Final Project

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

Course: Deep Learning (COMP 5413)

Faculty: Yimin yang

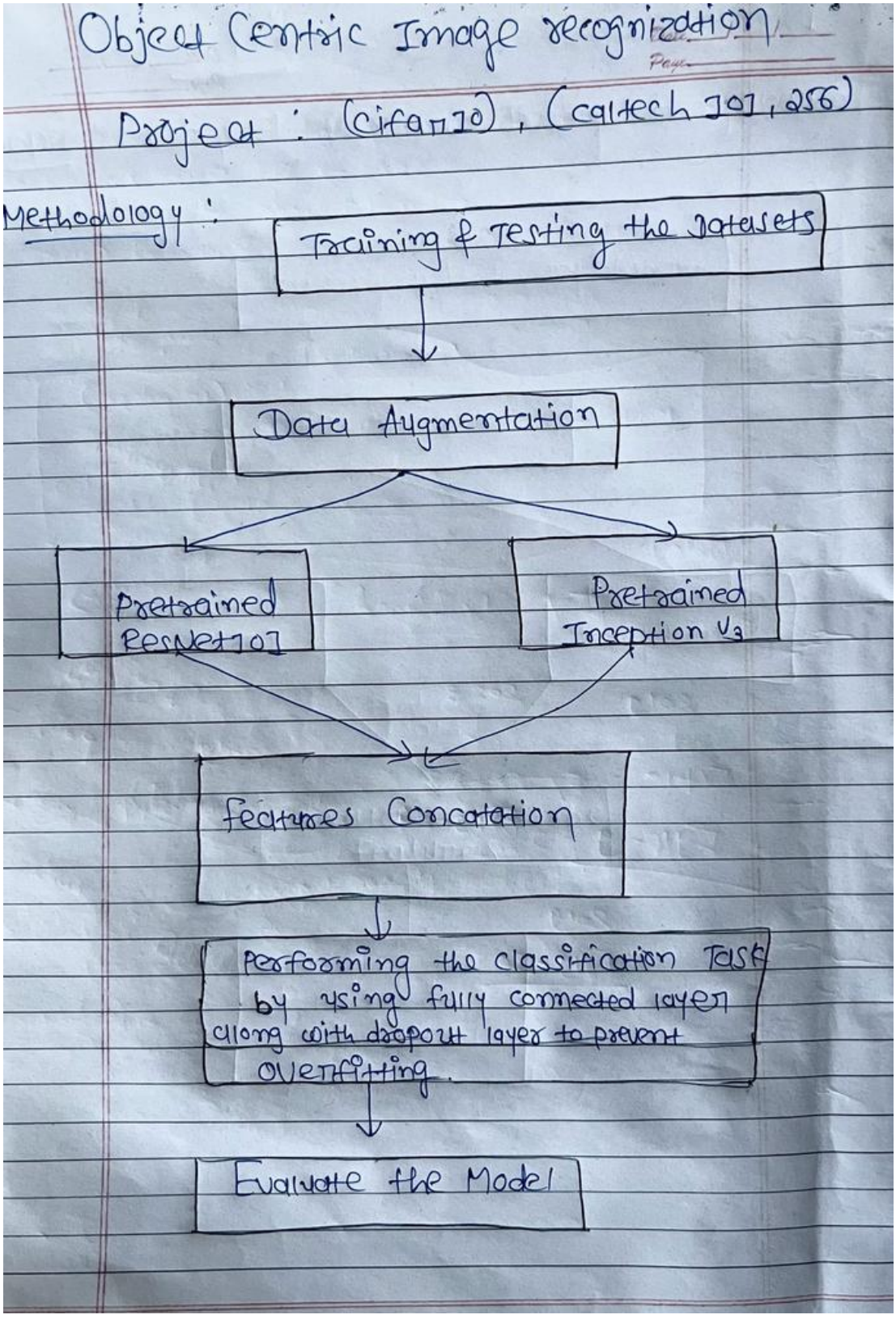
Department: Msc in CS

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')
```

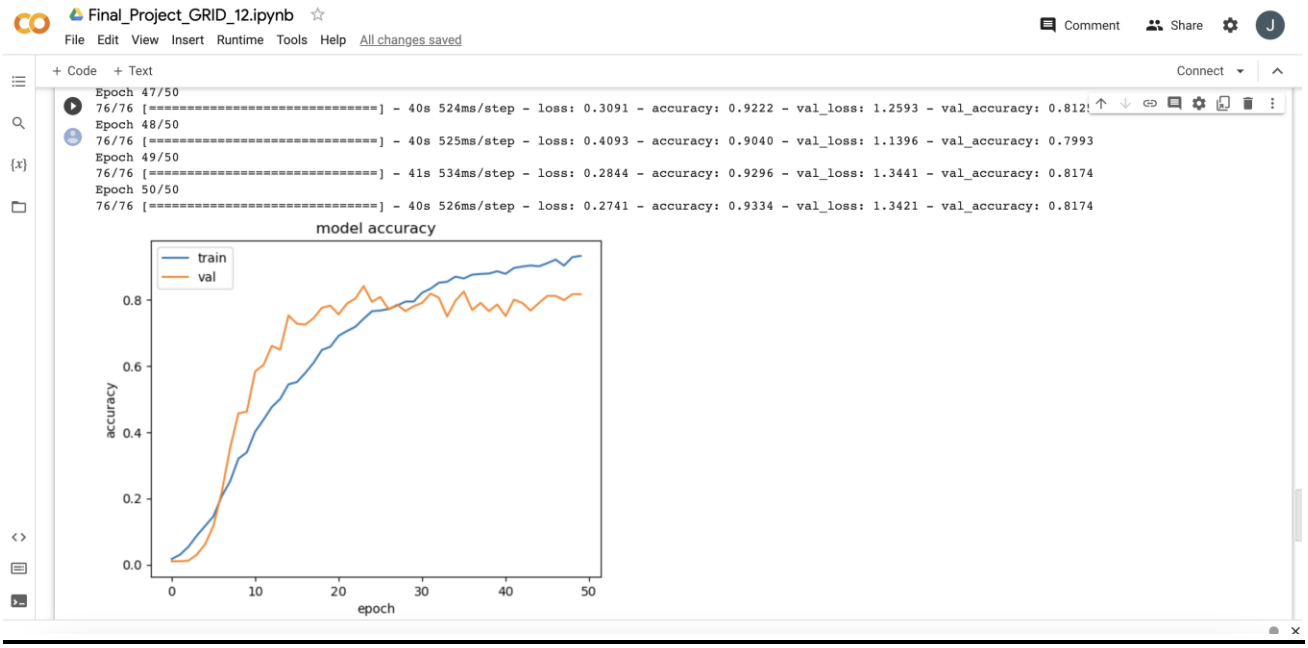
```
    plt.show()
"""# This is utility functions for time related operations"""
def print_time(start_time, end_time, description) :
    total_time_in_seconds = end_time - start_time
    total_time_in_minutes = total_time_in_seconds / 60
    print(description.format(total_time_in_minutes))
"""# This 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()
```

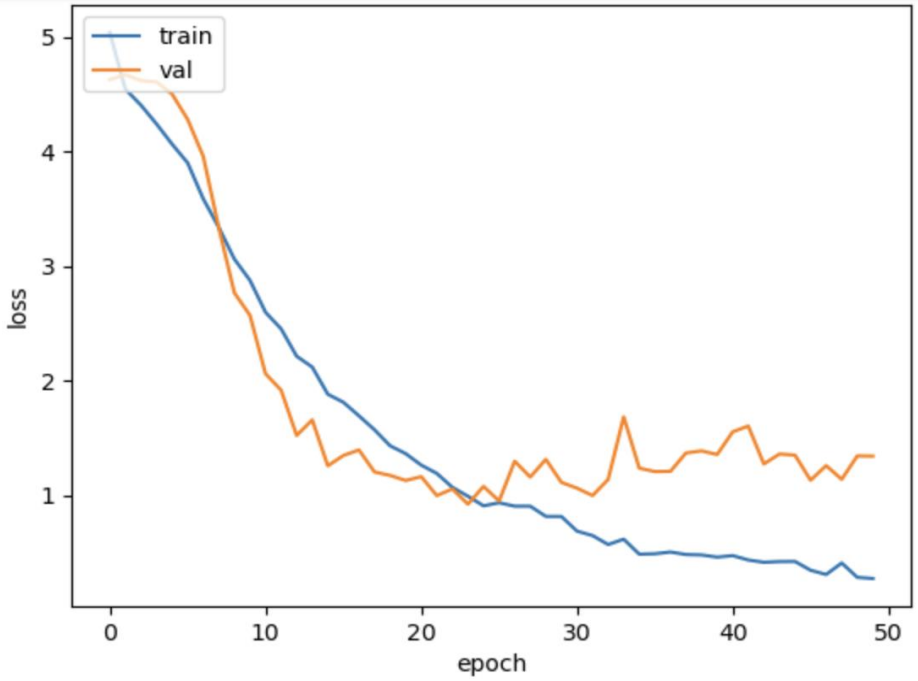
```
    print_time(start_time,vgg_end_time, "Total time spend during feature extraction
from the VGG16 model is {} minutes")
     resnet_model = ResNet50(weights='imagenet', include_top=False,
input_tensor=input_shape)
    resnet_features = resnet_model.output
    resnet_features = Flatten()(resnet_features)
    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




```
Total time spend during training is 39.66049718062083 minutes
Total time spend during testing is 0.686924417813619 minutes
Total time spend during testing is 0.685551917552948 minutes
Accuracy during the testing : 85.27%
Loss during the testing is 1.1003481149673462
Total time for pipeline is 41.07277849117915 minutes
```

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"""
```

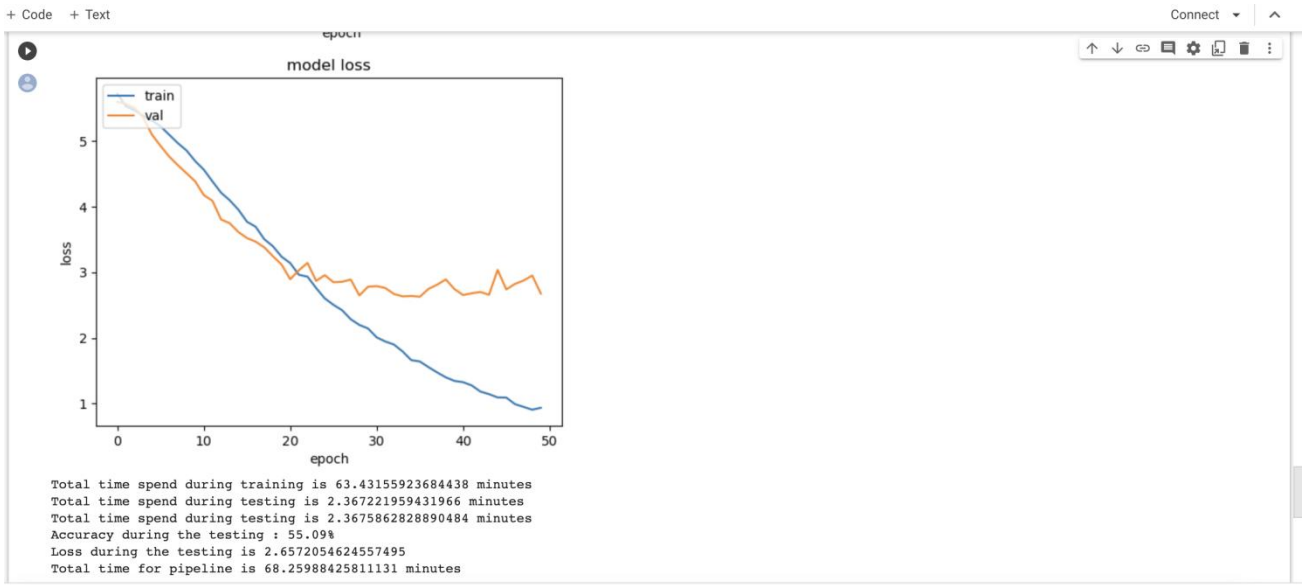
```
def print_time(start_time, end_time, description) :
    total_time_in_seconds = end_time - start_time
    total_time_in_minutes = total_time_in_seconds / 60
    print(description.format(total_time_in_minutes))

"""# 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')
```

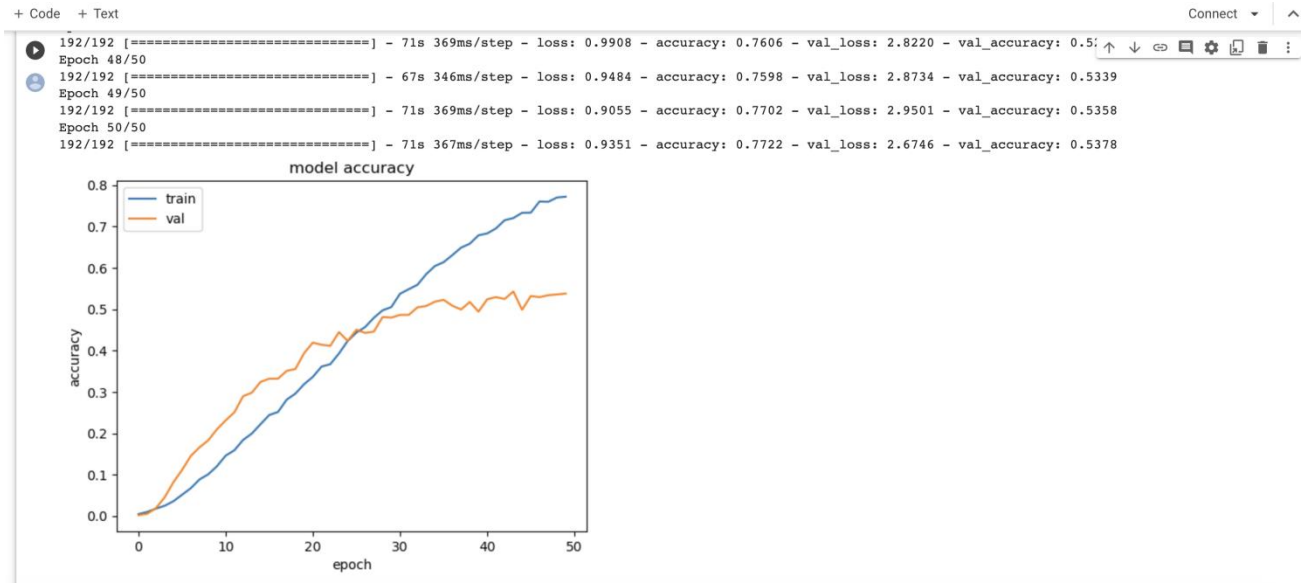
```
    test_generator = test_datagen.flow_from_directory(
            testing_data_set_folder,
            target_size=input_size,
            batch_size=batch_size,
            class_mode='categorical')

    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:]
```

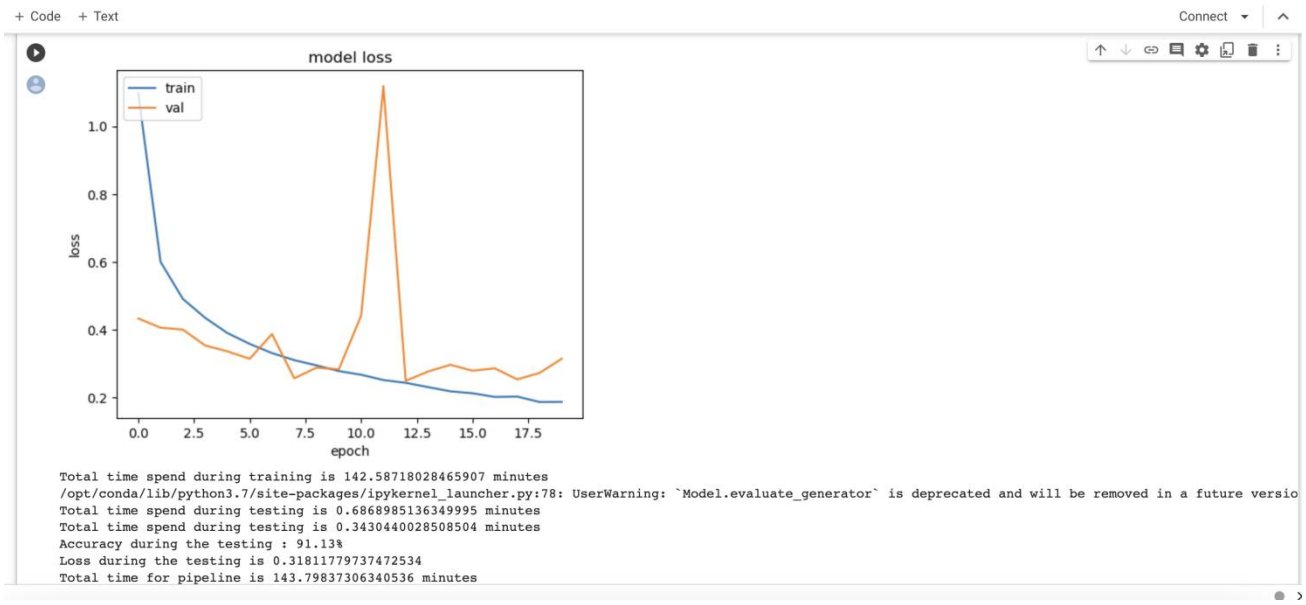
```
            # Create validation folder
            for filename in validation_images:
                src = os.path.join(category_path, filename)
                dst = os.path.join(category_validation_path, filename)
                shutil.copy(src, dst)

            # 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()
```

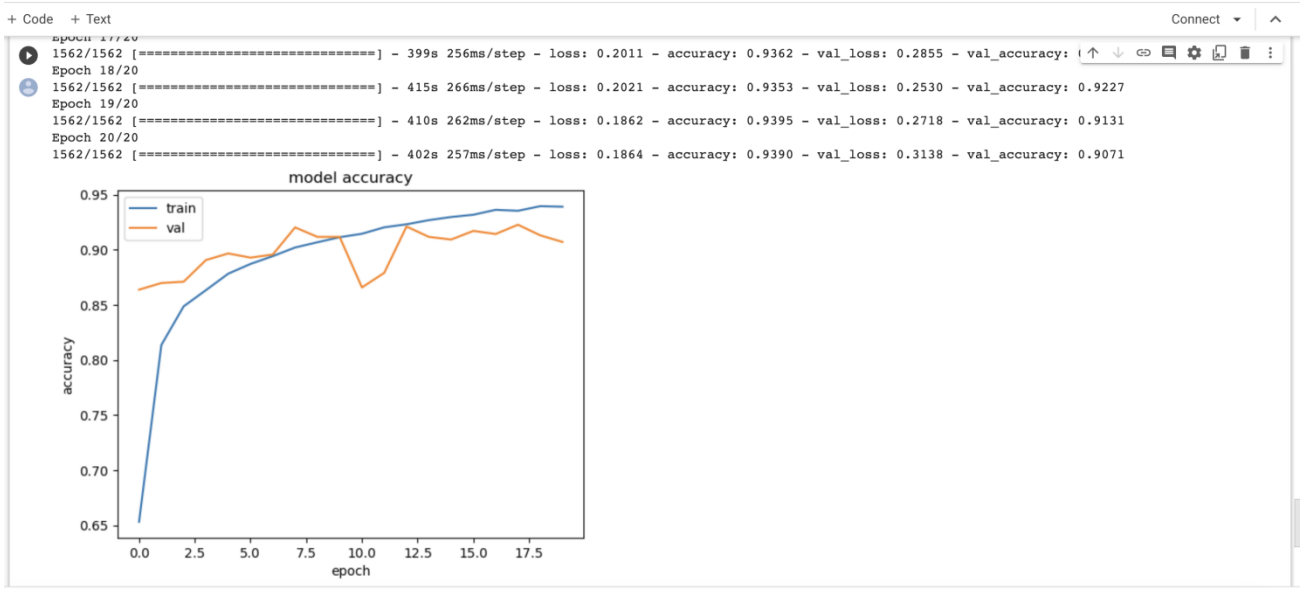
```
    print_time(resnet_end_time,training_end_time, "Total time spend during training is
{} minutes")

    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)


+ Code
+ Text
Connect
model loss
train
val
loss
epoch
Total time spend during training is 142.58718028465907 minutes
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:78: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future versio
Total time spend during testing is 0.6868985136349995 minutes
Total time spend during testing is 0.3430440028508504 minutes
Accuracy during the testing : 91.13%
Loss during the testing is 0.31811779737472534
Total time for pipeline is 143.79837306340536 minutes

Output:3.2


+ Code
+ Text
Connect
1562/1562 [==========] - 399s 256ms/step - loss: 0.2011 - accuracy: 0.9362 - val_loss: 0.2855 - val_accuracy:
Epoch 18/20
1562/1562 [==========] - 415s 266ms/step - loss: 0.2021 - accuracy: 0.9353 - val_loss: 0.2530 - val_accuracy: 0.9227
Epoch 19/20
1562/1562 [==========] - 410s 262ms/step - loss: 0.1862 - accuracy: 0.9395 - val_loss: 0.2718 - val_accuracy: 0.9131
Epoch 20/20
1562/1562 [==========] - 402s 257ms/step - loss: 0.1864 - accuracy: 0.9390 - val_loss: 0.3138 - val_accuracy: 0.9071
model accuracy
train
val
accuracy
epoch

Output:

$$\text{Average accuracy} = \frac{(\text{Cifar10}+\text{Caltech101}+\text{Caltech256})}{3}$$

$$= \frac{(91.13 + 85.27 + 55.09)}{3}$$

$$= 77.23$$