**Semester 6**

**RUSCS604 :- DATA SCIENCE**

**MEMBERS :\_**

4801 Samiksha Advilkar

4803 Sahil Bhilare

4808 Sharvin Desai

4821 Subhalaxmi Mohanty

4836 Prerana Vyavhare

**TOPIC :- NEURAL NETWORK MODELS FOR CASE STUDY**

**Which models are most suitable to psychological case studies** :

* In the realm of psychological studies from images, several neural network models have shown promising results.
* The choice of model depends on various factors such as the size of the dataset, computational resources available, and specific requirements of the application.

Here are some commonly used models:

**A basic overview of Artificial Neural Networks :-**

Artificial Neural Networks (ANNs) are computational models inspired by the human brain's neural networks. They consist of interconnected nodes organized in layers. Neurons in one layer are connected to neurons in the next layer through connections with weights and biases. ANNs use activation functions to introduce nonlinearity into the network. During training, ANNs adjust the weights and biases based on the error between predicted and true outputs using a process called backpropagation. ANNs are used in various applications, including image recognition, speech recognition, natural language processing, and more.

We can more learn about the neural networks from the following website giving actual visualization of how artificial neural networks works

<https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.48539&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>

1. **Convolutional Neural Networks (CNNs):**

Convolutional Neural Networks (CNNs) are a class of deep neural networks primarily designed for analyzing visual imagery

**What ?**

CNNs are a type of artificial neural network inspired by the organization and functioning of the animal visual cortex.

They consist of multiple layers of neurons, including convolutional layers, pooling layers, and fully connected layers.

CNNs are particularly effective at capturing spatial hierarchies of features in images, making them well-suited for tasks like image classification and object recognition

**How it works ?**

Convolutional Layers:-

Convolutional layers apply a set of learnable filters (kernels) to the input image.

- Each filter slides across the input image, performing element-wise multiplication and summing the results to produce feature maps.

- This process allows the network to automatically learn important features such as edges, textures, and patterns present in the input images.

Pooling Layers:-

Pooling layers down-sample the feature maps produced by convolutional layers, reducing their spatial dimensions.

Common pooling operations include max pooling and average pooling, which help to retain important information while reducing computational complexity and controlling overfitting.

Fully Connected Layers:

Fully connected layers take the flattened feature maps from the previous layers and perform classification or regression based on learned features.

These layers enable the network to combine and interpret the learned features to make predictions.



**WHERE USED ?**

* Image Classification
* Object Detection
* Facial Recognition
* Medical Image Analysis
* Natural Language Processing (NLP)

1. **Recurrent Neural Networks (RNNs):**

Recurrent Neural Networks (RNNs) are a type of artificial neural network designed to effectively process sequence data by maintaining an internal state or memory.

**What ?**

RNNs are a class of neural networks that have connections that form directed cycles, allowing them to exhibit temporal dynamic behavior.

- Unlike feedforward neural networks, which process each input independently, RNNs maintain a memory of past inputs and use this information to influence the current output.

- RNNs have loops within their architecture, allowing information to persist over time and enabling them to capture temporal dependencies in sequential data.

**How it works ?**

Recurrent Connections:-

The defining characteristic of RNNs is their recurrent connections, which allow information to be passed from one time step to the next.

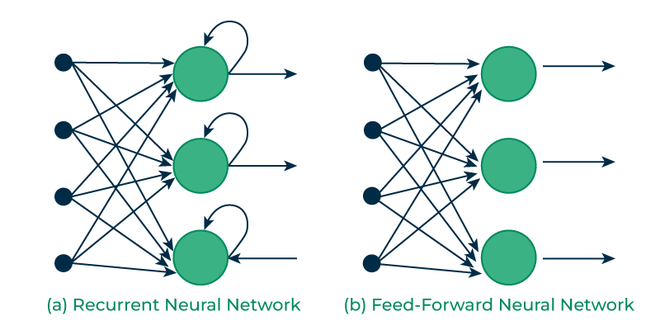
- At each time step, the RNN receives an input and its current internal state (or memory) as inputs. It combines these inputs to produce an output and update its internal state.

- The internal state serves as a memory that retains information about past inputs, allowing the network to capture context and dependencies over time.

Training:

- RNNs are typically trained using backpropagation through time (BPTT), an extension of the backpropagation algorithm.

- BPTT unrolls the network over time, treating it as a deep feedforward network with shared weights. Gradients are computed at each time step and propagated backward through time to update the network's parameters.



**Where used ?**

* Natural Language Processing (NLP)
* Speech Recognition
* Time Series Analysis
* Generative Models

Overall, RNNs are powerful tools for modeling and processing sequential data, making them essential in a wide range of applications where understanding temporal dynamics is crucial.

1. **Capsule Networks:**

Capsule Networks (CapsNets) are a type of neural network architecture proposed by Geoffrey Hinton and his colleagues in 2017 as a potential successor to Convolutional Neural Networks (CNNs). CapsNets aim to overcome some limitations of CNNs, particularly in handling hierarchical relationships among features

**What ?**

Capsule Networks are designed to better model spatial hierarchies and relationships within images, focusing on the relationships between parts and wholes.

- They use "capsules" as the basic building blocks instead of individual neurons. Each capsule represents a specific part of an object and encodes various properties of that part, such as pose, deformation, and texture.

- Capsules are organized into hierarchies, allowing higher-level capsules to represent more abstract features by aggregating information from lower-level capsules.

**How it works ?**

Primary Capsules:

- The input image is processed by a series of convolutional layers to extract low-level features.

- These features are then passed through primary capsules, which represent local patterns and features in the image.

Routing by Agreement:

- Capsule Networks use a routing mechanism called "routing by agreement" to determine the relationships between capsules in different layers.

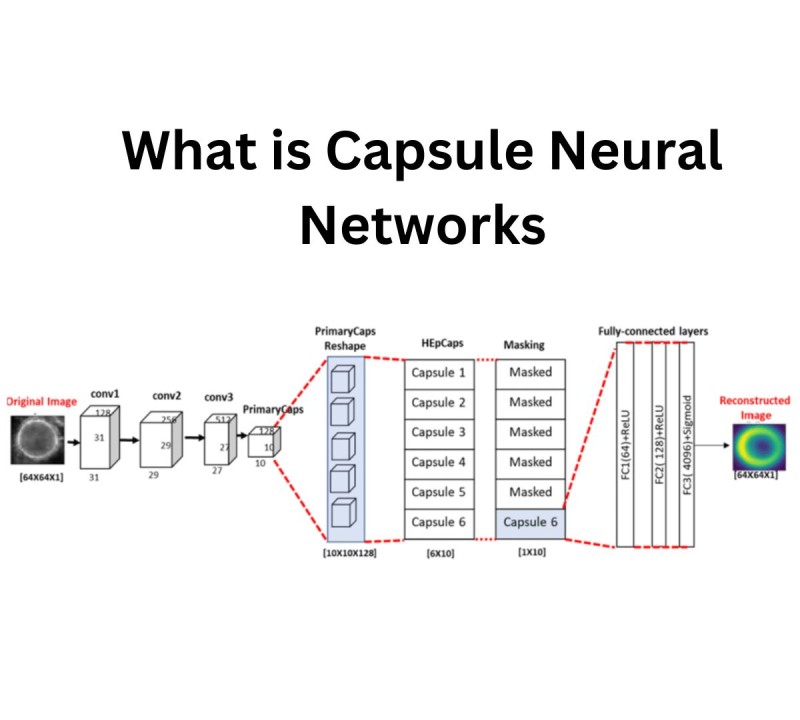
- During training, the agreement between the output of lower-level capsules and the predictions of higher-level capsules is computed.

- Capsules that agree with each other are reinforced, while those that disagree are suppressed.

Dynamic Routing:

- Dynamic routing iteratively updates the coupling coefficients between capsules based on the agreement between their predictions and actual output.

- This process allows the network to learn to assign higher weights to capsules that are in agreement and lower weights to those that are not.



**Where used ?**

* Image Recognition
* Enhanced Interpretability
* Improved Generalization
* Medical Imaging

1. **Hybrid Models:**

Hybrid models refer to neural network architectures that combine elements of different types of models, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and other architectures, to leverage their respective strengths and address specific challenges.

**What ?**

Hybrid models integrate multiple neural network architectures or components into a single unified framework.

- They combine the strengths of different types of models to enhance performance, address specific tasks, or overcome limitations associated with individual architectures.

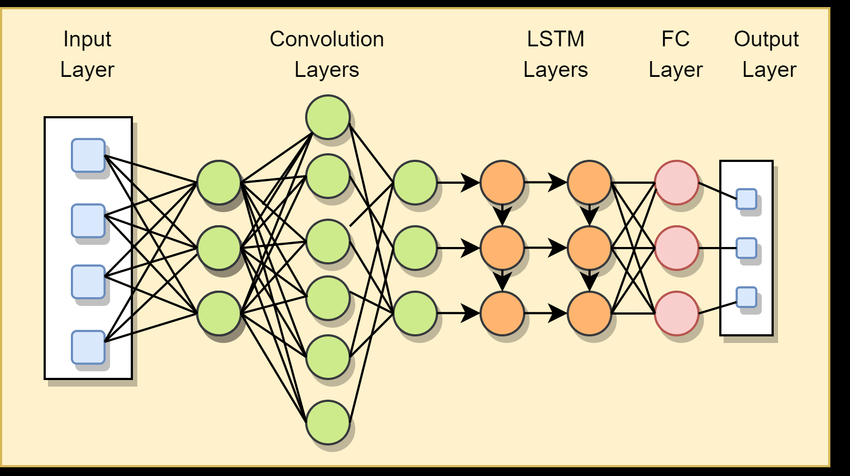
- Hybrid models can include combinations of CNNs, RNNs, fully connected layers, attention mechanisms, and other components, depending on the requirements of the task.

**How it works ?**

Hybrid models leverage the unique capabilities of each component architecture to tackle specific aspects of the problem at hand.

- For example, in a hybrid model for image captioning, a CNN may be used for image feature extraction, while an RNN generates textual descriptions based on the extracted features.

- The different components of the hybrid model are typically connected in a way that facilitates information flow and interaction between them, allowing for joint learning and optimization



**Where used ?**

* Image and Text Processing
* Video Analysis
* Speech Recognition
* Multi-modal Learning

Hybrid model provides enhanced performance dure to maximum leverage of multiple architectures , it provides increased flexibility and improved robustness .on the other hand one can may face challenges like more complex while training the models, since integration of tow models take place that affects on the time complexity of the training

But overall they give a powerful architecture

**CASE STUDY :-**

Certainly! Let's outline a case study for emotion detection using an image frame:

**Introduction :-**

An important topic of study in the fields of computer vision and artificial intelligence is the identification of emotion from images. It entails creating models and algorithms that can automatically identify and categorize human emotions based on visual signals found in pictures. In recent years, there has been a significant growth in demand for automatic emotion recognition systems due to the ubiquitous availability of digital cameras and social media platforms.

**Objective :-**

To develop a deep learning model capable of accurately detecting emotions in image

Steps :-

Data science process to carry out emotion detection implementation

Gather a diverse dataset of image frames ,here we are using ck+ dataset

Data Collection

Resize the image, rotate, flip, and zoom to increase robustness and prevent overfitting.

Monitoring & maintenance

Data Preprocessing

Continuously collect feedback from users and update the model to address any issues or improvements.

Here we choose a convolutional neural network (CNN) architecture suitable for image classification tasks.

Deployment

Model Selection

1)Convolutional layers:- 2)Pooling layers:

3)Fully connected layers: 4)Output layer:

Deploy the trained model for real-time or batch inference on new image frames.

Optimization

Model Architecture

Fine-tune like learning rate, batch size, and optimizer choice to optimize performance.

Split the dataset into training, validation, and testing sets.

Evaluate the trained model on the testing set using metrics such as accuracy, precision, recall

Evaluation

Training

Conclusion:

This case study demonstrates the development of an emotion detection system capable of accurately classifying emotions in image frames extracted from videos. By leveraging deep learning techniques and convolutional neural networks, the model achieves robust performance across diverse emotional expressions, paving the way for applications in areas like video content analysis, human-computer interaction, and affective computing.

Exploratory Data Analysis

**Source code :**

Imports :-

import cv2

import os

import numpy as np

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.optimizers import Adam

import tensorflow as tf

Data collection :-

from google.colab import drive

drive.mount('/content/drive',force\_remount=True)

dataset\_dir = "/content/drive/My Drive/images"

output :-



Data preprocessing :-

def preprocess\_images(dataset\_dir, img\_size=(48, 48)):

    images = []

    labels = []

    # Define the emotions

    emotions = ["anger", "contempt", "disgust", "fear", "happy", "sadness", "surprise"]

    # Iterate through the emotion folders

    for i, emotion in enumerate(emotions):

        # Get the path to the emotion folder

        emotion\_dir = os.path.join(dataset\_dir, emotion)

        # Check if the emotion folder exists

        if os.path.exists(emotion\_dir):

            # Iterate through the images in the emotion folder

            for image\_name in os.listdir(emotion\_dir):

                # Load and preprocess the image

                img\_path = os.path.join(emotion\_dir, image\_name)

                img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

                img = cv2.resize(img, img\_size) / 255.0

                # Append the preprocessed image and label to the lists

                images.append(img)

                labels.append(i)  # Use the index of the emotion as the label

    return np.array(images), np.array(labels)

images, labels = preprocess\_images(dataset\_dir)

print("Images shape:", images.shape)

print("Labels shape:", labels.shape)

output :-

A number on a white background

Description automatically generated

Splitting the datasets

train\_images, test\_images, train\_labels, test\_labels = train\_test\_split(images, labels, test\_size=0.2, random\_state=42)

print(train\_images.shape)

print(test\_images.shape)

A group of numbers with different expressions

Description automatically generated with medium confidence

Defining model architecture

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input\_shape=(48, 48, 1)),

    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),

    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(128, activation='relu'),

    Dense(7, activation='softmax')  # 7 output classes (one for each emotion)

])

Compiling the architecture

model.compile(optimizer=Adam(), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

Fitting the dataset into the model

history = model.fit(train\_images, train\_labels, validation\_split=0.2, epochs=20, batch\_size=12)

Output :-

A table of numbers and letters

Description automatically generated with medium confidence

A close-up of a white page

Description automatically generated

Predicting values

predictions = model.predict(test\_images)

predicted\_labels = np.argmax(predictions, axis=1)

print(predicted\_labels)

Output :-

Output :-



A number grid with black numbers

Description automatically generated with medium confidence

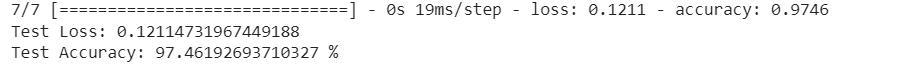
evaluating the model

test\_loss, test\_accuracy = model.evaluate(test\_images, test\_labels)

print("Test Loss:", test\_loss)

print("Test Accuracy:", test\_accuracy\*100,"%")

Output :-



Saving the model

model.save('model\_1.h5')



A number on a white background

Description automatically generated

actual implementation of the model :-

import cv2

import numpy as np

from keras.models import load\_model

# Load the pre-trained emotion detection model

model = load\_model('model\_1.h5')

# Define the emotion labels

emotion\_labels = {0: 'Angry', 1: 'Disgust', 2: 'Fear', 3: 'Happy', 4: 'Sad', 5: 'Surprise', 6: 'contempt'}

# Load the pre-trained face detection classifier

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

# Function to predict emotion from the input face image

def predict\_emotion(face\_img):

# Preprocess the face image

face\_img = cv2.resize(face\_img, (48, 48))

face\_img = cv2.cvtColor(face\_img, cv2.COLOR\_BGR2GRAY)

face\_img = np.expand\_dims(face\_img, axis=0)

face\_img = np.expand\_dims(face\_img, axis=-1)

face\_img = face\_img / 255.0

# Predict the emotion

emotions = model.predict(face\_img)

predicted\_label = np.argmax(emotions)

return emotion\_labels[predicted\_label]

# Function to detect faces in the input frame and predict emotions

def detect\_faces(frame):

# Convert frame to grayscale

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Detect faces in the frame

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Draw rectangles around the detected faces and predict emotions

for (x, y, w, h) in faces:

face\_region = frame[y:y+h, x:x+w]

emotion\_label = predict\_emotion(face\_region)

cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

cv2.putText(frame, emotion\_label, (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (0, 255, 0), 2)

return frame

# Function to start the camera and perform emotion detection

def start\_emotion\_detection():

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

if not ret:

break

frame = detect\_faces(frame)

cv2.imshow('Emotion Detection', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

# Main function

if \_\_name\_\_ == '\_\_main\_\_':

start\_emotion\_detection()

Final output :-

Disgust

A person taking a selfie

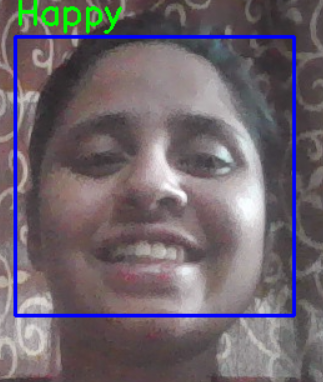
Description automatically generated

Contempt :-

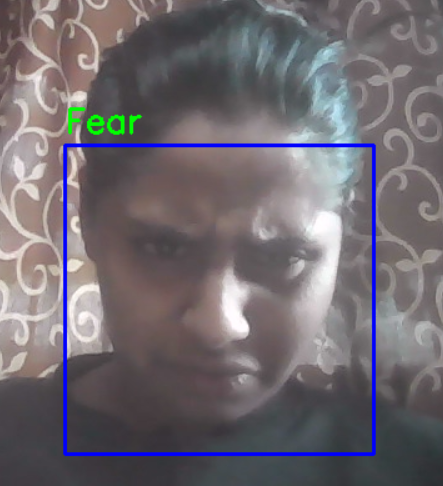
A selfie of a person

Description automatically generated

Happy :-



Fear :-



Angry :-

A person taking a selfie

Description automatically generated

Sad :-

A person making a face

Description automatically generated

Here the model Is getting trained continuously and recognizing the emotions more accurateky

