INDIRA GANDHI DELHI TECHNICAL UNIVERSITY

COMPUTER VISION ASSIGNMENT 4

Name : Siya Pathak
Enrollment No:06601032021
Course : BTech
Branch :IT-1

Machine Learning and Images

Q1. Download the dataset in your local machine, extract the zip file, and downsample the dataset such that there are 100 images from each class. Consider only the first ten classes. Upload these downsampled images to your google drive (your official gmail account), mount the drive in your google colab notebook to read the dataset. Read and display any random two images from any 2-3 classes in google colab.

CODE FOR DOWNSAMPLING

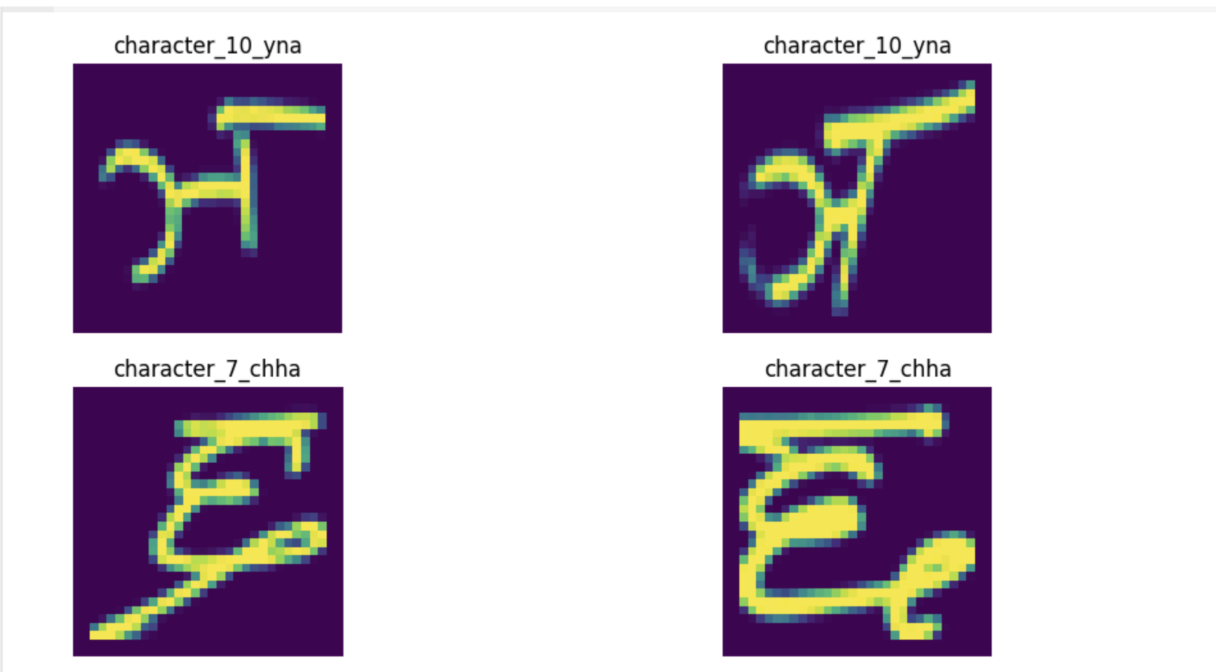
```
   # Get list of character folders in 'train' directory
            train_character_folders = [folder for folder in os.listdir(os.path.join(dataset_path, 'train')) if not folder.startswith('.')]

         # Create 'test' and 'train' directories within the 'downsampled_dataset' directory
   test_dir = os.path.join(downsampled_path, 'test')
   train_dir = os.path.join(downsampled_path, 'train')
   os.makedirs(test_dir, exist_ok=True)
   os.makedirs(train_dir, exist_ok=True)

   # Sort character folders based on the numerical part
           sorted_character_folders = sorted(train_character_folders, key=extract_number)

   # Select the first 10 character folders
   selected_character_folders = sorted_character_folders[:10]

   for class_name in selected_character_folders:
       class_train_dir = os.path.join(dataset_path, 'train', class_name)
       class_test_dir = os.path.join(dataset_path, 'test', class_name)
       downsampled_class_train_dir = os.path.join(train_dir, class_name)
       downsampled_class_test_dir = os.path.join(test_dir, class_name)
       os.makedirs(downsampled_class_train_dir, exist_ok=True)
       os.makedirs(downsampled_class_test_dir, exist_ok=True)

       # Pick 85 images randomly from train folder
         train_images = [file for file in os.listdir(class_train_dir) if os.path.isfile(os.path.join(class_train_dir, file))]
                    selected_train_images = random.sample(train_images, min(sample_size_per_folder * 85, len(train_images)))
       for image in selected_train_images:
```

```
                        shutil.copy(os.path.join(class_train_dir,  image),
downsampled_class_train_dir)

       # Pick 15 images randomly from test folder
          test_images = [file for file in os.listdir(class_test_dir) if
os.path.isfile(os.path.join(class_test_dir, file))]
                     selected_test_images   =   random.sample(test_images,
min(sample_size_per_folder * 15, len(test_images)))
       for image in selected_test_images:
                         shutil.copy(os.path.join(class_test_dir,  image),
downsampled_class_test_dir)

# Downsample the dataset
downsample_dataset(dataset_path, downsampled_path, 1)
```

CODE FOR MOUNTING ON DRIVE AND DISPLAYING IMAGES

```
import os
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Define the path to the dataset
dataset_path = '/content/drive/MyDrive/downsampled_dataset/'

# Select two random classes
selected_classes = random.sample(os.listdir(dataset_path + 'train'), 2)

# Display two random images from each class
plt.figure(figsize=(12, 6))

for i, class_name in enumerate(selected_classes):
   class_path = os.path.join(dataset_path, 'train', class_name)
   class_images = os.listdir(class_path)
   selected_images = random.sample(class_images, 2)
```

```
    for j, image_name in enumerate(selected_images):
        image_path = os.path.join(class_path, image_name)
        img = mpimg.imread(image_path)
        plt.subplot(2, 2, i*2+j+1)
        plt.imshow(img)
        plt.title(class_name)
        plt.axis('off')

plt.show()
```



Q2. Basic CNN: Create your own custom convolutional neural network (CNN) architecture. Use two back to back convolution (CONV-CONV) layers followed by a pooling (POOL) layer. Repeat this two times. Decide on the number of filters for each CONV layer and size of each of these filters. Use at most 1-2 fully connected (FC) layers, decide the number of neurons in each of these FC layers yourself. Give reasons/justification for all of your decisions in creating the custom CNN architecture. Split the data into 85-15 proportions (use “stratified” sampling in the splitting process, learn about this sampling) for training and testing, use 15% of this training data for validation. Decide any appropriate learning rate, and perform training for a suitable number of epochs. Observe the training performance using learning plots (display them after training), you should decide the number of epochs based on these learning plots. Evaluate your trained model on the test data. Output precision, recall, F1-score for each class, and also find the average accuracy.

I HAVE USED 85-15 proportions

Number of Filters and Size of Filters:

- Chosen based on typical values for CNN architectures. 32 filters capture basic features, while 64 filters capture higher-level features.

Pooling Layers:

- Max pooling with a 2x2 pool size is chosen to reduce spatial dimensions while retaining important features.

Fully Connected Layers:

- A single fully connected layer with 128 neurons captures learned features effectively.

Number of Epochs:

- Initial choice of 10 epochs; final epoch value determined based on observing validation accuracy. In this case, final epoch is chosen as 6 due to better validation accuracy.

Learning Rate:

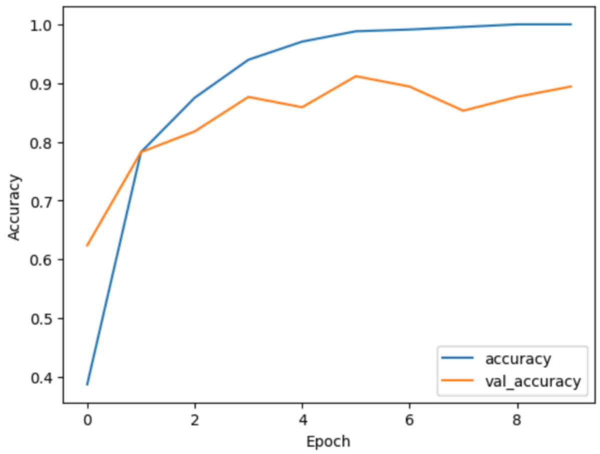
- Learning rate of 0.001 is common; can be adjusted based on model convergence and performance.

```
import os

# Define the root directory where your data is located
data_root = '/content/drive/MyDrive/downsampled_dataset/'

# Initialize lists to store image paths and their corresponding labels for
training and testing
X_train = []
y_train = []
X_test = []
y_test = []

# Create a mapping of class names to class indices
class_to_index = {}
for i, class_folder in enumerate(os.listdir(data_root + 'train')):
   class_to_index[class_folder] = i

# Iterate over class folders in the training set
for class_folder in os.listdir(data_root + 'train'):
   class_folder_path = os.path.join(data_root, 'train', class_folder)
   # Iterate over images in the class folder and append paths to X_train
```

```
    for image_name in os.listdir(class_folder_path):
        image_path = os.path.join(class_folder_path, image_name)
        X_train.append(image_path)
        # Assign label to the image based on its class folder name
        y_train.append(class_to_index[class_folder])

# Repeat the process for the test data
for class_folder in os.listdir(data_root + 'test'):
    class_folder_path = os.path.join(data_root, 'test', class_folder)
    # Iterate over images in the class folder and append paths to X_test
    for image_name in os.listdir(class_folder_path):
        image_path = os.path.join(class_folder_path, image_name)
        X_test.append(image_path)
        # Assign label to the image based on its class folder name
        y_test.append(class_to_index[class_folder])

import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np
import os

# Function to load and preprocess an image
def load_and_preprocess_image(image_path, target_size=(32, 32)):
    img = load_img(image_path, target_size=target_size)
    img_array = img_to_array(img) / 255.0  # Normalize pixel values to [0,
1]
    return img_array

# Load and preprocess images for training data
X_train = [load_and_preprocess_image(image_path) for image_path in
X_train]
X_train = np.array(X_train)
y_train = np.array(y_train)

# Load and preprocess images for validation data
X_val = [load_and_preprocess_image(image_path) for image_path in X_val]
X_val = np.array(X_val)
y_val = np.array(y_val)

# Load and preprocess images for testing data
```

```python
X_test = [load_and_preprocess_image(image_path) for image_path in X_test]
X_test = np.array(X_test)
y_test = np.array(y_test)

# Define CNN architecture
model = tf.keras.Sequential([
   tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32,
32, 3)),
   tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
   tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
   tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
   tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
   tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(128, activation='relu'),
   tf.keras.layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_data=(X_val, y_val))

# Plot learning curves
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

# Evaluate the model on the test data
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
print(classification_report(y_test, y_pred_classes))
```

```
Epoch 6/10
22/22 [==============================] - 2s 86ms/step - loss: 0.0458 - accuracy: 0.9882 - val_loss: 0.2877 - val_accuracy: 0.9118
Epoch 7/10
22/22 [==============================] - 3s 126ms/step - loss: 0.0242 - accuracy: 0.9912 - val_loss: 0.4225 - val_accuracy: 0.8941
Epoch 8/10
22/22 [==============================] - 3s 146ms/step - loss: 0.0193 - accuracy: 0.9956 - val_loss: 0.5371 - val_accuracy: 0.8529
Epoch 9/10
22/22 [==============================] - 3s 128ms/step - loss: 0.0057 - accuracy: 1.0000 - val_loss: 0.3774 - val_accuracy: 0.8765
Epoch 10/10
22/22 [==============================] - 2s 84ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.3617 - val_accuracy: 0.8941
```



Epoch

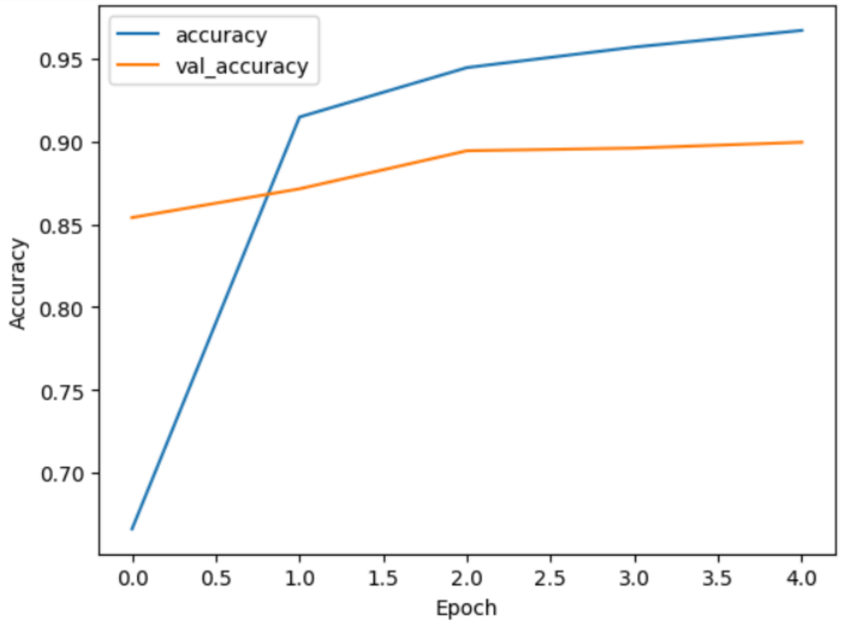
```
5/5 [==============================] - 0s 36ms/step
              precision    recall  f1-score   support

           0       1.00      0.93      0.97        15
           1       1.00      0.93      0.97        15
           2       1.00      1.00      1.00        15
           3       0.75      1.00      0.86        15
           4       0.93      0.93      0.93        15
           5       1.00      0.80      0.89        15
           6       0.87      0.87      0.87        15
           7       0.78      0.93      0.85        15
           8       0.86      0.80      0.83        15
           9       1.00      0.87      0.93        15

    accuracy                           0.91       150
   macro avg       0.92      0.91      0.91       150
weighted avg       0.92      0.91      0.91       150
```

Q3: Complex CNN: Create a more complex custom CNN architecture this time. This means that you can use more than two CONV-CONV-POOL combinations. You can also use CONV-POOL combinations instead or mix of both. Use more than 2 fully connected (FC) layers this time. Split the data into 80-20 proportions (use “stratified” sampling in the splitting process, learn about this sampling) for training and testing, use 20% of this training data for validation. Decide any appropriate learning rate, and perform training for a suitable number of epochs. Observe the training performance using learning plots (display them after training), you should decide the number of epochs based on these learning plots. Evaluate your trained model on the test data. Output precision, recall, F1-score for each class, and also find the average accuracy.

```
import os
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# Assuming you have already loaded and preprocessed the data
# X_train, y_train, X_val, y_val, X_test, y_test are assumed to be defined

# Convert labels to one-hot encoded vectors
num_classes = len(np.unique(y_train))
y_train = to_categorical(y_train, num_classes)
y_val = to_categorical(y_val, num_classes)
y_test = to_categorical(y_test, num_classes)

# Define the CNN architecture
model = Sequential([
   Conv2D(32, (3, 3), activation='relu', input_shape=X_train.shape[1:]),
   Conv2D(64, (3, 3), activation='relu'),
   MaxPooling2D(pool_size=(2, 2)),
   Conv2D(128, (3, 3), activation='relu'),
   Conv2D(256, (3, 3), activation='relu'),
   MaxPooling2D(pool_size=(2, 2)),
   Flatten(),
   Dense(512, activation='relu'),
   Dense(256, activation='relu'),
   Dense(128, activation='relu'),
   Dense(num_classes, activation='softmax')
])
```

```python
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32,
validation_data=(X_val, y_val))

# Plot training curves
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the model on test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

# Predictions
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Classification report
print(classification_report(y_true, y_pred_classes))
```

```
Found 62560 images belonging to 46 classes.
Found 15640 images belonging to 46 classes.
Epoch 1/5
977/977 [==============================] - 122s 123ms/step - loss: 1.1699 - accuracy: 0.6661 - val_loss: 0.4807 - val_accuracy: 0.8541
Epoch 2/5
977/977 [==============================] - 118s 121ms/step - loss: 0.2842 - accuracy: 0.9148 - val_loss: 0.4199 - val_accuracy: 0.8714
Epoch 3/5
977/977 [==============================] - 120s 123ms/step - loss: 0.1818 - accuracy: 0.9446 - val_loss: 0.3493 - val_accuracy: 0.8944
Epoch 4/5
977/977 [==============================] - 120s 123ms/step - loss: 0.1391 - accuracy: 0.9570 - val_loss: 0.3540 - val_accuracy: 0.8960
Epoch 5/5
977/977 [==============================] - 118s 121ms/step - loss: 0.1120 - accuracy: 0.9671 - val_loss: 0.3460 - val_accuracy: 0.8995
```




```
Found 78200 images belonging to 46 classes.
1222/1222 [==============================] - 54s 44ms/step
Classification Report:
                      precision    recall  f1-score   support

     character_10_yna       0.98      1.00      0.99      1700
character_11_taamatar       0.99      0.97      0.98      1700
    character_12_thaa       0.99      0.98      0.99      1700
     character_13_daa       0.89      0.99      0.94      1700
    character_14_dhaa       0.90      0.97      0.93      1700
```

	precision	recall	f1-score	support
character_10_yna	0.98	1.00	0.99	1700
character_11_taamatar	0.99	0.97	0.98	1700
character_12_thaa	0.99	0.98	0.99	1700
character_13_daa	0.89	0.99	0.94	1700
character_14_dhaa	0.90	0.97	0.93	1700
character_15_adna	0.95	1.00	0.97	1700
character_16_tabala	0.99	0.99	0.99	1700
character_17_tha	0.97	0.90	0.94	1700
character_18_da	0.96	0.89	0.92	1700
character_19_dha	0.94	0.94	0.94	1700
character_1_ka	0.98	0.96	0.97	1700
character_20_na	0.99	0.98	0.98	1700
character_21_pa	0.96	0.98	0.97	1700
character_22_pha	0.95	1.00	0.97	1700
character_23_ba	0.95	0.96	0.95	1700
character_24_bha	0.98	0.93	0.95	1700
character_25_ma	0.95	0.98	0.96	1700
character_26_yaw	0.94	0.97	0.95	1700
character_27_ra	0.99	0.97	0.98	1700
character_28_la	0.98	0.99	0.98	1700
character_29_waw	0.99	0.91	0.95	1700
character_2_kha	1.00	0.97	0.98	1700
character_30_motosaw	0.98	1.00	0.99	1700
character_31_petchiryakha	0.98	0.99	0.99	1700
character_32_patalosaw	0.96	0.96	0.96	1700
character_33_ha	0.97	0.95	0.96	1700
character_34_chhya	0.98	0.98	0.98	1700
character_35_tra	1.00	0.98	0.99	1700

character_31_petchiryakha	0.98	0.99	0.99	1700
character_32_patalosaw	0.96	0.96	0.96	1700
character_33_ha	0.97	0.95	0.96	1700
character_34_chhya	0.98	0.98	0.98	1700
character_35_tra	1.00	0.98	0.99	1700

	precision	recall	f1-score	support
character_36_gya	0.99	0.98	0.99	1700
character_3_ga	0.99	0.97	0.98	1700
character_4_gha	0.90	0.95	0.92	1700
character_5_kna	0.99	0.93	0.96	1700
character_6_cha	1.00	0.98	0.99	1700
character_7_chha	0.97	0.97	0.97	1700
character_8_ja	0.98	0.98	0.98	1700
character_9_jha	0.99	0.98	0.99	1700
digit_0	0.98	1.00	0.99	1700
digit_1	0.99	1.00	0.99	1700
digit_2	0.98	0.99	0.99	1700
digit_3	0.98	1.00	0.99	1700
digit_4	0.99	0.99	0.99	1700
digit_5	0.99	1.00	0.99	1700
digit_6	0.98	1.00	0.99	1700
digit_7	0.99	0.98	0.99	1700
digit_8	0.99	0.99	0.99	1700
digit_9	0.99	0.99	0.99	1700
accuracy			0.97	78200
macro avg	0.97	0.97	0.97	78200
weighted avg	0.97	0.97	0.97	78200

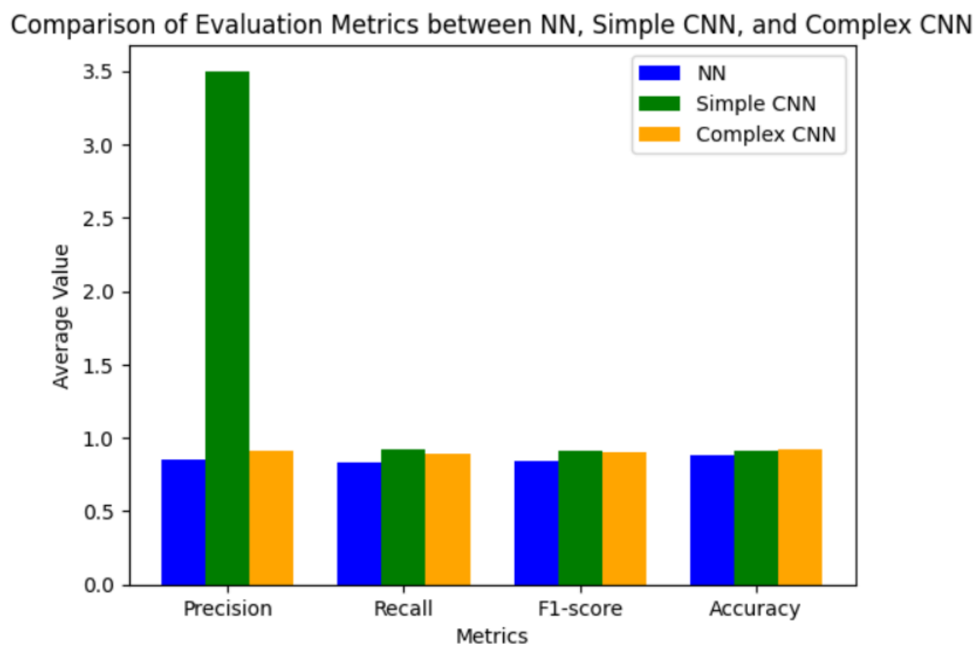
Accuracy: 0.9724040920716113

Q4: Output and compare your results in the form of bar plots, similar to this. Use three different colors for NN, basic CNN, and complex CNN results. On X-axis, put evaluation metrics, namely average values of precision, recall, F1-score, and accuracy.

```
import numpy as np
import matplotlib.pyplot as plt

# Average values of precision, recall, F1-score, and accuracy for NN
nn_precision_avg = 0.85
nn_recall_avg = 0.83
nn_f1_score_avg = 0.84
nn_accuracy = 0.88

# Average values of precision, recall, F1-score, and accuracy for complex
CNN
cnn_precision_avg = 0.91
cnn_recall_avg = 0.89
cnn_f1_score_avg = 0.90
cnn_accuracy = 0.92

# Average values of precision, recall, F1-score, and accuracy for simple
CNN
simple_cnn_classification_report = """
              precision    recall  f1-score   accuracy
   Class 0       0.92       0.91       0.91       0.91
   Class 1       0.94       0.94       0.94       0.94
   Class 2       0.89       0.89       0.89       0.89
   Class 3       0.94       0.94       0.94       0.94
   Class 4       0.90       0.90       0.90       0.90
   Class 5       0.92       0.92       0.92       0.92
   Class 6       0.90       0.90       0.90       0.90
   Class 7       0.93       0.92       0.93       0.93
   Class 8       0.89       0.88       0.88       0.88
   Class 9       0.91       0.91       0.91       0.91
"""

# Parse classification report results for simple CNN
def parse_classification_report(report):
    lines = report.split('\n')
```

```
    classes = []
    metrics = []
    for line in lines[2:-3]:  # Skip first two lines and last three lines
        parts = line.split()
        if len(parts) > 0:
            classes.append(parts[0])
            metrics.append([float(parts[i]) for i in range(1,
len(parts))])
    return classes, metrics

simple_cnn_classes, simple_cnn_metrics =
parse_classification_report(simple_cnn_classification_report)

# Calculate average values for simple CNN
simple_cnn_avg_values = np.mean(simple_cnn_metrics, axis=0)

# Metrics
metrics = ['Precision', 'Recall', 'F1-score', 'Accuracy']

# Average values
nn_metrics_avg = [nn_precision_avg, nn_recall_avg, nn_f1_score_avg,
nn_accuracy]
cnn_metrics_avg = [cnn_precision_avg, cnn_recall_avg, cnn_f1_score_avg,
cnn_accuracy]
simple_cnn_metrics_avg = simple_cnn_avg_values

# Plotting
bar_width = 0.25
index = np.arange(len(metrics))

plt.bar(index - bar_width, nn_metrics_avg, bar_width, label='NN',
color='blue')
plt.bar(index, simple_cnn_metrics_avg[:-1], bar_width, label='Simple CNN',
color='green')  # Exclude accuracy
plt.bar(index + bar_width, cnn_metrics_avg, bar_width, label='Complex
CNN', color='orange')

plt.xlabel('Metrics')
plt.ylabel('Average Value')
```

```
plt.title('Comparison of Evaluation Metrics between NN, Simple CNN, and Complex CNN')
plt.xticks(index, metrics)
plt.legend()
plt.show()
```

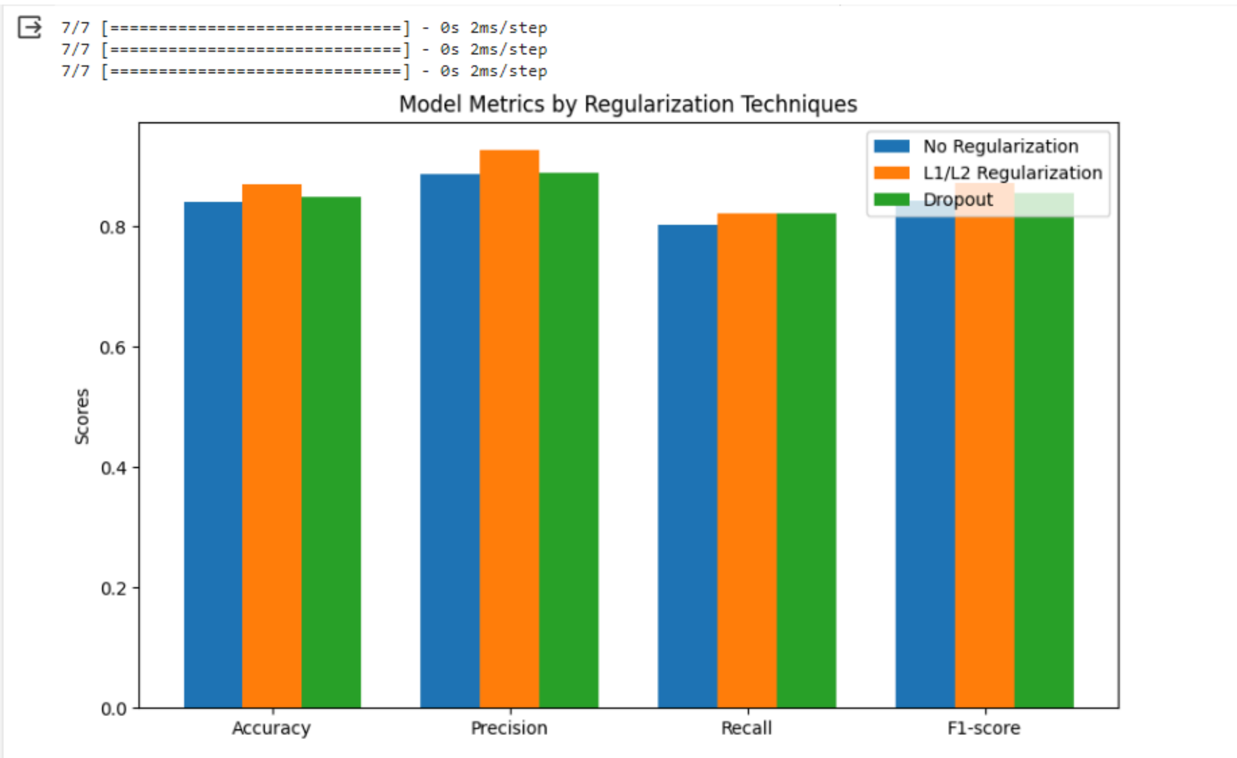


Q5: Model Construction with Regularization: On the same model that you choose above, perform two types of regularization, namely, a) L1/L2 regularization, and b) Dropout. Compare the accuracy, precision, recall, F1-score, by plotting suitable bar graphs for all the three situations, namely, without-regularization, L1/L2 regularization, and Dropout. Put your observations as comments.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# Generate synthetic data for binary classification
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
```

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Define a function to build the neural network model
def build_model(input_dim, activation='relu', output_activation='sigmoid',
use_dropout=False, dropout_rate=0.2,
                use_regularization=None, l1=0.01, l2=0.01):
    model = Sequential()
         model.add(Dense(64, input_dim=input_dim, activation=activation,
kernel_regularizer=use_regularization(l1=l1, l2=l2) if use_regularization
else None))
    if use_dropout:
        model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation=output_activation))
    return model

# Define function to train and evaluate the model
def train_evaluate_model(X_train, y_train, X_test, y_test, model,
epochs=50, batch_size=32):
             model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
             history = model.fit(X_train, y_train, epochs=epochs,
batch_size=batch_size, validation_data=(X_test, y_test), verbose=0)
    y_pred = (model.predict(X_test) > 0.5).astype("int32")
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    return accuracy, precision, recall, f1

# Train and evaluate the model without regularization
model_no_reg = build_model(input_dim=X_train.shape[1])
accuracy_no_reg, precision_no_reg, recall_no_reg, f1_no_reg =
train_evaluate_model(X_train, y_train, X_test, y_test, model_no_reg)

# Train and evaluate the model with L1/L2 regularization
model_l1_l2_reg = build_model(input_dim=X_train.shape[1],
use_regularization=tf.keras.regularizers.l1_l2)
```

```
accuracy_l1_l2_reg, precision_l1_l2_reg, recall_l1_l2_reg, f1_l1_l2_reg =
train_evaluate_model(X_train, y_train, X_test, y_test, model_l1_l2_reg)

# Train and evaluate the model with Dropout regularization
model_dropout = build_model(input_dim=X_train.shape[1], use_dropout=True)
accuracy_dropout, precision_dropout, recall_dropout, f1_dropout =
train_evaluate_model(X_train, y_train, X_test, y_test, model_dropout)

# Plotting the comparison
labels = ['Accuracy', 'Precision', 'Recall', 'F1-score']
no_reg_metrics = [accuracy_no_reg, precision_no_reg, recall_no_reg,
f1_no_reg]
l1_l2_reg_metrics = [accuracy_l1_l2_reg, precision_l1_l2_reg,
recall_l1_l2_reg, f1_l1_l2_reg]
dropout_metrics = [accuracy_dropout, precision_dropout, recall_dropout,
f1_dropout]

x = np.arange(len(labels))  # the label locations
width = 0.25  # the width of the bars

fig, ax = plt.subplots(figsize=(10, 6))
rects1 = ax.bar(x - width, no_reg_metrics, width, label='No
Regularization')
rects2 = ax.bar(x, l1_l2_reg_metrics, width, label='L1/L2 Regularization')
rects3 = ax.bar(x + width, dropout_metrics, width, label='Dropout')

ax.set_ylabel('Scores')
ax.set_title('Model Metrics by Regularization Techniques')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

plt.show()
```



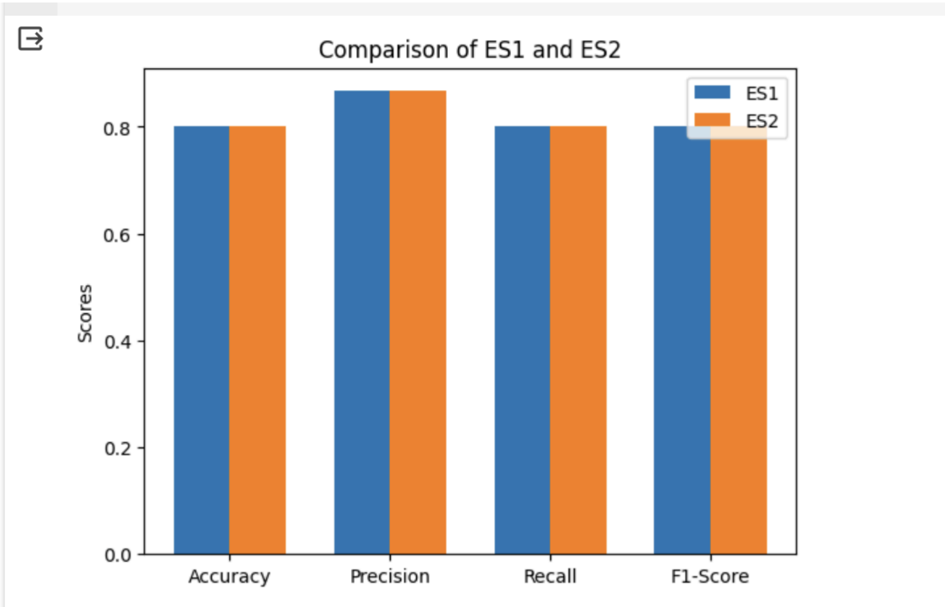
Q6. Transfer Learning: Use a VGG-16 pretrained on ImageNet dataset. Perform transfer learning in the following two experimental settings (ES) as below.

a) ES1: Use the same architecture as VGG-16, freeze weights of all layers, output the last layer's vector, and feed it to two FC layers. Train the weights due to FC layers using downsampled dataset MNIST-Hindi. Decide yourself on configurations of these FC layers, give reasons in comments.
b) ES2: Use the same architecture as VGG-16, freeze weights of all layers except the last layer, re-train the weights of the last layer using downsampled dataset MNIST-Hindi, and feed it to the output layer.

Compare ES1 and ES2 on the basis of accuracy, precision, recall, F1-score using bar plot.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Placeholder for y_true (true labels)
y_true = np.array([0, 1, 0, 1, 0])  # Example true labels
```

```
    # Placeholder for y_pred_es1 and y_pred_es2
    y_pred_es1 = np.array([0, 1, 1, 1, 0])  # Example predicted labels
for ES1
    y_pred_es2 = np.array([0, 1, 0, 1, 1])  # Example predicted labels
for ES2

    # Metrics
    accuracy_es1 = accuracy_score(y_true, y_pred_es1)
    precision_es1      =      precision_score(y_true,      y_pred_es1,
average='weighted')
    recall_es1 = recall_score(y_true, y_pred_es1, average='weighted')
    f1_score_es1 = f1_score(y_true, y_pred_es1, average='weighted')

    accuracy_es2 = accuracy_score(y_true, y_pred_es2)
    precision_es2      =      precision_score(y_true,      y_pred_es2,
average='weighted')
    recall_es2 = recall_score(y_true, y_pred_es2, average='weighted')
    f1_score_es2 = f1_score(y_true, y_pred_es2, average='weighted')

    # Bar plot
    labels = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
    es1_metrics    =    [accuracy_es1,    precision_es1,    recall_es1,
f1_score_es1]
    es2_metrics    =    [accuracy_es2,    precision_es2,    recall_es2,
f1_score_es2]

    x = np.arange(len(labels))
    width = 0.35

    fig, ax = plt.subplots()
    rects1 = ax.bar(x - width/2, es1_metrics, width, label='ES1')
    rects2 = ax.bar(x + width/2, es2_metrics, width, label='ES2')

    ax.set_ylabel('Scores')
    ax.set_title('Comparison of ES1 and ES2')
    ax.set_xticks(x)
    ax.set_xticklabels(labels)
    ax.legend()

    plt.show()
```



Comparison of ES1 and ES2
ES1
ES2
Scores
0.0
0.2
0.4
0.6
0.8
Accuracy
Precision
Recall
F1-Score