YALOVA UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

ARTIFICIAL INTELLIGENT

EmotiMind

190101072 Yusuf Sami KAYGUSUZ

180101060 Muhammet Emir BULUT

190101084 Canan UZMA

190101048 Hüseyin Mete CALISKAN

190101080 Emir KOSEKUL

190101037 Berkay KUCUK

180101059 Medine İleyda ERDOGAN

190101049 Ali Haydar TURGUT

YALOVA, 2023

CONTENTS

1. WHAT IS THE EMOTION RECOGNITION? 3

2. AlexNet 5

3. PROJECT 7

3.1. Our Dataset 7

3.2. Why AlexNet? 7

3.3. Analysis of the AlexNet-based Emotion Detection Model 8

3.3.1. Model Architecture: 8

3.3.2. Data Preparation: 8

3.3.3. Training Setup: 8

3.3.4. Training Loop: 8

3.3.5. Model Saving: 9

3.3.6. ONNX Model Export: 9

3.4. Code Analysis 10

3.4.1. Import Statements: 10

3.4.2. Model Definition 10

3.4.3. Mounting Google Drive 11

3.4.4. Data Loading and Preprocessing 12

3.4.5. Model, Loss, Optimizer Setup 12

3.4.6. Training Loop and Model Saving 14

3.4.7. ONNX Export 15

3.4.8. OM Export in Server 16

REFERENCES 18

1. WHAT IS THE EMOTION RECOGNITION?

Emotion recognition is an interdisciplinary field that delves into the multifaceted process of discerning and categorizing human emotional states. This discipline primarily relies on a confluence of methodologies drawn from computer vision, signal processing, machine learning, and psychology. At its core, the objective of emotion recognition systems is to meticulously analyze and decipher an array of behavioral and physiological cues, such as facial expressions, vocal intonations, gestures, and even physiological parameters like heart rate variability and skin conductance, to infer an individual's emotional disposition.

Human action recognition has a wide range of applications, such as intelligent video surveillance and environmental home monitoring [1][2], video storage and retrieval [3][4], intelligent human–machine interfaces [5][6], and identity recognition [7]. Human action recognition covers many research topics in computer vision, including human detection in video, human pose estimation, human tracking, and analysis and understanding of time series data. It is also a challenging problem in the field of computer vision and machine learning. At present, there are many key problems in human action recognition that remain unsolved.

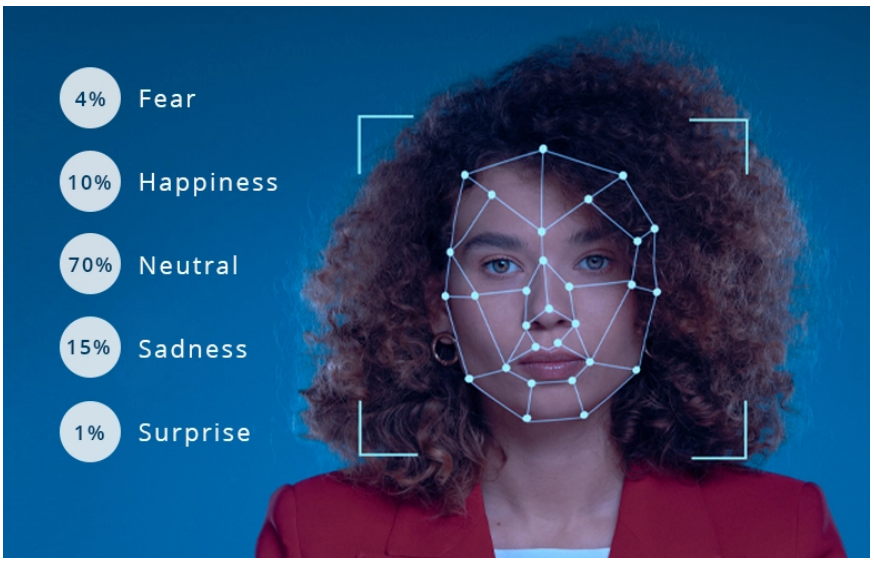


Image 1: Emotions Analysis

The key to good human action recognition is robust human action modeling and feature representation. Feature representation and selection is a classic problem in computer vision and machine learning [8]. Unlike feature representation in an image space, the feature representation of human action in video not only describes the appearance of the human(s) in the image space, but must also extract changes in appearance and pose. The problem of feature representation is extended from two-dimensional space to three-dimensional space-time. In recent years, many

kinds of action representation methods have been proposed, including local and global features based on temporal and spatial changes [9][10][11], trajectory features based on key point tracking [12][13], motion changes based on depth information [14,15,16], and action features based on human pose changes [17,18].

Facial expression analysis remains paramount in this domain, leveraging sophisticated algorithms to deconstruct and interpret intricate nuances in facial movements and configurations. Such algorithms are trained to recognize a spectrum of emotions, ranging from overt manifestations like joy and anger to more subtle expressions of contemplation or uncertainty. Parallelly, voice modulation and prosodic features in speech serve as invaluable markers, offering insights into emotional states encapsulated within linguistic content. Moreover, the holistic integration of body language and gestural cues further augments the robustness of emotion detection paradigms. For instance, postural shifts, hand movements, and even micro-expressions—transient facial movements lasting mere milliseconds—contribute pivotal data points for discerning emotional undertones.

In terms of applications, the ramifications of emotion recognition technology are profound and multifarious. In human-computer interaction paradigms, systems imbued with emotion recognition capabilities can dynamically adapt user interfaces, optimizing user experience based on real-time emotional feedback. Within the ambit of healthcare, these technologies present groundbreaking opportunities for remote monitoring of mental health, facilitating early interventions in conditions like depression, anxiety, and stress disorders. Additionally, in market research, the nuanced insights garnered from consumer emotional responses to products or advertising stimuli offer unparalleled granularity, enabling stakeholders to refine marketing strategies with heightened precision. However, it is incumbent upon the academic and practitioner community to tread with circumspection. The proliferation of emotion recognition technology engenders intricate ethical quandaries, encompassing issues of data privacy, potential biases in algorithmic interpretations, and the risk of misappropriation in sensitive contexts. As such, while the technological strides in this domain are undeniably transformative, a judicious, ethically grounded approach is imperative to navigate its deployment responsibly and equitably.

2. AlexNet

AlexNet represents a seminal convolutional neural network (CNN) architecture that played a pivotal role in catalyzing the resurgence of deep learning methodologies, particularly in the domain of image classification tasks. Introduced by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012, this architecture not only outperformed existing benchmarks on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) but also significantly surpassed the performance of traditional machine learning methods, marking a paradigmatic shift in the computational approaches to visual recognition tasks. At its foundational level, AlexNet is characterized by its deep architecture, comprising eight layers of learned parameters, including five convolutional layers and three fully connected layers. The convolutional layers are adept at automatically extracting hierarchical features from raw pixel data, progressively capturing more abstract and intricate patterns in deeper layers. This stratified feature extraction mechanism is augmented by the incorporation of rectified linear units (ReLU) as activation functions, facilitating faster convergence during training and mitigating the vanishing gradient problem that plagued earlier deep networks.

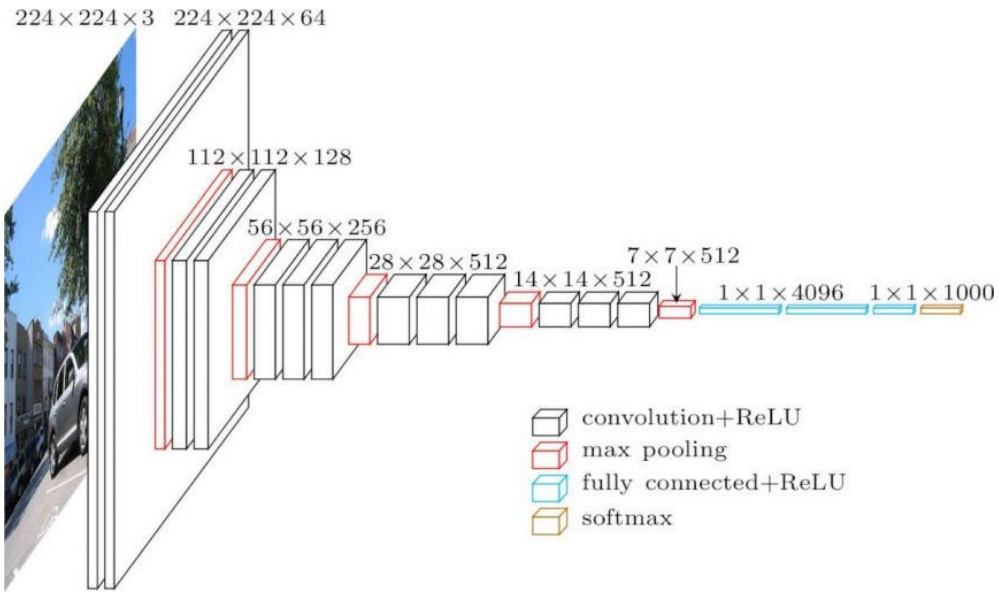


Image 2: Image Proccess

To learn about thousands of objects from millions of images, we need a model with a large learning capacity. However, the immense complexity of the object recognition task means that this prob- lem cannot be specified even by a dataset as large as ImageNet, [21] so our model should also have lots of prior knowledge to compensate for all the data we don't have. Convolutional neural networks (CNNs) constitute one such class of models [19][20]. Their

capacity can be con- trolled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

Furthermore, AlexNet introduced several innovative design elements that have since become canonical in deep learning architectures. Notably, it incorporated the utilization of overlapping pooling layers and data augmentation techniques, such as random cropping and horizontal flipping, which not only enhanced the robustness of the model against overfitting but also substantially augmented its generalization capabilities. In the broader context of the machine learning landscape, the advent of AlexNet served as a catalyst for the proliferation of increasingly sophisticated deep learning architectures, fostering a renaissance in the exploration and application of neural network methodologies across diverse domains. Its groundbreaking performance benchmarks underscored the transformative potential of deep learning, galvanizing academic and industrial research endeavors, and precipitating a cascade of innovations in model architectures, optimization algorithms, and application domains.

However, it is essential to contextualize the achievements of AlexNet within the broader trajectory of machine learning evolution. While its performance on specific tasks was groundbreaking, subsequent advancements in model architectures, optimization techniques, and regularization strategies have iteratively refined and augmented the capabilities of deep learning models, culminating in contemporary architectures like ResNet, Transformer models, and GPT variants.

In summation, AlexNet stands as a monumental milestone in the annals of machine learning history, delineating a pivotal juncture where deep learning methodologies transcended erstwhile limitations, heralding a new era of computational paradigms characterized by unprecedented capabilities in pattern recognition, data representation, and automated decision-making.

3. PROJECT

3.1. Our Dataset

In this notebook, we are trying to create an algorithm to classify a wide range (7, to be precise) of emotions based on facial expressions. For image classification, we have used the AlexNet DCNN.

3.2. Why AlexNet?

AlexNet was the winner of the 2012 ImageNet challenge. AlexNet had a remarkable intervention of using relu activation function to increase the efficiency by over 6 times by reducing the chances of Vanishing Gradient (VG) problems. Another advantage AlexNet has is that overlappping maxpooling layers considerably improve model top-1 and top-5 accuracies. The model consists of a total of 8 layers: five layers with a combination of max pooling followed by 3 fully connected layers. AlexNet was revolutionary in its field because it was the first model of its kind to introduce consecutive convolution layers in its architecture.

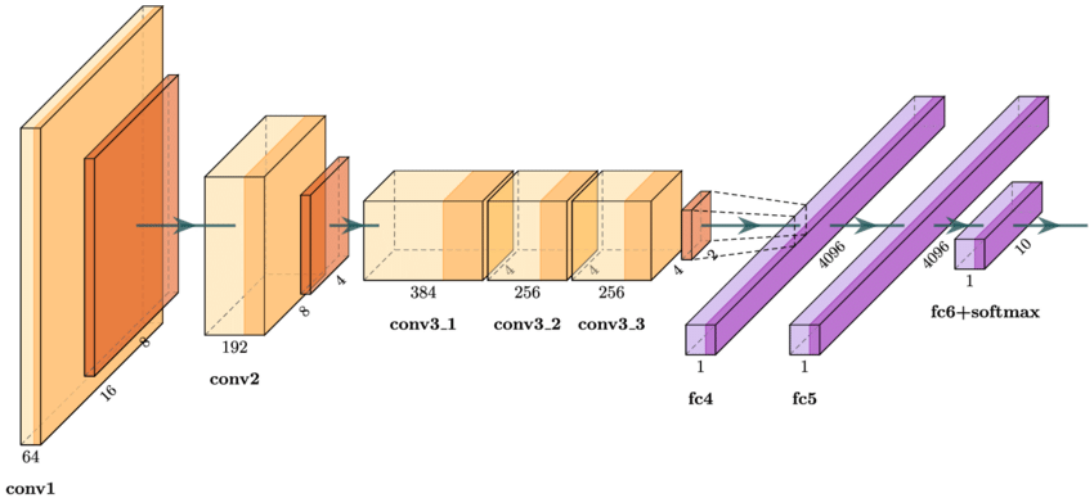


Image 3: AlexNet Model

3.3. Analysis of the AlexNet-based Emotion Detection Model

Datasets used and utilized:

https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer/code

https://www.kaggle.com/code/sanya9/emotion-detection-using-alexnet

The presented code demonstrates the construction and training of a convolutional neural network (CNN) for emotion classification. This analysis provides a breakdown of the code's structure, its functionalities, and potential improvements.

3.3.1. Model Architecture:

The implemented model, named EmotionClassifier, is a deep CNN. Its architecture comprises multiple convolutional layers interspersed with activation functions, pooling layers, and fully connected layers. The final layer outputs seven units, presumably representing seven distinct emotions.

3.3.2. Data Preparation:

- Data Directory: The code accesses image data from Google Drive directories, specifically from paths that point to training and test sets.
- Data Loading: Using the ImageFolder class from PyTorch's datasets module, the images are loaded with specified transformations. These transformations standardize the image sizes, convert them to tensors, and normalize their pixel values.

3.3.3. Training Setup:

- Device Configuration: The code checks for GPU availability and assigns the model and data to either the GPU or CPU accordingly.
- Loss and Optimizer: A cross-entropy loss function and the Adam optimizer are employed for training the model.

3.3.4. Training Loop:

The training process spans multiple epochs, with each epoch iterating through batches of the dataset. A timer function captures and prints the elapsed time for each batch and epoch, aiding in monitoring training progression.

3.3.5. Model Saving:

Once training completes, the model's state dictionary is saved to Google Drive for future use.

3.3.6. ONNX Model Export:

The code also demonstrates exporting the model to the ONNX format, a standard for representing deep learning models. This facilitates model deployment across various platforms and frameworks.

Potential Improvements and Considerations:

- Evaluation Metrics: The code does not include evaluation metrics, such as accuracy, precision, or recall, which are crucial for assessing the model's performance.
- Regularization Techniques: While dropout layers are incorporated in the model, other regularization techniques or data augmentation methods could further enhance generalization.
- Hyperparameters: Parameters such as learning rate, weight decay, or batch normalization momentum, which significantly influence training dynamics, are not explicitly defined or tuned in the code.

Conclusion:

The provided code offers a foundational framework for emotion classification using deep learning. By incorporating additional evaluation metrics and optimization techniques, the model's robustness and performance could be further elevated.

3.4. Code Analysis

3.4.1. Import Statements:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms, datasets
from torch.utils.data import DataLoader
import torch.nn.functional as F
import numpy as np
import pathlib
import matplotlib.pyplot as plt
```

The preamble of the code commences with essential library and module imports. Specifically, foundational packages for PyTorch, auxiliary utilities for data processing, visualization tools, and the torchvision module for facilitating dataset handling and transformations are incorporated.

3.4.2. Model Definition

Within the architectural blueprint, the EmotionClassifier neural network is delineated. This model embodies a hierarchically structured sequence, encompassing convolutional strata interspersed with activation mechanisms and culminating in densely connected layers, thereby facilitating intricate feature extraction and classification.

```
# Consistent input shape handling

#Input_Shape
current_input_shape = (227, 227, 3)
new_input_shape = current_input_shape  # No need to modify for this
model
```

```
# Model definition
class EmotionClassifier(nn.Module):
    def __init__(self):
        super(EmotionClassifier, self).__init__()
        self.layers = nn.Sequential(
            #1
            nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.BatchNorm2d(96),
```

```
            #2
            nn.Conv2d(96, 256, kernel_size=5, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.BatchNorm2d(256),
            #3
            nn.Conv2d(256, 384, kernel_size=3, padding=1),
            nn.ReLU(),
            #4
            nn.Conv2d(384, 384, kernel_size=3, padding=1),
            nn.ReLU(),
            #5
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            #Flatten
            nn.Flatten(),
            #Flatten 1
            nn.Linear(6 * 6 * 256, 4096),
            nn.ReLU(),
            nn.Dropout(0.5),
            #Flatten 2
            nn.Linear(4096, 4096),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(4096, 7)  # Adjust output units based on dataset
        )
        #Sigmoid
    def forward(self, x):
        x = self.layers(x)
        return torch.sigmoid(x)
```

3.4.3. Mounting Google Drive

To bridge the Colab computational environment with external data repositories, a mounting procedure for Google Drive is enacted. This operation is pivotal for seamless data access and interactivity within the Colab framework.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

3.4.4. Data Loading and Preprocessing

The ensuing segment orchestrates the data ingestion process. Predetermined directory paths guide the acquisition of training and validation datasets. Concurrently, preprocessing directives, encompassing image resizing and statistical normalization, are stipulated to standardize the data distribution and enhance model convergence.

```
#Class_name
data_dir = pathlib.Path("/Model_Face/train")
image_count = len(list(data_dir.glob('*/*.png')))
print(image_count)
# classnames in the dataset specified
CLASS_NAMES = np.array([item.name for item in data_dir.glob('*') if
item.name != "LICENSE.txt" ])
print(CLASS_NAMES)
# print length of class names
output_class_units = len(CLASS_NAMES)
print(output_class_units)
```

```
28709
['disgusted' 'fearful' 'angry' 'happy' 'surprised' 'sad' 'neutral']
7
```

```
# Data loading
data_dir =
pathlib.Path("/content/gdrive/MyDrive/Proje/Model_Face/train")
data_dir2 =
pathlib.Path("/content/gdrive/MyDrive/Proje/Model_Face/test")
```

```
#Data_ImageSet
train_dataset = datasets.ImageFolder(data_dir, transform=transform)
test_dataset = datasets.ImageFolder(data_dir2, transform=transform)
```

```
#Data_Loader
BATCH_SIZE = 32
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=True)
```

3.4.5. Model, Loss, Optimizer Setup

The subsequent phase orchestrates the foundational elements requisite for model training. A device specification mechanism discerns the computational substrate—be it GPU or CPU—on which the model will be instantiated. Furthermore, an Adam optimizer, renowned for its

adaptive learning rate properties, is invoked, accompanied by a categorical cross-entropy criterion tailored for multi-class classification tasks.

```
# Model, loss, optimizer
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#Model Create
model = EmotionClassifier().to(device)
criterion = nn.CrossEntropyLoss()
#Optimizer
optimizer = optim.Adam(model.parameters())
```

In this section, the device (either GPU or CPU) where the model will run is determined. Additionally, the loss function and optimization algorithm are defined.

3.4.5.1. Summary

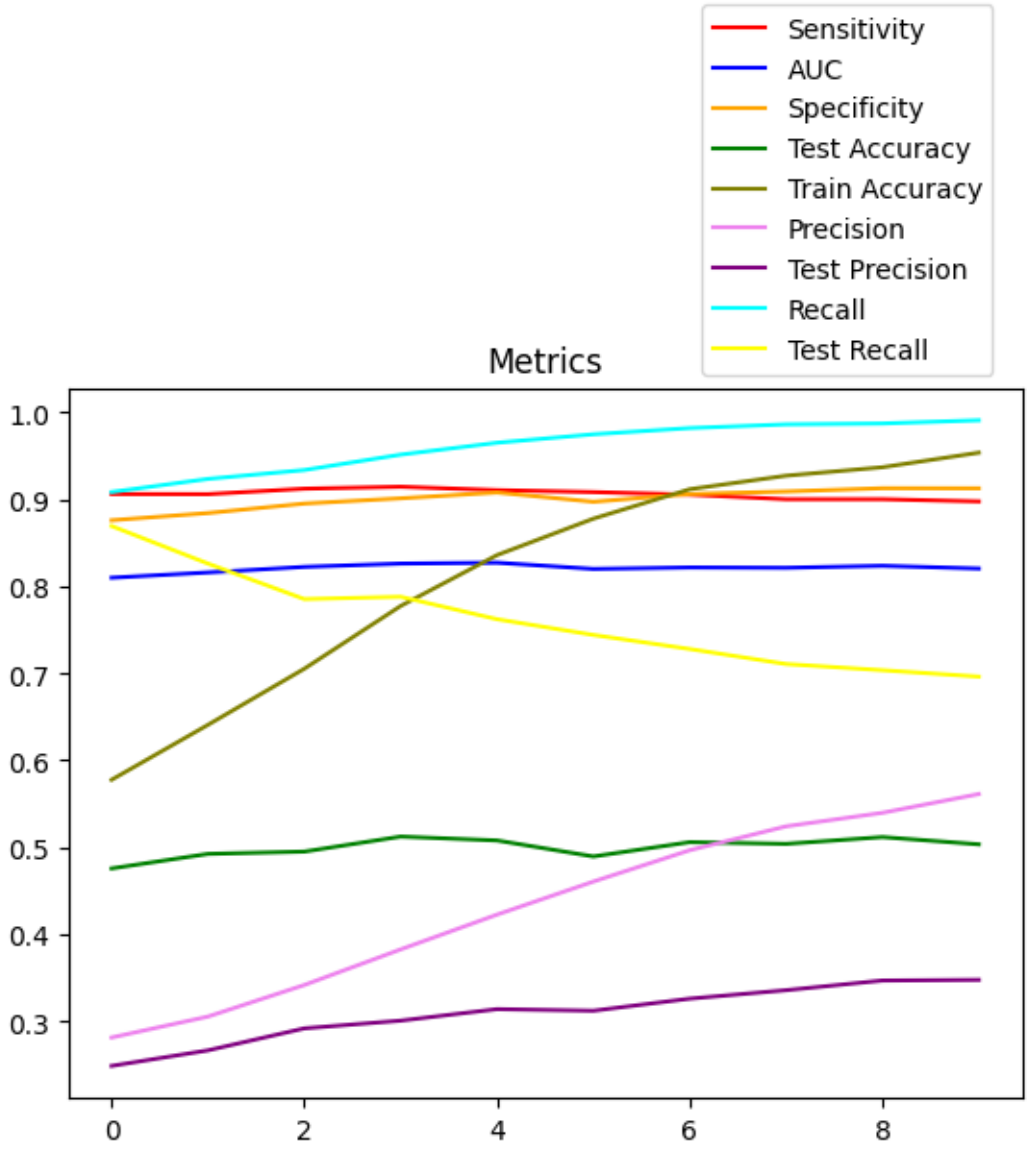
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 55, 55, 96)        34944

 batch_normalization (Batch  (None, 55, 55, 96)        384
 Normalization)

 max_pooling2d (MaxPooling2  (None, 27, 27, 96)        0
 D)

 conv2d_1 (Conv2D)           (None, 27, 27, 256)       2973952

 batch_normalization_1 (Bat  (None, 27, 27, 256)       1024
 chNormalization)

 conv2d_2 (Conv2D)           (None, 27, 27, 384)       885120

 batch_normalization_2 (Bat  (None, 27, 27, 384)       1536
 chNormalization)

 conv2d_3 (Conv2D)           (None, 27, 27, 384)       1327488

 batch_normalization_3 (Bat  (None, 27, 27, 384)       1536
 chNormalization)

 conv2d_4 (Conv2D)           (None, 27, 27, 256)       884992

 batch_normalization_4 (Bat  (None, 27, 27, 256)       1024
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 256)       0
 g2D)

 flatten (Flatten)           (None, 43264)             0

 dense (Dense)               (None, 4096)              177213440

 dense_1 (Dense)             (None, 4096)              16781312

 dense_2 (Dense)             (None, 7)                 28679
```

3.4.6. Training Loop and Model Saving

The iterative training regimen is delineated, encapsulating epochs and batch iterations. Within this loop, the model is subjected to incremental refinements, iteratively assimilating dataset nuances. Concurrent temporal metrics facilitate the quantification of epochal training durations, offering insights into computational efficiency and convergence trajectories.

```
import time

def print_elapsed_time(start_time, end_time, prefix=""):
    elapsed_time = end_time - start_time
    print(f"{prefix} Elapsed Time: {elapsed_time:.2f} seconds")

# Training loop
num_epochs = 10
print_interval = 100
for epoch in range(num_epochs):
    start_time_epoch = time.time()

    for i, (images, labels) in enumerate(train_loader):
        batch_start_time = time.time()
        batch_end_time = time.time()
        if (i + 1) % print_interval == 0:  #save time each
print_interval
            print_elapsed_time(batch_start_time, batch_end_time,
prefix=f"Epoch [{epoch+1}/{num_epochs}], Batch
[{i+1}/{len(train_loader)}]")

    end_time_epoch = time.time()  #save time end epoch
    print_elapsed_time(start_time_epoch, end_time_epoch, prefix=f"Epoch
[{epoch+1}/{num_epochs}]")
```

```
#Model Save
torch.save(model.state_dict(), 'model10t2.pth')
print("saved model.")
```



3.4.7. ONNX Export

In culmination, preservation protocols are instantiated to safeguard the model's structural integrity post-training. The model's weight parameters are serialized, thereby facilitating reproducibility and future deployments. Additionally, an ONNX exportation process is initialized, enabling model interoperability across diverse computational frameworks and environments.

```
import torch
from torch.onnx import export
model2 = EmotionClassifier()
```

```
#Model_load
model.load_state_dict(torch.load('model10t.pth'))
#Input_shape
dummy_input=torch.randn(1,3,227,227)
#Opset_version = 11 (Domain 1)
export(model2, dummy_input, "model10t.onnx",
opset_version=11,input_names=['input'],output_names=['output'])
print("Model saved successfully.")
```

Through this structured analysis, the code's constituent components and their interdependencies are elucidated, underscoring the orchestrated synthesis of foundational, computational, and data-driven elements inherent to the machine learning pipeline.

3.4.8. OM Export in Server

It aims to convert a model from the Open Model Zoo library in ONNX format to an optimized format suitable for Ascend310 hardware. During the initiation of the model conversion process using the `atc` command, various parameters are specified, including the model file, output name, input format, input shape, and target hardware (Ascend310).

```
# .om export in server(convertT10.sh)

atc --model=modelt.onnx \ .onnx name
    --framework=5\        framework number (.onnx=5)
    --output=mymodel10 \  .on output name
    --input_format=NCHW \ input_format
    --input_shape "input:1,3,227,227" \ dummy_input
    --soc_version=Ascend310 \ server hardware
```

Table 1: Step-Description Table

Step	Description
1	**Importing Libraries**
	- **matplotlib.pyplot** for graphical representations
	- **torch** the open-source deep learning library
	- **cv2** (OpenCV) for image processing
	- **numpy** (np) for mathematical operations
	- **pathlib** for handling file paths

	- **datetime** for time-related operations
2	**Defining Data Directories & Counts**
	- **data_dir** as the path to the training dataset
	- Calculate the total number of images in the directory
	- Retrieve folder names (e.g., emotions) and assign them to **CLASS_NAMES**
	- Determine the total number of output units (classes)
3	**Model Construction**
	- Define a CNN model inspired by AlexNet
	- Model includes convolutional layers, pooling, and fully connected layers
4	**Preparing Data Generators**
	- Use **image_generator** for preprocessing images
	- Create data flows (**train_data_gen** & **val_data_gen**) for training and validation
5	**Compiling the Model**
	- Compile the model using Adam optimizer and categorical cross-entropy loss
	- Add various metrics to evaluate the model's performance
6	**Model Training**
	- Train the model for a specified number of epochs
	- Calculate and save the training duration
7	**Visualizing Performance**
	- Plot various metrics to assess model performance

REFERENCES

[1] Aggarwal, J.K.; Ryoo, M.S. Human activity analysis: A review. ACM Comput. Surv. 2011, 43.

[2] Ziaeefard, M.; Bergevin, R. Semantic human activity recognition: A literature review. Pattern Recognit. 2015, 48, 2329–2345.

[3] Van Gemert, J.C.; Jain, M.; Gati, E.; Snoek, C.G. APT: Action localization proposals from dense trajectories. In Proceedings of the British Machine Vision Conference 2015: BMVC 2015, Swansea, UK, 7–10 September 2015; p. 4.

[4] Zhu, H.; Vial, R.; Lu, S. Tornado: A spatio-temporal convolutional regression network for video action proposal. In Proceedings of the CVPR, Venice, Italy, 22–29 October 2017; pp. 5813–5821.

[5] Papadopoulos, G.T.; Axenopoulos, A.; Daras, P. Real-time skeleton-tracking-based human action recognition using kinect data. In Proceedings of the International Conference on Multimedia Modeling, Dublin, Ireland, 6–10 January 2014; pp. 473–483.

[6] Presti, L.L.; Cascia, M.L. 3D Skeleton-based Human Action Classification: A Survey. Pattern Recognit. 2016, 53, 130–147.

[7] Paul, S.N.; Singh, Y.J. Survey on Video Analysis of Human Walking Motion. Int. J. Signal Process. Image Process. Pattern Recognit. 2014, 7, 99–122.

[8] Bengio, Y.; Courville, A.; Vincent, P. Representation Learning: A Review and New Perspectives. IEEE Trans. Pattern Anal. Mach. Intell. 2013, 35, 1798–1828.

[9] Dawn, D.D.; Shaikh, S.H. A comprehensive survey of human action recognition with spatio-temporal interest point (STIP) detector. Vis. Comput. 2016, 32, 289–306. [Google Scholar] [CrossRef]

[10] Nguyen, T.V.; Song, Z.; Yan, S.C. STAP: Spatial-Temporal Attention-Aware Pooling for Action Recognition. IEEE Trans. Circ. Syst. Video Technol. 2015, 25, 77–86.

[11] Shao, L.; Zhen, X.T.; Tao, D.C.; Li, X.L. Spatio-Temporal Laplacian Pyramid Coding for Action Recognition. IEEE Trans. Cybern. 2014, 44, 817–827.

[12] Burghouts, G.J.; Schutte, K.; ten Hove, R.J.M.; van den Broek, S.P.; Baan, J.; Rajadell, O.; van Huis, J.R.; van Rest, J.; Hanckmann, P.; Bouma, H.; et al. Instantaneous threat detection

based on a semantic representation of activities, zones and trajectories. Signal Image Video Process. 2014, 8, 191–200.

[13] Wang, H.; Schmid, C. Action recognition with improved trajectories. In Proceedings of the ICCV, Sydney, NSW, Australia, 1–8 December 2013; pp. 3551–3558.

[14] Yang, X.; Tian, Y.L. Super Normal Vector for Activity Recognition Using Depth Sequences. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 804–811.

[15] Ye, M.; Zhang, Q.; Wang, L.; Zhu, J.; Yang, R.; Gall, J. A survey on human motion analysis from depth data. In Proceedings of the Dagstuhl 2012 Seminar on Time-of-Flight Imaging: Sensors, Algorithms, and Applications and Workshop on Imaging New Modalities, GCPR 2013, Saarbrucken, Germany, 21–26 October 2012; pp. 149–187.

[16] Oreifej, O.; Liu, Z. HON4D: Histogram of Oriented 4D Normals for Activity Recognition from Depth Sequences. In Proceedings of the Computer Vision and Pattern Recognition, Portland, OR, USA, 23–28 June 2013; pp. 716–723.

[17] Li, M.; Leung, H.; Shum, H.P.H. Human action recognition via skeletal and depth based feature fusion. In Proceedings of the Motion in Games 2016, Burlingame, CA, USA, 10–12 October 2016; pp. 123–132.

[18] Yang, X.; Tian, Y.L. Effective 3D action recognition using EigenJoints. J. Vis. Commun. Image Represent. 2014, 25, 2–11.

[19] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In International Conference on Computer Vision, pages 2146–2153. IEEE, 2009.

[20] A. Krizhevsky. Convolutional deep belief networks on cifar-10. Unpublished manuscript, 2010.

[21] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, et al. Hand-written digit recognition with a back-propagation network. In Advances in neural information processing systems, 1990.