#RESNET-50#

import tensorflow as tf

def residual\_block(x, filters, kernel\_size=3):

    y = x

    x = tf.keras.layers.Conv2D(filters, kernel\_size, padding='same')(x)

    x = tf.keras.layers.BatchNormalization()(x)

    x = tf.keras.layers.ReLU()(x)

    x = tf.keras.layers.Conv2D(filters, kernel\_size, padding='same')(x)

    x = tf.keras.layers.BatchNormalization()(x)

    if x.shape[-1] != y.shape[-1]:

        y = tf.keras.layers.Conv2D(filters, kernel\_size=1, padding='same')(y)

    x = tf.keras.layers.Add()([x, y])

    x = tf.keras.layers.ReLU()(x)

    return x

input\_shape = (224, 224, 3)

input\_tensor = tf.keras.layers.Input(shape=input\_shape)

output = residual\_block(input\_tensor, filters=64)

model = tf.keras.models.Model(inputs=input\_tensor, outputs=output)

model.summary()

#AUTOENCODER#

from keras.layers import Dense,Conv2D,MaxPooling2D,UpSampling2D

from keras import Input, Model

from keras.datasets import mnist

import numpy as np

import matplotlib.pyplot as plt

encoding\_dim = 15

input\_img = Input(shape=(784,))

encoded = Dense(encoding\_dim, activation='relu')(input\_img)

decoded = Dense(784, activation='sigmoid')(encoded)

autoencoder = Model(input\_img, decoded)

encoder = Model(input\_img, encoded)

encoded\_input = Input(shape=(encoding\_dim,))

decoder\_layer = autoencoder.layers[-1]

decoder = Model(encoded\_input, decoder\_layer(encoded\_input))

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

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

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))

x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))

print(x\_train.shape)

print(x\_test.shape)

autoencoder.fit(x\_train, x\_train, epochs=15, batch\_size=256, validation\_data=(x\_test, x\_test))

encoded\_img = encoder.predict(x\_test)

decoded\_img = decoder.predict(encoded\_img)

plt.figure(figsize=(20, 4))

for i in range(5):

    # Display original

    ax = plt.subplot(2, 5, i + 1)

    plt.imshow(x\_test[i].reshape(28, 28))

    plt.gray()

    ax.get\_xaxis().set\_visible(False)

    ax.get\_yaxis().set\_visible(False)

    # Display reconstruction

    ax = plt.subplot(2, 5, i + 1 + 5)

    plt.imshow(decoded\_img[i].reshape(28, 28))

    plt.gray()

    ax.get\_xaxis().set\_visible(False)

    ax.get\_yaxis().set\_visible(False)

plt.show()

#ENSEMBLE NETWORK#

import keras,os

from keras.models import Sequential

from keras.layers import Dense, Conv2D, MaxPool2D , Flatten

model = Sequential()

model.add(Conv2D(input\_shape=(224,224,3),filters=64,kernel\_size=(3,3),padding="same", activation="relu"))

model.add(Conv2D(filters=64,kernel\_size=(3,3),padding="same", activation="relu"))

model.add(MaxPool2D(pool\_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=128, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=128, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(MaxPool2D(pool\_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=256, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=256, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=256, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(MaxPool2D(pool\_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(MaxPool2D(pool\_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(MaxPool2D(pool\_size=(2,2),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=4096,activation="relu"))

model.add(Dense(units=4096,activation="relu"))

model.add(Dense(units=1, activation="sigmoid"))

model.summary()

from keras.layers import Input, Lambda, Dense, Flatten

from keras.models import Model

from keras.applications.vgg16 import VGG16

from keras.applications.vgg16 import preprocess\_input

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

import numpy as np

from keras.preprocessing.image import ImageDataGenerator

IMAGE\_SIZE = [224, 224]

vgg = VGG16(input\_shape=IMAGE\_SIZE + [3], weights='imagenet', include\_top=False)

vgg.summary()

for layer in vgg.layers:

  layer.trainable = False

  x = Flatten()(vgg.output)

prediction = Dense(1, activation='sigmoid')(x)

model = Model(inputs=vgg.input, outputs=prediction)

model.summary()

model.compile(

  loss='binary\_crossentropy',

  optimizer='adam',

  metrics=['accuracy']

)

train\_path = '/content/drive/MyDrive/Pets/images'

valid\_path = '/content/drive/MyDrive/archive/Pet\_Breeds/abyssinian'

train\_datagen = ImageDataGenerator(rescale = 1./255, shear\_range = 0.2, zoom\_range = 0.2, horizontal\_flip = True)

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

training\_set = train\_datagen.flow\_from\_directory(train\_path, target\_size = (224, 224), batch\_size = 16, class\_mode = 'binary')

test\_set = test\_datagen.flow\_from\_directory(valid\_path, target\_size = (224, 224), batch\_size = 16, class\_mode = 'binary')

r = Model.fit\_generator(training\_set, validation\_data=test\_set, epochs=15, steps\_per\_epoch=len(training\_set), validation\_steps=len(test\_set))

from IPython.display import Image

from keras.preprocessing import image

import tensorflow

img\_path = "/content/drive/MyDrive/archive/Pet\_Breeds/cat\_or\_dog\_4.jpg"

test\_image = image.load\_img(img\_path, target\_size = [224,224])

test\_image = image.img\_to\_array(test\_image)

test\_image = np.expand\_dims(test\_image, axis = 0)

result = model.predict(test\_image)

if result==0:

    print("CAT")

else:

    print("DOG")

#RESNET-50#

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

def identity\_block(x, filter):

    x\_skip = x

    x = tf.keras.layers.Conv2D(filter, (3,3), padding = 'same')(x)

    x = tf.keras.layers.BatchNormalization(axis=3)(x)

    x = tf.keras.layers.Activation('relu')(x)

    x = tf.keras.layers.Conv2D(filter, (3,3), padding = 'same')(x)

    x = tf.keras.layers.BatchNormalization(axis=3)(x)

    x = tf.keras.layers.Add()([x, x\_skip])

    x = tf.keras.layers.Activation('relu')(x)

    return x

def convolutional\_block(x, filter):

    x\_skip = x

    x = tf.keras.layers.Conv2D(filter, (3,3), padding = 'same', strides = (2,2))(x)

    x = tf.keras.layers.BatchNormalization(axis=3)(x)

    x = tf.keras.layers.Activation('relu')(x)

    x = tf.keras.layers.Conv2D(filter, (3,3), padding = 'same')(x)

    x = tf.keras.layers.BatchNormalization(axis=3)(x)

    x\_skip = tf.keras.layers.Conv2D(filter, (1,1), strides = (2,2))(x\_skip)

    x = tf.keras.layers.Add()([x, x\_skip])

    x = tf.keras.layers.Activation('relu')(x)

    return x

def ResNet34(shape = (32, 32, 3), classes = 10):

    x\_input = tf.keras.layers.Input(shape)

    x = tf.keras.layers.ZeroPadding2D((3, 3))(x\_input)

    x = tf.keras.layers.Conv2D(64, kernel\_size=7, strides=2, padding='same')(x)

    x = tf.keras.layers.BatchNormalization()(x)

    x = tf.keras.layers.Activation('relu')(x)

    x = tf.keras.layers.MaxPool2D(pool\_size=3, strides=2, padding='same')(x)

    block\_layers = [3, 4, 6, 3]

    filter\_size = 64

    for i in range(4):

        if i == 0:

            for j in range(block\_layers[i]):

                x = identity\_block(x, filter\_size)

        else:

            filter\_size = filter\_size\*2

            x = convolutional\_block(x, filter\_size)

            for j in range(block\_layers[i] - 1):

                x = identity\_block(x, filter\_size)

    x = tf.keras.layers.AveragePooling2D((2,2), padding = 'same')(x)

    x = tf.keras.layers.Flatten()(x)

    x = tf.keras.layers.Dense(512, activation = 'relu')(x)

    x = tf.keras.layers.Dense(classes, activation = 'softmax')(x)

    model = tf.keras.models.Model(inputs = x\_input, outputs = x, name = "ResNet34")

    return model

model.summary()

#AUTOENCODER#

import tensorflow as tf

from tensorflow import keras

import numpy as np

input\_dim = 784

encoding\_dim = 32

encoder\_input = keras.layers.Input(shape=(input\_dim,))

encoder\_output = keras.layers.Dense(encoding\_dim, activation='relu')(encoder\_input)

encoder = keras.models.Model(inputs=encoder\_input, outputs=encoder\_output)

decoder\_input = keras.layers.Input(shape=(encoding\_dim,))

decoder\_output = keras.layers.Dense(input\_dim, activation='sigmoid')(decoder\_input)

decoder = keras.models.Model(inputs=decoder\_input, outputs=decoder\_output)

autoencoder = keras.models.Model(inputs=encoder\_input, outputs=decoder(encoder\_output))

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

data = np.random.random(size=(1, input\_dim))

autoencoder.fit(data, data, epochs=50, batch\_size=1)

encoded\_data = encoder.predict(data)

decoded\_data = decoder.predict(encoded\_data)

#ENSEMBLE NETWORK#

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

data = load\_iris()

X, y = data.data, data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

svm\_classifier = SVC(kernel='linear')

svm\_classifier.fit(X\_train, y\_train)

y\_pred = svm\_classifier.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy \* 100:.2f}%")