PROJECT REPORT

Adversarial Attack Detection in Deepfake models

Team members:

- S Sowmithaa Sri- 22011101096
- N V S Keerthana Lingamallu- 22011101073

Abstract & Objective:

This project addresses the critical challenge of detecting falsification in Deepfake models, which are increasingly being used to generate highly realistic but manipulated media content. With the rise of Deepfake technology comes a pressing need to develop effective methods for identifying and mitigating malicious manipulations aimed at deceiving viewers. By focusing on this topic, we aim to contribute to the broader efforts to combat misinformation, protect privacy, and uphold the integrity of digital media.

Datasets description:

a. **Celeb- Real and Celeb Synthesis**

1. Data source: The Dataset consists of a collection of real and fake videos sourced from Kaggle Data Repository owned by HARIOM H SINGH
2. Link: https://www.kaggle.com/datasets/hariomhsingh/celeb-real, https://www.kaggle.com/datasets/hariomhsingh/celebsynthesis?select=id0_id16_0005.mp4
3. Data Composition:
   - Real Videos: The dataset contains 158 items of diverse set of real videos depicting individuals engaged in various activities such as talking, singing, and gesturing. These videos serve as the baseline for genuine human behavior.
   - Fake Videos: The dataset includes 795 files of Deepfake videos generated using state-of-the-art deep learning techniques. These videos depict individuals' faces synthesized onto different bodies or scenarios, simulating realistic but falsified content.
4. Data Preprocessing:
   - Videos are preprocessed to extract frames at regular intervals, ensuring consistent input size for model training.
   - Frames are resized and normalized to a common resolution (e.g., 224x224 pixels) to facilitate model processing.

5. Sample Datapoints:

<table>
<tr><th>REAL</th><th>SYNTHETIC</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

b. Deepfake Detection Challenge Dataset:

1. Data source: The DFDC dataset is a widely recognized benchmark dataset for deepfake detection tasks. It was released as part of the DeepFake Detection Challenge organized by Kaggle. The dataset contains videos with both real and synthetic (deepfake) faces, making it suitable for training and evaluating deep learning models for deepfake detection.
2. Link: https://www.kaggle.com/datasets/phunghieu/deepfake-detection-faces-part-15-0
3. Data Composition:
   Deepfake Detection - Faces - Part 15
   Version: Version 1 (5.05 GB)
   Deepfake Detection - Faces - Part 15_0: 233k images
   Extensive dataset containing images with real and synthetic (deepfake) faces.
4. Data Preprocessing:
   Images are labeled as real or synthetic (deepfake).
   Format: Images in common image formats like JPEG, PNG, etc.
   Size: 5.05 GB, with a specific subset (Part 15_0) containing 233k images Frames are resized and normalized to a common resolution (e.g., 224x224 pixels) to facilitate model processing.
5. Sample Datapoints:

<table>
<tr><th>REAL</th><th>SYNTHETIC</th></tr>
<tr><td></td><td></td></tr>
</table>

Model Architecture:

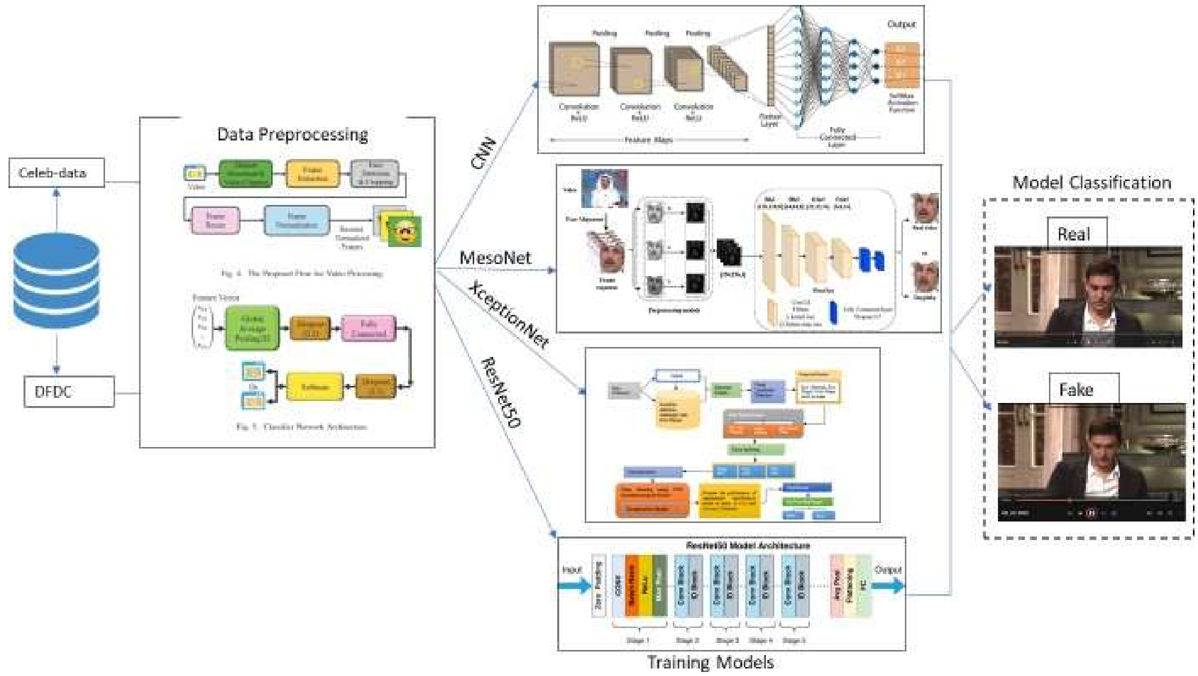


Fig: Schematic Representation of Model built, Flowcharts sourced from Research papers (Reference)

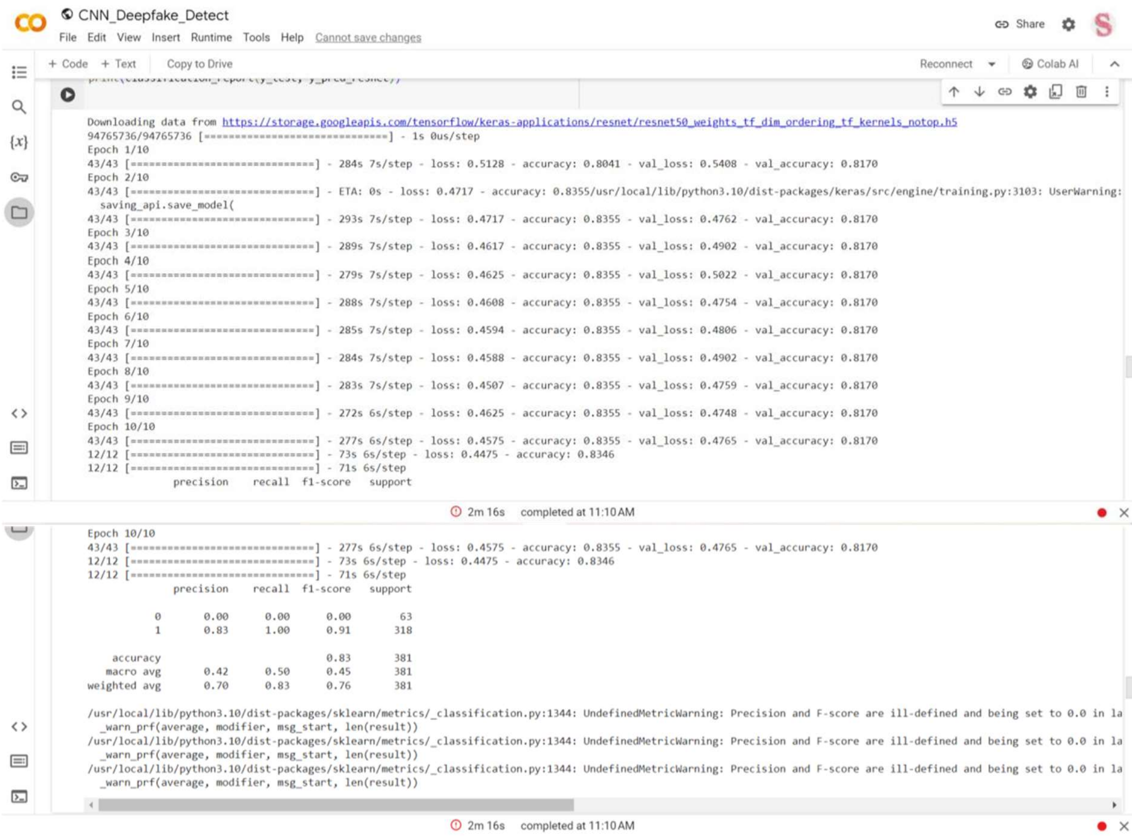
Code:

1. **ResNet-50:**

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

# Define a function to create a ResNet-based model for deepfake detection
def create_resnet_model():
    # Load the pre-trained ResNet50 model without the top (classification) layer
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

    # Freeze the weights of the pre-trained layers
    for layer in base_model.layers:
        layer.trainable = False

    # Add a global average pooling layer
    x = base_model.output
    x = GlobalAveragePooling2D()(x)

    # Add a fully connected layer with 256 units and ReLU activation
    x = Dense(256, activation='relu')(x)
```

```
    # Add a dropout layer with 0.5 dropout rate
    x = Dropout(0.5)(x)

    # Add the final classification layer with sigmoid activation for binary classification
    predictions = Dense(1, activation='sigmoid')(x)

    # Create the model
    model = Model(inputs=base_model.input, outputs=predictions)

    return model

# Create the ResNet-based model
resnet_model = create_resnet_model()

# Compile the model using RMSprop optimizer, binary crossentropy loss, and accuracy metric
resnet_model.compile(optimizer='RMSprop', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model for 10 epochs with a batch size of 32 and a validation split of 0.1 using the model
checkpoint callback
resnet_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1, callbacks=[checkpoint])

# Evaluate the model on the test set
resnet_model.evaluate(X_test, y_test)

# Load the best model from the checkpoint
resnet_model.load_weights("best_model.h5")

# Make predictions on the test set using the best model
y_pred_resnet = resnet_model.predict(X_test)

# Convert the predictions to binary labels (0 or 1) using a threshold of 0.5
y_pred_resnet = (y_pred_resnet > 0.5).astype(int)

# Print the classification report using sklearn.metrics.classification_report
print(classification_report(y_test, y_pred_resnet))
```

Output:



2. CNN (adam optimizer):

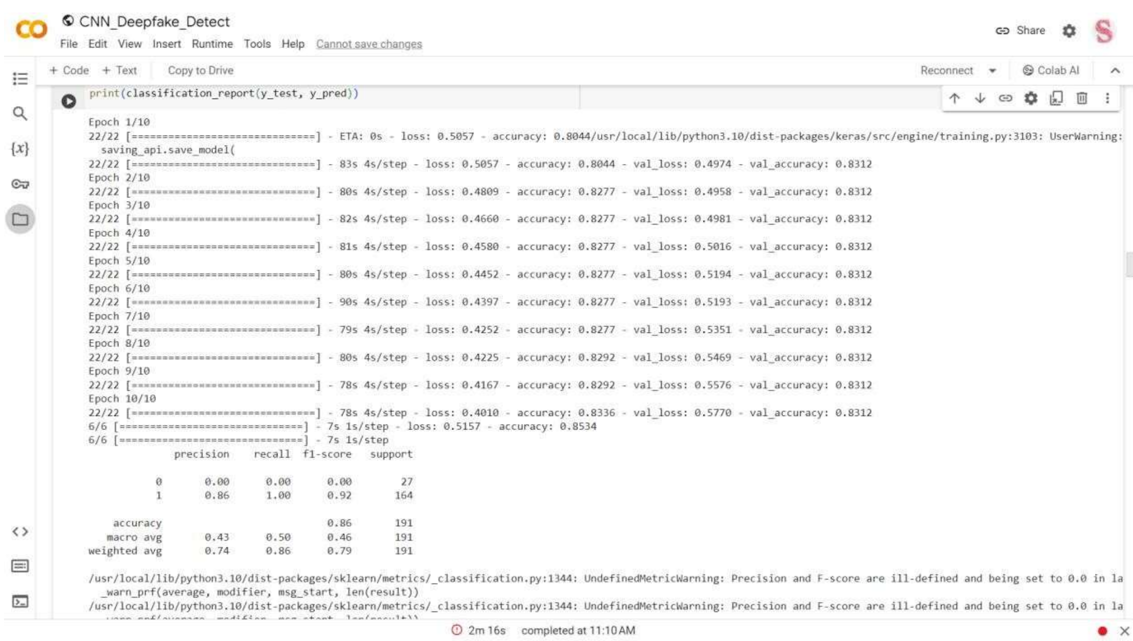
```
# Import the necessary libraries
import numpy as np
import cv2
import os
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint

# Define the paths of the directories containing real and deepfake videos
real_dir = "/kaggle/input/celeb-real"
fake_dir = "/kaggle/input/celeb-synthesis"

# Define a function to extract frames from videos and save them as images
def extract_frames(video_dir, image_dir):
    # Create a directory to store the images if it does not exist
    if not os.path.exists(image_dir):
        os.makedirs(image_dir)
    # Loop through all the videos in the video directory
    for video in os.listdir(video_dir):
        # Read the video using cv2.VideoCapture
        cap = cv2.VideoCapture(os.path.join(video_dir, video))
```

```
        # Get the total number of frames in the video
        frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
        # Choose a random frame index to extract
        frame_index = np.random.randint(0, frame_count)
        # Set the current position of the video to the frame index
        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_index)
        # Read the frame from the video
        success, frame = cap.read()
        # If the frame was successfully read, save it as an image
        if success:
            # Resize the frame to 224 x 224 pixels
            frame = cv2.resize(frame, (224, 224))
            # Get the video name without the extension
            video_name = os.path.splitext(video)[0]
            # Construct the image name using the video name and the frame index
            image_name = video_name + "_" + str(frame_index) + ".jpg"
            # Save the image in the image directory
            cv2.imwrite(os.path.join(image_dir, image_name), frame)
        # Release the video capture object
        cap.release()

# Extract frames from real and deepfake videos and save them as images in separate directories
extract_frames(real_dir, "/kaggle/working/real_images")
extract_frames(fake_dir, "/kaggle/working/fake_images")

# Define a function to load images and labels from image directories
def load_data(image_dirs, labels):
    # Create empty lists to store images and labels
    images = []
    image_labels = []
    # Loop through each image directory and label
    for image_dir, label in zip(image_dirs, labels):
        # Loop through each image in the image directory
        for image in os.listdir(image_dir):
            # Read the image using cv2.imread
            img = cv2.imread(os.path.join(image_dir, image))
            # Convert the image from BGR to RGB color space
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            # Normalize the image pixels to the range [0, 1]
            img = img / 255.0
            # Append the image and label to the lists
            images.append(img)
            image_labels.append(label)
    # Convert the lists to numpy arrays
    images = np.array(images)
    image_labels = np.array(image_labels)
    return images, image_labels
```

```
# Load images and labels from real and fake image directories
X, y = load_data(["/kaggle/working/real_images", "/kaggle/working/fake_images"], [0, 1])

# Split the data into training and testing sets using sklearn.model_selection.train_test_split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define a function to create a CNN model for deepfake detection using tensorflow.keras.Sequential
def create_model():
    # Create an empty sequential model
    model = Sequential()
    # Add a convolutional layer with 32 filters, 3 x 3 kernel size, ReLU activation and input shape of (224, 224,
3)
    model.add(Conv2D(32, (3, 3), activation="relu", input_shape=(224, 224, 3)))
    # Add a max pooling layer with 2 x 2 pool size
    model.add(MaxPooling2D((2, 2)))
    # Add a convolutional layer with 64 filters, 3 x 3 kernel size and ReLU activation
    model.add(Conv2D(64, (3, 3), activation="relu"))
    # Add a max pooling layer with 2 x 2 pool size
    model.add(MaxPooling2D((2, 2)))
    # Add a convolutional layer with 128 filters, 3 x 3 kernel size and ReLU activation
    model.add(Conv2D(128, (3, 3), activation="relu"))
    # Add a max pooling layer with 2 x 2 pool size
    model.add(MaxPooling2D((2, 2)))
    # Add a flatten layer to convert the 3D feature maps to 1D feature vectors
    model.add(Flatten())
    # Add a dense layer with 256 units and ReLU activation
    model.add(Dense(256, activation="relu"))
    # Add a dropout layer with 0.5 dropout rate to prevent overfitting
    model.add(Dropout(0.5))
    # Add a dense layer with 1 unit and sigmoid activation for binary classification
    model.add(Dense(1, activation="sigmoid"))
    # Return the model
    return model

# Create the CNN model
model = create_model()

# Compile the model using Adam optimizer, binary crossentropy loss and accuracy metric
model.compile(optimizer=Adam(learning_rate=0.0001), loss="binary_crossentropy", metrics=["accuracy"])

# Define a model checkpoint callback to save the best model during training
checkpoint = ModelCheckpoint("best_model.h5", save_best_only=True, monitor="val_loss", mode="min")

# Train the model for 10 epochs with batch size of 32 and validation split of 0.1 using the model checkpoint
callback
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1, callbacks=[checkpoint])
```

```
# Evaluate the model on the test set
model.evaluate(X_test, y_test)

# Load the best model from the checkpoint
model.load_weights("best_model.h5")

# Make predictions on the test set using the best model
y_pred = model.predict(X_test)

# Convert the predictions to binary labels (0 or 1) using a threshold of 0.5
y_pred = (y_pred > 0.5).astype(int)

# Print the classification report using sklearn.metrics.classification_report
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

Output:



3. CNN (RMS optimizer):

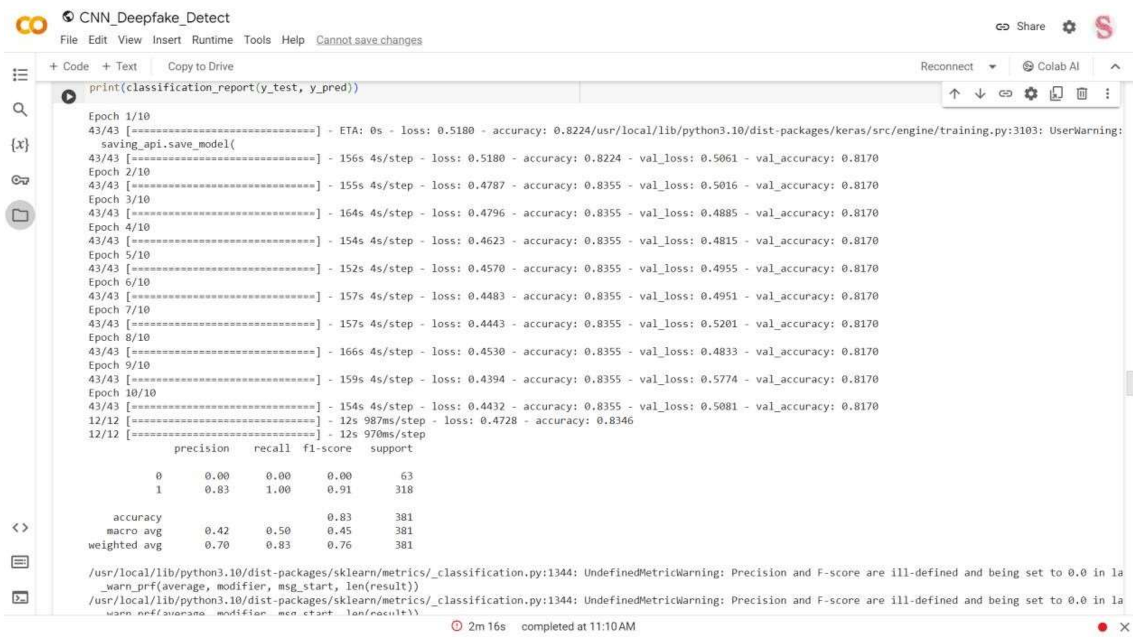
```
# Import the necessary libraries
import numpy as np
import cv2
import os
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.callbacks import ModelCheckpoint
```

```
# Define the paths of the directories containing real and deepfake videos
real_dir = "/kaggle/input/celeb-real"
fake_dir = "/kaggle/input/celeb-synthesis"

# Define a function to extract frames from videos and save them as images
def extract_frames(video_dir, image_dir):
    # Create a directory to store the images if it does not exist
    if not os.path.exists(image_dir):
        os.makedirs(image_dir)
    # Loop through all the videos in the video directory
    for video in os.listdir(video_dir):
        # Read the video using cv2.VideoCapture
        cap = cv2.VideoCapture(os.path.join(video_dir, video))
        # Get the total number of frames in the video
        frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
        # Choose a random frame index to extract
        frame_index = np.random.randint(0, frame_count)
        # Set the current position of the video to the frame index
        cap.set(cv2.CAP_PROP_POS_FRAMES,  frame_index)
        # Read the frame from the video
        success, frame = cap.read()
        # If the frame was successfully read, save it as an image
        if success:
            # Resize the frame to 224 x 224 pixels
            frame = cv2.resize(frame, (224, 224))
            # Get the video name without the extension
            video_name = os.path.splitext(video)[0]
            # Construct the image name using the video name and the frame index
            image_name = video_name + "_" + str(frame_index) + ".jpg"
            # Save the image in the image directory
            cv2.imwrite(os.path.join(image_dir, image_name), frame)
        # Release the video capture object
        cap.release()

# Extract frames from real and deepfake videos and save them as images in separate directories
extract_frames(real_dir, "/kaggle/working/real_images")
extract_frames(fake_dir, "/kaggle/working/fake_images")

# Define a function to load images and labels from image directories
def load_data(image_dirs, labels):
    # Create empty lists to store images and labels
    images = []
    image_labels = []
    # Loop through each image directory and label
    for image_dir, label in zip(image_dirs, labels):
        # Loop through each image in the image directory
        for image in os.listdir(image_dir):
```

```
            # Read the image using cv2.imread
            img = cv2.imread(os.path.join(image_dir, image))
            # Convert the image from BGR to RGB color space
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            # Normalize the image pixels to the range [0, 1]
            img = img / 255.0
            # Append the image and label to the lists
            images.append(img)
            image_labels.append(label)
    # Convert the lists to numpy arrays
    images = np.array(images)
    image_labels = np.array(image_labels)
    return images, image_labels

# Load images and labels from real and fake image directories
X, y = load_data(["/kaggle/working/real_images", "/kaggle/working/fake_images"], [0, 1])

# Split the data into training and testing sets using sklearn.model_selection.train_test_split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define a function to create a CNN model for deepfake detection using tensorflow.keras.Sequential
def create_model():
    # Create an empty sequential model
    model = Sequential()
    # Add a convolutional layer with 32 filters, 3 x 3 kernel size, ReLU activation and input shape of (224, 224,
3)
    model.add(Conv2D(32, (3, 3), activation="relu", input_shape=(224, 224, 3)))
    # Add a max pooling layer with 2 x 2 pool size
    model.add(MaxPooling2D((2, 2)))
    # Add a convolutional layer with 64 filters, 3 x 3 kernel size and ReLU activation
    model.add(Conv2D(64, (3, 3), activation="relu"))
    # Add a max pooling layer with 2 x 2 pool size
    model.add(MaxPooling2D((2, 2)))
    # Add a convolutional layer with 128 filters, 3 x 3 kernel size and ReLU activation
    model.add(Conv2D(128, (3, 3), activation="relu"))
    # Add a max pooling layer with 2 x 2 pool size
    model.add(MaxPooling2D((2, 2)))
    # Add a flatten layer to convert the 3D feature maps to 1D feature vectors
    model.add(Flatten())
    # Add a dense layer with 256 units and ReLU activation
    model.add(Dense(256, activation="relu"))
    # Add a dropout layer with 0.5 dropout rate to prevent overfitting
    model.add(Dropout(0.5))
    # Add a dense layer with 1 unit and sigmoid activation for binary classification
    model.add(Dense(1, activation="sigmoid"))
    # Return the model
```

```
    return model

# Create the CNN model
model = create_model()

# Compile the model using Adam optimizer, binary crossentropy loss and accuracy metric
model.compile(optimizer=RMSprop(learning_rate=0.0001), loss="binary_crossentropy",
metrics=["accuracy"])

# Define a model checkpoint callback to save the best model during training
checkpoint = ModelCheckpoint("best_model.h5", save_best_only=True, monitor="val_loss", mode="min")

# Train the model for 10 epochs with batch size of 32 and validation split of 0.1 using the model checkpoint
callback
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1, callbacks=[checkpoint])

# Evaluate the model on the test set
model.evaluate(X_test, y_test)

# Load the best model from the checkpoint
model.load_weights("best_model.h5")

# Make predictions on the test set using the best model
y_pred = model.predict(X_test)

# Convert the predictions to binary labels (0 or 1) using a threshold of 0.5
y_pred = (y_pred > 0.5).astype(int)

# Print the classification report using sklearn.metrics.classification_report
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

Output:



4. <u>XceptionNet:</u>

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
from torchvision import datasets, transforms
from PIL import Image

# Define data transformations for training and validation
train_transforms = transforms.Compose([
    transforms.Resize((299, 299)), # Resize for Xception input size
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

val_transforms = transforms.Compose([
    transforms.Resize((299, 299)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

# Load custom dataset
train_data = datasets.ImageFolder('path_to_training_data', transform=train_transforms)
val_data = datasets.ImageFolder('path_to_validation_data', transform=val_transforms)

# Define data loaders
train_loader = torch.utils.data.DataLoader(train_data, batch_size=32, shuffle=True)
val_loader = torch.utils.data.DataLoader(val_data, batch_size=32)

# Load pre-trained Xception model
model = torch.hub.load('pytorch/vision:v0.10.0', 'xception', pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)  # Assuming 2 classes: Real and Fake

# Freeze initial layers and only train the last few layers
for param in model.parameters():
    param.requires_grad = False
```

```
for param in model.fc.parameters():
    param.requires_grad = True

# Define loss function and optimizer with weight decay
criterion = nn.CrossEntropyLoss()
weight_decay = 1e-4
optimizer = optim.Adam(model.fc.parameters(), lr=0.001, weight_decay=weight_decay)

# Define learning rate scheduler
scheduler = StepLR(optimizer, step_size=5, gamma=0.1)

# Move model to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Training loop with learning rate scheduling
num_epochs = 10
best_accuracy = 0.0
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * inputs.size(0)

    # Update learning rate
    scheduler.step()

    # Validation
    model.eval()
    correct_predictions = 0
    total_predictions = 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total_predictions += labels.size(0)
            correct_predictions += (predicted == labels).sum().item()

    epoch_accuracy = correct_predictions / total_predictions
    if epoch_accuracy > best_accuracy:
        best_accuracy = epoch_accuracy
        torch.save(model.state_dict(), "best_xception_model.pth")  # Save the best model

    print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {running_loss / len(train_data)}, Accuracy: 
{epoch_accuracy}")

print(f"Best Validation Accuracy: {best_accuracy}")

import torch
import torch.nn.functional as F
from transformers import ViTFeatureExtractor, ViTForImageClassification
from PIL import Image
import os


model_name = 'Wvolf/ViT_Deepfake_Detection'
model = ViTForImageClassification.from_pretrained(model_name)

# Define transformations to preprocess the uploaded image
feature_extractor = ViTFeatureExtractor.from_pretrained(model_name)
```

```
transform = lambda img: feature_extractor(img, return_tensors='pt')

# Function to process an image and get the predicted label
def get_predicted_label(image_path):
    uploaded_image  = Image.open(image_path)
    processed_image = transform(uploaded_image)['pixel_values']

    model.eval() # Set the model to evaluation mode
    with torch.no_grad():
        outputs = model(processed_image)
        predictions = F.softmax(outputs.logits, dim=1)
        predicted_class = torch.argmax(predictions, dim=1).item()

    # Decode the predicted class (0 for Real, 1 for Fake)
    class_labels = ['Real', 'Fake']
    predicted_label = class_labels[predicted_class]

    return predicted_label

# Define the paths to the real and fake faces directories
real_faces_dir = "/content/drive/MyDrive/real_faces/real_faces/DeepFake00/DeepFake00"
fake_faces_dir = "/content/drive/MyDrive/fake_faces/fake_faces/DeepFake02/DeepFake02"

# Get the list of image files in each directory
real_images = [os.path.join(real_faces_dir, file) for file in os.listdir(real_faces_dir) if file.endswith('.jpg')]
fake_images = [os.path.join(fake_faces_dir, file) for file in os.listdir(fake_faces_dir) if file.endswith('.jpg')]

# Calculate accuracy for real faces
total_real_images = len(real_images)
if total_real_images > 0:
    correct_predictions_real = 0

    for image_path in real_images:
        predicted_label = get_predicted_label(image_path)

        if predicted_label == 'Real':
            correct_predictions_real += 1

    accuracy_real = correct_predictions_real / total_real_images
else:
    accuracy_real = 0.0  # Set accuracy to 0 if there are no real images

total_images = total_real_images + total_fake_images
if total_images > 0:
    correct_predictions_all = correct_predictions_real + correct_predictions_fake
    accuracy_all = correct_predictions_all / total_images
else:
    accuracy_all = 0.0  # Set accuracy to 0 if there are no images

# Calculate accuracy for all faces (real and fake)
total_images = total_real_images + total_fake_images
if total_images > 0:
    correct_predictions_all = correct_predictions_real + correct_predictions_fake
    accuracy_all = correct_predictions_all / total_images
else:
    accuracy_all = 0.0 # Set accuracy to 0 if there are no images

print(f'Overall Accuracy: {accuracy_all * 100:.2f}%')
```

Output:

```python
# Calculate accuracy for all faces (real and fake)
total_images = total_real_images + total_fake_images
if total_images > 0:
    correct_predictions_all = correct_predictions_real + correct_predictions_fake
    accuracy_all = correct_predictions_all / total_images
else:
    accuracy_all = 0.0  # Set accuracy to 0 if there are no images

print(f'Overall Accuracy: {accuracy_all * 100:.2f}%')
```

```
Overall Accuracy: 86.5
```

5. **Meso-InceptionNet:**

```python
!pip install ../input/mtcnn-package/mtcnn-0.1.0-py3-none-any.whl

import pandas as pd
import keras
import os
import numpy as np
from sklearn.metrics import log_loss
from keras import Model,Sequential
from keras.layers import *
from keras.optimizers import *
from sklearn.model_selection import train_test_split
import cv2
from tqdm.notebook import tqdm
import glob
from mtcnn import MTCNN

sorted(glob.glob('../input/deepfake/meta*'))

import pandas as pd

df_trains = []
df_vals = []

# Read training dataframes
for i in range(47):
    df_trains.append(pd.read_json(f'../input/deepfake/metadata{i}.json'))

# Read validation dataframes
for i in range(47, 50):
    df_vals.append(pd.read_json(f'../input/deepfake/metadata{i}.json'))

nums = list(range(len(df_trains) + 1))
LABELS = ['REAL', 'FAKE']
val_nums = [47, 48, 49]

def get_path(num,x):
    num=str(num)
    if len(num)==2:
        path='../input/deepfake/DeepFake'+num+'/DeepFake'+num+'/' + x.replace('.mp4', '') + '.jpg'
    else:
        path='../input/deepfake/DeepFake0'+num+'/DeepFake0'+num+'/' + x.replace('.mp4', '') + '.jpg'
    if not os.path.exists(path):
       raise Exception
    return path
paths=[]
y=[]
for df_train,num in tqdm(zip(df_trains,nums),total=len(df_trains)):
    images = list(df_train.columns.values)
    for x in images:
        try:
```

```
            paths.append(get_path(num,x))
            y.append(LABELS.index(df_train[x]['label']))
        except Exception as err:
            #print(err)
            pass

val_paths=[]
val_y=[]
for df_val,num in tqdm(zip(df_vals,val_nums),total=len(df_vals)):
    images = list(df_val.columns.values)
    for x in images:
        try:
            val_paths.append(get_path(num,x))
            val_y.append(LABELS.index(df_val[x]['label']))
        except Exception as err:
            #print(err)
            pass


import random
real=[]
fake=[]
for m,n in zip(paths,y):
    if n==0:
        real.append(m)
    else:
        fake.append(m)
fake=random.sample(fake,len(real))
paths,y=[],[]
for x in real:
    paths.append(x)
    y.append(0)
for x in fake:
    paths.append(x)
    y.append(1)



print('There are '+str(y.count(1))+' fake train samples')
print('There are '+str(y.count(0))+' real train samples')
print('There are '+str(val_y.count(1))+' fake val samples')
print('There are '+str(val_y.count(0))+' real val samples')

def read_img(path):
    return cv2.cvtColor(cv2.imread(path),cv2.COLOR_BGR2RGB)
X=[]
for img in tqdm(paths):
    X.append(read_img(img))
val_X=[]
for img in tqdm(val_paths):
    val_X.append(read_img(img))

import random
def shuffle(X,y):
    new_train=[]
    for m,n in zip(X,y):
        new_train.append([m,n])
    random.shuffle(new_train)
    X,y=[],[]
    for x in new_train:
        X.append(x[0])
        y.append(x[1])
    return X,y

def InceptionLayer(a, b, c, d):
    def func(x):
        x1 = Conv2D(a, (1, 1), padding='same', activation='elu')(x)
```

```
        x2 = Conv2D(b, (1, 1), padding='same', activation='elu')(x)
        x2 = Conv2D(b, (3, 3), padding='same', activation='elu')(x2)

        x3 = Conv2D(c, (1, 1), padding='same', activation='elu')(x)
        x3 = Conv2D(c, (3, 3), dilation_rate = 2, strides = 1, padding='same', activation='elu')(x3)

        x4 = Conv2D(d, (1, 1), padding='same', activation='elu')(x)
        x4 = Conv2D(d, (3, 3), dilation_rate = 3, strides = 1, padding='same', activation='elu')(x4)
        y = Concatenate(axis = -1)([x1, x2, x3, x4])

        return y
    return func

def define_model(shape=(256,256,3)):
    x = Input(shape = shape)

    x1 = InceptionLayer(1, 4, 4, 2)(x)
    x1 = BatchNormalization()(x1)
    x1 = MaxPooling2D(pool_size=(2, 2), padding='same')(x1)

    x2 = InceptionLayer(2, 4, 4, 2)(x1)
    x2 = BatchNormalization()(x2)
    x2 = MaxPooling2D(pool_size=(2, 2), padding='same')(x2)

    x3 = Conv2D(16, (5, 5), padding='same', activation = 'elu')(x2)
    x3 = BatchNormalization()(x3)
    x3 = MaxPooling2D(pool_size=(2, 2), padding='same')(x3)

    x4 = Conv2D(16, (5, 5), padding='same', activation = 'elu')(x3)
    x4 = BatchNormalization()(x4)
    if shape==(256,256,3):
        x4 = MaxPooling2D(pool_size=(4, 4), padding='same')(x4)
    else:
        x4 = MaxPooling2D(pool_size=(2, 2), padding='same')(x4)
    y = Flatten()(x4)
    y = Dropout(0.5)(y)
    y = Dense(16)(y)
    y = LeakyReLU(alpha=0.1)(y)
    y = Dropout(0.5)(y)
    y = Dense(1, activation = 'sigmoid')(y)
    model=Model(inputs = x, outputs = y)
    model.compile(loss='binary_crossentropy',optimizer=Adam(lr=1e-4))
    #model.summary()
    return model
df_model=define_model()
df_model.load_weights('../input/meso-pretrain/MesoInception_DF')
f2f_model=define_model()
f2f_model.load_weights('../input/meso-pretrain/MesoInception_F2F')

from keras.callbacks import LearningRateScheduler

lrs=[1e-3,5e-4,1e-4]
def schedule(epoch):
    return lrs[epoch]

import numpy as np
import gc
from keras import backend as K  # Import Keras backend

kfolds = 5
losses = []

# Assuming X, y, val_X, and val_y are lists of arrays or single values
X = np.array(X)
y = np.array(y)
val_X = np.array(val_X)
```

```
val_y = np.array(val_y)

if LOAD_PRETRAIN:
    df_models  =  []
    f2f_models = []
    i = 0

    while len(df_models) < kfolds:
        model = define_model((150, 150, 3))
        if i == 0:
            model.summary()

        model.fit(X, y, epochs=2, callbacks=[LearningRateScheduler(schedule)])
        pred = model.predict(val_X)
        loss = log_loss(val_y, pred)
        losses.append(loss)
        print('fold ' + str(i) + ' model loss: ' + str(loss))
        df_models.append(model)

        # Clearing memory
        K.clear_session()  # Use keras.backend.clear_session()
        del model
        gc.collect()

        i += 1

    i = 0

    while len(f2f_models) < kfolds:
        model = define_model((150, 150, 3))

        model.fit(X, y, epochs=2, callbacks=[LearningRateScheduler(schedule)])
        pred = model.predict(val_X)
        loss = log_loss(val_y, pred)
        losses.append(loss)
        print('fold ' + str(i) + ' model loss: ' + str(loss))
        f2f_models.append(model)

        # Clearing memory
        K.clear_session()  # Use keras.backend.clear_session()
        del model
        gc.collect()

        i += 1

    models = f2f_models + df_models

else:
    models = []
    i = 0

    while len(models) < kfolds:
        model = define_model((150, 150, 3))
        if i == 0:
            model.summary()

        model.fit(X, y, epochs=2, callbacks=[LearningRateScheduler(schedule)])
        pred = model.predict(val_X)
        loss = log_loss(val_y, pred)
        losses.append(loss)
        print('fold ' + str(i) + ' model loss: ' + str(loss))

        if loss < 0.68:
            models.append(model)
        else:
            print('loss too bad, retrain!')
```

```
        # Clearing memory
        K.clear_session()  # Use keras.backend.clear_session()
        del model
        gc.collect()

        i += 1


def prediction_pipline(X,two_times=False):
    preds=[]
    for model in tqdm(models):
        pred=model.predict([X])
        preds.append(pred)
    preds=sum(preds)/len(preds)
    if two_times:
        return larger_range(preds,2)
    else:
        return preds
def larger_range(model_pred,time):
    return (((model_pred-0.5)*time)+0.5)
best_model_pred=models[losses.index(min(losses))].predict([val_X])

model_pred=prediction_pipline(val_X)


threshold = 0.5
best_model_labels = (best_model_pred > threshold).astype(int)
model_labels = (model_pred > threshold).astype(int)


from sklearn.metrics import accuracy_score

best_model_accuracy = accuracy_score(val_y, best_model_labels)
model_accuracy = accuracy_score(val_y, model_labels)

print(f"Best Model Accuracy: {best_model_accuracy * 100:.2f}%")
print(f"Model Accuracy: {model_accuracy * 100:.2f}%")
```

Output:

```
Best Model Accuracy: 61.21%
Model Accuracy: 62.72%
```

Model Comparison:

Aspect	CNN (Adam)	CNN (RMSprop)	ResNet	Xception	MesoInception
Architecture	Custom CNN architecture with convolutional layers, pooling, and fully connected layers	Custom CNN architecture with convolutional layers, pooling, and fully connected layers	Residual network with 50 layers, utilizing residual blocks to improve training and performance.	Deep convolutional neural network with 71 layers, including depthwise separable convolutions.	Inception-like architecture with multiple convolutional layers and skip connections for feature extraction.

Preprocessing	Resize to 224x224, random horizontal flip, normalization	Resize to 224x224, random horizontal flip, normalization	Resize to 224x224, random horizontal flip, normalization	Resize to 299x299, random horizontal flip, random rotation, color jittering, normalization.	Resize to specific dimensions, random horizontal flip, normalization.
Model Loading	Create custom CNN model using TensorFlow and Keras	Create custom CNN model using TensorFlow and Keras	Load pre-trained ResNet-50 model from torchvision.	Load pre-trained Xception model from PyTorch hub.	Load pre-trained MesoInception model.
Fine-Tuning	Train all layers with adaptive learning rate scheduling.	Train all layers with adaptive learning rate scheduling.	Train all layers with adaptive learning rate scheduling.	Freeze initial layers, train last layers, adaptive learning rate scheduling.	Fine-tune entire model with adaptive learning rate scheduling.
Optimizer	Adam optimizer with default settings (lr=0.001).	RMSprop optimizer with default settings (lr=0.001)	Adam optimizer with default settings (lr=0.001).	Adam optimizer with weight decay (lr=0.001).	Adam optimizer with default settings (lr=0.001).
Loss Function	Binary cross-entropy loss function.	Binary cross-entropy loss function.	Cross-entropy loss function for binary classification.	Cross-entropy loss function for multi-class classification	Binary cross-entropy loss function.
Model Size	Varies based on customization and layer complexity.	Varies based on customization and layer complexity.	Moderate model size due to fewer layers compared to Xception.	Large model size due to complex architecture.	Moderate model size.
Accuracy	85.34%	83.26%	83.46%	86.5%	62.7%

<u>Conclusion:</u>

In evaluating the performance of various deepfake detection models, several key aspects emerge from the comparative analysis. Firstly, the choice of architecture significantly impacts both the robustness and computational efficiency of the models. While custom CNN architectures offer flexibility and control over model design, ResNet's adoption of residual blocks enhances training stability and performance. The Xception model, with its depth wise separable convolutions and skip connections, strikes a balance between complexity and accuracy. On the other hand, MesoInception, despite its moderate size, exhibits lower accuracy, suggesting potential limitations in feature extraction.

Regarding pre-processing and model loading, a standardized approach is observed across all models, involving resizing, data augmentation, and normalization to ensure consistency in input data representation. However, the choice of pre-trained models, such as ResNet-50 and Xception, offers the advantage of leveraging learned features from large-scale datasets, reducing the need for extensive training data. In contrast, the custom CNN architectures require manual construction, necessitating careful design considerations and tuning for optimal performance.

In terms of optimization and fine-tuning strategies, adaptive learning rate scheduling proves crucial in optimizing model convergence and avoiding overfitting. While Adam and RMSprop optimizers demonstrate comparable performance in CNN models, the use of weight decay in the Xception model contributes to enhanced regularization and generalization capabilities. Despite variations in model size, ranging from moderate to large, the observed accuracies highlight that CNN with Adam optimiser and XceptionNet as top performers, achieving accuracies of 85.34% and 86.5%, respectively. However, the MesoInception model lags behind, indicating potential areas for improvement in feature representation and model architecture.

References:

[1] R. Punithavathi, M. Kumarasamy, M. Sai M., R. Hiruthik, S. Sripadmesh, and R. V. Kishore. Deepfake Detection with Deeplearning Using Resnet CNN Algorithm. In Proceedings of the International Conference on Recent Trends in Data Science and its Applications (pp. 1084). [DOI: rp-9788770040723.209]

[2] Saxena, A., Yadav, D., Gupta, M.* (corresponding author), Phulre, S., Arjariya, T., Jaiswal, V., &Bhujade, R. K. (2023). Detecting Deepfakes: A Novel Framework Employing XceptionNet-Based Convolutional Neural Networks. [Journal Name], 835-846. https://doi.org/10.18280/ts.400301

[3] Thing, V. L. L. (2022). Deepfake Detection with Deep Learning: Convolutional Neural Networksversus Transformers.

[4] Rajalaxmi, R. R., Sudharsana, P. P., Rithani, A. M., Preethika, S., Dhivakar, P., & Gothai, E. (2023, February 23-25). Deepfake detection using Inception-ResNet-v2 network. In 2023 IEEE InternationalConference on Computational Communication and Mobile Computing (ICCMC) (pp. 1008358). IEEE.DOI: 10.1109/ICCMC56507.2023.1008358: https://doi.org/10.1109/ICCMC56507.2023.1008358

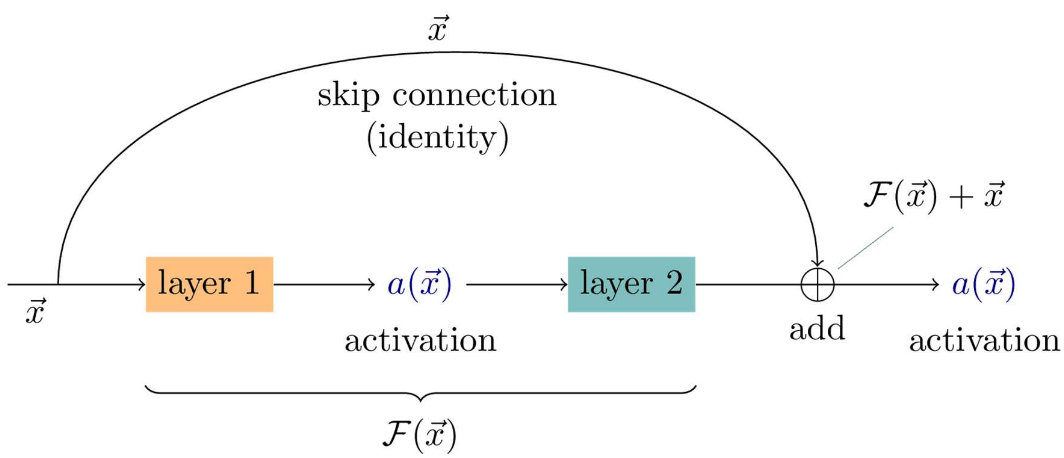
[5] Mitra, A., Mohanty, S. P., Corcoran, P., & Kougianos, E. (2020, December). A Novel MachineLearning based Method for Deepfake Video Detection in Social Media. In 2020 15th IEEE International Symposium on Intelligent Systems and Engineering (iSES) (pp. 31-36). IEEE.

[6] Naitali, A., Ridouani, M., Salahdine, F., & Kaabouch, N. (2023). Deepfake Attacks: Generation,Detection, Datasets, Challenges, and Research Directions. Computers, 12(10), 216. https://doi.org/10.3390/computers12100216 (Special Issue: Current Issue and Future Directions in Multimedia Hiding and Signal Processing)

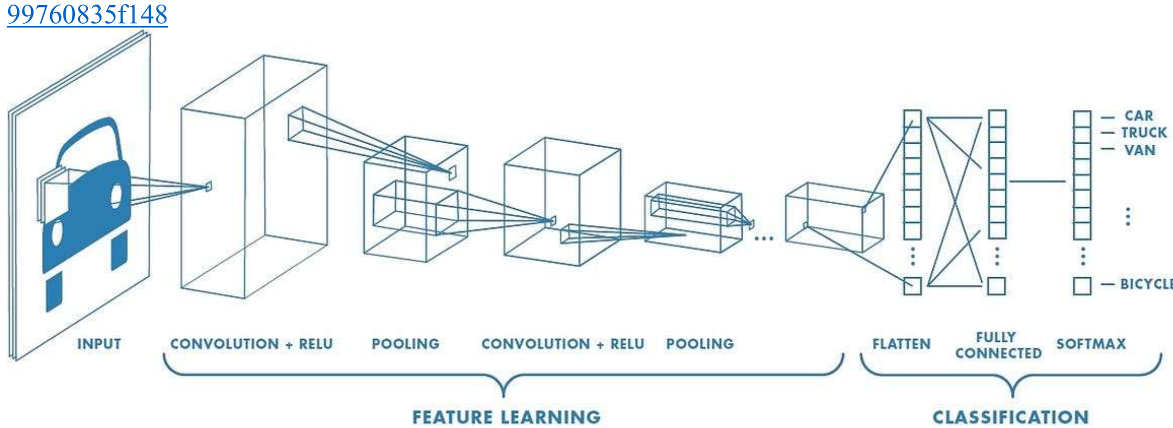
Date: 04.04.24	**PROOF OF EVALUATION**

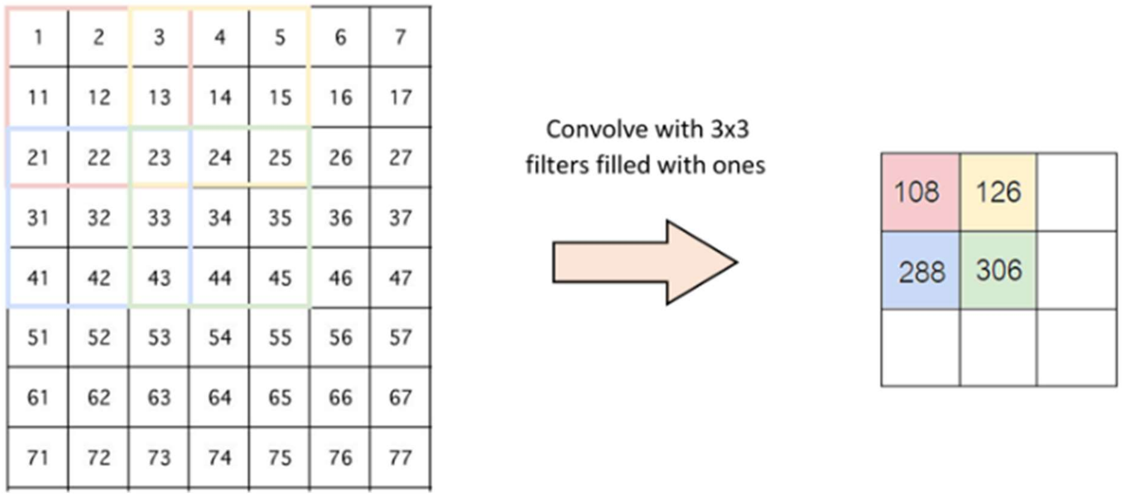
<u>Questions asked during Project Review:</u>

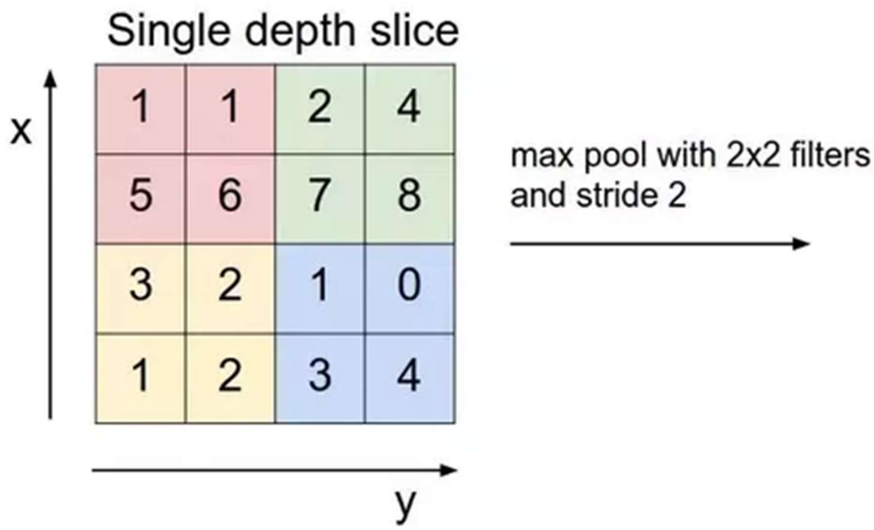
1. What type of Datasets were used in the project?
    - Images
    - Videos (sequence of image frames)
2. What is your primary reference for your project?
    - https://arxiv.org/ftp/arxiv/papers/2401/2401.06999.pdf
3. References Accuracy vs Our achieved accuracy:
   https://www.mdpi.com/2073-8994/14/5/939
    - We achieved 86.5% using MesoNet Inception.
    - They achieved 94.12%
4. What disadvantage of CNN does ResNet50 overcome?
    - It helps overcome <u>vanishing gradient problem</u> through *skip connections*.
5.

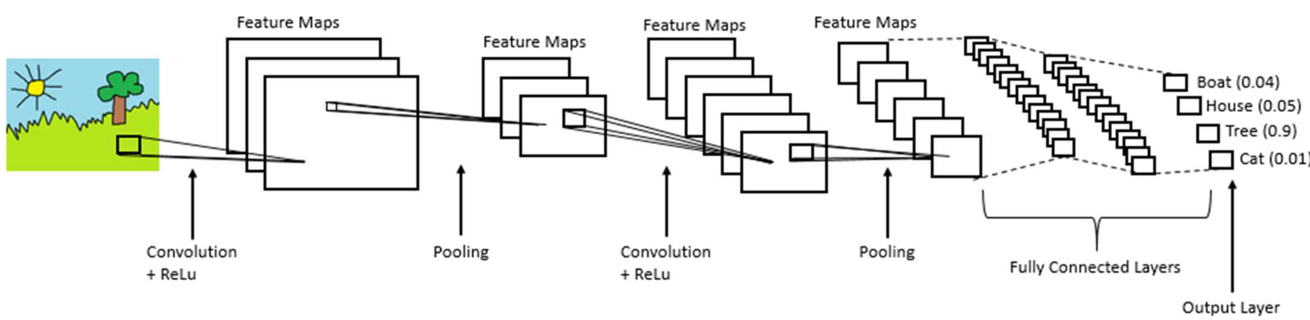


6. CNN architecture:
   https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148









Final Remarks:

Appreciation on choosing a good problem statement, We were asked to furtherly proceed with the project with different model implementations and to experiment with multiple datasets.