Q3. Train a convolutional neural network (CNN) on the CIFAR-10 dataset with and without batch normalization layers. Analyze and compare the convergence behavior and final accuracy of both models. Present your findings with appropriate visualizations and interpretations.

#CNN Model with Batch Normalization

import tensorflow as tf

from tensorflow.keras import Sequential

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

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

import numpy as np

import os

def create\_cnn\_model(input\_shape=(32, 32, 3), num\_classes=10):

    model = Sequential()

    model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=input\_shape))

    model.add(BatchNormalization())

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(64, (3, 3), activation='relu'))

    model.add(BatchNormalization())

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(128, (3, 3), activation='relu'))

    model.add(BatchNormalization())

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Flatten())

    model.add(Dense(256, activation='relu'))

    model.add(Dropout(0.5))

    model.add(Dense(num\_classes, activation='softmax'))

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

    return model

if \_\_name\_\_ == "\_\_main\_\_":

    (x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

    x\_train = x\_train.astype('float32') / 255.0

    x\_test = x\_test.astype('float32') / 255.0

    y\_train = to\_categorical(y\_train, 10)

    y\_test = to\_categorical(y\_test, 10)

    model = create\_cnn\_model(input\_shape=(32, 32, 3), num\_classes=10)

    model.summary()

    history = model.fit(x\_train, y\_train, batch\_size=32, epochs=50, validation\_data=(x\_test, y\_test))

import matplotlib.pyplot as plt

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

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

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Training and Validation Accuracy')

plt.show()

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

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

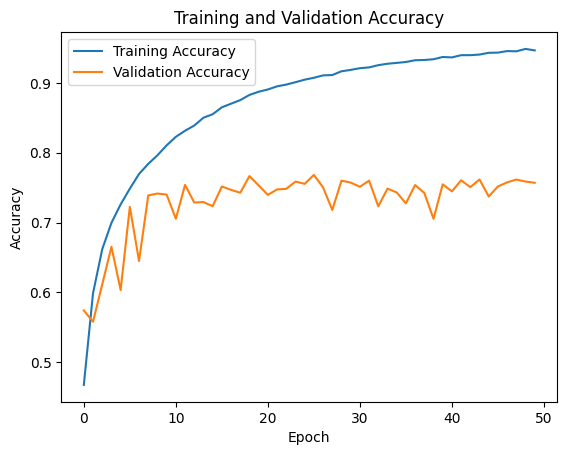
plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.title('Training and Validation Loss')

plt.show()

Output(Graphs Visualizations):-

#CNN model without Batch Normalization

import tensorflow as tf

from tensorflow.keras import Sequential

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

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

import numpy as np

import os

def create\_cnn\_model(input\_shape=(32, 32, 3), num\_classes=10):

    model = Sequential()

    model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=input\_shape))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(64, (3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Conv2D(128, (3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Flatten())

    model.add(Dense(256, activation='relu'))

    model.add(Dropout(0.5))

    model.add(Dense(num\_classes, activation='softmax'))

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

    return model

if \_\_name\_\_ == "\_\_main\_\_":

    (x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

    x\_train = x\_train.astype('float32') / 255.0

    x\_test = x\_test.astype('float32') / 255.0

    y\_train = to\_categorical(y\_train, 10)

    y\_test = to\_categorical(y\_test, 10)

    model = create\_cnn\_model(input\_shape=(32, 32, 3), num\_classes=10)

    model.summary()

    history = model.fit(x\_train, y\_train, batch\_size=32, epochs=50, validation\_data=(x\_test, y\_test))

import matplotlib.pyplot as plt

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

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

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Training and Validation Accuracy')

plt.show()

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

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

plt.xlabel('Epoch')

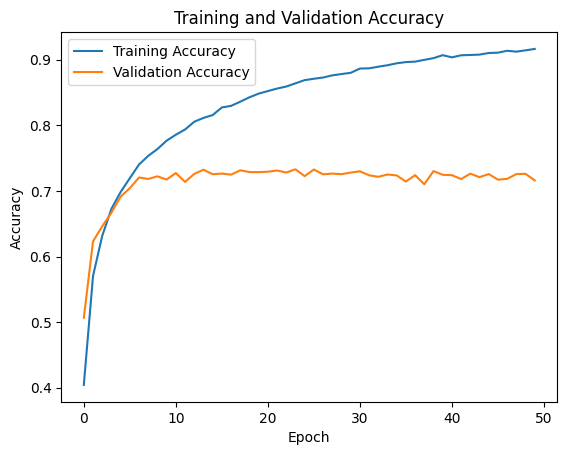
plt.ylabel('Loss')

plt.legend()

plt.title('Training and Validation Loss')

plt.show()

Output(Graph Visualization):-



Q5. Utilize a pre-trained deep learning model (e.g., VGG16 or ResNet) for classifying a custom image dataset such as flowers or animals. Compare the performance between fixed feature extraction and fine-tuning. Present your evaluation results and explain the observed differences.

# Fixed Feature Extraction

import tensorflow as tf

from tensorflow.keras.applications import VGG16

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import Adam

import matplotlib.pyplot as plt

def create\_fixed\_model(input\_shape=(224, 224, 3), num\_classes=2):

    base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    base\_model.trainable = False

    model = Sequential([

        base\_model,

        Conv2D(128, (3, 3), activation='relu', padding='same'),

        BatchNormalization(),

        MaxPooling2D(pool\_size=(2, 2)),

        Flatten(),

        Dense(256, activation='relu'),

        Dropout(0.5),

        Dense(num\_classes, activation='softmax')

    ])

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

    return model

if \_\_name\_\_ == "\_\_main\_\_":

    train\_path = 'animal/train'

    test\_path = 'animal/test'

    train\_datagen = ImageDataGenerator(rescale=1./255, validation\_split=0.2)

    train\_generator = train\_datagen.flow\_from\_directory(

        train\_path, target\_size=(224, 224), batch\_size=32,

        class\_mode='categorical', subset='training'

    )

    val\_generator = train\_datagen.flow\_from\_directory(

        train\_path, target\_size=(224, 224), batch\_size=32,

        class\_mode='categorical', subset='validation'

    )

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

    test\_generator = test\_datagen.flow\_from\_directory(

        test\_path, target\_size=(224, 224), batch\_size=32,

        class\_mode='categorical'

    )

    model = create\_fixed\_model()

    history = model.fit(train\_generator, validation\_data=val\_generator, epochs=10)

    loss, acc = model.evaluate(test\_generator)

    print(f"Test Accuracy (Fixed): {acc:.4f}")

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

plt.title('Validation Accuracy (Fixed Feature)')

plt.xlabel('Epochs')

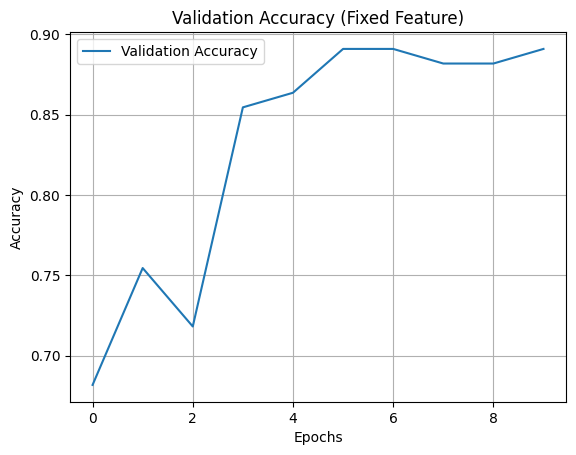
plt.ylabel('Accuracy')

plt.legend()

plt.grid(True)

plt.show()

Graph(Output):-



# Fine Tuning

import tensorflow as tf

from tensorflow.keras.applications import VGG16

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import Adam

import matplotlib.pyplot as plt

def create\_finetune\_model(input\_shape=(224, 224, 3), num\_classes=2):

    base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    for layer in base\_model.layers[:-4]:

        layer.trainable = False

    for layer in base\_model.layers[-4:]:

        layer.trainable = True

    model = Sequential([

        base\_model,

        Conv2D(128, (3, 3), activation='relu', padding='same'),

        BatchNormalization(),

        MaxPooling2D(pool\_size=(2, 2)),

        Flatten(),

        Dense(256, activation='relu'),

        Dropout(0.5),

        Dense(num\_classes, activation='softmax')

    ])

    model.compile(optimizer=Adam(learning\_rate=1e-4), loss='categorical\_crossentropy', metrics=['accuracy'])

    return model

if \_\_name\_\_=="\_\_main\_\_":

    train\_path = 'animal/train'

    test\_path = 'animal/test'

    train\_datagen = ImageDataGenerator(rescale=1./255, validation\_split=0.2)

    train\_generator = train\_datagen.flow\_from\_directory(

        train\_path, target\_size=(224, 224), batch\_size=32,

        class\_mode='categorical', subset='training'

    )

    val\_generator = train\_datagen.flow\_from\_directory(

        train\_path, target\_size=(224, 224), batch\_size=32,

        class\_mode='categorical', subset='validation'

    )

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

    test\_generator = test\_datagen.flow\_from\_directory(

        test\_path, target\_size=(224, 224), batch\_size=32,

        class\_mode='categorical'

    )

    model = create\_finetune\_model()

    history = model.fit(train\_generator, validation\_data=val\_generator, epochs=10)

    loss, acc = model.evaluate(test\_generator)

    print(f"Test Accuracy (Fine-Tuned): {acc:.4f}")

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

plt.title('Validation Accuracy (Fine-Tuning)')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.grid(True)

plt.show()

Graph(Output):-

