DEEP LEARNING

ASSIGNMENT-2

1.Your CNN model for classifying animals in the wild achieves only 70% accuracy. What techniques would you apply to improve the model’s performance?

AIM:

The aim of this project is to **classify images** from the **CIFAR-10 dataset** using a **Convolutional Neural Network (CNN)** with **transfer learning**. Specifically, the pre-trained **ResNet50 model** will be used to extract features, and the model will be fine-tuned for the classification task using the CIFAR-10 dataset, which consists of 60,000 images in 10 classes. The primary goal is to achieve high classification accuracy using data augmentation and regularization techniques.

PROCEDURE:

1. Setting Up the Environment:

* Google Collab will be used as the environment for training the model, as it provides free access to powerful GPUs.
* Install TensorFlow: TensorFlow is the framework used for training the CNN model, which includes tools like Image Data Generator for data augmentation and pre-trained models like ResNet50.
* **2. Import Necessary Libraries:**
* Import all the required libraries for loading the dataset, data augmentation, model building, and visualization.

#### 3. ****Load the CIFAR-10 Dataset:****

The CIFAR-10 dataset is available as part of TensorFlow/Keras. We can load it directly into our environment.

* This dataset is split into:
* **x\_train** and **x\_val**: Image data (training and validation).
* **y\_train** and **y\_val**: Labels for each image (0-9, representing 10 different classes).

**4. Preprocessing the Data:**

* Normalize the image data to a range between 0 and 1.
* Convert the labels to one-hot encoding for multi-class classification.

#### 5. ****Data Augmentation:****

To improve model generalization and avoid overfitting, apply **data augmentation** to the training set (rotations, shifts, zooms, and flips).

**6. Model Building Using Transfer Learning (ResNet50):**

We use the **ResNet50 model** pre-trained on ImageNet as the base, and remove its top layers (classification layers). We then add new layers on top to perform classification specific to CIFAR-10 classes.

* Freeze the layers of the ResNet50 model to retain the pre-trained features.
* Add a **global average pooling layer**, a fully connected layer, a **dropout layer** for regularization, and an output layer with 10 classes for classification.

**7. Callbacks for Training:**

* **Learning Rate Scheduler**: Gradually decay the learning rate after certain epochs to help the model converge.
* **Early Stopping**: Stop training early if validation performance stops improving to avoid overfitting.

#### 8. ****Model Training:****

Train the model on the CIFAR-10 training data while validating on the validation set. Use the defined **callbacks** to adjust the learning rate and stop early if necessary.

**9. Model Evaluation:**

After training, evaluate the model on the validation data to check its performance.

**10. Plot Training and Validation Metrics:**

To visualize how well the model is performing, plot the **accuracy** and **loss** during training and validation.

CODING:

!pip install tensorflow

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import ResNet50

from tensorflow.keras import layers, models, regularizers

from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping

import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_val, y\_val) = tf.keras.datasets.cifar10.load\_data()

x\_train, x\_val = x\_train / 255.0, x\_val / 255.0

y\_train = tf.keras.utils.to\_categorical(y\_train, 10)

y\_val = tf.keras.utils.to\_categorical(y\_val, 10)

train\_datagen = ImageDataGenerator(

    rotation\_range=40,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    shear\_range=0.2,

    zoom\_range=0.2,

    horizontal\_flip=True,

    fill\_mode='nearest'

)

val\_datagen = ImageDataGenerator(rescale=1./255)

base\_model = ResNet50(weights='imagenet', include\_top=False, input\_shape=(32, 32, 3))

base\_model.trainable = False

model = models.Sequential([

    base\_model,

    layers.GlobalAveragePooling2D(),

    layers.Dense(512, activation='relu', kernel\_regularizer=regularizers.l2(0.01)),

    layers.Dropout(0.5),

    layers.Dense(10, activation='softmax')  # Output layer for 10 classes

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

def lr\_scheduler(epoch, lr):

    if epoch < 5:

        return lr

    else:

        return lr \* tf.math.exp(-0.1)

callbacks = [

    LearningRateScheduler(lr\_scheduler),

    EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

]

history = model.fit(

    x\_train, y\_train,

    epochs=5,

    batch\_size=64,

    validation\_data=(x\_val, y\_val),

    callbacks=callbacks

)

score = model.evaluate(x\_val, y\_val, verbose=1)

print(f"Test loss: {score[0]}")

print(f"Test accuracy: {score[1]}")

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Model Loss')

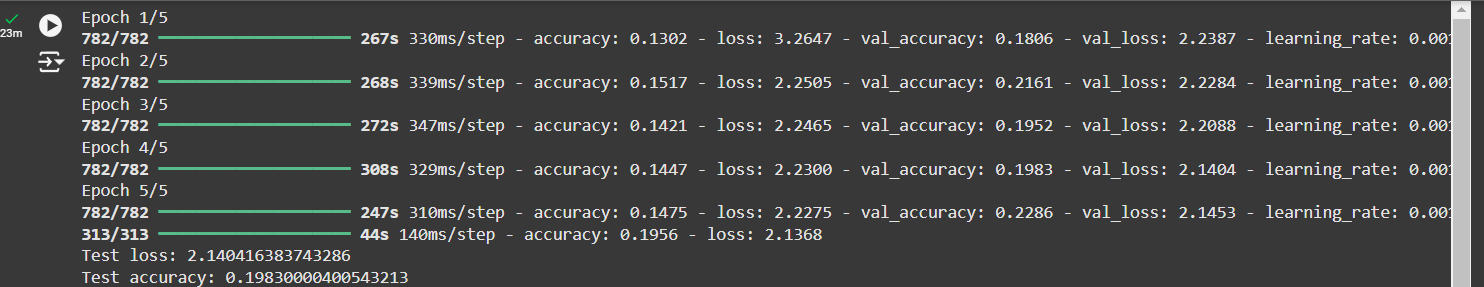
plt.xlabel('Epochs')

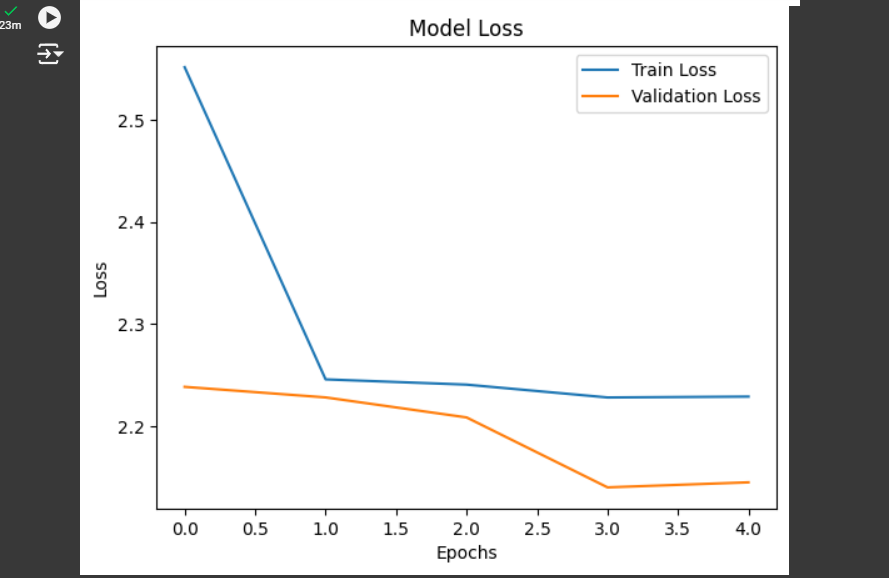
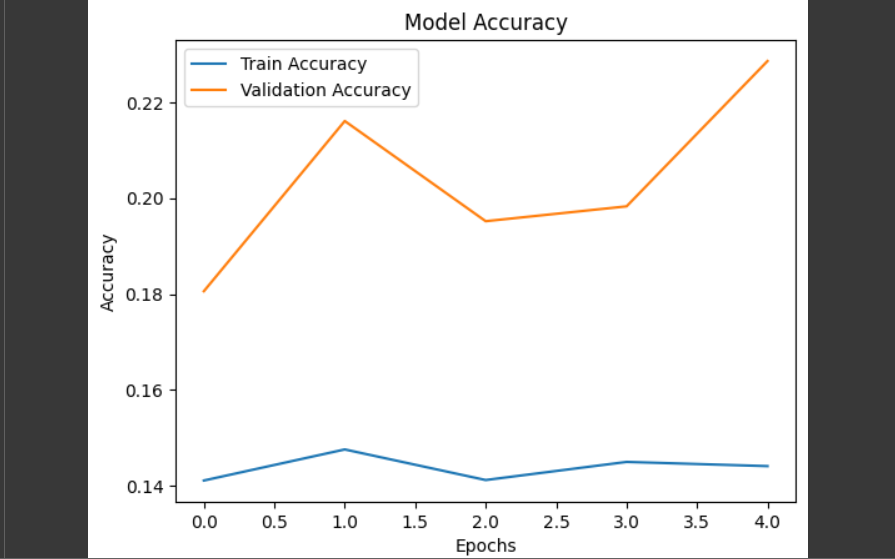
plt.ylabel('Loss')

plt.legend()

plt.show()

OUTPUT:





RESULT:

Thus the code was exectued successfully.