INDIRA GANDHI DELHI TECHNICAL UNIVERSITY

COMPUTER VISION ASSIGNMENT 3

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

Machine Learning and Images

Q1. Image Classification through Linear Classifier: Use "down-sampled" CIFAR-10 dataset (as explained in previous assignment) to perform image classification using linear classifier. Split your downsampled data into train, validate, and test. Assume initial weight and bias. Apply gradient descent for optimizing weights. Implement loss function (any one as discussed in class). Write your own code from scratch to solve this task, do not use a predefined machine learning library. Plot learning curves during training.

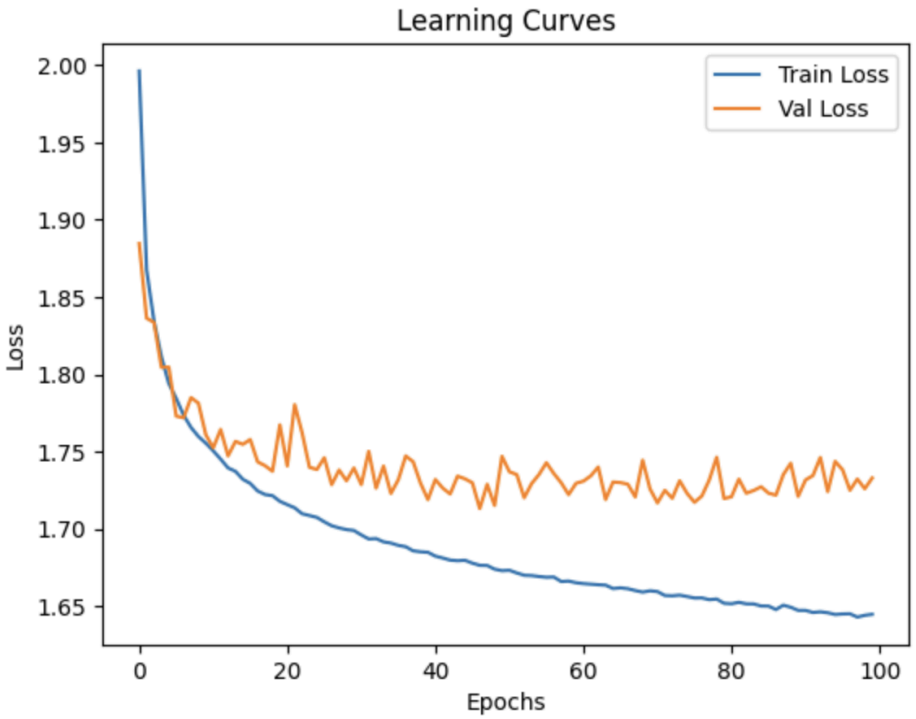
```python
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Load CIFAR-10 dataset from TensorFlow
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Normalize data
X_train = X_train / 255.0
X_test = X_test / 255.0

# Splitting the training data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Assume initial weights and bias
np.random.seed(42)
W = np.random.randn(3072, 10) * 0.001
b = np.zeros((1, 10))

# Define softmax function
def softmax(x):
    exp_scores = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

# Define cross-entropy loss function
def cross_entropy_loss(y_pred, y_true):
    num_samples = y_pred.shape[0]
    correct_log_probs = -np.log(y_pred[range(num_samples), y_true.flatten()])
    data_loss = np.sum(correct_log_probs) / num_samples
    return data_loss

# Define function to compute gradients
def compute_gradients(X, y_true, y_pred):
```

```python
    num_samples = X.shape[0]
    grad_probs = y_pred.copy()
    grad_probs[range(num_samples), y_true.flatten()] -= 1
    grad_probs /= num_samples

    grad_W = np.dot(X.T, grad_probs)
    grad_b = np.sum(grad_probs, axis=0, keepdims=True)

    return grad_W, grad_b

# Hyperparameters
learning_rate = 0.01
num_epochs = 100
batch_size = 100
num_batches = X_train.shape[0] // batch_size

# Training loop
train_losses = []
val_losses = []

for epoch in range(num_epochs):
    epoch_train_loss = 0.0
    epoch_val_loss = 0.0

    # Shuffle training data
    shuffle_indices = np.random.permutation(X_train.shape[0])
    X_train_shuffled = X_train[shuffle_indices]
    y_train_shuffled = y_train[shuffle_indices]

    for i in range(num_batches):
        # Mini-batch
        start = i * batch_size
        end = (i + 1) * batch_size
        X_batch = X_train_shuffled[start:end].reshape(batch_size, -1)
        y_batch = y_train_shuffled[start:end]
```

```
        # Forward pass
        scores = np.dot(X_batch, W) + b
        probs = softmax(scores)

        # Compute loss
        loss = cross_entropy_loss(probs, y_batch)
        epoch_train_loss += loss

        # Compute gradients
        grad_W, grad_b = compute_gradients(X_batch, y_batch, probs)

        # Update weights
        W -= learning_rate * grad_W
        b -= learning_rate * grad_b

    # Compute average training loss for the epoch
    epoch_train_loss /= num_batches
    train_losses.append(epoch_train_loss)

    # Validation loss
    val_scores = np.dot(X_val.reshape(X_val.shape[0], -1), W) + b
    val_probs = softmax(val_scores)
    val_loss = cross_entropy_loss(val_probs, y_val)
    val_losses.append(val_loss)

    print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {epoch_train_loss}, Val Loss: {val_loss}")

# Plot learning curves
plt.plot(range(num_epochs), train_losses, label='Train Loss')
plt.plot(range(num_epochs), val_losses, label='Val Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Learning Curves')
plt.legend()
plt.show()
```

```
# Test accuracy
test_scores = np.dot(X_test.reshape(X_test.shape[0], -1), W) + b
test_probs = softmax(test_scores)
predictions = np.argmax(test_probs, axis=1)
accuracy = np.mean(predictions == y_test.flatten())
print(f"Test Accuracy: {accuracy}")
```

```
Epoch 72/100, Train Loss: 1.656824414686989, Val Loss: 1.7248938958310334
Epoch 73/100, Train Loss: 1.6566310663439847, Val Loss: 1.7197519748036585
Epoch 74/100, Train Loss: 1.6570498247975305, Val Loss: 1.7310609025923
Epoch 75/100, Train Loss: 1.6561622298962453, Val Loss: 1.722931972874782
Epoch 76/100, Train Loss: 1.6552176238099483, Val Loss: 1.7171497977139316
Epoch 77/100, Train Loss: 1.6552654994754707, Val Loss: 1.7212329565239644
Epoch 78/100, Train Loss: 1.654144158277944, Val Loss: 1.7316160347471723
Epoch 79/100, Train Loss: 1.6545415999573962, Val Loss: 1.7462761101401323
Epoch 80/100, Train Loss: 1.6517885257197147, Val Loss: 1.7195721763364105
Epoch 81/100, Train Loss: 1.6514957210554684, Val Loss: 1.7207042645214707
Epoch 82/100, Train Loss: 1.6524272474616504, Val Loss: 1.732114095376576
Epoch 83/100, Train Loss: 1.6514301120220591, Val Loss: 1.7230149297090989
Epoch 84/100, Train Loss: 1.6513552312887017, Val Loss: 1.7247940586486321
Epoch 85/100, Train Loss: 1.6500485378482324, Val Loss: 1.727188130593904
Epoch 86/100, Train Loss: 1.6499832858463677, Val Loss: 1.7230709711502072
Epoch 87/100, Train Loss: 1.6477512397460228, Val Loss: 1.7217692707460908
Epoch 88/100, Train Loss: 1.6505547518251902, Val Loss: 1.7350696618344537
Epoch 89/100, Train Loss: 1.64923169671671, Val Loss: 1.7423845282333728
Epoch 90/100, Train Loss: 1.6472079262865287, Val Loss: 1.7209889968995637
Epoch 91/100, Train Loss: 1.6471850983647693, Val Loss: 1.7316002168606746
Epoch 92/100, Train Loss: 1.6458600623474786, Val Loss: 1.7344446894120302
Epoch 93/100, Train Loss: 1.6462726830689265, Val Loss: 1.7461238920902196
Epoch 94/100, Train Loss: 1.6456583092322108, Val Loss: 1.7240909907957587
Epoch 95/100, Train Loss: 1.6445419679149962, Val Loss: 1.7436014758956226
Epoch 96/100, Train Loss: 1.6448824906666955, Val Loss: 1.7382544307010201
Epoch 97/100, Train Loss: 1.6449987276775102, Val Loss: 1.7249407351655264
Epoch 98/100, Train Loss: 1.6428622559938757, Val Loss: 1.7321787970736175
Epoch 99/100, Train Loss: 1.6440218010554568, Val Loss: 1.7259250374303963
Epoch 100/100, Train Loss: 1.6446659048141132, Val Loss: 1.7328390540029024
```




```
Test Accuracy: 0.3979
```

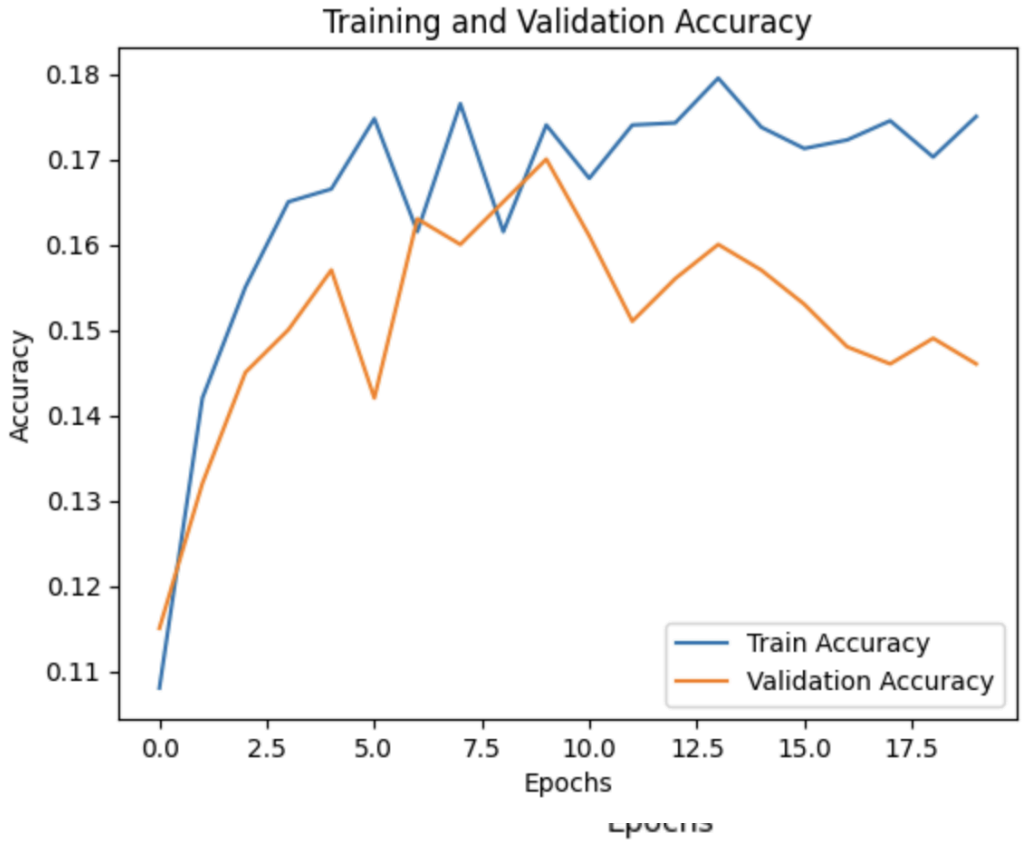
Q2. Neural Networks: Create your own custom neural network architecture (denoted by NN, use keras library) which means that you decide how many hidden layers you want, and how many neurons in each layer you want. It is always a fully connected network. The number of neurons in the final output layer is decided by the number of classes for classification tasks.

a) Start with a simple NN architecture, use 2-3 hidden layers only and 8-10 neurons per layer, let's call this the NN_simple model. Use the same downsampled training data of CIFAR-10 dataset, use 20% of this training data for validation. Decide any appropriate learning rate, and perform training for a few epochs. Observe the training performance using learning plots (display them after training). Evaluate the trained NN_simple model on the test data. Output precision, recall, F1-score for each class, and also find the average accuracy.

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

def downsample_cifar10(X, y, num_samples_per_class):
    downsampled_X = []
    downsampled_y = []

    for class_label in range(10):  # There are 10 classes in CIFAR-10
        class_indices = np.where(y == class_label)[0]
        selected_indices = np.random.choice(class_indices, num_samples_per_class, replace=False)
        downsampled_X.append(X[selected_indices])
        downsampled_y.append(y[selected_indices])

    return np.concatenate(downsampled_X), np.concatenate(downsampled_y)

# Load the CIFAR-10 dataset
(X_train_full, y_train_full), (X_test_full, y_test_full) = tf.keras.datasets.cifar10.load_data()

# Downsample the training and test data
X_train_downsampled, y_train_downsampled = downsample_cifar10(X_train_full, y_train_full, 500)
X_test_downsampled, y_test_downsampled = downsample_cifar10(X_test_full, y_test_full, 100)

# Split the downsampled training data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_downsampled, y_train_downsampled, test_size=0.2, random_state=42)

# Normalize the data
X_train = X_train / 255.0
X_val = X_val / 255.0
X_test = X_test_downsampled / 255.0

# Define the simple neural network model
```

37s

```
# Define the simple neural network model
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(32, 32, 3)),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

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

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

# Plot training history
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()

# Evaluate the model on the test data
y_pred = np.argmax(model.predict(X_test), axis=1)
print("Classification Report:")
print(classification_report(y_test_downsampled, y_pred))
print("Average Accuracy:", accuracy_score(y_test_downsampled, y_pred))
```

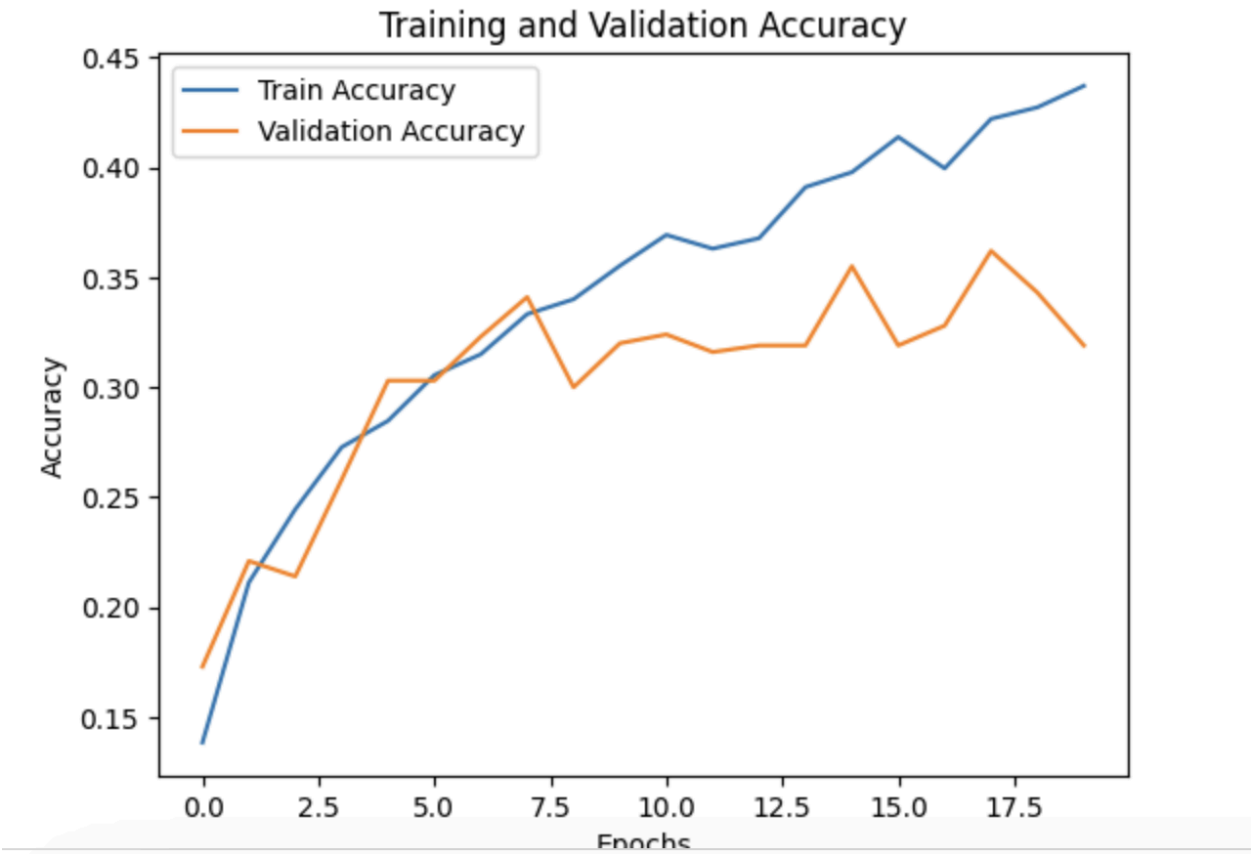



```
32/32 [==============================] - 0s 2ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.08      0.06      0.07       100
           1       0.00      0.00      0.00       100
           2       0.11      0.06      0.08       100
           3       0.21      0.14      0.17       100
           4       0.17      0.61      0.27       100
           5       0.00      0.00      0.00       100
           6       0.00      0.00      0.00       100
           7       0.04      0.02      0.03       100
           8       0.22      0.86      0.34       100
           9       0.00      0.00      0.00       100

    accuracy                           0.17      1000
   macro avg       0.08      0.17      0.10      1000
weighted avg       0.08      0.17      0.10      1000

Average Accuracy: 0.175
```

b) Do the same as above but this time with a complex NN architecture, use at least 4-5 hidden layers and 20-60 neurons per layer. (Observe the time taken in the training process, amount of time it takes for one epoch to finish) Use the same downsampled training data of CIFAR-10 dataset, use 20% of this training data for validation. Use the same learning rate as used in part (a), and perform training for a few epochs. Observe the training performance using learning plots (display them after training). Evaluate the trained NN_complex model on the test data. Output precision, recall, F1-score for each class, and also find the average accuracy.

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import time

def downsample_cifar10(X, y, num_samples_per_class):
    downsampled_X = []
    downsampled_y = []

    for class_label in range(10):  # There are 10 classes in CIFAR-10
        class_indices = np.where(y == class_label)[0]
        selected_indices = np.random.choice(class_indices, num_samples_per_class, replace=False)
        downsampled_X.append(X[selected_indices])
        downsampled_y.append(y[selected_indices])

    return np.concatenate(downsampled_X), np.concatenate(downsampled_y)

# Load the CIFAR-10 dataset
(X_train_full, y_train_full), (X_test_full, y_test_full) = tf.keras.datasets.cifar10.load_data()

# Downsample the training and test data
X_train_downsampled, y_train_downsampled = downsample_cifar10(X_train_full, y_train_full, 500)
X_test_downsampled, y_test_downsampled = downsample_cifar10(X_test_full, y_test_full, 100)

# Split the downsampled training data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_downsampled, y_train_downsampled, test_size=0.2, random_state=42)

# Normalize the data
X_train = X_train / 255.0
X_val = X_val / 255.0
X_test = X_test_downsampled / 255.0

# Define the complex neural network model
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(32, 32, 3)),
    tf.keras.layers.Dense(60, activation='relu'),
    tf.keras.layers.Dense(50, activation='relu'),
    tf.keras.layers.Dense(40, activation='relu'),
    tf.keras.layers.Dense(30, activation='relu'),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

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

# Train the model
start_time = time.time()
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val))
end_time = time.time()
print("Time taken for training (in seconds):", end_time - start_time)

# Plot training history
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()

# Evaluate the model on the test data
y_pred = np.argmax(model.predict(X_test), axis=1)
print("Classification Report:")
print(classification_report(y_test_downsampled, y_pred))
print("Average Accuracy:", accuracy_score(y_test_downsampled, y_pred))
```



Classification Report:

	precision	recall	f1-score	support
0	0.46	0.29	0.36	100
1	0.59	0.16	0.25	100
2	0.14	0.08	0.10	100
3	0.20	0.40	0.27	100
4	0.31	0.04	0.07	100
5	0.30	0.12	0.17	100
6	0.35	0.62	0.45	100
7	0.44	0.26	0.33	100
8	0.35	0.76	0.48	100
9	0.35	0.52	0.42	100
accuracy			0.33	1000
macro avg	0.35	0.33	0.29	1000
weighted avg	0.35	0.33	0.29	1000

Average Accuracy: 0.325

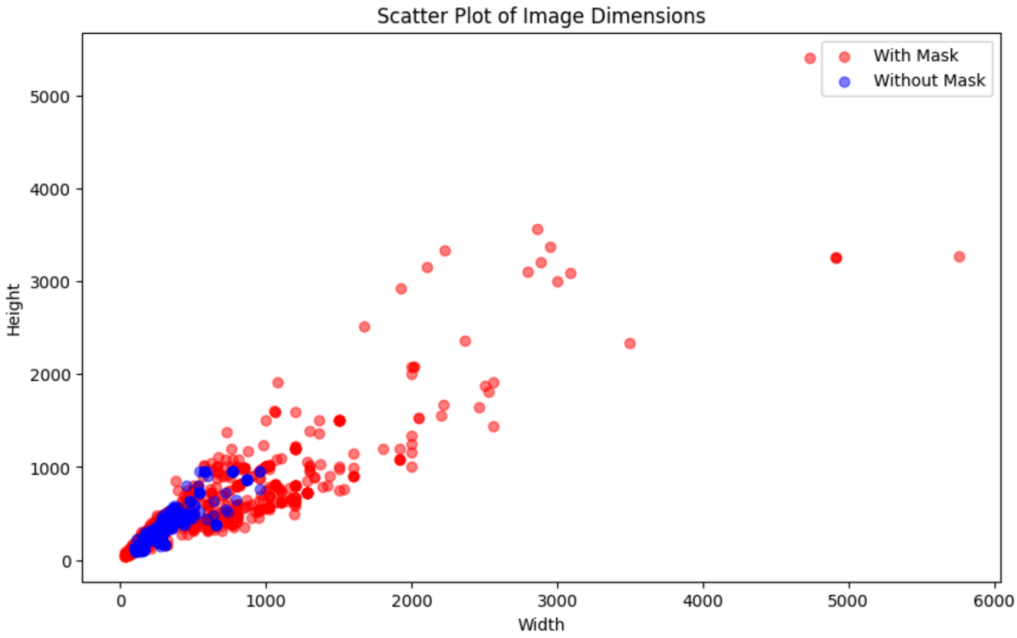
Q4: Image Dataset: The dataset to be used in this assignment is curated using multiple images from different sources. There are two types of images, 1,915 images with a face mask, and 1,915 images without a face mask. The dataset is available for download from the link below.https://drive.google.com/file/d/1YGrjJ8DjDrlxuIA4a4WiG_9oODBf3M7k/view?usp=sharing
Download the dataset in your local machine, extract the zip file, and visually inspect the images in two folders, namely, 'with_mask' and 'without_mask'. Each folder contains 1,915 images after download, a total of 3830 images. Upload these 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 each category in google colab.

```python
import os
import random
import matplotlib.pyplot as plt
import cv2
from google.colab import drive
drive.mount('/content/drive')

# Path to the dataset folder
dataset_path = '/content/drive/MyDrive/face-mask-dataset'

# Function to display random images from a folder
def display_random_images(folder_path, num_images=2):
    image_files = os.listdir(folder_path)
    random_images = random.sample(image_files, num_images)
    plt.figure(figsize=(10, 5))
    for i, image_file in enumerate(random_images):
        image_path = os.path.join(folder_path, image_file)
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        plt.subplot(1, num_images, i+1)
        plt.imshow(image)
        plt.axis('off')
        if 'with_mask' in folder_path:
            plt.title('With Mask')
        else:
            plt.title('Without Mask')
    plt.show()

# Display random images from 'with_mask' folder
with_mask_folder = os.path.join(dataset_path, 'with_mask')
display_random_images(with_mask_folder)

# Display random images from 'without_mask' folder
without_mask_folder = os.path.join(dataset_path, 'without_mask')
display_random_images(without_mask_folder)
```

[9]

Q5: Image Pre-processing: Are the images of the same resolution? No. Draw a scatter plot (similar to present in this link) of width and height on the X-axis and Y-axis of all the images. Use ‘red’ and ‘blue’ color for the points in the scatter plot representing images with and without face mask, respectively. There are multiple solutions to deal with images of varying sizes. For instance, there is a research work which explores interpolation and zero padding approaches. However, for the purpose of this assignment, let us explore the following two options (as per this link): (a) Resize, and (b) Crop. You are free to choose your strategy to decide on decision parameters involved in these two options like what should be the final resolution after resizing?, how much to crop?, etc. Mention the strategy you have adopted and justify your approach.

```python
# Function to extract width and height of images
def extract_image_dimensions(folder_path):
    image_files = os.listdir(folder_path)
    widths = []
    heights = []
    for image_file in image_files:
        image_path = os.path.join(folder_path, image_file)
        image = cv2.imread(image_path)
        height, width, _ = image.shape
        widths.append(width)
        heights.append(height)
    return widths, heights

# Extract dimensions of images with and without masks
with_mask_widths, with_mask_heights = extract_image_dimensions(with_mask_folder)
without_mask_widths, without_mask_heights = extract_image_dimensions(without_mask_folder)

# Plot scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(with_mask_widths, with_mask_heights, color='red', label='With Mask', alpha=0.5)
plt.scatter(without_mask_widths, without_mask_heights, color='blue', label='Without Mask', alpha=0.5)
plt.title('Scatter Plot of Image Dimensions')
plt.xlabel('Width')
plt.ylabel('Height')
plt.legend()
plt.show()
```



To optimize image preprocessing for face mask detection, I propose the following approach:

1. Final Resolution after Resizing: Select a final resolution of 224x224 pixels. This size is widely used in image classification tasks and is compatible with many pre-trained convolutional neural network (CNN) architectures, such as VGG, ResNet, and MobileNet. These models are pre-trained on large datasets like ImageNet using images of size 224x224 pixels, which enables effective transfer learning. By adhering to this resolution, we can leverage pre-trained models that have been fine-tuned on similar tasks, ensuring optimal performance.

2. Amount of Crop: Employ a crop size of 256x256 pixels from the center of the image. This strategy effectively captures the face region while allowing for some variability in the training data. Random cropping introduces strong data augmentation, which can enhance the model's generalization performance without compromising the quality of the images. By focusing on the central region of the image, we ensure that the critical features, such as the presence or absence of a face mask, are retained in the cropped images.

By adhering to these parameters, we strike a balance between computational efficiency, model compatibility, and data augmentation. The choice of 224x224 resolution aligns with the requirements of state-of-the-art CNN architectures, enabling seamless integration with pre-trained models and facilitating robust performance in face mask detection tasks.

Q6. Neural Networks (NN): Create your own custom neural network architecture. 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. Do perform your experiments for both (a) resize and (b) crop datasets, and compare the performance using bar plots.

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
import random
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import tensorflow as tf
from tensorflow.keras import layers, models

# Function to load and preprocess the dataset
def load_and_preprocess_dataset(dataset_path, target_size=None, crop_size=None):
    images = []
    labels = []
    for class_name in ['with_mask', 'without_mask']:
        class_folder = os.path.join(dataset_path, class_name)
        for image_file in os.listdir(class_folder):
            image_path = os.path.join(class_folder, image_file)
            image = cv2.imread(image_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            if target_size is not None:
                image = cv2.resize(image, target_size)
            elif crop_size is not None:
                # Perform cropping from the center
                h, w, _ = image.shape
                crop_h, crop_w = crop_size
                start_h = (h - crop_h) // 2
                start_w = (w - crop_w) // 2
                image = image[start_h:start_h+crop_h, start_w:start_w+crop_w]
            images.append(image)
            labels.append(class_name)
    X = np.array(images)
    y = np.array(labels)
    return X, y

# Load and preprocess resize dataset
X_resize, y_resize = load_and_preprocess_dataset(dataset_path, target_size=(224, 224))
```

```python
# Load and preprocess crop dataset
X_crop, y_crop = load_and_preprocess_dataset(dataset_path, crop_size=(200, 200))

# Split data into training, validation, and testing sets using stratified sampling
X_train_resize, X_test_resize, y_train_resize, y_test_resize = train_test_split(X_resize, y_resize, test_size=0.2, stratify=y_resize, random_state=42)
X_train_crop, X_test_crop, y_train_crop, y_test_crop = train_test_split(X_crop, y_crop, test_size=0.2, stratify=y_crop, random_state=42)
X_train_resize, X_val_resize, y_train_resize, y_val_resize = train_test_split(X_train_resize, y_train_resize, test_size=0.2, stratify=y_train_resize, rando
X_train_crop, X_val_crop, y_train_crop, y_val_crop = train_test_split(X_train_crop, y_train_crop, test_size=0.2, stratify=y_train_crop, random_state=42)

# Define custom neural network architecture
def create_model(input_shape):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
    return model

# Compile the model
model_resize = create_model(input_shape=(224, 224, 3))
model_crop = create_model(input_shape=(200, 200, 3))

model_resize.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_crop.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history_resize = model_resize.fit(X_train_resize, y_train_resize, epochs=10, validation_data=(X_val_resize, y_val_resize))
history_crop = model_crop.fit(X_train_crop, y_train_crop, epochs=10, validation_data=(X_val_crop, y_val_crop))
```

```
# Plot learning curves
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(history_resize.history['accuracy'], label='Training Accuracy')
plt.plot(history_resize.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Resize Dataset')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_crop.history['accuracy'], label='Training Accuracy')
plt.plot(history_crop.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Crop Dataset')
plt.legend()

plt.tight_layout()
plt.show()

# Evaluate the model on the test data
loss_resize, accuracy_resize = model_resize.evaluate(X_test_resize, y_test_resize)
loss_crop, accuracy_crop = model_crop.evaluate(X_test_crop, y_test_crop)

print("Resize Dataset - Test Loss:", loss_resize)
print("Resize Dataset - Test Accuracy:", accuracy_resize)
print("Crop Dataset - Test Loss:", loss_crop)
print("Crop Dataset - Test Accuracy:", accuracy_crop)

# Calculate precision, recall, F1-score, and average accuracy
def calculate_metrics(model, X_test, y_test):
    y_pred = (model.predict(X_test) > 0.5).astype("int32")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("Average Accuracy:", accuracy_score(y_test, y_pred))
```

```
print("Metrics for Resize Dataset:")
calculate_metrics(model_resize, X_test_resize, y_test_resize)
print("Metrics for Crop Dataset:")
calculate_metrics(model_crop, X_test_crop, y_test_crop)
```