Final Project

**Task 1: Object-centric Image recognition task (Cifar10, Caltech101, Caltech256)**

## Course: Deep Learning (COMP 5413) Faculty: Yimin yang

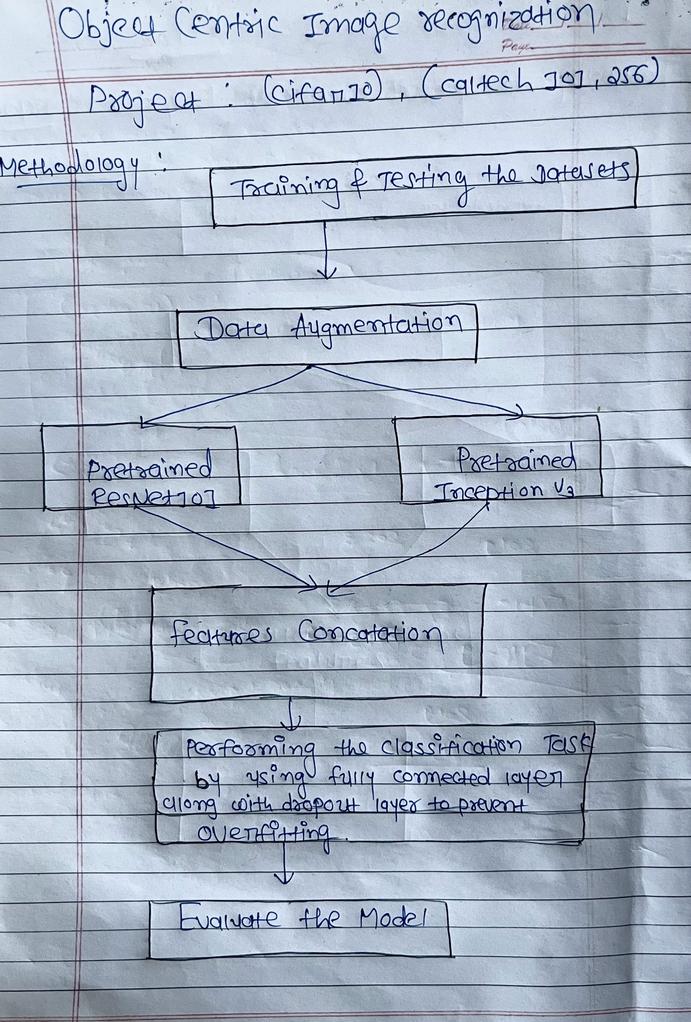
Department: Msc in CS

## Group members:

**Jay Chovatiya**(1161980)

**Rahul Kukadiya** (1193621)

**Methodology**:



* **We have followed the given pipeline steps:**
* Discussing about the Image Data generation in which the pipeline starts with image data generation, which is accomplished with Keras Image data generator class. Transformations such as rotation, shift, shear, zoom, and horizontal flip are used to augment the training dataset.
* In the following stage we have done feature extraction in which two pre-trained models (ResNet101,InceptionV3)heterogeneous features are used to extract features from the augmented data during the feature extraction phase. The features are then concatenated and fed into a fully connected layer for classification.
* In addition, to create a single feature map, the extracted features are concatenated along the third axis (depth) with np.concatenate(). The resulting features are reshaped to 64x64x1 and saved in x\_train and x\_test, which will later be used as input to the following task .
* During Classification task, a fully connected layer is used, which takes the features extracted in the previous step and classifies the images into one of the classes.Apart from that we used dropout layer also in order to prevent overfitting.
* In the last stage we performed Model Evaluation .The model's accuracy and loss are determined using a separate testing dataset.

**Implementation:**

**Caltech101.py**

# importing all the libraries

from keras.applications.vgg16 import VGG16

from tensorflow.keras.applications.resnet50 import ResNet50, preprocess\_input

from keras.layers import Input, Flatten, Dense, Concatenate,Dropout

from keras.models import Model

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

from keras.models import Model

from keras.models import Sequential

from keras.optimizers import Adam

from sklearn.utils import shuffle

import os

import random

import shutil

import time

from PIL import Image

from keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.preprocessing import image

from keras import optimizers

from keras import backend

import warnings

warnings.filterwarnings("ignore")

"""# This is utility functions for graph related operations"""

def plot\_graph(history) :

    plt.plot(history.history['accuracy'])

    plt.plot(history.history['val\_accuracy'])

    plt.title('model accuracy')

    plt.ylabel('accuracy')

    plt.xlabel('epoch')

    plt.legend(['train', 'val'], loc='upper left')

    plt.show()

   plt.plot(history.history['loss'])

    plt.plot(history.history['val\_loss'])

    plt.title('model loss')

    plt.ylabel('loss')

    plt.xlabel('epoch')

    plt.legend(['train', 'val'], loc='upper left')

    plt.show()

"""# This is utility functions for time related operations"""

def print\_time(start\_time, end\_time, description) :

    total\_time\_in\_seconds = end\_time - start\_time

    total\_time\_in\_minutes = total\_time\_in\_seconds / 60

    print(description.format(total\_time\_in\_minutes))

"""# This is utility functions for file and dataset related operations"""

def get\_save\_path(filename, dataset\_save\_folder = '/kaggle/working/') :

    return os.path.join(dataset\_save\_folder, filename)

def create\_dir(path\_dir) :

    if not os.path.exists(path\_dir) :

        os.makedirs(path\_dir)

def generate\_dataset(src\_dir, dest\_dir, image\_set\_number, validation\_image\_number = 24) :

    train\_dir = os.path.join(dest\_dir, 'train')

    test\_dir = os.path.join(dest\_dir, 'test')

    validation\_dir = os.path.join(dest\_dir, 'validation')

   create\_dir(train\_dir)

    create\_dir(test\_dir)

    create\_dir(validation\_dir)

    categories = os.listdir(src\_dir)

    for category in categories:

        category\_path = os.path.join(src\_dir, category)

        category\_train\_path = os.path.join(train\_dir, category)

        category\_test\_path = os.path.join(test\_dir, category)

        category\_validation\_path = os.path.join(validation\_dir, category)

         create\_dir(category\_train\_path)

        create\_dir(category\_test\_path)

        create\_dir(category\_validation\_path)

         if os.path.isdir(category\_path):

            image\_files = os.listdir(category\_path)

            random.shuffle(image\_files)

            train\_images = image\_files[:image\_set\_number]

            test\_images = image\_files[image\_set\_number:]

             # Create test folder

            for filename in test\_images:

                src = os.path.join(category\_path, filename)

                dst = os.path.join(category\_test\_path, filename)

                shutil.copy(src, dst)

           train\_set\_images = train\_images[:validation\_image\_number]

            validation\_images = train\_images[validation\_image\_number:]

            # Create validation folder

            for filename in validation\_images:

                src = os.path.join(category\_path, filename)

                dst = os.path.join(category\_validation\_path, filename)

                shutil.copy(src, dst)

            # Create train folder

            for filename in train\_set\_images:

                src = os.path.join(category\_path, filename)

                dst = os.path.join(category\_train\_path, filename)

                shutil.copy(src, dst)

"""# Clasification Model

def execute\_model(training\_data\_set\_folder, validation\_data\_set\_folder, testing\_data\_set\_folder, epochs = 50, batch\_size = 32) :

    backend.clear\_session()

    train\_datagen = ImageDataGenerator(

          rescale=1./255,

          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')

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

            training\_data\_set\_folder,

            target\_size=input\_size,

            batch\_size=32,

            class\_mode='categorical')

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

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

            validation\_data\_set\_folder,

            target\_size=input\_size,

            batch\_size=batch\_size,

            class\_mode='categorical')

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

            testing\_data\_set\_folder,

            target\_size=input\_size,

            batch\_size=batch\_size,

            class\_mode='categorical')

input\_shape = Input(shape=input\_size+(3,))

    start\_time = time.time()

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

    vgg\_features = vgg\_model.output

    vgg\_features = Flatten()(vgg\_features)

    vgg\_features = Dense(512, activation='relu')(vgg\_features)

    vgg\_end\_time = time.time()

    print\_time(start\_time,vgg\_end\_time, "Total time spend during feature extraction from the VGG16 model is {} minutes")

     resnet\_model = ResNet50(weights='imagenet', include\_top=False, input\_tensor=input\_shape)

    resnet\_features = resnet\_model.output

    resnet\_features = Flatten()(resnet\_features)

    resnet\_features = Dense(512, activation='relu')(resnet\_features)

     resnet\_end\_time = time.time()

    print\_time(vgg\_end\_time,resnet\_end\_time, "Total time spend during feature extraction from the Resnet model is {} minutes")

    merged = Concatenate()([vgg\_features, resnet\_features])

    merged = Dropout(0.5)(merged)

    merged = Dense(256, activation='relu')(merged)

    merged = Dropout(0.5)(merged)

    predictions = Dense(train\_generator.num\_classes, activation='softmax')(merged)

model = Model(inputs=input\_shape, outputs=predictions)

    model.compile(optimizer=Adam(lr=0.0001), loss='categorical\_crossentropy', metrics=['accuracy'])

    model.summary()

history = model.fit\_generator(

        train\_generator,

        steps\_per\_epoch=train\_generator.samples // batch\_size,

        epochs=epochs,

        validation\_data=validation\_generator,

        validation\_steps=validation\_generator.samples // batch\_size)

   plot\_graph(history)

   training\_end\_time = time.time()

    print\_time(resnet\_end\_time,training\_end\_time, "Total time spend during training is {} minutes")

   testing\_accuracy\_tracker = []

    testing\_losses\_tracker = []

    for i in range(2) :

        testing\_start\_time = time.time()

        loss, accuracy = model.evaluate\_generator(test\_generator, steps= test\_generator.samples // batch\_size)

        testing\_accuracy\_tracker.append(accuracy)

        testing\_losses\_tracker.append(loss)

        testing\_end\_time = time.time()

   print\_time(testing\_start\_time, testing\_end\_time, "Total time spend during testing is {} minutes")

   model\_end\_time = time.time()

    print("Accuracy during the testing : %.2f%%" %  (np.average(np.array(testing\_accuracy\_tracker)) \* 100))

    print(f'Loss during the testing is {np.average(np.array(testing\_losses\_tracker))}')

    print\_time(start\_time, model\_end\_time, "Total time for pipeline is {} minutes")

"""# Caltech 101 dataset"""

dataset\_folder = '/kaggle/input/caltech-101/caltech-101'

dataset\_save\_folder = '/kaggle/working/caltech-101'

selected\_images = 30

validation\_image\_number = 24

input\_size = (224, 224)

epoch = 50

batch\_size = 32

# if os.path.exists(dataset\_save\_folder) :

#     shutil.rmtree(dataset\_save\_folder)

# generate\_dataset(dataset\_folder, dataset\_save\_folder, selected\_images, validation\_image\_number)

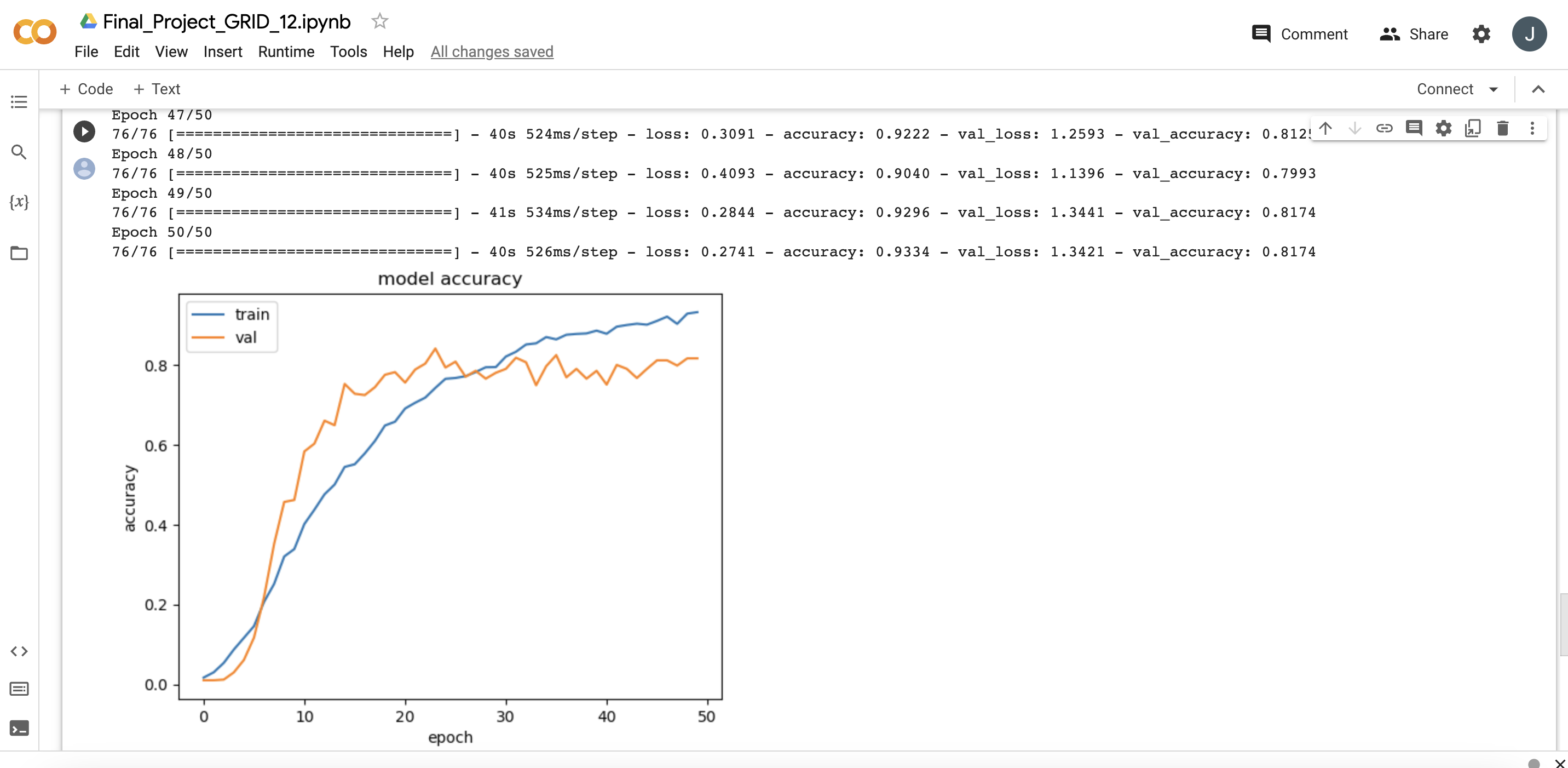
training\_data\_set\_folder = get\_save\_path('train', dataset\_save\_folder)

validation\_data\_set\_folder = get\_save\_path('validation', dataset\_save\_folder)

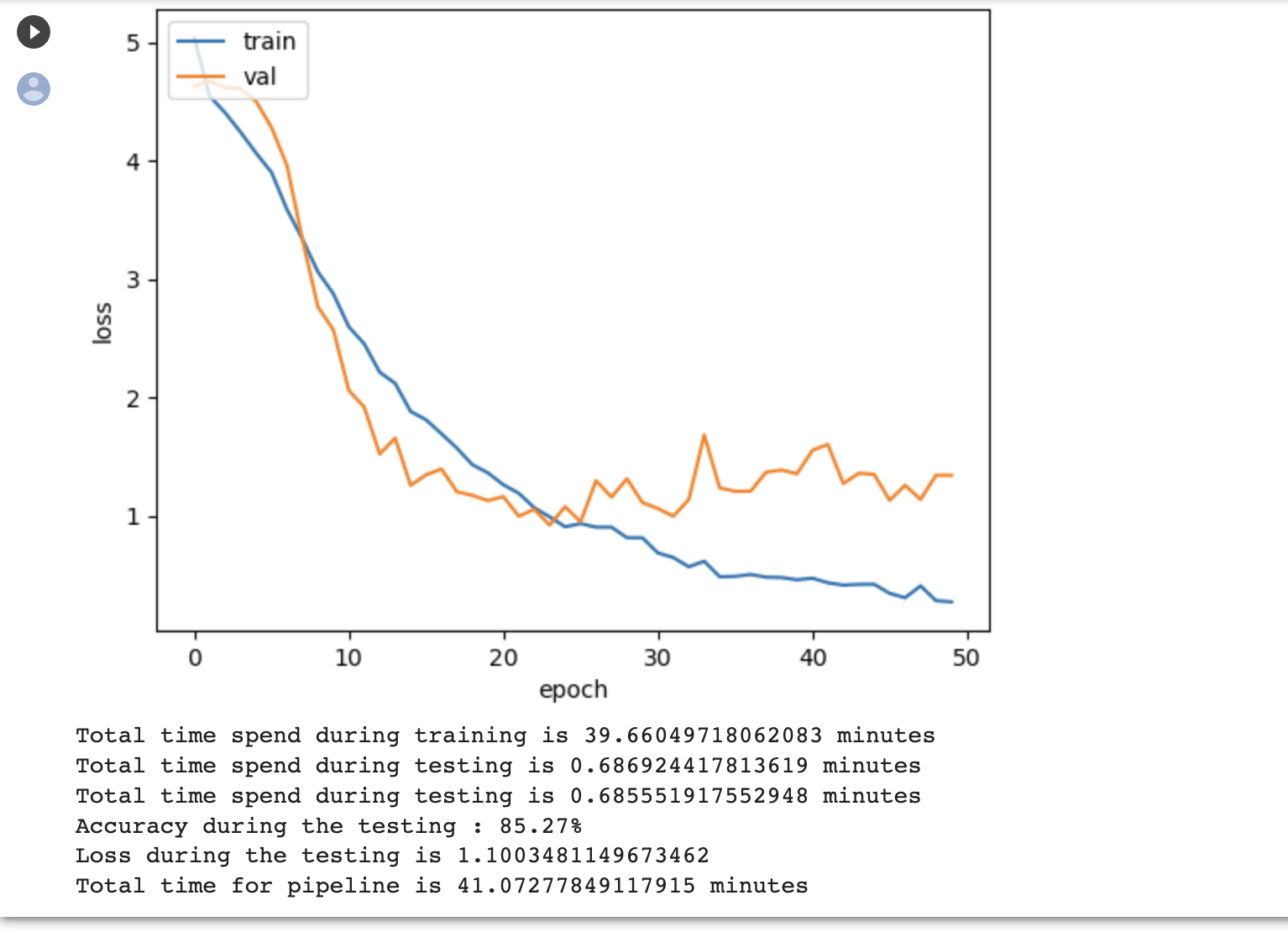
testing\_data\_set\_folder = get\_save\_path('test', dataset\_save\_folder)

execute\_model(training\_data\_set\_folder, validation\_data\_set\_folder, testing\_data\_set\_folder, epoch, batch\_size)

**Output: 1.1 Cifar101(2-run)**

****

**Output:1.2**

****

**Cifar10.Py**

from keras.applications.vgg16 import VGG16

from tensorflow.keras.applications.resnet50 import ResNet50, preprocess\_input

from keras.layers import Input, Flatten, Dense, Concatenate,Dropout

from keras.models import Model

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

from keras.models import Model

from keras.models import Sequential

from keras.optimizers import Adam

from sklearn.utils import shuffle

import os

import random

import shutil

import time

from PIL import Image

from keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.preprocessing import image

from keras import optimizers

from keras import backend

import warnings

warnings.filterwarnings("ignore")

"""# This is utility functions for graph related operations"""

def plot\_graph(history) :

    plt.plot(history.history['accuracy'])

    plt.plot(history.history['val\_accuracy'])

    plt.title('model accuracy')

    plt.ylabel('accuracy')

    plt.xlabel('epoch')

    plt.legend(['train', 'val'], loc='upper left')

    plt.show()

    plt.plot(history.history['loss'])

    plt.plot(history.history['val\_loss'])

    plt.title('model loss')

    plt.ylabel('loss')

    plt.xlabel('epoch')

    plt.legend(['train', 'val'], loc='upper left')

    plt.show()

"""# This is utility functions for time related operations"""

def print\_time(start\_time, end\_time, description) :

    total\_time\_in\_seconds = end\_time - start\_time

    total\_time\_in\_minutes = total\_time\_in\_seconds / 60

    print(description.format(total\_time\_in\_minutes))

"""# This is utility functions for file and dataset related operations"""

def get\_save\_path(filename, dataset\_save\_folder = '/kaggle/working/') :

    return os.path.join(dataset\_save\_folder, filename)

def create\_dir(path\_dir) :

    if not os.path.exists(path\_dir) :

        os.makedirs(path\_dir)

def generate\_dataset(src\_dir, dest\_dir, image\_set\_number, validation\_image\_number = 24) :

    train\_dir = os.path.join(dest\_dir, 'train')

    test\_dir = os.path.join(dest\_dir, 'test')

    validation\_dir = os.path.join(dest\_dir, 'validation')

    create\_dir(train\_dir)

    create\_dir(test\_dir)

    create\_dir(validation\_dir)

    categories = os.listdir(src\_dir)

    for category in categories:

        category\_path = os.path.join(src\_dir, category)

        category\_train\_path = os.path.join(train\_dir, category)

        category\_test\_path = os.path.join(test\_dir, category)

        category\_validation\_path = os.path.join(validation\_dir, category)

        create\_dir(category\_train\_path)

        create\_dir(category\_test\_path)

        create\_dir(category\_validation\_path)

        if os.path.isdir(category\_path):

            image\_files = os.listdir(category\_path)

            random.shuffle(image\_files)

            train\_images = image\_files[:image\_set\_number]

            test\_images = image\_files[image\_set\_number:]

            # Create test folder

            for filename in test\_images:

                src = os.path.join(category\_path, filename)

                dst = os.path.join(category\_test\_path, filename)

                shutil.copy(src, dst)

            train\_set\_images = train\_images[:validation\_image\_number]

            validation\_images = train\_images[validation\_image\_number:]

            # Create validation folder

            for filename in validation\_images:

                src = os.path.join(category\_path, filename)

                dst = os.path.join(category\_validation\_path, filename)

                shutil.copy(src, dst)

            # Create train folder

            for filename in train\_set\_images:

                src = os.path.join(category\_path, filename)

                dst = os.path.join(category\_train\_path, filename)

                shutil.copy(src, dst)

"""# Clasification Model

"""

def execute\_model(training\_data\_set\_folder, validation\_data\_set\_folder, testing\_data\_set\_folder, epochs = 50, batch\_size = 32) :

    backend.clear\_session()

    train\_datagen = ImageDataGenerator(

          rescale=1./255,

          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')

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

            training\_data\_set\_folder,

            target\_size=input\_size,

            batch\_size=32,

            class\_mode='categorical')

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

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

            validation\_data\_set\_folder,

            target\_size=input\_size,

            batch\_size=batch\_size,

            class\_mode='categorical')

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

            testing\_data\_set\_folder,

            target\_size=input\_size,

            batch\_size=batch\_size,

            class\_mode='categorical')

    input\_shape = Input(shape=input\_size+(3,))

    start\_time = time.time()

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

    vgg\_features = vgg\_model.output

    vgg\_features = Flatten()(vgg\_features)

    vgg\_features = Dense(512, activation='relu')(vgg\_features)

    vgg\_end\_time = time.time()

    print\_time(start\_time,vgg\_end\_time, "Total time spend during feature extraction from the VGG16 model is {} minutes")

    resnet\_model = ResNet50(weights='imagenet', include\_top=False, input\_tensor=input\_shape)

    resnet\_features = resnet\_model.output

    resnet\_features = Flatten()(resnet\_features)

    resnet\_features = Dense(512, activation='relu')(resnet\_features)

    resnet\_end\_time = time.time()

    print\_time(vgg\_end\_time,resnet\_end\_time, "Total time spend during feature extraction from the Resnet model is {} minutes")

    merged = Concatenate()([vgg\_features, resnet\_features])

    merged = Dropout(0.5)(merged)

    merged = Dense(256, activation='relu')(merged)

    merged = Dropout(0.5)(merged)

    predictions = Dense(train\_generator.num\_classes, activation='softmax')(merged)

    model = Model(inputs=input\_shape, outputs=predictions)

    model.compile(optimizer=Adam(lr=0.0001), loss='categorical\_crossentropy', metrics=['accuracy'])

    model.summary()

    history = model.fit\_generator(

        train\_generator,

        steps\_per\_epoch=train\_generator.samples // batch\_size,

        epochs=epochs,

        validation\_data=validation\_generator,

        validation\_steps=validation\_generator.samples // batch\_size)

    plot\_graph(history)

    training\_end\_time = time.time()

    print\_time(resnet\_end\_time,training\_end\_time, "Total time spend during training is {} minutes")

    testing\_accuracy\_tracker = []

    testing\_losses\_tracker = []

    for i in range(2) :

        testing\_start\_time = time.time()

        loss, accuracy = model.evaluate\_generator(test\_generator, steps= test\_generator.samples // batch\_size)

        testing\_accuracy\_tracker.append(accuracy)

        testing\_losses\_tracker.append(loss)

        testing\_end\_time = time.time()

        print\_time(testing\_start\_time, testing\_end\_time, "Total time spend during testing is {} minutes")

    model\_end\_time = time.time()

    print("Accuracy during the testing : %.2f%%" %  (np.average(np.array(testing\_accuracy\_tracker)) \* 100))

    print(f'Loss during the testing is {np.average(np.array(testing\_losses\_tracker))}')

    print\_time(start\_time, model\_end\_time, "Total time for pipeline is {} minutes")

"""# Caltech 256 Dataset"""

dataset\_folder = '/kaggle/input/caltech256/256\_ObjectCategories'

dataset\_save\_folder = '/kaggle/working/caltech256'

selected\_images = 30

validation\_image\_number = 24

input\_size = (150, 150)

epoch = 50

batch\_size = 32

# Generate the dataset as per the requirement

if os.path.exists(dataset\_save\_folder) :

    shutil.rmtree(dataset\_save\_folder)

generate\_dataset(dataset\_folder, dataset\_save\_folder, selected\_images, validation\_image\_number)

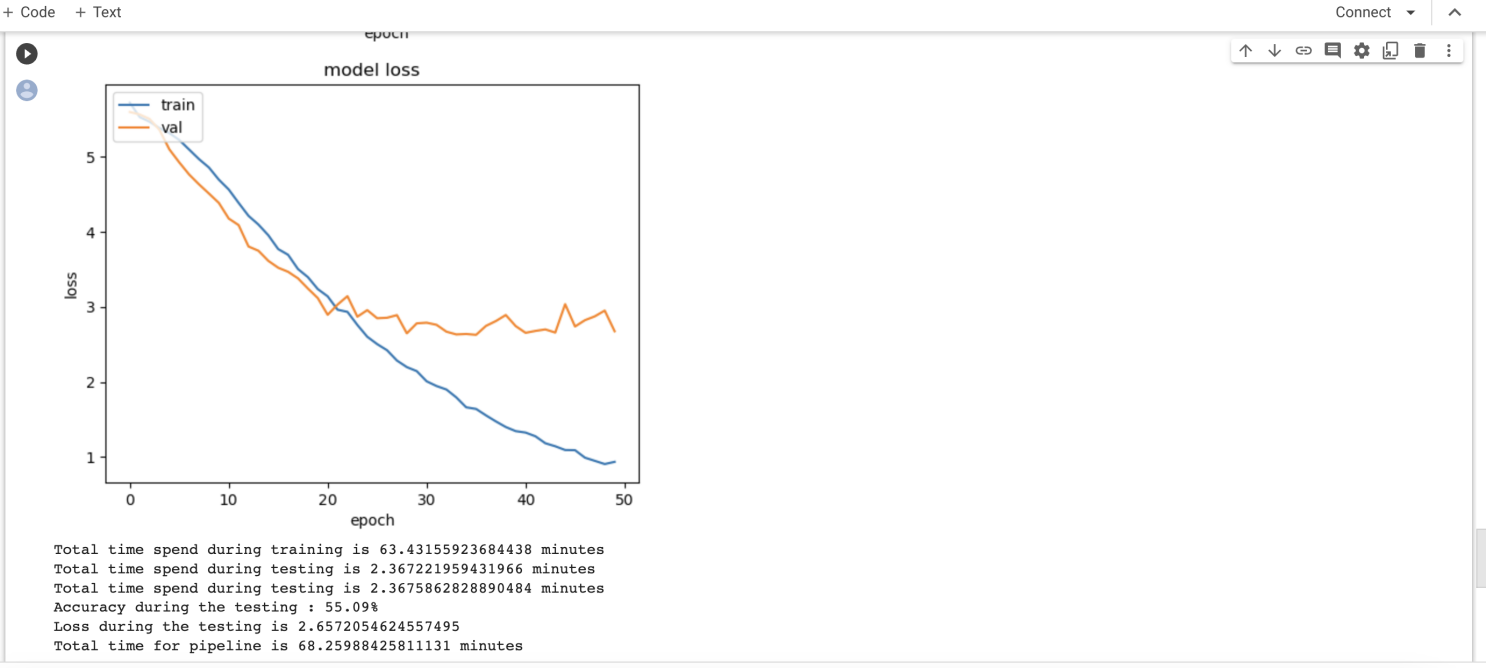
training\_data\_set\_folder = get\_save\_path('train', dataset\_save\_folder)

validation\_data\_set\_folder = get\_save\_path('validation', dataset\_save\_folder)

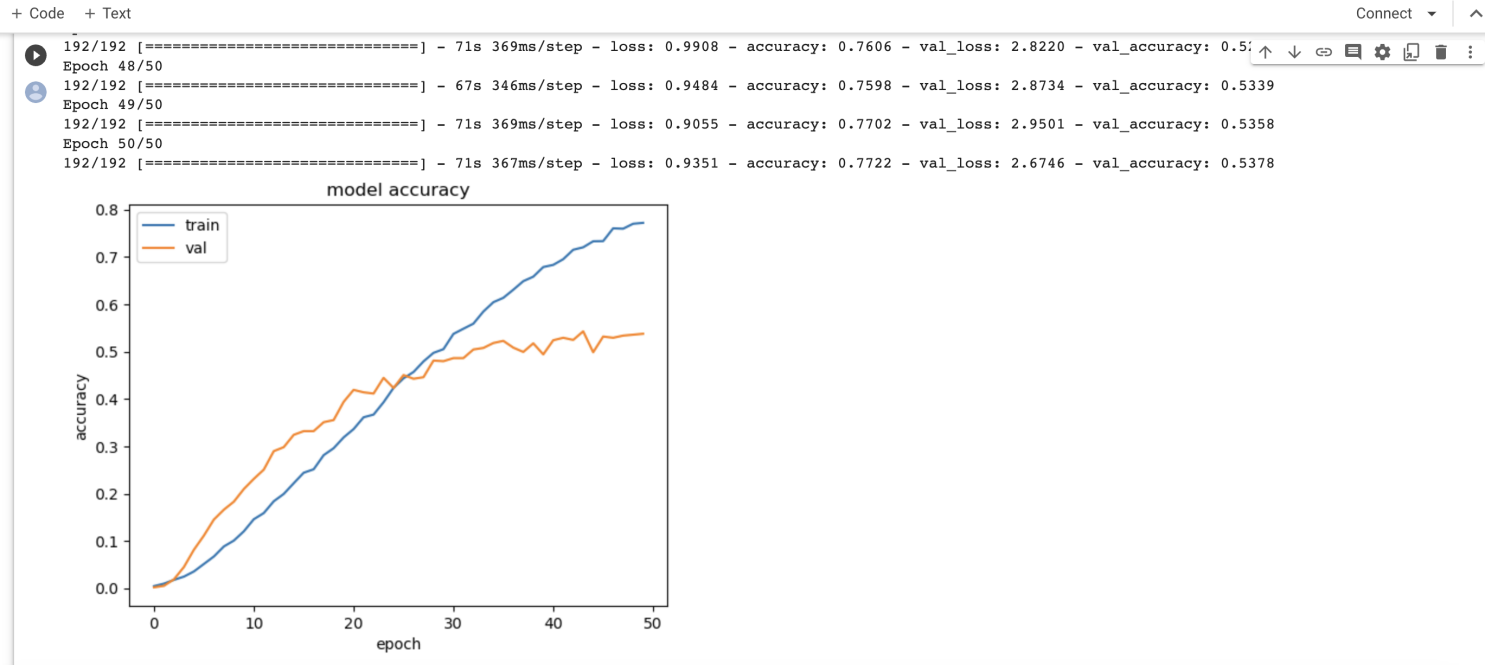
testing\_data\_set\_folder = get\_save\_path('test', dataset\_save\_folder)

execute\_model(training\_data\_set\_folder, validation\_data\_set\_folder, testing\_data\_set\_folder, epoch, batch\_size)

**Output:2.1 Cifar256(2-run)**

****

**Output:2.2**

****

**Caltech256.py**

from keras.applications.vgg16 import VGG16

from tensorflow.keras.applications.resnet50 import ResNet50, preprocess\_input

from keras.layers import Input, Flatten, Dense, Concatenate,Dropout

from keras.models import Model

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

from keras.models import Model

from keras.models import Sequential

from keras.optimizers import Adam

from sklearn.utils import shuffle

import os

import random

import shutil

import time

from PIL import Image

from keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.preprocessing import image

from keras import optimizers

from keras import backend

import warnings

warnings.filterwarnings("ignore")

"""# This is utility functions for graph related operations"""

def plot\_graph(history) :

    plt.plot(history.history['accuracy'])

    plt.plot(history.history['val\_accuracy'])

    plt.title('model accuracy')

    plt.ylabel('accuracy')

    plt.xlabel('epoch')

    plt.legend(['train', 'val'], loc='upper left')

    plt.show()

    plt.plot(history.history['loss'])

    plt.plot(history.history['val\_loss'])

    plt.title('model loss')

    plt.ylabel('loss')

    plt.xlabel('epoch')

    plt.legend(['train', 'val'], loc='upper left')

    plt.show()

"""# This is utility functions for time related operations"""

def print\_time(start\_time, end\_time, description) :

    total\_time\_in\_seconds = end\_time - start\_time

    total\_time\_in\_minutes = total\_time\_in\_seconds / 60

    print(description.format(total\_time\_in\_minutes))

"""# This is utility functions for file and dataset related operations"""

def get\_save\_path(filename, dataset\_save\_folder = '/kaggle/working/') :

    return os.path.join(dataset\_save\_folder, filename)

def create\_dir(path\_dir) :

    if not os.path.exists(path\_dir) :

        os.makedirs(path\_dir)

def generate\_dataset(src\_dir, dest\_dir, image\_set\_number, validation\_image\_number = 24) :

    train\_dir = os.path.join(dest\_dir, 'train')

    test\_dir = os.path.join(dest\_dir, 'test')

    validation\_dir = os.path.join(dest\_dir, 'validation')

    create\_dir(train\_dir)

    create\_dir(test\_dir)

    create\_dir(validation\_dir)

    categories = os.listdir(src\_dir)

    for category in categories:

        category\_path = os.path.join(src\_dir, category)

        category\_train\_path = os.path.join(train\_dir, category)

        category\_test\_path = os.path.join(test\_dir, category)

        category\_validation\_path = os.path.join(validation\_dir, category)

        create\_dir(category\_train\_path)

        create\_dir(category\_test\_path)

        create\_dir(category\_validation\_path)

        if os.path.isdir(category\_path):

            image\_files = os.listdir(category\_path)

            random.shuffle(image\_files)

            train\_images = image\_files[:image\_set\_number]

            test\_images = image\_files[image\_set\_number:]

            # Create test folder

            for filename in test\_images:

                src = os.path.join(category\_path, filename)

                dst = os.path.join(category\_test\_path, filename)

                shutil.copy(src, dst)

            train\_set\_images = train\_images[:validation\_image\_number]

            validation\_images = train\_images[validation\_image\_number:]

            # Create validation folder

            for filename in validation\_images:

                src = os.path.join(category\_path, filename)

                dst = os.path.join(category\_validation\_path, filename)

                shutil.copy(src, dst)

            # Create train folder

            for filename in train\_set\_images:

                src = os.path.join(category\_path, filename)

                dst = os.path.join(category\_train\_path, filename)

                shutil.copy(src, dst)

"""# Clasification Model

"""

def execute\_model(training\_data\_set\_folder, validation\_data\_set\_folder, testing\_data\_set\_folder, epochs = 50, batch\_size = 32) :

    backend.clear\_session()

    train\_datagen = ImageDataGenerator(

          rescale=1./255,

          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')

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

            training\_data\_set\_folder,

            target\_size=input\_size,

            batch\_size=32,

            class\_mode='categorical')

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

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

            validation\_data\_set\_folder,

            target\_size=input\_size,

            batch\_size=batch\_size,

            class\_mode='categorical')

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

            testing\_data\_set\_folder,

            target\_size=input\_size,

            batch\_size=batch\_size,

            class\_mode='categorical')

    input\_shape = Input(shape=input\_size+(3,))

    start\_time = time.time()

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

    vgg\_features = vgg\_model.output

    vgg\_features = Flatten()(vgg\_features)

    vgg\_features = Dense(512, activation='relu')(vgg\_features)

    vgg\_end\_time = time.time()

    print\_time(start\_time,vgg\_end\_time, "Total time spend during feature extraction from the VGG16 model is {} minutes")

    resnet\_model = ResNet50(weights='imagenet', include\_top=False, input\_tensor=input\_shape)

    resnet\_features = resnet\_model.output

    resnet\_features = Flatten()(resnet\_features)

    resnet\_features = Dense(512, activation='relu')(resnet\_features)

    resnet\_end\_time = time.time()

    print\_time(vgg\_end\_time,resnet\_end\_time, "Total time spend during feature extraction from the Resnet model is {} minutes")

    merged = Concatenate()([vgg\_features, resnet\_features])

    merged = Dropout(0.5)(merged)

    merged = Dense(256, activation='relu')(merged)

    merged = Dropout(0.5)(merged)

    predictions = Dense(train\_generator.num\_classes, activation='softmax')(merged)

    model = Model(inputs=input\_shape, outputs=predictions)

    model.compile(optimizer=Adam(lr=0.0001), loss='categorical\_crossentropy', metrics=['accuracy'])

    model.summary()

    history = model.fit\_generator(

        train\_generator,

        steps\_per\_epoch=train\_generator.samples // batch\_size,

        epochs=epochs,

        validation\_data=validation\_generator,

        validation\_steps=validation\_generator.samples // batch\_size)

    plot\_graph(history)

    training\_end\_time = time.time()

    print\_time(resnet\_end\_time,training\_end\_time, "Total time spend during training is {} minutes")

    testing\_accuracy\_tracker = []

    testing\_losses\_tracker = []

    for i in range(2) :

        testing\_start\_time = time.time()

        loss, accuracy = model.evaluate\_generator(test\_generator, steps= test\_generator.samples // batch\_size)

        testing\_accuracy\_tracker.append(accuracy)

        testing\_losses\_tracker.append(loss)

        testing\_end\_time = time.time()

        print\_time(testing\_start\_time, testing\_end\_time, "Total time spend during testing is {} minutes")

    model\_end\_time = time.time()

    print("Accuracy during the testing : %.2f%%" %  (np.average(np.array(testing\_accuracy\_tracker)) \* 100))

    print(f'Loss during the testing is {np.average(np.array(testing\_losses\_tracker))}')

    print\_time(start\_time, model\_end\_time, "Total time for pipeline is {} minutes")

"""# Caltech 101 dataset"""

dataset\_folder = '/kaggle/input/caltech-101/caltech-101'

dataset\_save\_folder = '/kaggle/working/caltech-101'

selected\_images = 30

validation\_image\_number = 24

input\_size = (224, 224)

epoch = 50

batch\_size = 32

if os.path.exists(dataset\_save\_folder) :

    shutil.rmtree(dataset\_save\_folder)

generate\_dataset(dataset\_folder, dataset\_save\_folder, selected\_images, validation\_image\_number)

training\_data\_set\_folder = get\_save\_path('train', dataset\_save\_folder)

validation\_data\_set\_folder = get\_save\_path('validation', dataset\_save\_folder)

testing\_data\_set\_folder = get\_save\_path('test', dataset\_save\_folder)

execute\_model(training\_data\_set\_folder, validation\_data\_set\_folder, testing\_data\_set\_folder, epoch, batch\_size)

"""# Caltech 256 Dataset"""

dataset\_folder = '/kaggle/input/caltech256/256\_ObjectCategories'

dataset\_save\_folder = '/kaggle/working/caltech256'

selected\_images = 30

validation\_image\_number = 24

input\_size = (150, 150)

epoch = 50

batch\_size = 32

# Generate the dataset as per the requirement

if os.path.exists(dataset\_save\_folder) :

    shutil.rmtree(dataset\_save\_folder)

generate\_dataset(dataset\_folder, dataset\_save\_folder, selected\_images, validation\_image\_number)

training\_data\_set\_folder = get\_save\_path('train', dataset\_save\_folder)

validation\_data\_set\_folder = get\_save\_path('validation', dataset\_save\_folder)

testing\_data\_set\_folder = get\_save\_path('test', dataset\_save\_folder)

execute\_model(training\_data\_set\_folder, validation\_data\_set\_folder, testing\_data\_set\_folder, epoch, batch\_size)

"""# Cifar 10 dataset"""

training\_data\_set\_folder = "/kaggle/input/cifar10-images-dataset/cifar-10/train"

validation\_data\_set\_folder = "/kaggle/input/cifar10-images-dataset/cifar-10/validate"

testing\_data\_set\_folder = "/kaggle/input/cifar10-images-dataset/cifar-10/test"

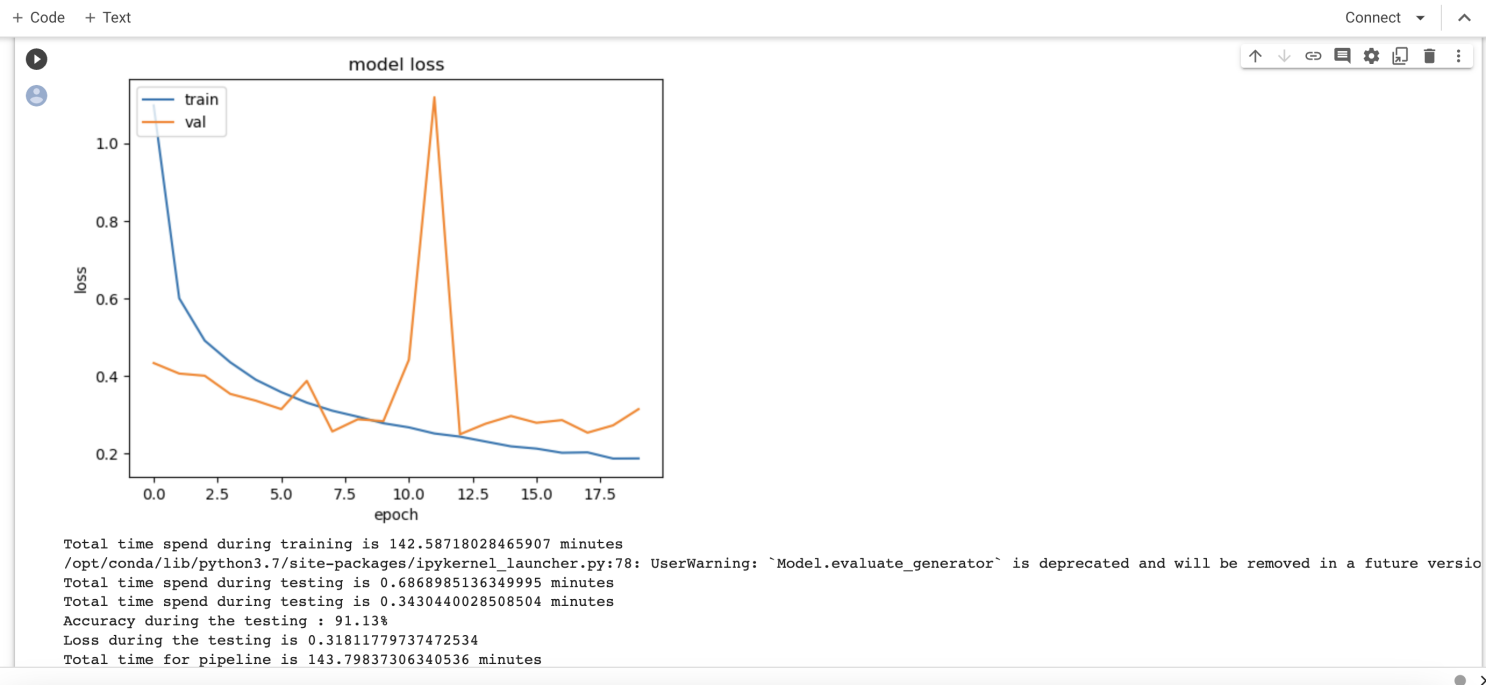
input\_size = (150, 150)

epoch = 20

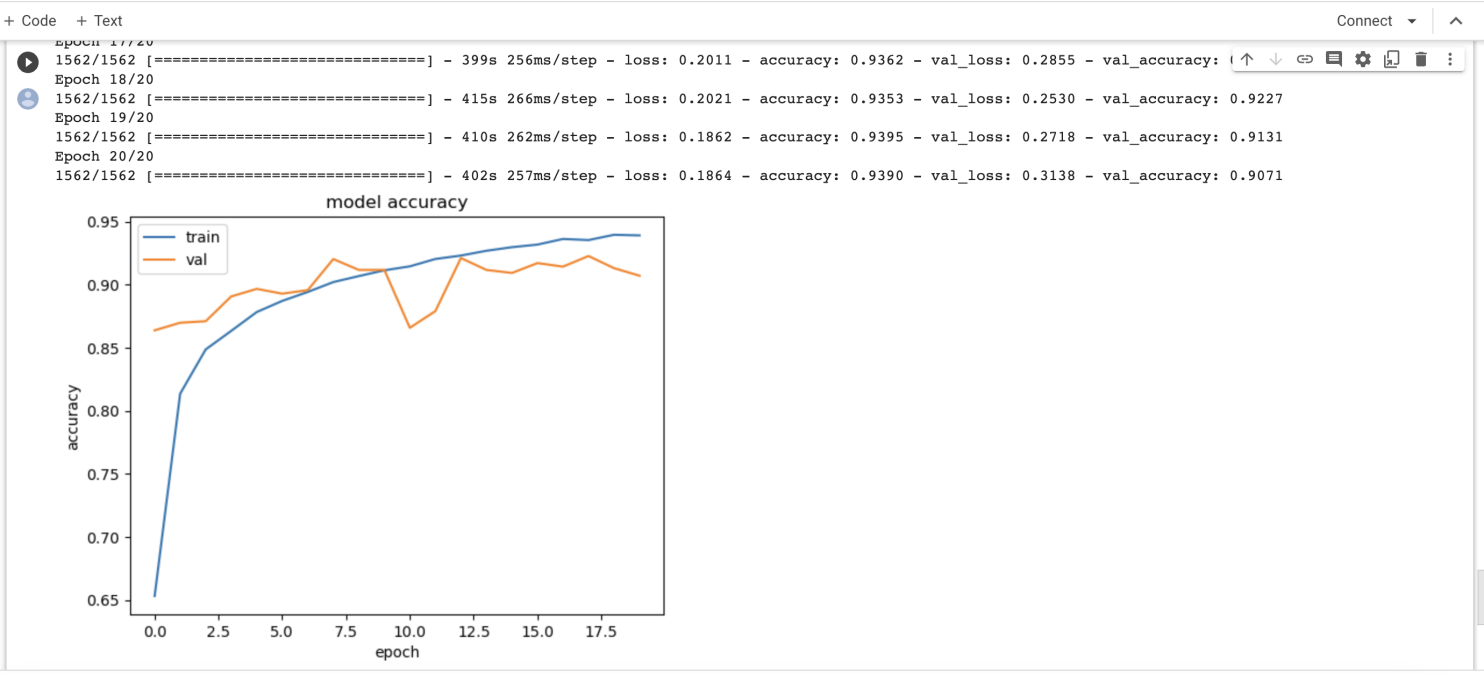
batch\_size = 32

execute\_model(training\_data\_set\_folder, validation\_data\_set\_folder, testing\_data\_set\_folder, epoch, batch\_size)

**Output:3.1 Cifar10(2-run)**

****

**Output:3.2**

****

**Output:**

# Average accuracy = (Cifar10+Caltech101+Caltech256)

**3**

# = (91.13+ 85.27 +55.09)

**3**

# = 77.23