial expression recognition tasks. It's a standard benchmark in this domain.



Accessing the FER-2013 Dataset

We can access the FER-2013 dataset from Kaggle. Here's a simple way to download and load the dataset:

1. Download the Dataset:
   - Visit the FER-2013 Kaggle page (https://www.kaggle.com/datasets/msambare/fer2013).
   - Download the dataset to your local machine.

2. Load the Dataset:
   - Unzip the dataset file.
   - The dataset contains two folders, train and test.

After extracting the `fer.zip` dataset from Kaggle, it's essential to organize the data into three distinct sets: training, testing, and validation. This organization is crucial for training and evaluating the model effectively. The primary reason for this is because there are both private test data and public test data in the test directory, we want to split this. Here's how the process is carried out:

1. **Setting Up Directories**

First, define the base directory where the dataset resides, and set up paths for the source (`test`) and the new `validation` directories:

```
base_dir = '/path/to/fer' # include the path to the dataset
source_dir = os.path.join(base_dir, 'test')
```

```
validation_dir = os.path.join(base_dir, 'validation')
```

2. **Creating Validation Set**

The FER dataset comes with predefined training and test sets. To create a validation set, we move a subset of the test images to a new validation directory. This helps in tuning the model parameters without using the test set, preserving its integrity for final model evaluation.

3. **Handling `.DS_Store` Issue**

While processing the dataset, especially on macOS, a `.DS_Store` file can be created automatically. This file should not be considered a class of images. Therefore, the script includes a check to skip this file to prevent it from being misinterpreted as an emotion class by data generators.

4. **Moving Files**

For each emotion class in the test directory, the script moves files starting with `PrivateTest` to the corresponding emotion class directory in the validation directory. This naming convention is specific to the FER dataset.

Here is the Python script used for the above steps:

```python
import os
import shutil

# Defining directories
base_dir = '/path/to/fer'
source_dir = os.path.join(base_dir, 'test')
validation_dir = os.path.join(base_dir, 'validation')

# Moving the PrivateTest files to the validation directory
for emotion_class in os.listdir(source_dir):
    if emotion_class == '.DS_Store':
        continue
    class_path = os.path.join(source_dir, emotion_class)
    validation_class_path = os.path.join(validation_dir, emotion_class)
    if not os.path.exists(validation_class_path):
        os.makedirs(validation_class_path)

    if os.path.isdir(class_path):
        for file in os.listdir(class_path):
            if file.startswith('PrivateTest'):
                                shutil.move(os.path.join(class_path,  file),
validation_class_path)

print("PrivateTest files have been moved to the 'validation' directory.")
```

Preprocessing

General Model Preprocessing

For a general model, we keep the same dimensions as the original dataset. Data augmentation techniques are applied to the training set to improve the model's ability to generalize. This includes rotation, width and height shifts, zoom, and horizontal flipping. All images are rescaled to have pixel values in the range [0, 1]. Below is the python code for the preprocessing:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def get_datagen(dataset, aug=False):
    if aug:
        datagen = ImageDataGenerator(
            rescale=1./255,
            rotation_range=10,
            width_shift_range=0.1,
            height_shift_range=0.1,
            zoom_range=0.1,
            horizontal_flip=True)
    else:
        datagen = ImageDataGenerator(rescale=1./255)

    return datagen.flow_from_directory(
        dataset,
        target_size=(48, 48),
        color_mode='grayscale',
        shuffle=True,
        class_mode='categorical',
        batch_size=64)
```

```
# Creating data generators
# only train generator is "True" saying it should be augmented
train_generator = get_datagen('/content/fer_data/train', True)
validation_generator = get_datagen('/content/fer_data/validation')
test_generator = get_datagen('/content/fer_data/test')
```

```
Found 28709 images belonging to 7 classes. # Train (Augmented)
Found 3589 images belonging to 7 classes. # Validation
Found 3589 images belonging to 7 classes. # Test
```

Transfer Learning Model Preprocessing

For the ResNet-50 model, which is typically used with RGB images, the preprocessing includes resizing the images to 197 x 197 pixels and converting them from grayscale to RGB. This is necessary as ResNet-50 is pre-trained on RGB images and the minimum we can input is 197 x 197. The rescaling and data augmentation techniques remain the same as for the general model.

Modifications for the ResNet-50 preprocessing include changing the `target_size` to (197, 197) and `color_mode` to 'rgb' in the `ImageDataGenerator`. For ResNet-50, the datagen function is modified like below:

```
    return datagen.flow_from_directory(
        dataset,
        target_size=(197, 197),
        color_mode='rgb',  # Changed to 'rgb'
        shuffle=True,
        class_mode='categorical',
        batch_size=64)
```

Models

Baseline Model Architecture

The baseline model for the emotion detection project is a straightforward yet effective convolutional neural network (CNN). This architecture is custom-designed, emphasizing simplicity while being capable of handling the complexity of facial emotion recognition. Below is a breakdown of the model:

Type: Custom Convolutional Neural Network (CNN).
Input Shape: The model accepts grayscale images of size 48x48 pixels.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dropout, Flatten,
Dense

Initialize the Sequential model
model = Sequential()
```

First Convolutional Layer
- 32 filters, each with a kernel size of 3x3
- ReLU activation function is used for non-linearity
- 'same' padding ensures the output feature map has the same width and height as the input
- Input shape is set for 48x48 pixel images with a single color channel (grayscale)

```
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', activation='relu',
input_shape=(48, 48, 1)))
```

First Max Pooling Layer
- Pool size of 2x2 reduces the spatial dimensions (width and height) by half

```
model.add(MaxPool2D(pool_size=(2, 2)))
```

First Dropout Layer
- Randomly sets 25% of input units to zero to prevent overfitting

```
model.add(Dropout(0.25))
```

Second Convolutional Layer
- Increases the depth of the feature maps with 64 filters

```
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
```

Second Max Pooling Layer

```
model.add(MaxPool2D(pool_size=(2, 2)))
```

Second Dropout Layer

```
model.add(Dropout(0.25))
```

Flatten Layer

- Flattens the 3D output to a 1D vector to feed into the dense layer

```
model.add(Flatten())
```

First Dense Layer

- Fully connected layer with 128 units

```
model.add(Dense(128, activation='relu'))
```

Third Dropout Layer

```
model.add(Dropout(0.25))
```

Output Dense Layer

- Outputs a probability distribution with 7 units, one for each class
- Softmax activation function is used for multi-class classification

```
model.add(Dense(7, activation='softmax'))
```

The total number of parameters (1,199,495) indicates the capacity of the model to learn from data. The trainable parameters (also 1,199,495) are the weights that will be updated during the training process. There are no non-trainable parameters, which means all parts of the model will be updated during training. The architecture is typical for image classification tasks, and it should serve well as a baseline for comparing more complex models or for further tuning to improve performance.

VGG-Style CNN

The VGG architecture, developed by the Visual Graphics Group at Oxford (hence VGG), is a prime example of a simple, yet powerful CNN. It standardizes the convolutional layer to use small (3 x 3) filters, followed by a max pooling layer. This approach makes the design of the network straightforward while maintaining high performance.

1. Model Architecture

Our model is a Sequential model in Keras, consisting of several convolutional stages, each followed by batch normalization and max pooling. The stages increase the depth of the network while reducing the spatial dimensions. We start by defining a Sequential model:

```
model = Sequential()
```

2. Convolutional Stages

The model consists of four convolutional stages. Each stage comprises Conv2D layers followed by BatchNormalization and MaxPooling2D.

Convolutional Stage 1

- Two Conv2D layers with 64 filters of size (3 x 3), using ReLU activation and 'same' padding.
- BatchNormalization after each Conv2D layer.
- A MaxPooling2D layer with a pool size of (2 x 2).

```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=input_shape))
model.add(BatchNormalization())
...
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Stages 2 to 4

- Similar structure as Stage 1 but with an increased number of filters: 128 in Stage 2, 256 in Stage 3, and 512 in Stage 4.

```
# Convolutional Stage 2
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
...
```

```
# Convolutional Stage 4
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
```

3. Regularization with Dropout and L2 Regularization

- A Dropout layer follows the last convolutional stage to prevent overfitting by randomly setting input units to 0 at each update during training.
- Dense layers use L2 regularization, which penalizes the weights during training, further helping to prevent overfitting.

```
model.add(Flatten())
model.add(Dropout(drop))
model.add(Dense(4096, activation='relu', kernel_regularizer=l2(l2_reg)))
```

4. Output Layer

The final Dense layer uses softmax activation suitable for multi-class classification. The number of units in the final layer corresponds to the number of classes, in this case it is 7 classes for the 7 emotions from the dataset.

```
model.add(Dense(num_classes,                          activation='softmax',
kernel_regularizer=l2(l2_reg)))
```

5. Compiling the Model

- The model is compiled with the Adam optimizer and categorical cross-entropy loss function. Adam optimizer is efficient for large datasets and high-dimensional parameter spaces. However, different optimizers might be used later on to improve validation accuracy. The learning rate is fixed at 0.001.

```
model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
```

6. Training the Model

- The model is trained using a fit method. Training data is provided through generators.

```
history = model.fit(
   train_generator,
   steps_per_epoch=train_generator.samples // train_generator.batch_size,
   validation_data=validation_generator,
                      validation_steps=validation_generator.samples       //
validation_generator.batch_size,
   epochs=100, verbose=1)
```

Deep Learning with ResNet and Transfer Learning

ResNet, short for Residual Networks, is a type of convolutional neural network (CNN) architecture that was designed to enable the training of very deep neural networks with the aim of achieving state-of-the-art results in various computer vision tasks. ResNet was introduced by Kaiming He et al. in their paper "Deep Residual Learning for Image Recognition" at the 2015 ICCV conference, where they won 1st place on the ILSVRC 2015 classification task.

Deep networks face two primary challenges as they grow in layers: gradient dissipation (or vanishing) and network degradation. Gradient dissipation refers to the instability of gradients in very deep networks, where they can become too small to carry significant information, leading to a halt in learning. Network degradation, on the other hand, is the phenomenon where the accuracy starts to saturate and then degrades rapidly as the network goes deeper, an issue not attributed to overfitting as it also affects training accuracy.

ResNet's architecture addresses these challenges by incorporating techniques like Batch Normalization and ReLU activation to mitigate gradient dissipation. But most importantly, it introduces a unique concept of learning residuals. Instead of functions that map inputs to outputs directly, ResNet layers are designed to learn the difference between the input and the output, termed as the residual. This is achieved through identity mapping (the input) and residual mapping (the learned difference), summed up to form the final output.
residual blocks with skip connections, which allow the network to learn residual functions with reference to the layer inputs, instead of learning unreferenced functions.

Mathematically, instead of aiming to learn a direct mapping of F(x) from input x , ResNet learns the residual function (F(x) - x). This approach helps in combating the vanishing gradient problem that often occurs with networks as they grow deeper.

Residual Block

Residual block consists of a stack of layers, followed by a shortcut connection that bypasses one or more layers. The bypassed layers are referred to as the "residual" of the block. The output of the block is obtained by adding the output from the bypassed layers to the output from the residual.

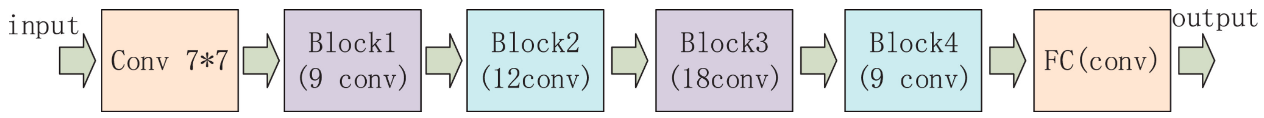


Figure 1: Residual Block

Advantages of ResNet

The residual learning framework eases the training of networks that are substantially deeper than those used previously. This depth is crucial for achieving high performance in complex visual tasks. Additionally, ResNet models have shown remarkable generalization performance, which means they perform well on new, unseen datasets.

Transfer Learning

Transfer learning is a technique where a model developed for one task is reused as the starting point for a model on a second task. It's particularly popular in deep learning where large datasets and extensive training are required to develop neural network models. Using a pretrained model like ResNet-50, which has already been trained on a large dataset (like ImageNet), saves time and computational resources, and often leads to better performance.

I decided to use transfer learning in this case study because of:

- **Efficiency**: Training a deep neural network from scratch requires a significant amount of data and computational power. Transfer learning allows us to start with patterns learned from solving a similar problem.
- **Speed:** It often takes less time to fine-tune a pre-trained network than to train a new one from scratch.
- **Performance:** Pre-trained networks have already learned a rich set of features that can be beneficial for the new task, often leading to improved performance.

Fine-Tuning a Pre-Trained ResNet-50 Model

Now, let's delve into the steps for fine-tuning a pre-trained ResNet-50 model.

Step 1: Loading the Pre-Trained Model

We begin by loading the `ResNet-50` model without its top layer since we aim to customize the classifier for our specific task. This is achieved using the `VGGFace` library:

```
from keras_vggface.vggface import VGGFace

vgg_notop     =     VGGFace(model='resnet50',     include_top=False,
input_shape=(Resize_pixelsize, Resize_pixelsize, 3), pooling='avg')
```

Step 2: Customizing the Model

We append our own fully connected layers to the base model. This allows the model to learn features specific to our new dataset.

We extend the model by adding several layers:

- Flatten: Converts the pooled feature map to a single column that is passed to the fully connected layers.
- Dense: Fully connected layers with ReLU activation.
- Dropout: A regularization technique where randomly selected neurons are ignored during training, preventing overfitting.

```
from keras.layers import Flatten, Dense, Dropout
from keras.models import Model
last_layer = vgg_notop.get_layer('avg_pool').output
x = Flatten(name='flatten')(last_layer)
x = Dropout(DROPOUT_RATE, name='dropout_fc6')(x)
x = Dense(4096, activation='relu', name='fc6')(x)
x = Dropout(DROPOUT_RATE, name='dropout_fc7')(x)
x = Dense(1024, activation='relu', name='fc7')(x)
x = Dropout(DROPOUT_RATE, name='dropout_fc8')(x)
out = Dense(7, activation='softmax', name='classifier')(x)

model = Model(inputs=vgg_notop.input, outputs=out)
```

Step 3: Freezing the Layers for Feature Extraction

The initial layers of a pretrained model capture universal features like edges and blobs. To keep these intact, we freeze them during the training of our new dataset. Freezing means that the weights of these layers will not be updated

```
for i in range(FROZEN_LAYER_NUM):
    if i not in batch_norm_indices:
        vgg_notop.layers[i].trainable = False
```

Step 4: Compiling the Model

We use SGD (Stochastic Gradient Descent) as the optimizer, with a specific learning rate and decay. The model is compiled with a loss function and metrics for performance evaluation.

```
model.compile(optimizer=sgd_optim,          loss='categorical_crossentropy',
metrics=['accuracy'])
```

Step 5: Training the Model

The model is trained using the `fit` method. We use `ReduceLROnPlateau` to reduce the learning rate when the validation accuracy plateaus, optimizing performance.

```
history = model.fit(...)
```

Step 6: Saving the Model

Finally, the model is saved for future use.

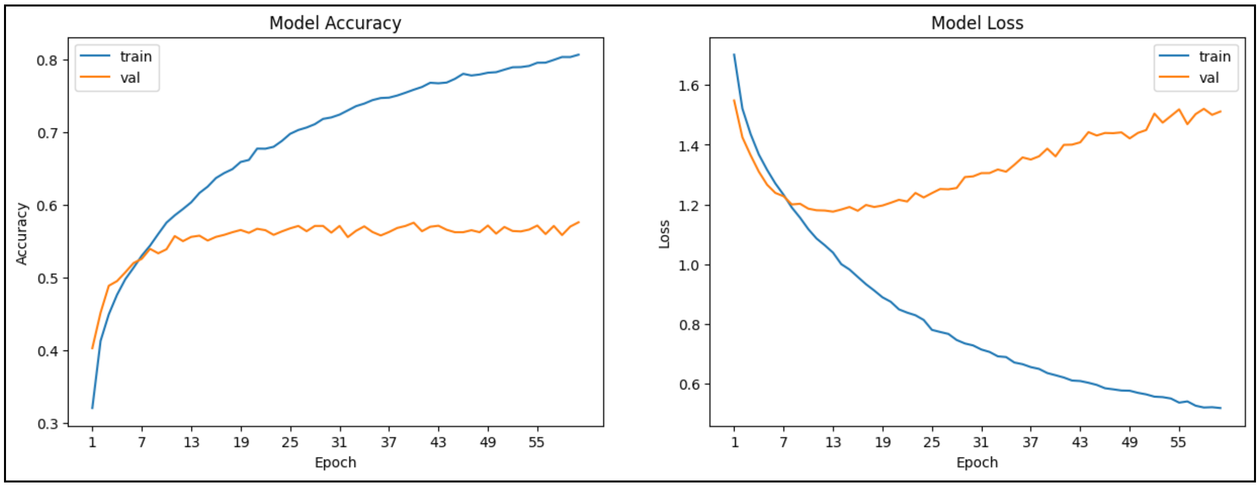
```
model.save('pretrained_Resnet.h5')
```

Optimizers

In this project, the Adam optimizer was initially chosen for the baseline CNN and VGGNet-style models due to its rapid convergence capabilities compared to SGD (Stochastic Gradient Descent). However, considering research by Hardt, Recht, & Singer (2016), which suggests SGD may yield better generalization and stability despite slower convergence, the ResNet-50 model was trained using the SGD optimizer. The intent was to enhance the model's accuracy even if the training process required more time.

Results and Evaluation

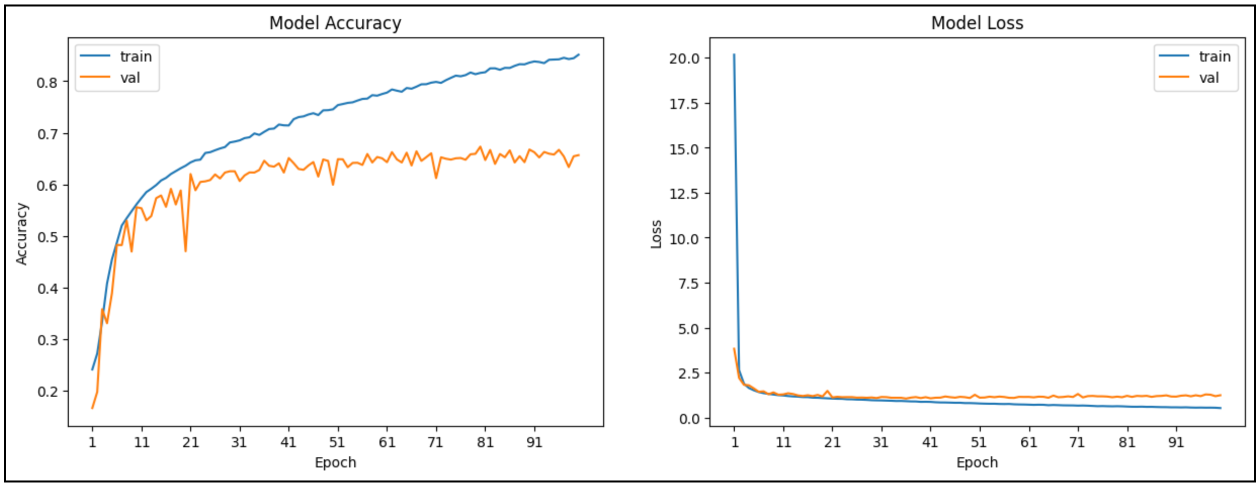
Accuracy and Loss Plots



Plot 1: Training and Validation accuracy and loss plot for baseline simple model

The training accuracy increases steadily, suggesting that the model is learning and improving from the training data. The training loss decreases, which is expected as the model optimizes its weights. When it comes to the validation performance, the accuracy plateaus around 0.6, which is significantly lower than the training accuracy. This indicates that the model may not generalize as well to unseen data, which is a sign of overfitting. The validation loss decreases initially but starts to increase after a certain number of epochs, this is a clear sign of overfitting.

```
# Baseline CNN
Test accuracy: 0.5617163777351379
Test loss: 1.581886887550354
```

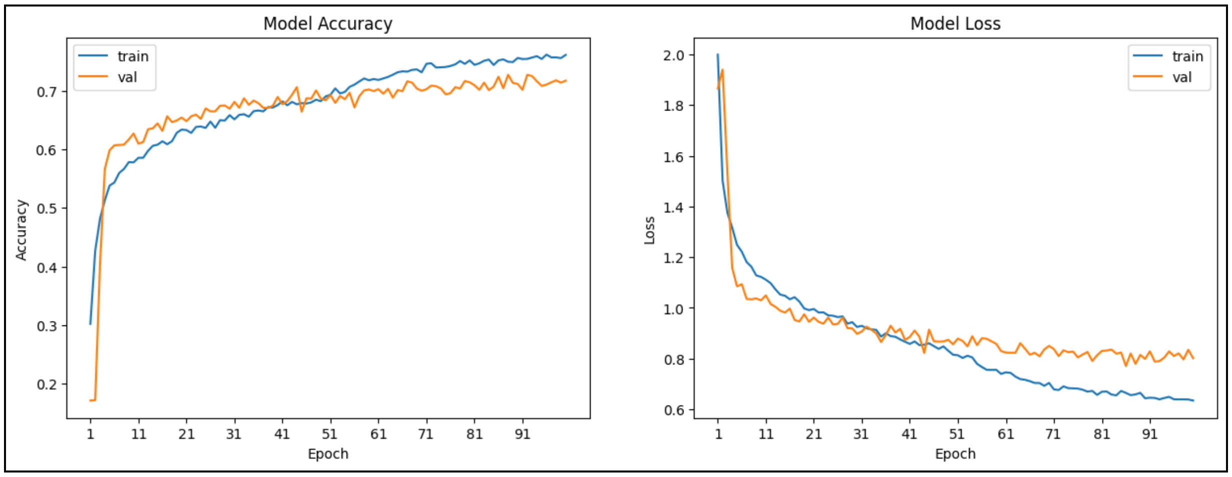


Plot 2: Training and Validation accuracy and loss plot for VGG model with L2 regularizer

In regards to the validation accuracy compared to the previous model, there is a significant improvement, which is more aligned with the training accuracy. This suggests that the VGG model is generalizing better than the simple CNN model. Both training and validation losses have decreased and stabilized, indicating that the model is converging well and not overfitting as much as the simple CNN model. This VGG model seems to have a more stable learning curve in both accuracy and loss plots, with fewer fluctuations, suggesting that the model is learning more effectively over time.

The increased performance in this model can be attributed to its depth, using many layers of convolutional filters. This depth allows the network to learn a hierarchy of features at multiple levels of abstraction. In our case study of recognition of emotions, the lower layers might learn to detect simple features such as edges and corners relevant to facial structures. The intermediate layers could learn to combine these features to detect more complex structures, like eyes, noses, or mouths. Higher Layers might abstract these features further to understand entire facial expressions associated with emotions.

```
#VGG Model
Test accuracy: 0.6667595505714417
Test loss: 1.3561335802078247
```



Plot 3: Training and Validation accuracy and loss plot for transfer learning ResNet50 model using SGD optimizer

Based on the plots, the validation accuracy for the ResNet-50 model is higher and closer to the training accuracy, indicating better generalization than both the simple CNN and VGG models. Both training and validation loss decrease steadily and converge closely, which implies that the fine-tuning process effectively adapted the pre-trained weights to the specific task of emotion detection without overfitting.

The use of a pre-trained ResNet-50 model has leveraged the power of transfer learning, where the model has been pre-trained on a large dataset (VGGFace) and has learned a rich representation of features. These features are transferable to the task of emotion detection, even with fine-tuning on a smaller dataset.

ResNet-50 is known for its depth and the use of residual connections that help in training deep networks by allowing gradients to flow through the network without disappearing. This depth and architecture facilitate the learning of more abstract features that are beneficial for complex tasks like emotion recognition. The fine-tuning has allowed the pre-trained model to adjust its learned features to better suit the emotion detection task. This is evident from the improved accuracy and loss metrics. It is important to note that the SGD optimizer also resulted in a higher accuracy.

```
#ResNet50
Test accuracy: 0.7093898057937622
Test loss: 0.845494270324707
```

Baseline CNN
Test Accuracy: **56.17%**
Test Loss: **1.58**

The Baseline CNN provides a modest accuracy that can be considered as a starting point for emotion detection tasks. The relatively high loss indicates that there is significant room for improvement. This model might have limitations in terms of its ability to capture the complexity of the data due to its simpler architecture.

VGG Model
- Test Accuracy: **66.68%**
- Test Loss: **1.36**

The VGG model marks a substantial improvement over the Baseline CNN, with a 10% increase in accuracy. This jump highlights the benefits of using a deeper and more complex network for feature extraction. The lower loss suggests that the model has a better understanding of the data, although there may still be some issues with overfitting or inadequate feature representation, as indicated by the fact that the loss remains above 1.

ResNet50
- Test Accuracy: **70.94%**
- Test Loss: **0.85**

The ResNet-50 model, utilizing transfer learning, shows the best performance with nearly 71% accuracy. This is a significant improvement over both the Baseline CNN and the VGG model, demonstrating the effectiveness of transfer learning from a pre-trained network. The considerable reduction in loss indicates that the model predictions are more confident and closer to the true labels.

Confusion Matrix

Baseline CNN Model:

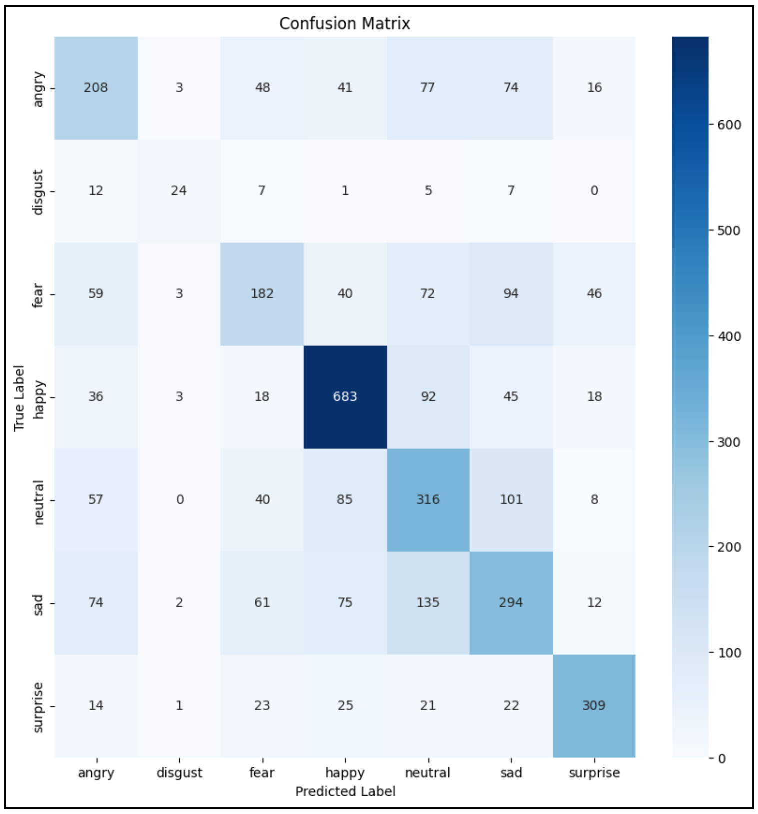


Figure 2: Confusion matrix for Baseline CNN Model for all 7 classes (emotions)

This model performs best in recognizing 'happy' emotions, with 683 correct predictions. This indicates that the 'happy' emotion has distinctive features that this CNN is able to learn effectively.

However, there are some confusions between categories, for 'Angry' and 'Surprise' emotions the model often confuses 'angry' with 'surprise', with 77 instances incorrectly labeled as 'surprise'. There is a notable confusion between 'fear' and 'sad' emotions, which could be due to similarities in facial expressions like furrowed brows or downturned mouths. The 'neutral' class has a relatively high number of misclassifications across other emotions, indicating that the model may be overfitting to more expressive features and neglecting subtler ones. The diagonal elements of the matrix represent correct predictions and are mostly the highest numbers in their respective rows, showing that the model has a decent predictive capability. However, the spread of misclassifications suggests that there is room for improvement, possibly through more complex models, data augmentation, or improved feature extraction.

VGGNet Model:

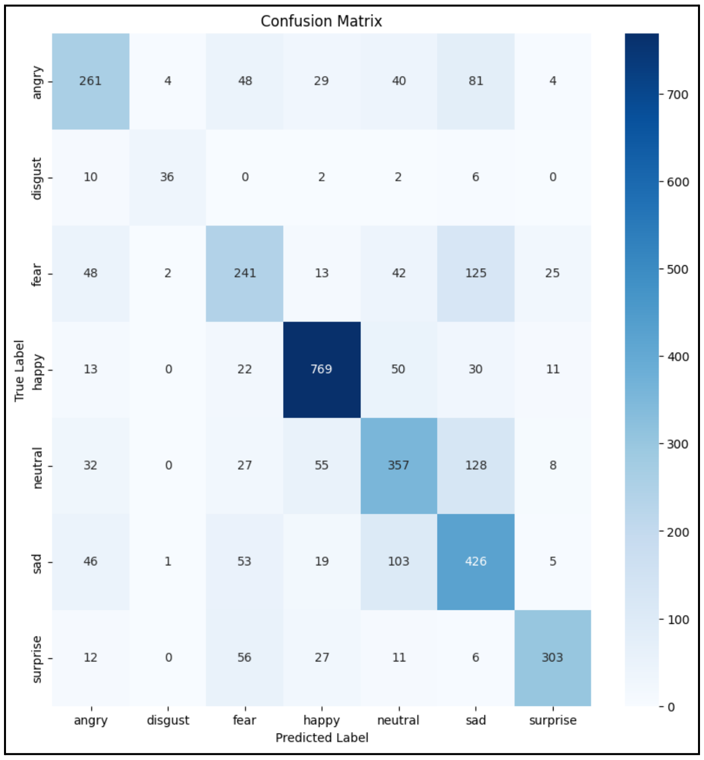


Figure 3: Confusion matrix for modified VGGnet Model for all 7 classes (emotions)

The number of correct predictions for 'happy' expressions has increased to 769 from 683, showing that the VGG model captures the features related to happiness more accurately.
This model also has fewer misclassifications, the confusion between 'angry' and 'surprise' has decreased (from 77 to 40 instances misclassified as 'surprise'), suggesting better feature distinction by the VGG model. Misclassifications for 'neutral' have also decreased, especially when compared to 'happy' and 'sad' categories, indicating improved feature learning for subtler expressions. The VGG model generally shows an increase in diagonal elements (correct predictions) across all emotions when compared to the simple CNN model. This implies better performance and learning of discriminative features for each emotion category.

ResNet50 Model:

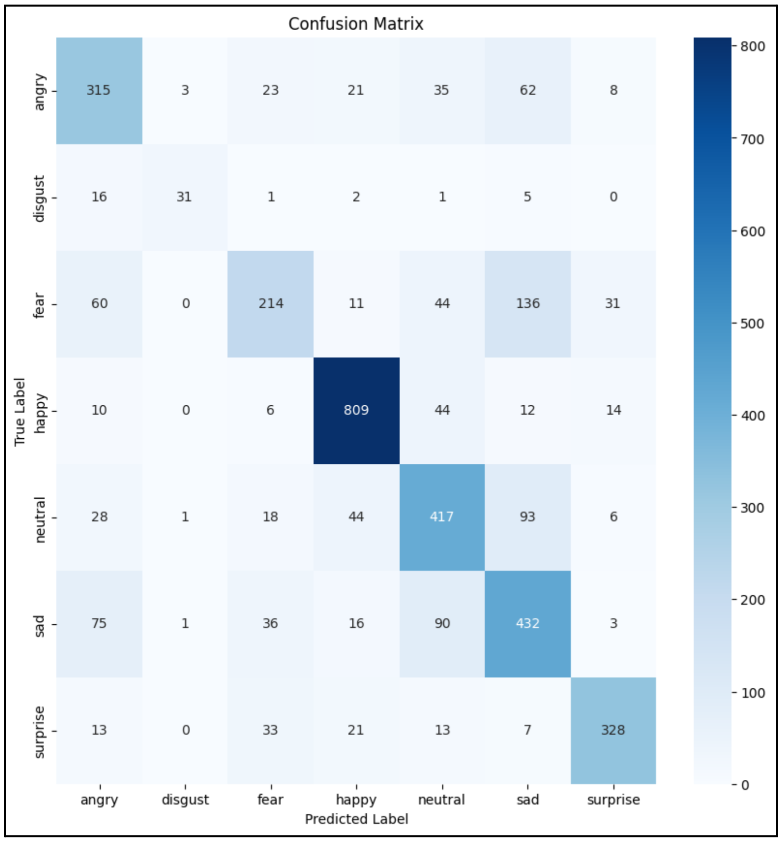


Figure 4: Confusion matrix for pretrained ResNet50 Model for all 7 classes (emotions)

Finally, for the ResNet50 model, there's a noticeable increase in the number of correct predictions for most emotions, especially 'happy', which went up to 809 from 769 in the VGG model and 683 in the simple CNN model. This suggests that the ResNet-50 model, with its deeper architecture and residual connections, is better at capturing the complex features that characterize different emotions. The model has fewer misclassifications between most emotion pairs compared to both the simple CNN and VGG models. For instance, the confusion between 'angry' and 'surprise' has been further reduced. The increase in diagonal elements indicates that the ResNet-50 model outperforms the previous models in correctly identifying all the emotion categories.

```
| Emotion   | Precision | Recall   | F1-score |
|----------|-----------|----------|----------|
| Angry    | 0.609284  | 0.674518 | 0.640244 |
| Disgust  | 0.861111  | 0.553571 | 0.673913 |
| Fear     | 0.646526  | 0.431452 | 0.517533 |
| Happy    | 0.875541  | 0.903911 | 0.889500 |
| Neutral  | 0.647516  | 0.686985 | 0.666667 |
| Sad      | 0.578313  | 0.661562 | 0.617143 |
| Surprise | 0.841026  | 0.790361 | 0.814907 |
|----------|-----------|----------|----------|
| Accuracy |           |          | 0.709390 |
| Macro avg| 0.722760  | 0.671766 | 0.688558 |
| Weighted avg | 0.712385 | 0.709390 | 0.706431 |
```

The model demonstrates the highest precision with 'Disgust' (0.861111) and the best recall for 'Happy' (0.903911), indicating a strong ability to correctly identify these emotions without many false negatives. The overall accuracy of the model stands at 0.709390, showing a good level of correctness across all emotion predictions, with 'Happy' having the highest F1-score (0.889500), suggesting a balanced precision-recall performance for this category.

Visualizations

Saliency Map

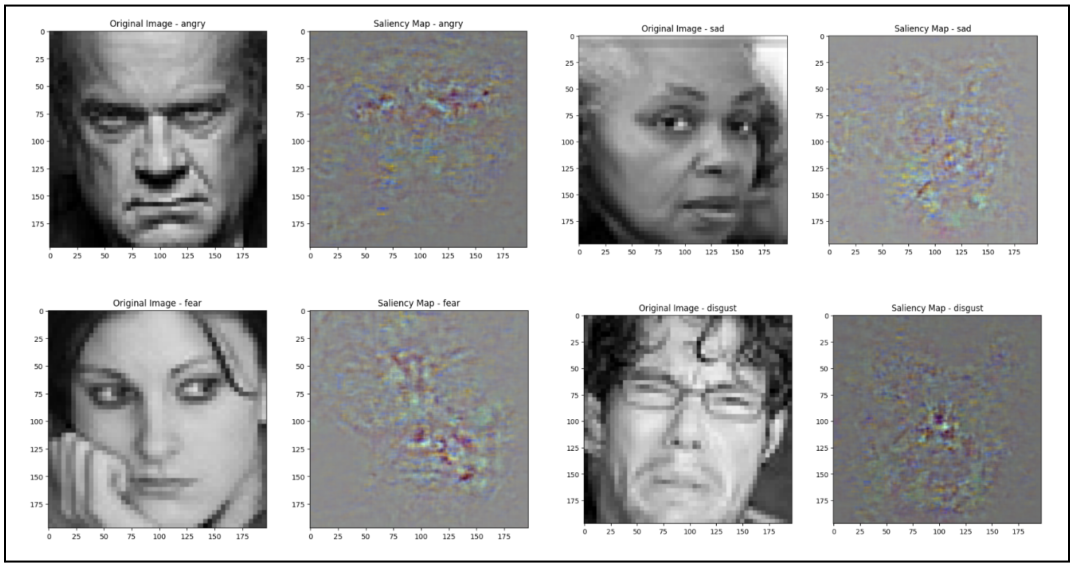


Figure 5: Saliency Maps for 4 of the 7 classes, indicating the interpretability of the model

After training our models and evaluating them, it is very important to try and interpret our model and visualize how the model is able to classify the images based on emotions. One commonly used visualization technique in deep neural networks is known as a saliency map. This method involves backpropagating the loss to the pixel values of an image, which results in a saliency map that highlights the pixels with the greatest impact on the loss value. Overall, it helps reveal the visual features that the convolutional neural network has extracted from the input image. This process helps us gain a better understanding of the significance of each feature within the original image in relation to the final classification decision. For example, we can see that for the 'Angry' emotion, the model primarily looks at the

From the saliency maps, we can observe that for each emotion the neural network is able to pick up on features around the face in order to detect the emotion.

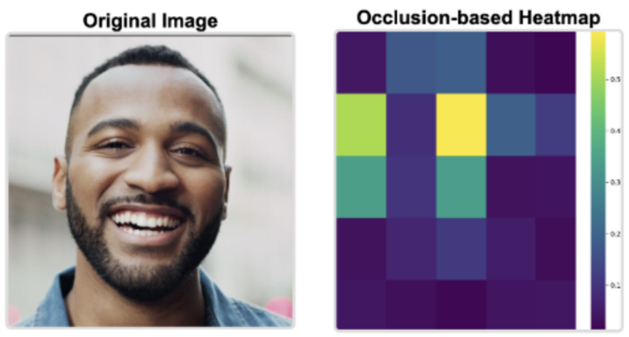


Figure 6: Occlusion based heatmap on Baseline CNN

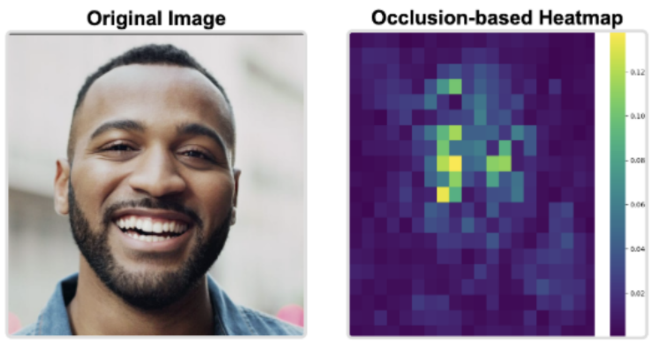
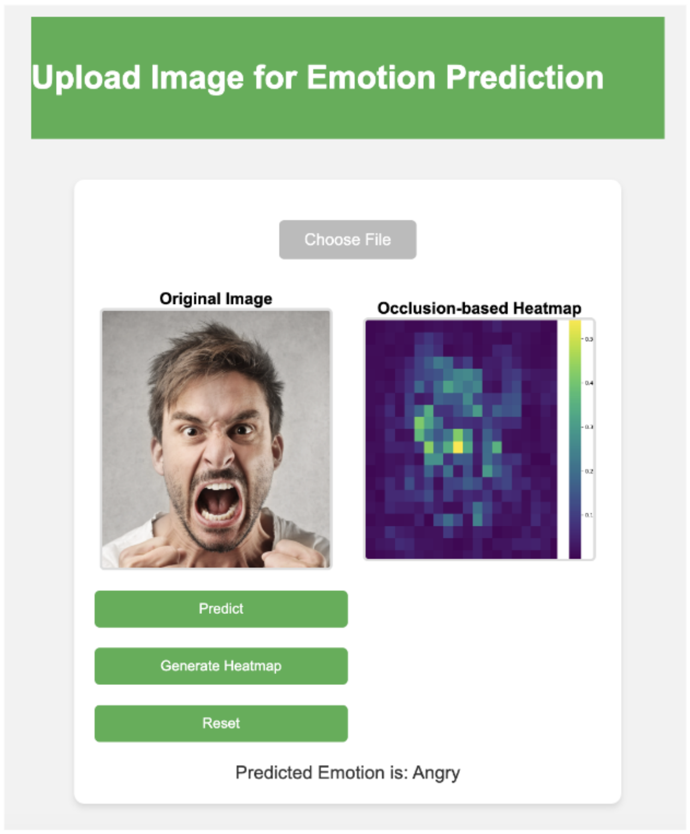


Figure 7: Occlusion based heatmap on Resnet50 Model

I also created occlusion based heatmaps as a way to understand how our model is working in order to detect emotion based on images. I did a test using images that I found online instead of using the test set. I tried to classify emotions using both the baseline CNN model and the best performing ResNet50 pretrained model in order to see the difference in the complexity and performance of the two models. Observing the occlusion-based heatmaps from both models, it's noticeable that the heatmap from the ResNet50 model appears more focused around central facial features, which likely indicates that ResNet50 is more precise in identifying the areas of the face that are most indicative of emotion. In contrast, the baseline CNN's heatmap is more dispersed, suggesting a less targeted approach to feature recognition in emotion detection. It also picks up on less detail and nuance as compared to the ResNet50 model.

Webcam Detection and Flask API App

After successfully fine tuning the ResNet50 model to classify images from the FER-2013 dataset, we achieved a test accuracy of 70.9%. This is the best performing model and for this project I decided to save this model and implement it in an app to showcase the models ability to classify emotions. In order to do this I utilized Flask API to create a web app to upload any image and have a functionality to predict the emotion in the image and also generate the occlusion based heatmap for that image to gain more insight visually on how the model is working. This implementation was very successful and I learned a lot about deploying machine learning models into an application, rather than letting the models sit in the notebook.



Predicted Emotion is: Angry

Figure 8: Emotion Detection Flask Web App using ResNet50 Pretrained Model

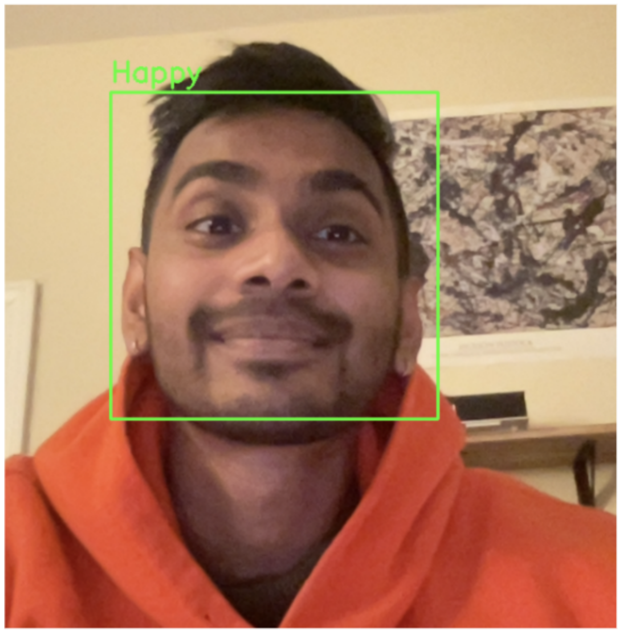


Figure 9: Real Time Live Webcam Emotion Detection using ResNet50 Pretrained Model

Limitations and Further Work

There were several limitations that I faced during this project, firstly I was limited by GPU computing power and was limited to a single T4 GPU on google colab. In order to fine tune the pretrained ResNet50 model, I had to upgrade to Colab pro to get access to a better GPU. I used the V100 GPU with high RAM to train the ResNet50 model which took around 4 hours. This took up all of my computing credits and I was unable to retrain my model. Due to limited GPU power, I also had to reduce the number of epochs to 100. In the papers that I have read, the authors mentioned that they conducted their experiments over 300 epochs. This would take too much time on my system and I do not have access to GPUs on AWS services under the educational account. Another limitation that I faced was that the dataset was not balanced, the lack of data for some classes like 'disgust' and 'fear' made it quite difficult for the model to predict those emotions, unless the input image is very close to what the model has understood. Incorporating more data for these classes would greatly increase the accuracy for detecting said emotions.

I look forward to building on this code to try more pretrained models like SetNet50 and also facial landmark detection and alignment, attention layers in CNNs and ensemble methods where we combine many models to increase the accuracy and develop a very high performing model for emotion recognition, which can then be implemented healthcare and educational applications.

References

LiB. et al. (2021). "Facial expression recognition via ResNet-50." International Journal of Cognitive Computing in Engineering.

Bangar, Siddhesh. (2022, June 28). "VGG-Net Architecture Explained." Visual Geometry Group by Oxford University

Park, Sieun. (2021, June 20). "A 2021 Guide to improving CNNs - Optimizers: Adam vs SGD." Published in Geek Culture.

Pramerdorfer, Christopher, & Kampel, Martin. (2016, December 9). "Facial Expression Recognition using Convolutional Neural Networks: State of the Art." Computer Vision Lab, TU Wien.

Vemou, Konstantina, & Horvath, Anna. (2021). "Facial Emotion Recognition." Issue 1. TechDispatch.

O. Khajuria, R. Kumar and M. Gupta, "Facial Emotion Recognition using CNN and VGG-16," 2023 International Conference on Inventive Computation Technologies (ICICT), Lalitpur, Nepal, 2023, pp. 472-477, doi: 10.1109/ICICT57646.2023.10133972.

B.-K. Kim, S.-Y. Dong, J. Roh, G. Kim, and S.-Y. Lee, "Fusing Aligned and Non-Aligned Face Information for Automatic Affect Recognition in the Wild: A Deep Learning Approach," in IEEE Conf. Computer Vision and Pattern Recognition (CVPR) Workshops, 2016, pp. 48–57.

Brechet P., Chen Z., Jakob N., Wagner S., "Transfer Learning for Facial Expression Classification". https://github.com/EmCity/transfer-learning-fer2013

Turcian D, Stoicu-Tivadar V. Real-Time Detection of Emotions Based on Facial Expression for Mental Health. Stud Health Technol Inform. 2023 Oct 20;309:272-276. doi: 10.3233/SHTI230795. PMID: 37869856.