# Advanced Deep Learning

**By Rudra Rao – CS21020**

# R.D. & S.H. NATIONAL COLLEGE
# & S. W.A. SCIENCE COLLEGE,

**Bandra, Mumbai – 400050.**

## Department of Computer Science

## CERTIFICATE

This is to certify that **Mr. Rudra Rao** of **M.Sc. Part II(Sem IV)** class has satisfactorily completed **9** Practicals in the subject of **Advanced Deep Learning** as a part of M.Sc. Degree Course in Computer Science during the academic year 2022 – 2023.

**Date of Submission:**

**Faculty Incharge**

**Co-ordinator,**
**Department of Computer Science**

**Signature of External Examiner**

# INDEX

| | | | |
|---|---|---|---|
| 8 | **Practical 8:** Write a program to implement a simple form of a recurrent neural network.<br>1. (4-to-1 RNN) to show that the quantity of rain on a certain day also depends on the values of the previous day<br>2. LSTM for sentiment analysis on datasets like UMICH SI650 for similar. | 25/04/2023 | |
| 9 | **Practical 9:** Write a program for object detection from the image/video. | 02/05/2023 | |

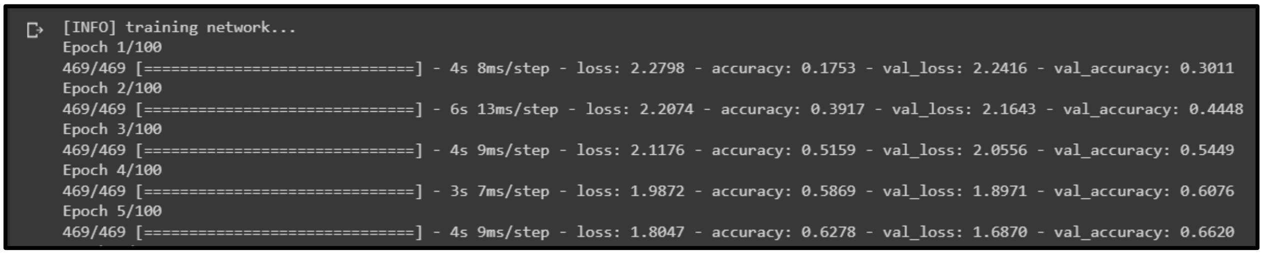
# Practical 1

**Aim: Implement Feed-forward Neural Network and train the network with different optimizers and compare the results.**

**Source Code:**

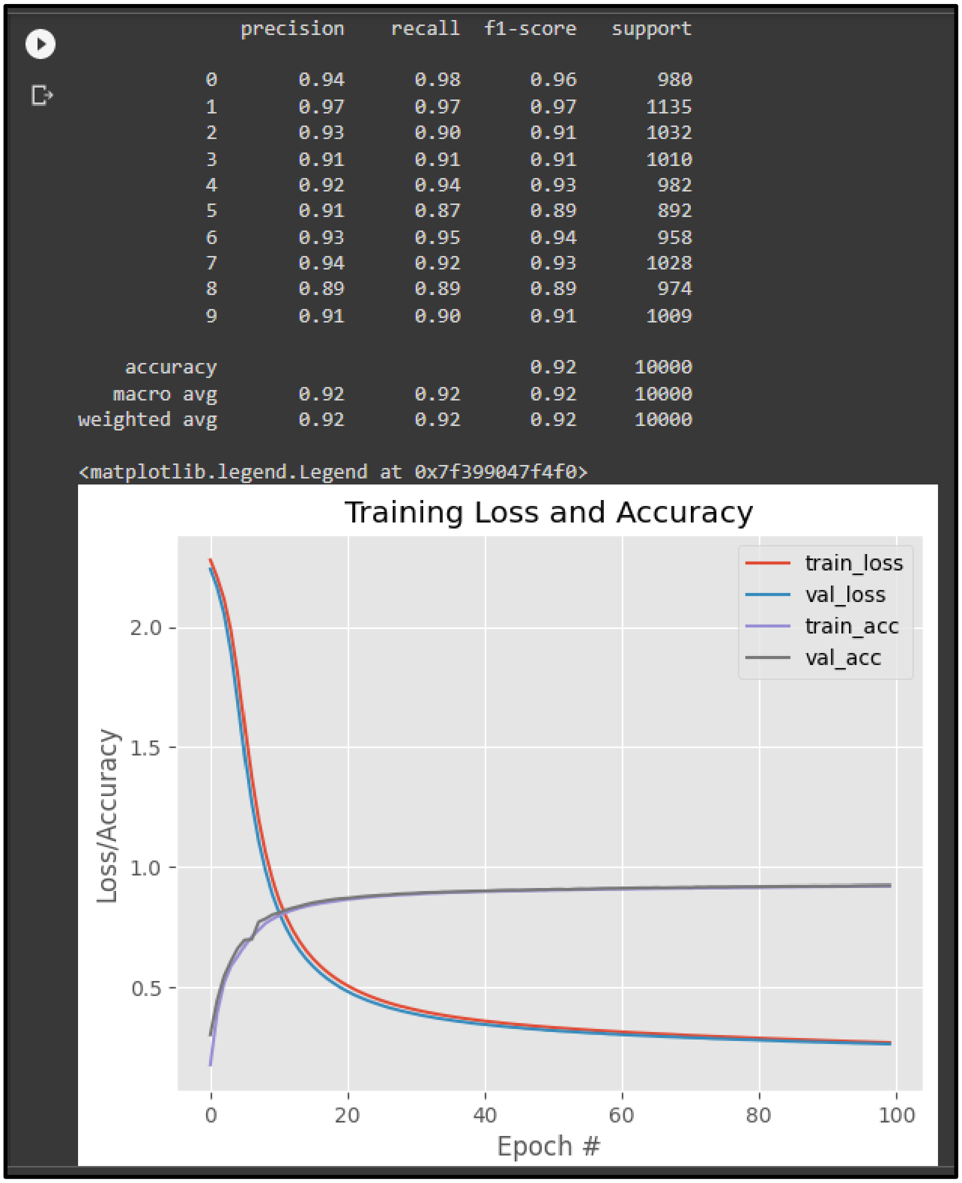
```
# import the necessary packages
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.datasets import mnist
from tensorflow.keras import backend as K
import matplotlib.pyplot as plt
import numpy as np
import argparse
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-o", "--output", required=True,help="path to the output loss/accuracy plot")
args = vars(ap.parse_args())

# grab the MNIST dataset (if this is your first time using this
# dataset then the 11MB download may take a minute)
print("[INFO] accessing MNIST...")
((trainX, trainY), (testX, testY)) = mnist.load_data()
# each image in the MNIST dataset is represented as a 28x28x1
# image, but in order to apply a standard neural network we must
# first "flatten" the image to be simple list of 28x28=784 pixels
trainX = trainX.reshape((trainX.shape[0], 28 * 28 * 1))
testX = testX.reshape((testX.shape[0], 28 * 28 * 1))
# scale data to the range of [0, 1]
trainX = trainX.astype("float32") / 255.0
testX = testX.astype("float32") / 255.0

# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)
```

```
model = Sequential()
model.add(Dense(256, input_shape=(784,), activation="sigmoid"))
model.add(Dense(128, activation="sigmoid"))
model.add(Dense(10, activation="softmax"))

print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd,     metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY),          epochs=100, batch_size=128)
```



```
[INFO] training network...
Epoch 1/100
469/469 [==============================] - 4s 8ms/step - loss: 2.2798 - accuracy: 0.1753 - val_loss: 2.2416 - val_accuracy: 0.3011
Epoch 2/100
469/469 [==============================] - 6s 13ms/step - loss: 2.2074 - accuracy: 0.3917 - val_loss: 2.1643 - val_accuracy: 0.4448
Epoch 3/100
469/469 [==============================] - 4s 9ms/step - loss: 2.1176 - accuracy: 0.5159 - val_loss: 2.0556 - val_accuracy: 0.5449
Epoch 4/100
469/469 [==============================] - 3s 7ms/step - loss: 1.9872 - accuracy: 0.5869 - val_loss: 1.8971 - val_accuracy: 0.6076
Epoch 5/100
469/469 [==============================] - 4s 9ms/step - loss: 1.8047 - accuracy: 0.6278 - val_loss: 1.6870 - val_accuracy: 0.6620
```

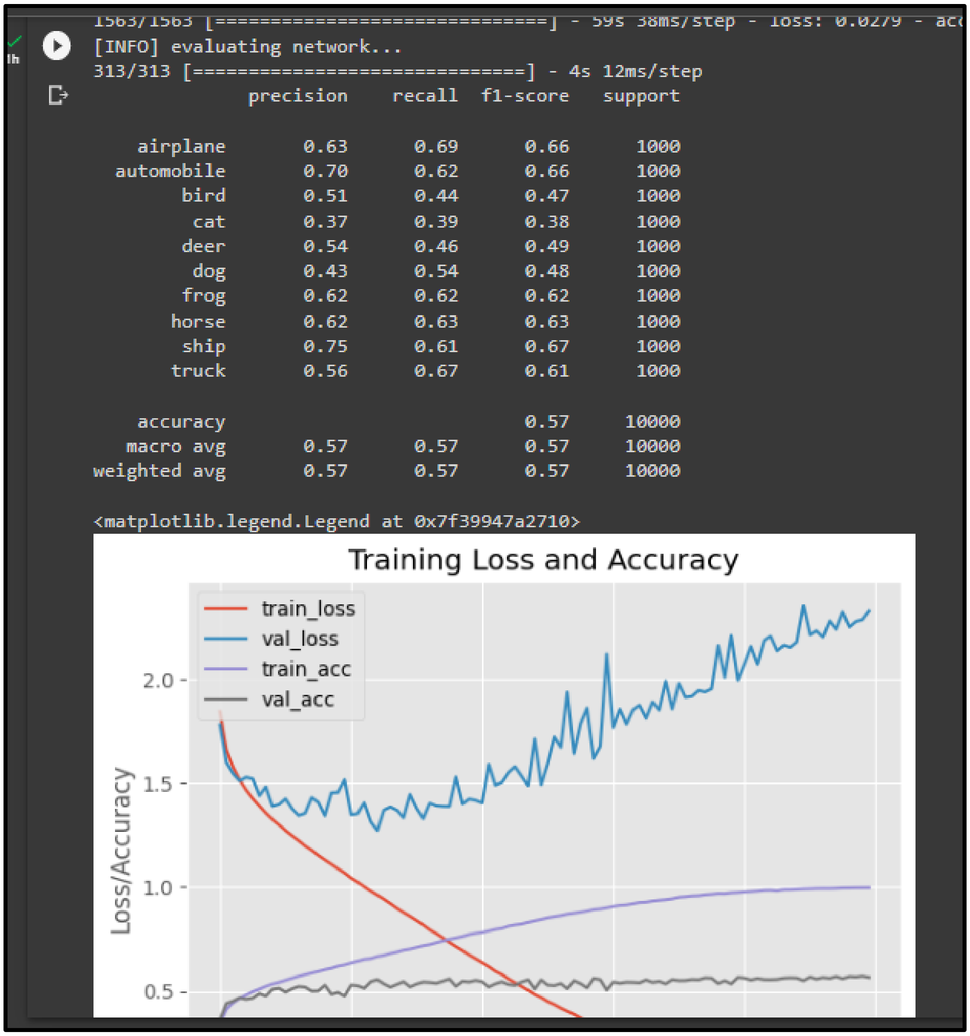
```
# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=128)
print(classification_report(testY.argmax(axis=1),     predictions.argmax(axis=1),
        target_names=[str(x) for x in lb.classes_]))

plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 100), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 100), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 100), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 100), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```



```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       980
           1       0.97      0.97      0.97      1135
           2       0.93      0.90      0.91      1032
           3       0.91      0.91      0.91      1010
           4       0.92      0.94      0.93       982
           5       0.91      0.87      0.89       892
           6       0.93      0.95      0.94       958
           7       0.94      0.92      0.93      1028
           8       0.89      0.89      0.89       974
           9       0.91      0.90      0.91      1009

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000

<matplotlib.legend.Legend at 0x7f399047f4f0>
```

```
# import the necessary packages
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt
import numpy as np
import argparse

print("[INFO] loading CIFAR-10 data...")
((trainX, trainY), (testX, testY)) = cifar10.load_data()
```

```
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0
trainX = trainX.reshape((trainX.shape[0], 3072))
testX = testX.reshape((testX.shape[0], 3072))

lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)
# initialize the label names for the CIFAR-10 dataset
labelNames = ["airplane", "automobile", "bird", "cat", "deer","dog", "frog", "horse", "ship",
"truck"]

model = Sequential()
model.add(Dense(1024, input_shape=(3072,), activation="relu"))
model.add(Dense(512, activation="relu"))
model.add(Dense(10, activation="softmax"))

print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd,     metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY),          epochs=100, batch_size=32)

print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),predictions.argmax(axis=1),
target_names=labelNames))

plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 100), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 100), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 100), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 100), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```



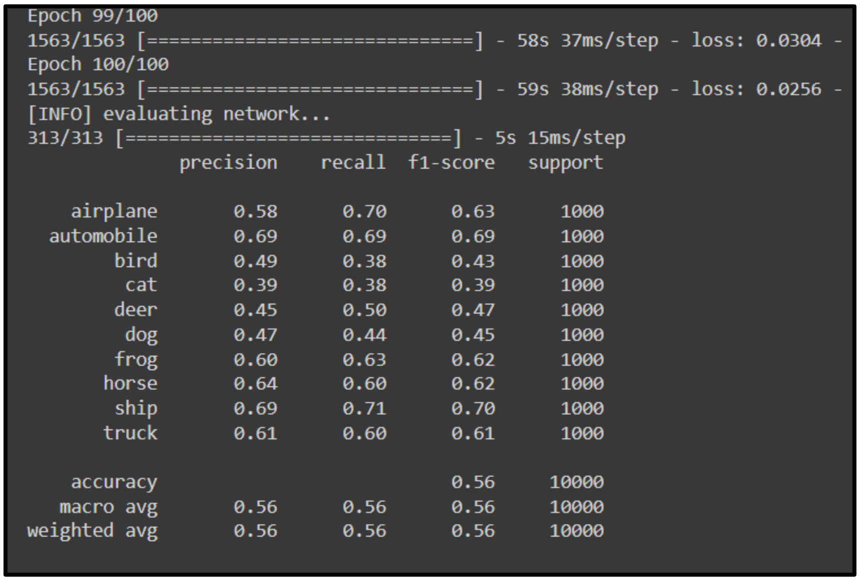
```
1563/1563 [==============================] - 59s 38ms/step - loss: 0.0279 - acc
[INFO] evaluating network...
313/313 [==============================] - 4s 12ms/step
              precision    recall  f1-score   support

    airplane       0.63      0.69      0.66      1000
  automobile       0.70      0.62      0.66      1000
        bird       0.51      0.44      0.47      1000
         cat       0.37      0.39      0.38      1000
        deer       0.54      0.46      0.49      1000
         dog       0.43      0.54      0.48      1000
        frog       0.62      0.62      0.62      1000
       horse       0.62      0.63      0.63      1000
        ship       0.75      0.61      0.67      1000
       truck       0.56      0.67      0.61      1000

    accuracy                           0.57     10000
   macro avg       0.57      0.57      0.57     10000
weighted avg       0.57      0.57      0.57     10000

<matplotlib.legend.Legend at 0x7f39947a2710>
```

from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt
import numpy as np
import argparse

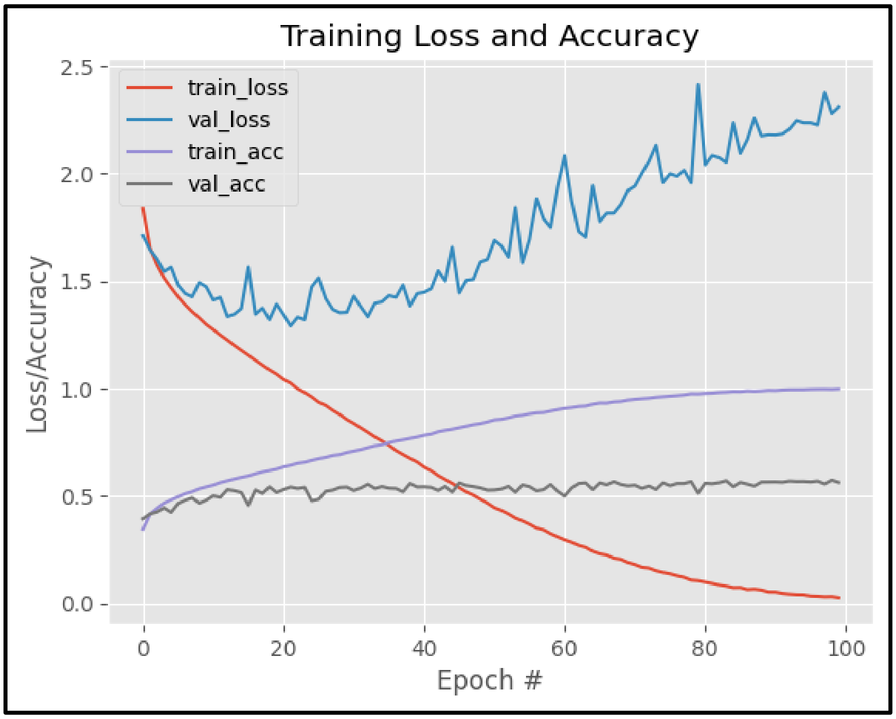
# load the training and testing data, scale it into the range [0, 1],
# then reshape the design matrix
print("[INFO] loading CIFAR-10 data...")
((trainX, trainY), (testX, testY)) = cifar10.load_data()
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0

```
trainX = trainX.reshape((trainX.shape[0], 3072))
testX = testX.reshape((testX.shape[0], 3072))

# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)
# initialize the label names for the CIFAR-10 dataset
labelNames = ["airplane", "automobile", "bird", "cat", "deer",
        "dog", "frog", "horse", "ship", "truck"]

# define the 3072-1024-512-10 architecture using Keras
model = Sequential()
model.add(Dense(1024, input_shape=(3072,), activation="relu"))
model.add(Dense(512, activation="relu"))
model.add(Dense(10, activation="softmax"))

# train the model using SGD
print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd,     metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY),          epochs=100, batch_size=32)

print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),
        predictions.argmax(axis=1), target_names=labelNames))

# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 100), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 100), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 100), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 100), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig(args["output"])
```

**Output:**



```
Epoch 99/100
1563/1563 [==============================] - 58s 37ms/step - loss: 0.0304 -
Epoch 100/100
1563/1563 [==============================] - 59s 38ms/step - loss: 0.0256 -
[INFO] evaluating network...
313/313 [==============================] - 5s 15ms/step
              precision    recall  f1-score   support

    airplane       0.58      0.70      0.63      1000
  automobile       0.69      0.69      0.69      1000
        bird       0.49      0.38      0.43      1000
         cat       0.39      0.38      0.39      1000
        deer       0.45      0.50      0.47      1000
         dog       0.47      0.44      0.45      1000
        frog       0.60      0.63      0.62      1000
       horse       0.64      0.60      0.62      1000
        ship       0.69      0.71      0.70      1000
       truck       0.61      0.60      0.61      1000

    accuracy                           0.56     10000
   macro avg       0.56      0.56      0.56     10000
weighted avg       0.56      0.56      0.56     10000
```



# Practical 2

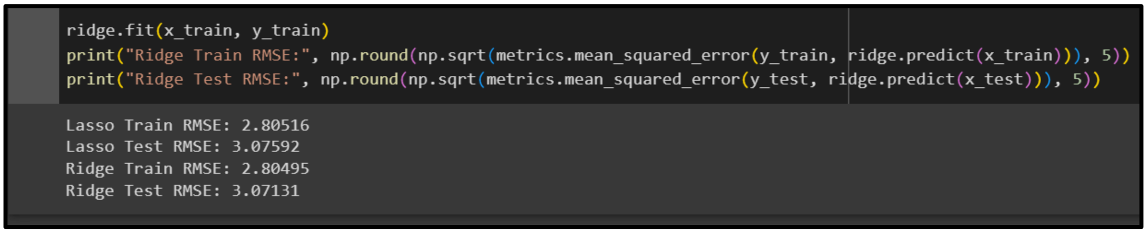
**Aim: Write a Program to implement regularization to prevent the model from overfitting**

**Source Code:**

```
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.linear_model import Lasso

df_train = pd.read_csv('/content/sample_data/train.csv')
df_test = pd.read_csv('/content/sample_data/test.csv')

df_train = df_train.dropna()
df_test = df_test.dropna()

x_train = df_train['x']
x_train = x_train.values.reshape(-1,1)
y_train = df_train['y']
y_train = y_train.values.reshape(-1,1)

x_test = df_test['x']
x_test = x_test.values.reshape(-1,1)
y_test = df_test['y']
y_test = y_test.values.reshape(-1,1)

lasso = Lasso()

lasso.fit(x_train, y_train)

print("Lasso Train RMSE:", np.round(np.sqrt(metrics.mean_squared_error(y_train,
lasso.predict(x_train))), 5))
print("Lasso Test RMSE:", np.round(np.sqrt(metrics.mean_squared_error(y_test,
lasso.predict(x_test))), 5))

import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.linear_model import Ridge

df_train = pd.read_csv('/content/sample_data/train.csv')
```

```
df_test = pd.read_csv('/content/sample_data/test.csv')

df_train = df_train.dropna()
df_test = df_test.dropna()

x_train = df_train['x']
x_train = x_train.values.reshape(-1,1)
y_train = df_train['y']
y_train = y_train.values.reshape(-1,1)

x_test = df_test['x']
x_test = x_test.values.reshape(-1,1)
y_test = df_test['y']
y_test = y_test.values.reshape(-1,1)

ridge = Ridge()

ridge.fit(x_train, y_train)
print("Ridge Train RMSE:", np.round(np.sqrt(metrics.mean_squared_error(y_train,
ridge.predict(x_train))), 5))
print("Ridge Test RMSE:", np.round(np.sqrt(metrics.mean_squared_error(y_test,
ridge.predict(x_test))), 5))
```

**Output:**



```
ridge.fit(x_train, y_train)
print("Ridge Train RMSE:", np.round(np.sqrt(metrics.mean_squared_error(y_train, ridge.predict(x_train))), 5))
print("Ridge Test RMSE:", np.round(np.sqrt(metrics.mean_squared_error(y_test, ridge.predict(x_test))), 5))
```

```
Lasso Train RMSE: 2.80516
Lasso Test RMSE: 3.07592
Ridge Train RMSE: 2.80495
Ridge Test RMSE: 3.07131
```

# Practical 3

**Aim: Implement deep learning for recognizing classes for datasets like CIFAR-10 images for previously unseen images and assign them to one of the 10 classes**

**Source Code:**

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np

(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape

X_test.shape

y_train.shape

y_train[:5]

y_train = y_train.reshape(-1,)
y_train[:5]

y_test = y_test.reshape(-1,)

classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]

def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])

plot_sample(X_train, y_train, 0)

X_train = X_train / 255.0
X_test = X_test / 255.0

ann = models.Sequential([
                layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)

from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))

cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

cnn.fit(X_train, y_train, epochs=10)

cnn.evaluate(X_test,y_test)

y_pred = cnn.predict(X_test)
y_pred[:5]

y_classes = [np.argmax(element) for element in y_pred]
```

y_classes[:5]

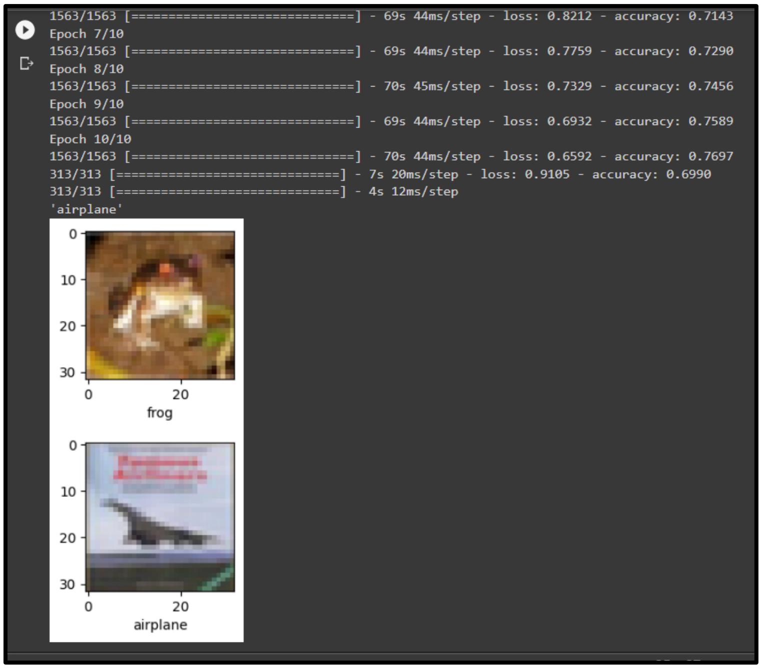
y_test[:5]

plot_sample(X_test, y_test,3)

classes[y_classes[3]]

classes[y_classes[3]]

**Output:**



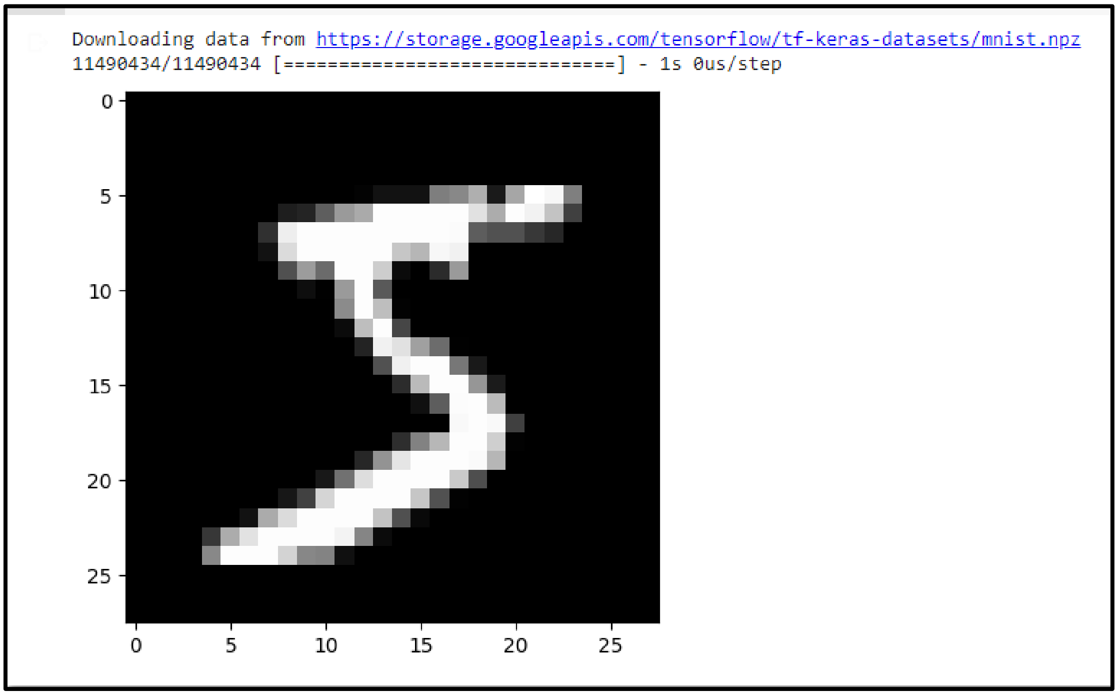
```
1563/1563 [==============================] - 69s 44ms/step - loss: 0.8212 - accuracy: 0.7143
Epoch 7/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.7759 - accuracy: 0.7290
Epoch 8/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.7329 - accuracy: 0.7456
Epoch 9/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.6932 - accuracy: 0.7589
Epoch 10/10
1563/1563 [==============================] - 70s 44ms/step - loss: 0.6592 - accuracy: 0.7697
313/313 [==============================] - 7s 20ms/step - loss: 0.9105 - accuracy: 0.6990
313/313 [==============================] - 4s 12ms/step
'airplane'
```

# Practical 4

**Aim: Implement deep learning for the Prediction of the autoencoder from the test data (e.g. MNIST data set)**

**Source Code:**

```
!pip install tensorflow-gpu==2.0.0b1
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Dense, Input, Flatten,\
                               Reshape, LeakyReLU as LR,\
                               Activation, Dropout
from tensorflow.keras.models import Model, Sequential
from matplotlib import pyplot as plt
from IPython import display # If using IPython, Colab or Jupyter
import numpy as np

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train/255.0
x_test = x_test/255.0
# Plot image data from x_train
plt.imshow(x_train[0], cmap = "gray")
plt.show()
```

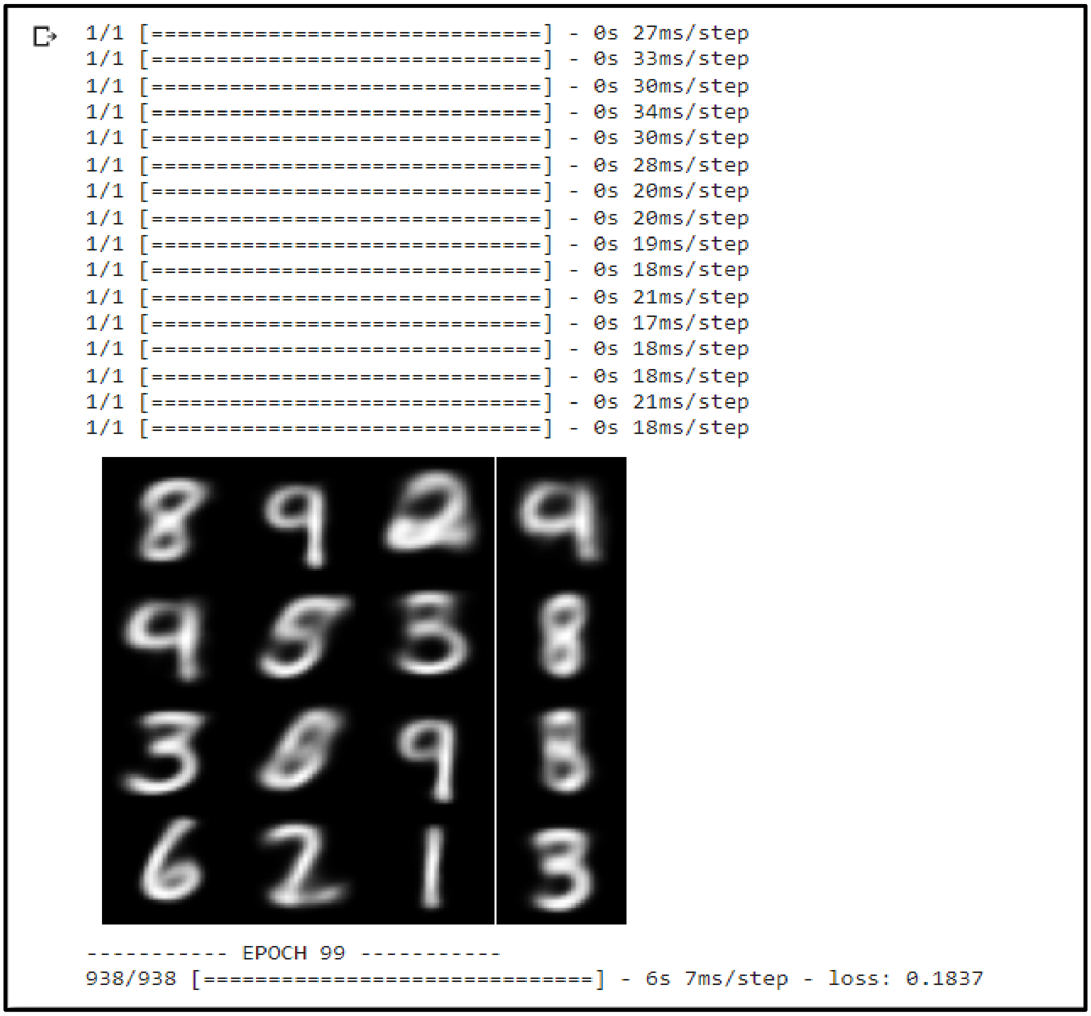


```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```

```
LATENT_SIZE = 32
```

```
encoder = Sequential([
    Flatten(input_shape = (28, 28)),
    Dense(512),
    LR(),
    Dropout(0.5),
    Dense(256),
    LR(),
    Dropout(0.5),
    Dense(128),
    LR(),
    Dropout(0.5),
    Dense(64),
    LR(),
    Dropout(0.5),
    Dense(LATENT_SIZE, activation="sigmoid"),
])

decoder = Sequential([
    Dense(64, input_shape = (LATENT_SIZE,)),
    LR(),
    Dropout(0.5),
    Dense(128),
    LR(),
    Dropout(0.5),
    Dense(256),
    LR(),
    Dropout(0.5),
    Dense(512),
    LR(),
    Dropout(0.5),
    Dense(784),
    Activation("sigmoid"),
    Reshape((28, 28))
])

img = Input(shape = (28, 28))
latent_vector = encoder(img)
output = decoder(latent_vector)
model = Model(inputs = img, outputs = output)
model.compile("nadam", loss = "binary_crossentropy")
```

```
EPOCHS = 100
#Only do plotting if you have IPython, Jupyter, or using Colab
for epoch in range(EPOCHS):
    fig, axs = plt.subplots(4, 4, figsize=(4,4))
    rand = x_test[np.random.randint(0, 10000, 16)].reshape((4, 4, 1, 28, 28))

    display.clear_output() # If you imported display from IPython

    for i in range(4):
        for j in range(4):
            axs[i, j].imshow(model.predict(rand[i, j])[0], cmap = "gray")
            axs[i, j].axis("off")

    plt.subplots_adjust(wspace = 0, hspace = 0)
    plt.show()
    print("-----------", "EPOCH", epoch, "-----------")
    model.fit(x_train, x_train, batch_size = 64)
```



```
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 18ms/step
```

```
----------- EPOCH 99 -----------
938/938 [==============================] - 6s 7ms/step - loss: 0.1837
```

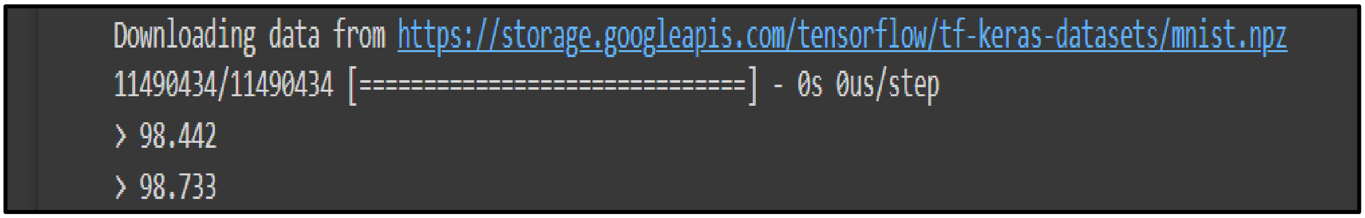
# Practical 5

**Aim: Implement Convolutional Neural Network for Digit Recognition on the MNIST Dataset.**

**Source Code:**

```
# baseline cnn model for mnist
from numpy import mean
from numpy import std
from matplotlib import pyplot as plt
from sklearn.model_selection import KFold
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD

# load train and test dataset
def load_dataset():
 # load dataset
 (trainX, trainY), (testX, testY) = mnist.load_data()
 # reshape dataset to have a single channel
 trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
 testX = testX.reshape((testX.shape[0], 28, 28, 1))
 # one hot encode target values
 trainY = to_categorical(trainY)
 testY = to_categorical(testY)
 return trainX, trainY, testX, testY

# scale pixels
def prep_pixels(train, test):
 # convert from integers to floats
 train_norm = train.astype('float32')
 test_norm = test.astype('float32')
 # normalize to range 0-1
 train_norm = train_norm / 255.0
 test_norm = test_norm / 255.0
 # return normalized images
 return train_norm, test_norm

# define cnn model
def define_model():
 model = Sequential()
 model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
input_shape=(28, 28, 1)))
 model.add(MaxPooling2D((2, 2)))
 model.add(Flatten())
 model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
 model.add(Dense(10, activation='softmax'))
 # compile model
 opt = SGD(learning_rate=0.01, momentum=0.9)
 model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
 return model

# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
 scores, histories = list(), list()
 # prepare cross validation
 kfold = KFold(n_folds, shuffle=True, random_state=1)
 # enumerate splits
 for train_ix, test_ix in kfold.split(dataX):
 # define model
  model = define_model()
 # select rows for train and test
  trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
 # fit model
  history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY),
verbose=0)
 # evaluate model
  _, acc = model.evaluate(testX, testY, verbose=0)
  print('> %.3f' % (acc * 100.0))
 # stores scores
 scores.append(acc)
 histories.append(history)
 return scores, histories

def summarize_diagnostics(histories):
 for i in range(len(histories)):
  # plot loss
  plt.subplot(2, 1, 1)
  plt.title('Cross Entropy Loss')
  plt.plot(histories[i].history['loss'], color='blue', label='train')
  plt.plot(histories[i].history['val_loss'], color='orange', label='test')
  # plot accuracy
  plt.subplot(2, 1, 2)
  plt.title('Classification Accuracy')
  plt.plot(histories[i].history['accuracy'], color='blue', label='train')
  plt.plot(histories[i].history['val_accuracy'], color='orange', label='test')
  plt.show()

# summarize model performance
def summarize_performance(scores):
 # print summary
 print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scores)))
 # box and whisker plots of results
 plt.boxplot(scores)
 plt.show()

def run_test_harness():
 # load dataset
 trainX, trainY, testX, testY = load_dataset()
 # prepare pixel data
 trainX, testX = prep_pixels(trainX, testX)
 # evaluate model
 scores, histories = evaluate_model(trainX, trainY)
 # learning curves
 summarize_diagnostics(histories)
 # summarize estimated performance
 summarize_performance(scores)

run_test_harness()
```

**Output:**



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
> 98.442
> 98.733
```

# Practical 6

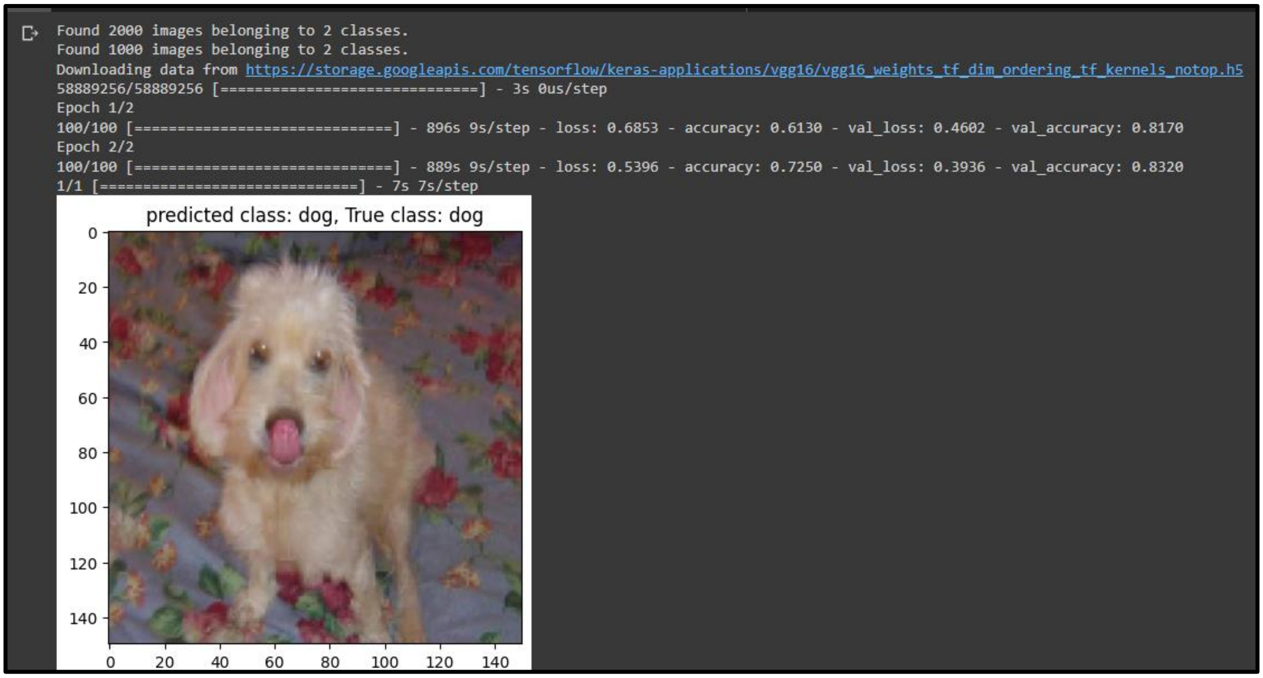
**Aim**: **Write a program to implement Transfer Learning on the suitable dataset (e.g. classify the cats versus dogs dataset from Kaggle).**

**Source Code:**

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
import zipfile
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16

filename = "cats_and_dogs_filtered.zip"

with zipfile.ZipFile("/content/drive/MyDrive/cats_and_dogs_filtered.zip", "r") as zip_ref :
  zip_ref.extractall()

train_dir = os.path.join(os.getcwd(),"cats_and_dogs_filtered","train")
train_dir

validation_dir = os.path.join(os.getcwd(),"cats_and_dogs_filtered","validation")

train_datagen = ImageDataGenerator(rescale=1./255,rotation_range=20,width_shift_range=0.2,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)

validate_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir,target_size=(150,150),batch_size=20,class_mode="binary")

validation_generator = validate_datagen.flow_from_directory(validation_dir,target_size=(150,150),batch_size=20,class_mode="binary")

conv_base = VGG16(weights="imagenet",include_top=False, input_shape=(150,150,3))

conv_base.trainable = False
```

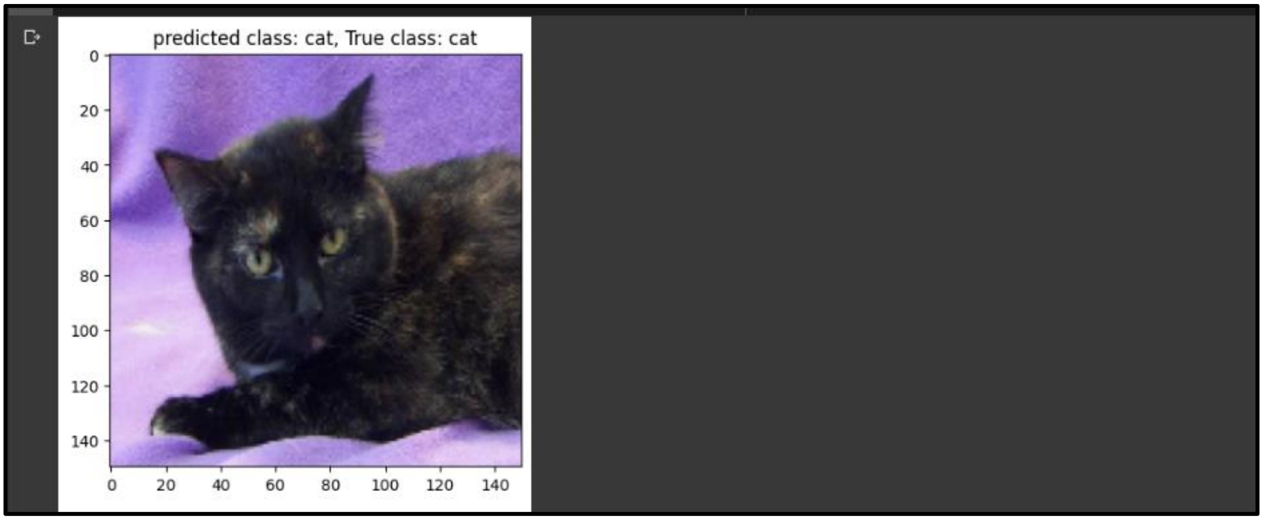
```
model = tf.keras.models.Sequential()
model.add(conv_base)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256,activation ="relu"))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(1,activation = "sigmoid"))

model.compile(loss="binary_crossentropy",optimizer=tf.keras.optimizers.RMSprop(learning_rate=2e-5),metrics=["accuracy"])

history = model.fit(train_generator,steps_per_epoch=100, epochs=2,
validation_data=validation_generator,validation_steps=50)

x, y_true = next(validation_generator)
y_pred = model.predict(x)
class_names = ['cat', 'dog']
for i in range(len(x)):
  plt.imshow(x[i])
  plt.title(f'predicted class: {class_names[int(round(y_pred[i][0]))]}, True class: {class_names[int(y_true[i])]}')
  plt.show()

acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]

epochs = range(1, len(acc) +1)
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="validation acc")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```

**Output:**

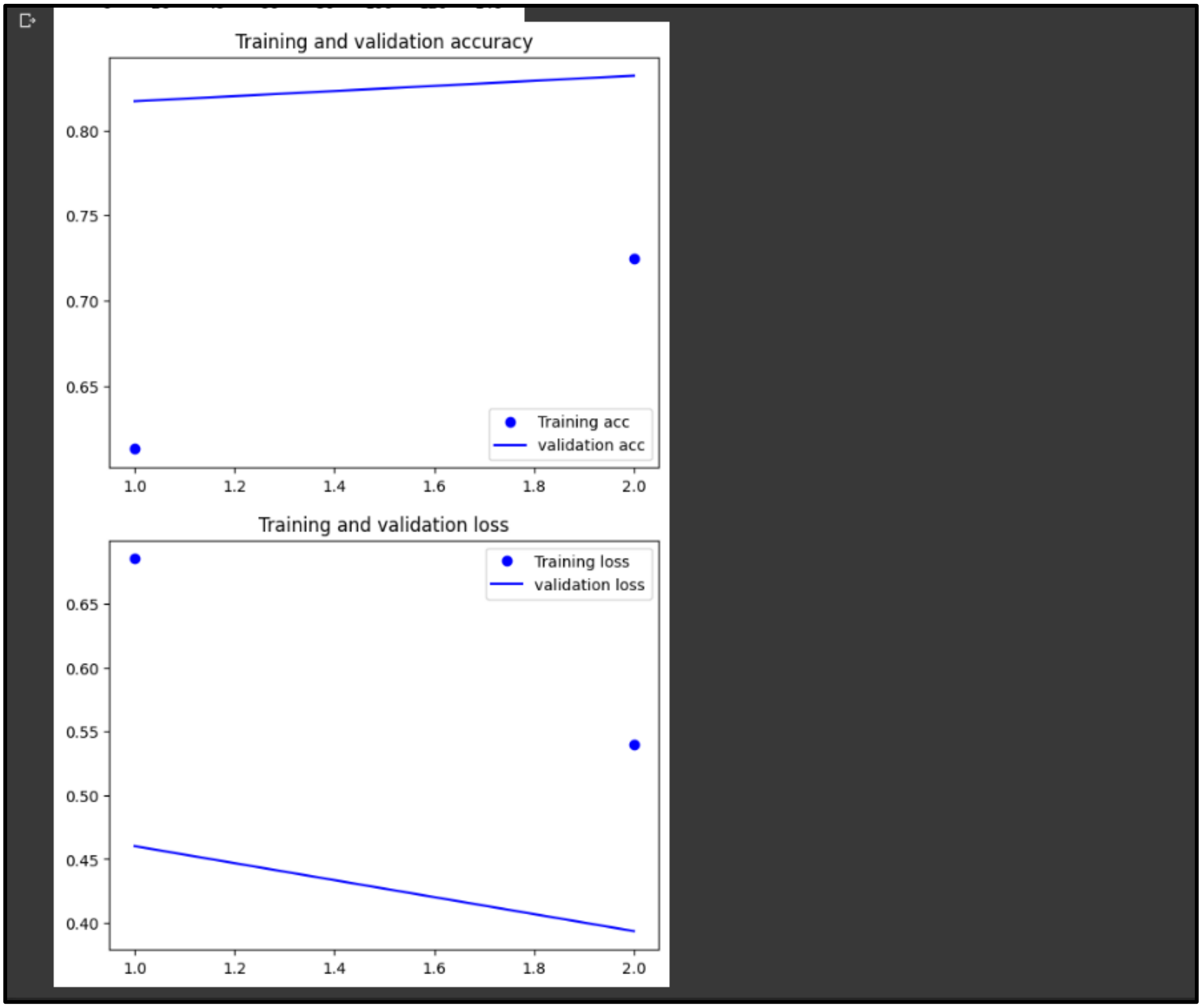


```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 3s 0us/step
Epoch 1/2
100/100 [==============================] - 896s 9s/step - loss: 0.6853 - accuracy: 0.6130 - val_loss: 0.4602 - val_accuracy: 0.8170
Epoch 2/2
100/100 [==============================] - 889s 9s/step - loss: 0.5396 - accuracy: 0.7250 - val_loss: 0.3936 - val_accuracy: 0.8320
1/1 [==============================] - 7s 7s/step
```

predicted class: dog, True class: dog



predicted class: cat, True class: cat


Training and validation accuracy
0.80
0.75
0.70
0.65
● Training acc
— validation acc
1.0
1.2
1.4
1.6
1.8
2.0
Training and validation loss
● Training loss
— validation loss
0.65
0.60
0.55
0.50
0.45
0.40
1.0
1.2
1.4
1.6
1.8
2.0

# Practical 7

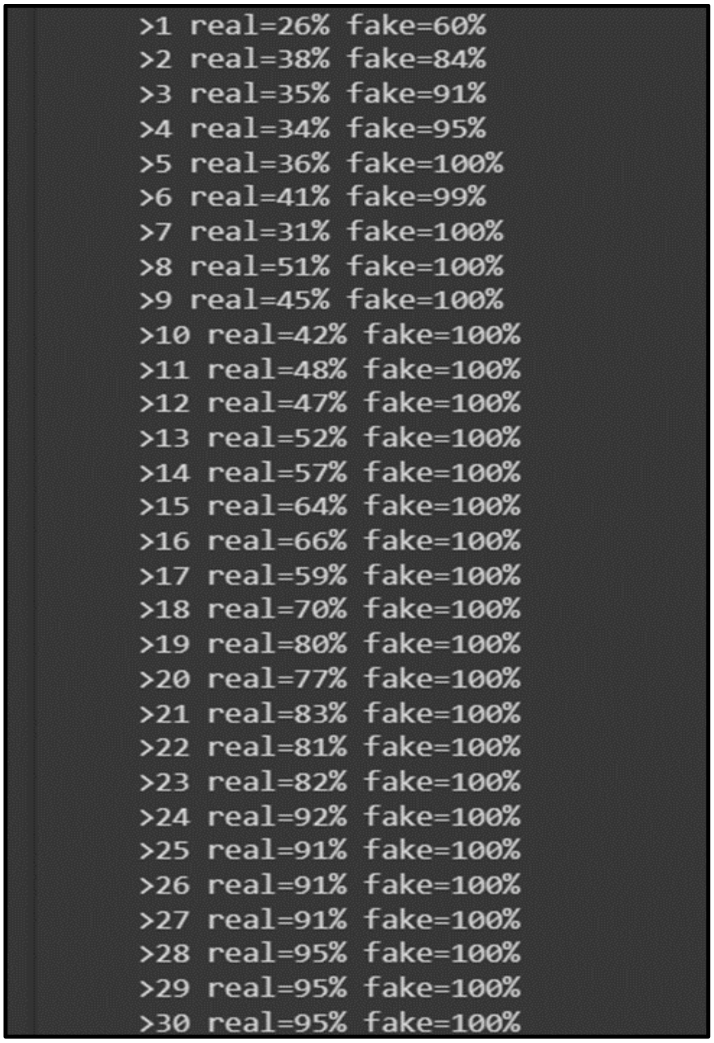
**Aim: Write a program for the Implementation of a Generative Adversarial Network for generating synthetic shapes (like digits)**

**Source Code:**

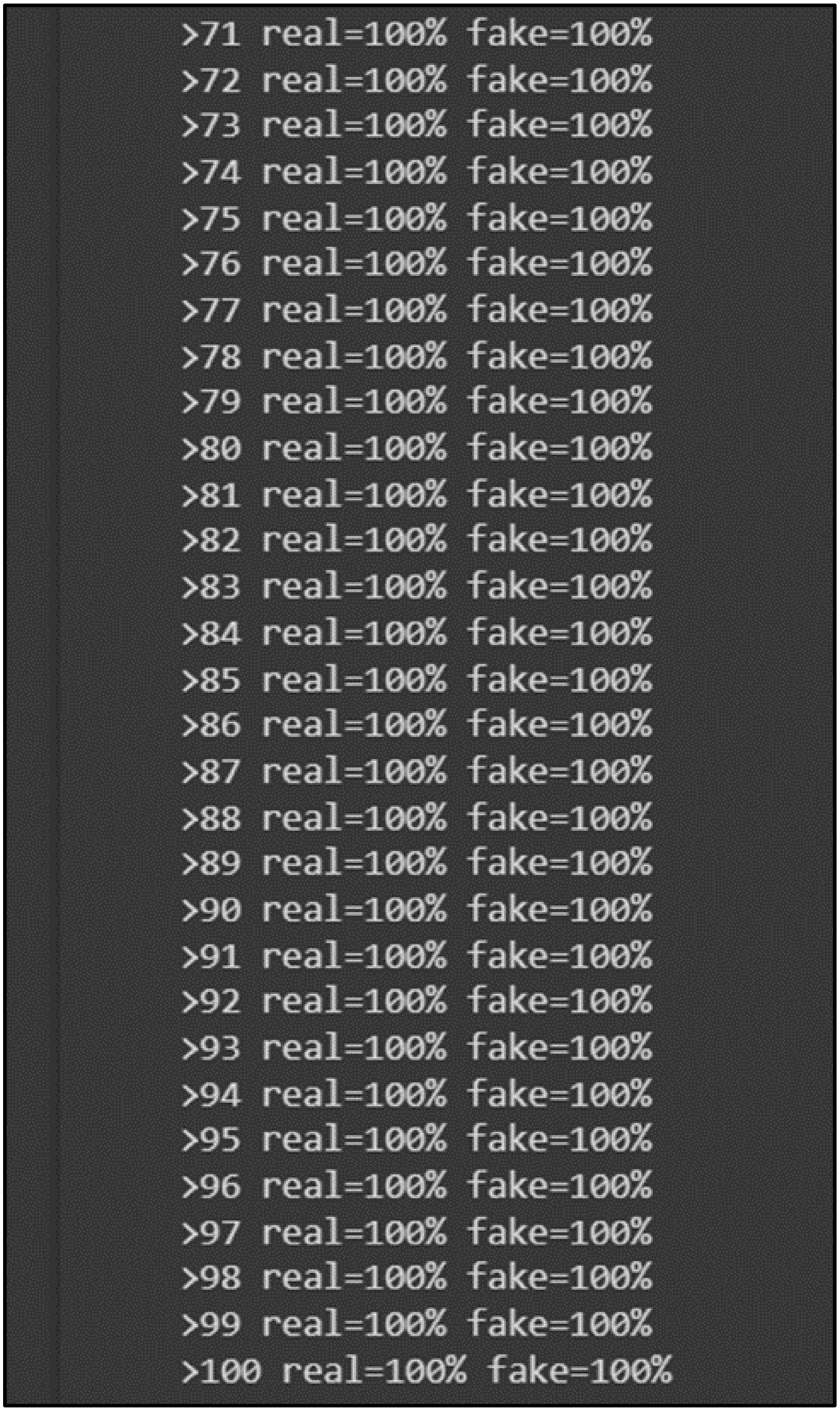
```
from numpy import expand_dims
from numpy import ones
from numpy import zeros
from numpy.random import rand
from numpy.random import randint
from keras.datasets.mnist import load_data
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LeakyReLU

# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,1)):
 model = Sequential()
 model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same', input_shape=in_shape))
 model.add(LeakyReLU(alpha=0.2))
 model.add(Dropout(0.4))
 model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same'))
 model.add(LeakyReLU(alpha=0.2))
 model.add(Dropout(0.4))
 model.add(Flatten())
 model.add(Dense(1, activation='sigmoid'))
 # compile model
 opt = Adam(lr=0.0002, beta_1=0.5)
 model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
 return model

# load and prepare mnist training images
def load_real_samples():
 # load mnist dataset
 (trainX, _), (_, _) = load_data()
 # expand to 3d, e.g. add channels dimension
 X = expand_dims(trainX, axis=-1)
 # convert from unsigned ints to floats
 X = X.astype('float32')
 # scale from [0,255] to [0,1]
 X = X / 255.0
 return X

# select real samples
def generate_real_samples(dataset, n_samples):
 # choose random instances
 ix = randint(0, dataset.shape[0], n_samples)
 # retrieve selected images
 X = dataset[ix]
 # generate 'real' class labels (1)
 y = ones((n_samples, 1))
 return X, y

# generate n fake samples with class labels
def generate_fake_samples(n_samples):
 # generate uniform random numbers in [0,1]
 X = rand(28 * 28 * n_samples)
 # reshape into a batch of grayscale images
 X = X.reshape((n_samples, 28, 28, 1))
 # generate 'fake' class labels (0)
 y = zeros((n_samples, 1))
 return X, y

# train the discriminator model
def train_discriminator(model, dataset, n_iter=100, n_batch=256):
 half_batch = int(n_batch / 2)
 # manually enumerate epochs
 for i in range(n_iter):
 # get randomly selected 'real' samples
  X_real, y_real = generate_real_samples(dataset, half_batch)
 # update discriminator on real samples
  _, real_acc = model.train_on_batch(X_real, y_real)
 # generate 'fake' examples
  X_fake, y_fake = generate_fake_samples(half_batch)
 # update discriminator on fake samples
```

```
  _, fake_acc = model.train_on_batch(X_fake, y_fake)
 # summarize performance
  print('>%d real=%.0f%% fake=%.0f%%' % (i+1, real_acc*100, fake_acc*100))

# define the discriminator model
model = define_discriminator()
# load image data
dataset = load_real_samples()
# fit the model
train_discriminator(model, dataset)
```

**Output:**



```
>1 real=26% fake=60%
>2 real=38% fake=84%
>3 real=35% fake=91%
>4 real=34% fake=95%
>5 real=36% fake=100%
>6 real=41% fake=99%
>7 real=31% fake=100%
>8 real=51% fake=100%
>9 real=45% fake=100%
>10 real=42% fake=100%
>11 real=48% fake=100%
>12 real=47% fake=100%
>13 real=52% fake=100%
>14 real=57% fake=100%
>15 real=64% fake=100%
>16 real=66% fake=100%
>17 real=59% fake=100%
>18 real=70% fake=100%
>19 real=80% fake=100%
>20 real=77% fake=100%
>21 real=83% fake=100%
>22 real=81% fake=100%
>23 real=82% fake=100%
>24 real=92% fake=100%
>25 real=91% fake=100%
>26 real=91% fake=100%
>27 real=91% fake=100%
>28 real=95% fake=100%
>29 real=95% fake=100%
>30 real=95% fake=100%
```



```
>71 real=100% fake=100%
>72 real=100% fake=100%
>73 real=100% fake=100%
>74 real=100% fake=100%
>75 real=100% fake=100%
>76 real=100% fake=100%
>77 real=100% fake=100%
>78 real=100% fake=100%
>79 real=100% fake=100%
>80 real=100% fake=100%
>81 real=100% fake=100%
>82 real=100% fake=100%
>83 real=100% fake=100%
>84 real=100% fake=100%
>85 real=100% fake=100%
>86 real=100% fake=100%
>87 real=100% fake=100%
>88 real=100% fake=100%
>89 real=100% fake=100%
>90 real=100% fake=100%
>91 real=100% fake=100%
>92 real=100% fake=100%
>93 real=100% fake=100%
>94 real=100% fake=100%
>95 real=100% fake=100%
>96 real=100% fake=100%
>97 real=100% fake=100%
>98 real=100% fake=100%
>99 real=100% fake=100%
>100 real=100% fake=100%
```

# Practical 8

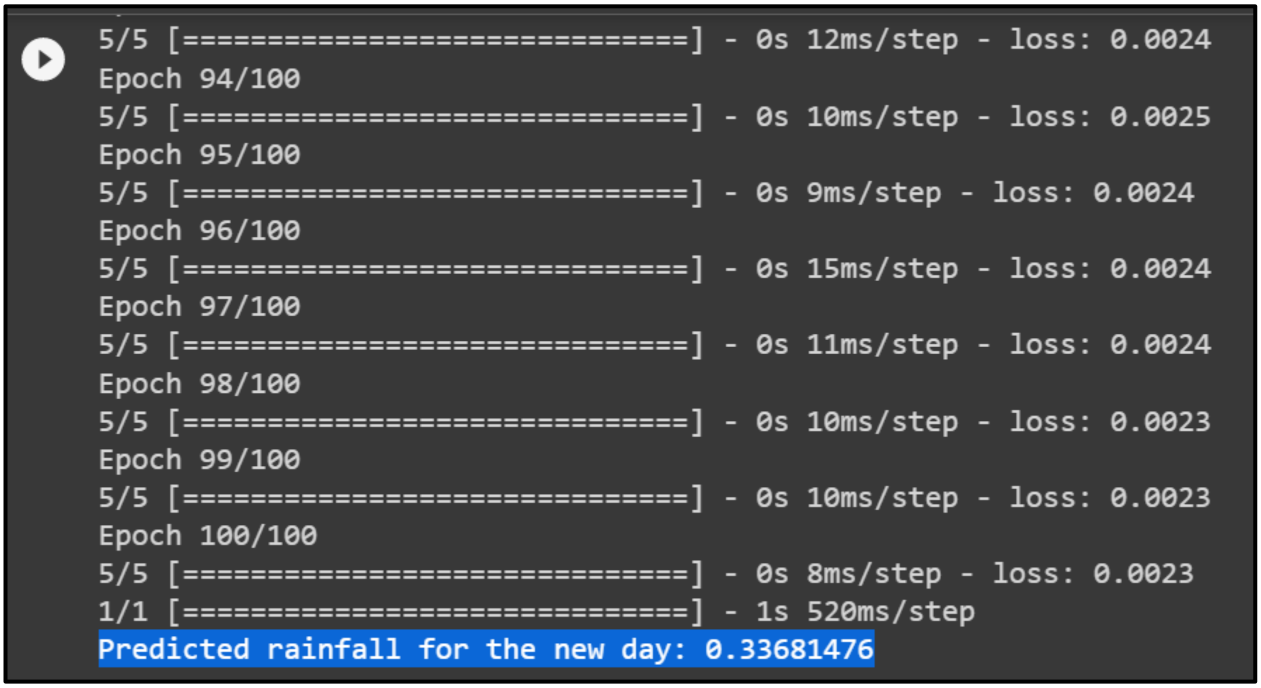
**Aim: Write a program to implement a simple form of a recurrent neural network.**

1.**(4-to-1 RNN) to show that the quantity of rain on a certain day also depends on the values of the previous day**

**Source Code:**

```
import numpy as np
import tensorflow as tf

# Define the training data
rainfall_data = np.array([[0.2, 0.3, 0.1, 0.5, 0.4],
                          [0.1, 0.4, 0.5, 0.2, 0.3],
                          [0.3, 0.2, 0.4, 0.3, 0.1],
                          [0.4, 0.1, 0.3, 0.4, 0.2],
                          [0.5, 0.5, 0.2, 0.1, 0.5]])

# Prepare the input and output data
input_data = rainfall_data[:, :-1]  # Previous four days' rainfall values
output_data = rainfall_data[:, -1]  # Current day's rainfall value

# Define the RNN model
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(10, input_shape=(4, 1)),
    tf.keras.layers.Dense(1)
])

# Compile the model
model.compile(loss='mse', optimizer='adam')

# Train the model
model.fit(np.expand_dims(input_data, axis=2), output_data, epochs=100, batch_size=1)

# Predict the rainfall for a new day
new_input = np.array([[0.3, 0.2, 0.1, 0.4]])  # Previous four days' rainfall values
predicted_rainfall = model.predict(np.expand_dims(new_input, axis=2))
print("Predicted rainfall for the new day:", predicted_rainfall[0][0])
```

**Output:**



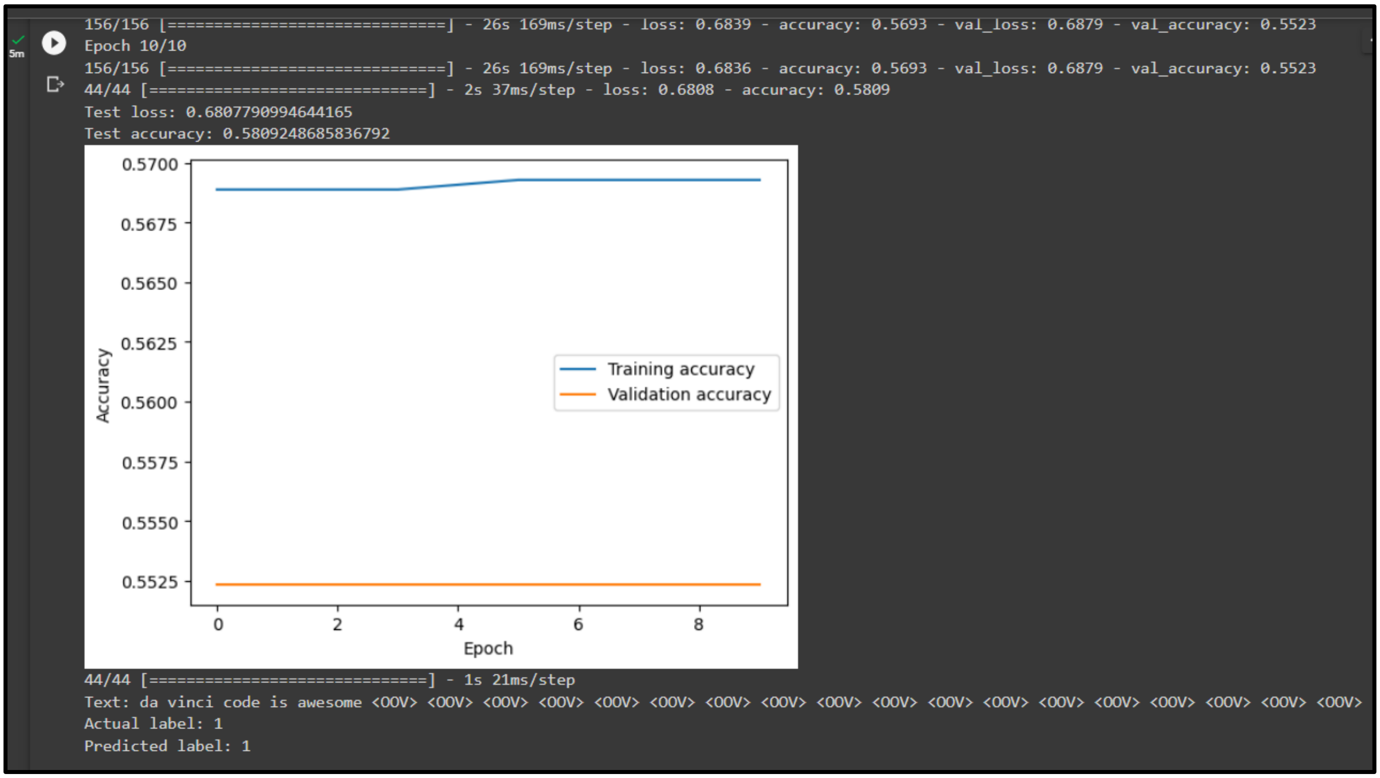
```
5/5 [==============================] - 0s 12ms/step - loss: 0.0024
Epoch 94/100
5/5 [==============================] - 0s 10ms/step - loss: 0.0025
Epoch 95/100
5/5 [==============================] - 0s 9ms/step - loss: 0.0024
Epoch 96/100
5/5 [==============================] - 0s 15ms/step - loss: 0.0024
Epoch 97/100
5/5 [==============================] - 0s 11ms/step - loss: 0.0024
Epoch 98/100
5/5 [==============================] - 0s 10ms/step - loss: 0.0023
Epoch 99/100
5/5 [==============================] - 0s 10ms/step - loss: 0.0023
Epoch 100/100
5/5 [==============================] - 0s 8ms/step - loss: 0.0023
1/1 [==============================] - 1s 520ms/step
Predicted rainfall for the new day: 0.33681476
```

**Source Code:**

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

data = pd.read_csv("/content/sample_data/training.txt", delimiter="\t", names=["label", "text"])

X_train, X_test, y_train, y_test = train_test_split(data["text"],data["label"], test_size=0.2,
random_state=42)

tokenizer = Tokenizer(num_words=5000, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

max_length = 100
```

```
X_train_pad = pad_sequences(X_train_seq, maxlen=max_length,
padding="post",truncating="post")
X_test_pad = pad_sequences(X_test_seq, maxlen=max_length,
padding="post",truncating="post")

model = tf.keras.models.Sequential([tf.keras.layers.Embedding(input_dim=5000,
output_dim=32,input_length=max_length),tf.keras.layers.LSTM(units=64, dropout=0.2,
recurrent_dropout=0.2),tf.keras.layers.Dense(1, activation="sigmoid")])

model.compile(optimizer="adam", loss="binary_crossentropy",metrics=["accuracy"])

history = model.fit(X_train_pad, y_train, epochs=10, batch_size=32,validation_split=0.1)

loss, accuracy = model.evaluate(X_test_pad, y_test)
print("Test loss:", loss)
print("Test accuracy:", accuracy)

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.show()

predictions = model.predict(X_test_pad)

index = np.random.randint(0, len(X_test_pad))
text = tokenizer.sequences_to_texts([X_test_pad[index]])[0]
label = y_test.values[index]
prediction = predictions[index][0]
print("Text:", text)
print("Actual label:", label)
print("Predicted label:", round(prediction))
```

**Output:**



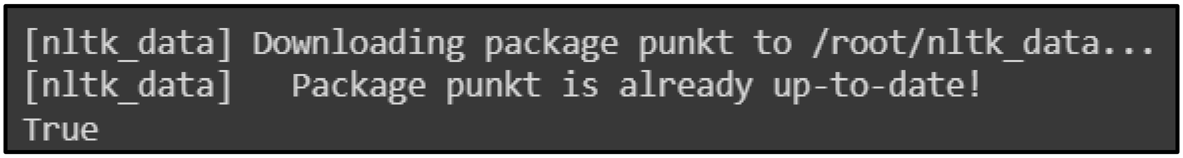
```
156/156 [==============================] - 26s 169ms/step - loss: 0.6839 - accuracy: 0.5693 - val_loss: 0.6879 - val_accuracy: 0.5523
Epoch 10/10
156/156 [==============================] - 26s 169ms/step - loss: 0.6836 - accuracy: 0.5693 - val_loss: 0.6879 - val_accuracy: 0.5523
44/44 [==============================] - 2s 37ms/step - loss: 0.6808 - accuracy: 0.5809
Test loss: 0.6807790994644165
Test accuracy: 0.5809248685836792
```

```
44/44 [==============================] - 1s 21ms/step
Text: da vinci code is awesome <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV>
Actual label: 1
Predicted label: 1
```

2.**LSTM for sentiment analysis on datasets like UMICH SI650 for similar.**

**Source Code:**

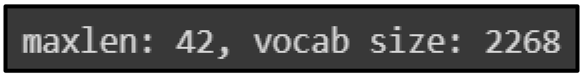
```
from __future__ import division, print_function
from keras.layers.core import Dense, Activation
from keras.layers import Embedding
from keras.layers import LSTM
from keras.models import Sequential
from keras.preprocessing import sequence
from sklearn.model_selection import train_test_split
import collections
import nltk
import numpy as np
nltk.download('punkt')
```



```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
maxlen = 0
```

```
word_freqs = collections.Counter()
num_recs = 0

ftrain = open("/content/umich-sentiment-train.txt", "r")  # Open the file in text mode

for line in ftrain:
    label, sentence = line.strip().split("\t")
    words = nltk.word_tokenize(sentence.lower())

    if len(words) > maxlen:
        maxlen = len(words)

    for word in words:
        word_freqs[word] += 1

    num_recs += 1

ftrain.close()

# print some statistics about our data, that will drive our parameters
print("maxlen: %d, vocab size: %d" % (maxlen, len(word_freqs)))
```



```
maxlen: 42, vocab size: 2268
```

# Practical 9

**Aim: Write a program for object detection from the image/video.**

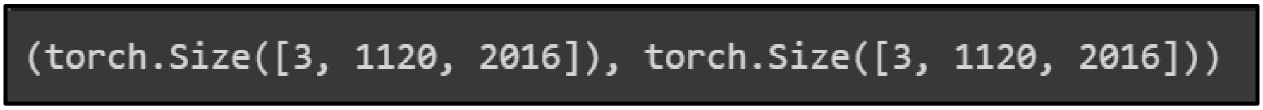
**Source Code:**

```
import torch
import torchvision
import pycocotools
from PIL import Image

holiday = Image.open("/content/kids.jpg").convert('RGB')
holiday
```

```
kids_playing = Image.open("/content/kids.jpg").convert('RGB')
kids_playing
```

```
holiday_tensor_int = pil_to_tensor(holiday)
kids_playing_tensor_int = pil_to_tensor(kids_playing)
```

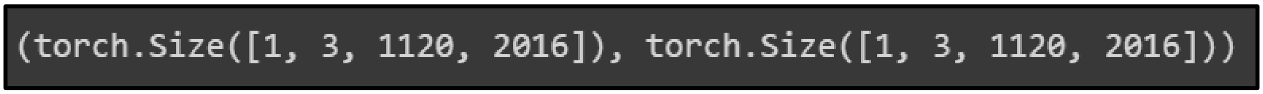
```
holiday_tensor_int.shape, kids_playing_tensor_int.shape
```



```
(torch.Size([3, 1120, 2016]), torch.Size([3, 1120, 2016]))
```

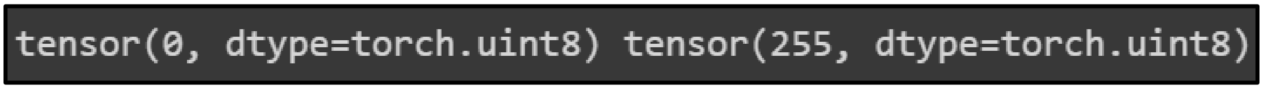
```
holiday_tensor_int = holiday_tensor_int.unsqueeze(dim=0)
kids_playing_tensor_int = kids_playing_tensor_int.unsqueeze(dim=0)
```

```
holiday_tensor_int.shape, kids_playing_tensor_int.shape
```



```
(torch.Size([1, 3, 1120, 2016]), torch.Size([1, 3, 1120, 2016]))
```

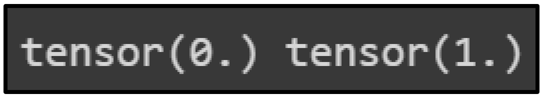
```
print(holiday_tensor_int.min(), holiday_tensor_int.max())
```



```
tensor(0, dtype=torch.uint8) tensor(255, dtype=torch.uint8)
```

```
holiday_tensor_float = holiday_tensor_int / 255.0
kids_playing_tensor_float = kids_playing_tensor_int / 255.0
```

```
print(holiday_tensor_float.min(), holiday_tensor_float.max())
```



```
tensor(0.) tensor(1.)
```

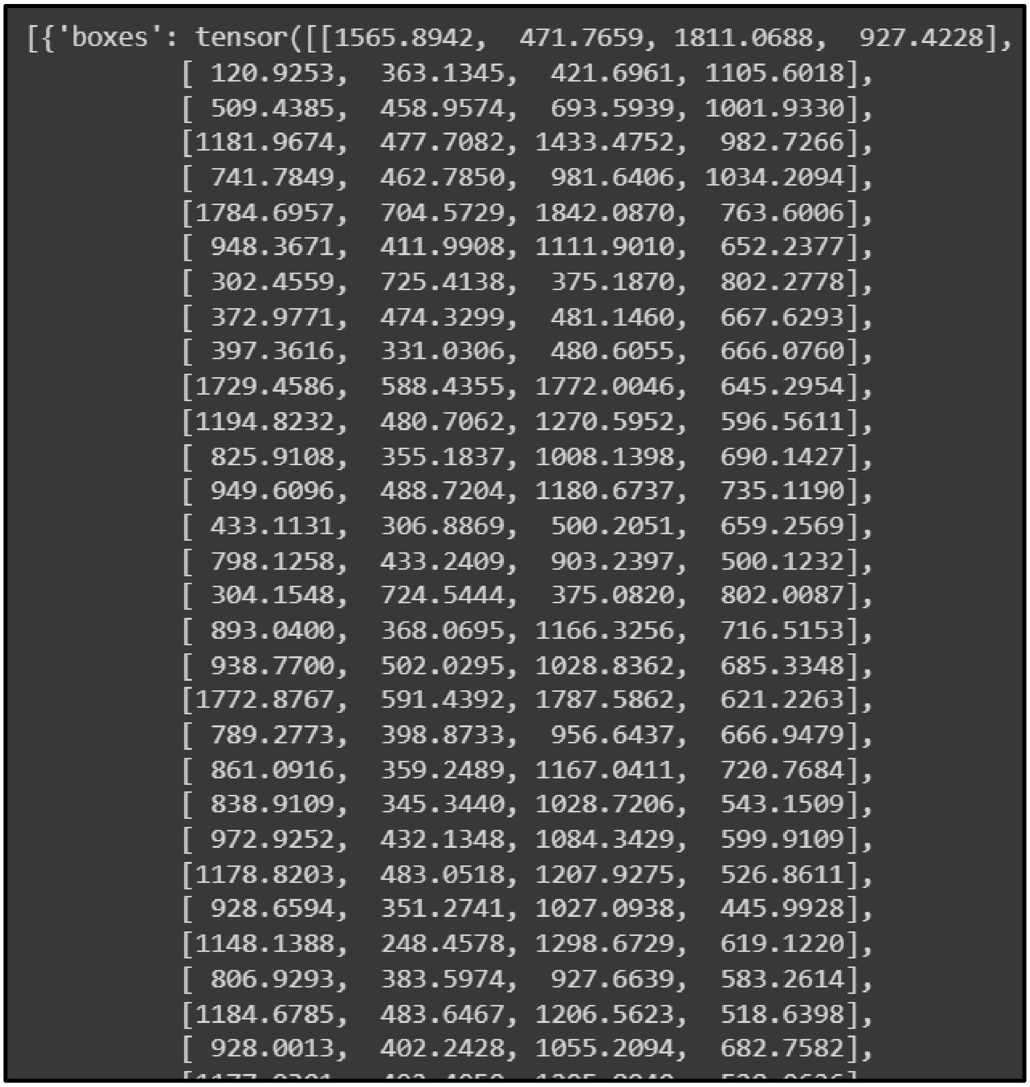
from torchvision.models.detection import fasterrcnn_resnet50_fpn

object_detection_model = fasterrcnn_resnet50_fpn(pretrained=True, progress=False)

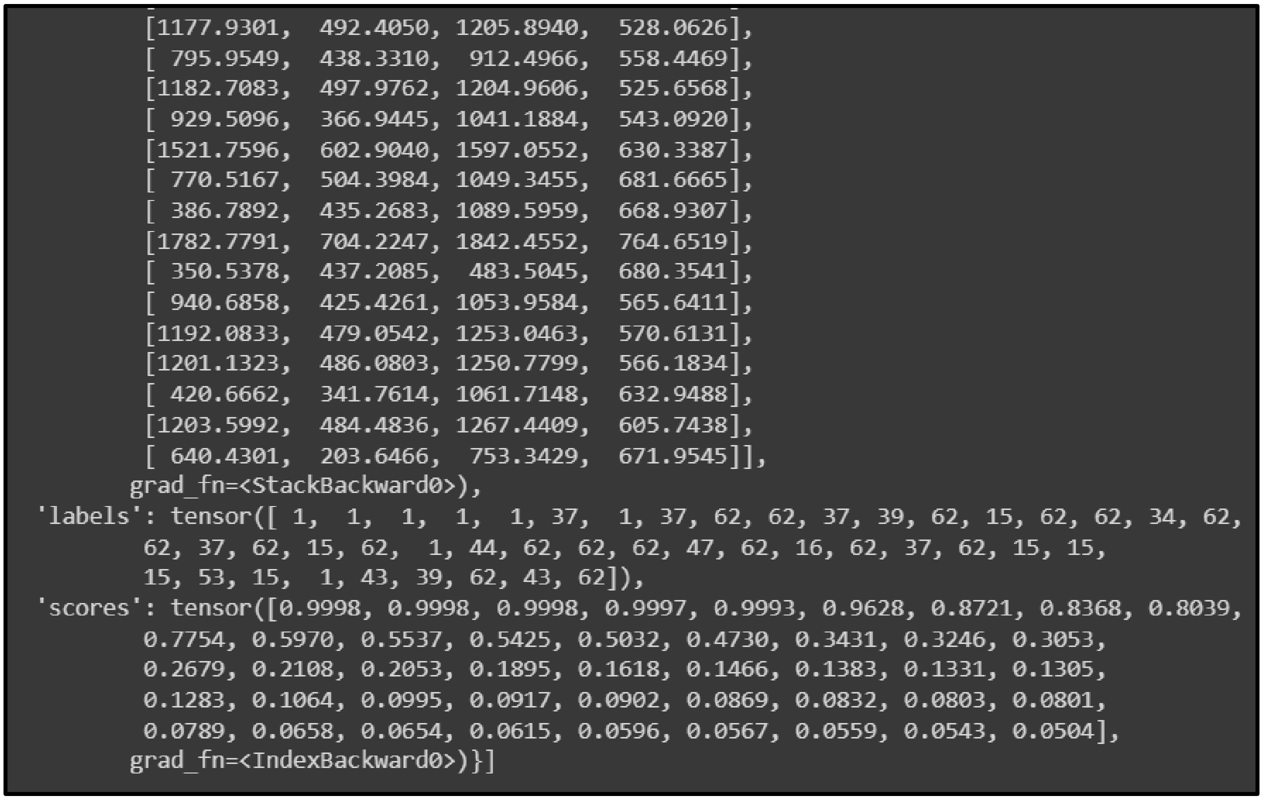
object_detection_model.eval(); ## Setting Model for Evaluation/Prediction

holiday_preds = object_detection_model(holiday_tensor_float)

holiday_preds



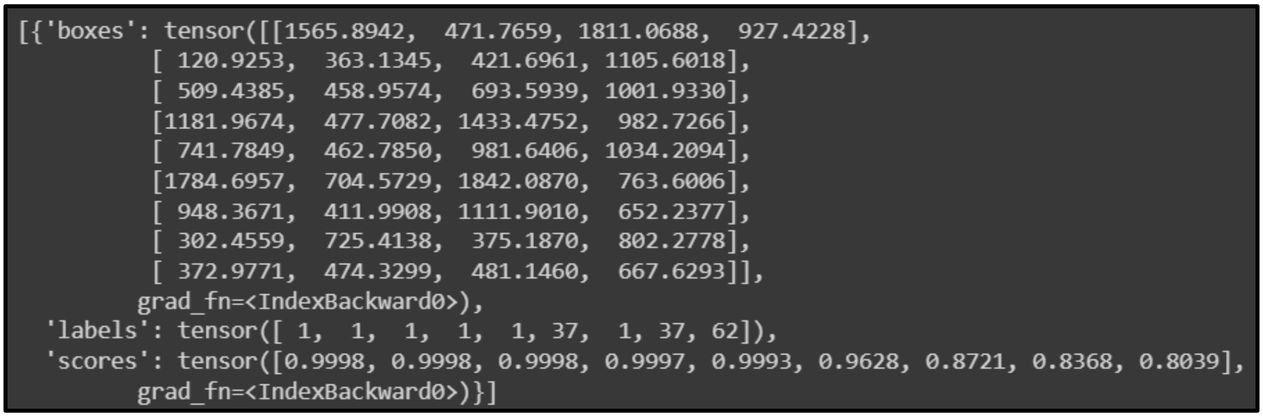
```
[{'boxes': tensor([[1565.8942,  471.7659, 1811.0688,  927.4228],
           [ 120.9253,  363.1345,  421.6961, 1105.6018],
           [ 509.4385,  458.9574,  693.5939, 1001.9330],
           [1181.9674,  477.7082, 1433.4752,  982.7266],
           [ 741.7849,  462.7850,  981.6406, 1034.2094],
           [1784.6957,  704.5729, 1842.0870,  763.6006],
           [ 948.3671,  411.9908, 1111.9010,  652.2377],
           [ 302.4559,  725.4138,  375.1870,  802.2778],
           [ 372.9771,  474.3299,  481.1460,  667.6293],
           [ 397.3616,  331.0306,  480.6055,  666.0760],
           [1729.4586,  588.4355, 1772.0046,  645.2954],
           [1194.8232,  480.7062, 1270.5952,  596.5611],
           [ 825.9108,  355.1837, 1008.1398,  690.1427],
           [ 949.6096,  488.7204, 1180.6737,  735.1190],
           [ 433.1131,  306.8869,  500.2051,  659.2569],
           [ 798.1258,  433.2409,  903.2397,  500.1232],
           [ 304.1548,  724.5444,  375.0820,  802.0087],
           [ 893.0400,  368.0695, 1166.3256,  716.5153],
           [ 938.7700,  502.0295, 1028.8362,  685.3348],
           [1772.8767,  591.4392, 1787.5862,  621.2263],
           [ 789.2773,  398.8733,  956.6437,  666.9479],
           [ 861.0916,  359.2489, 1167.0411,  720.7684],
           [ 838.9109,  345.3440, 1028.7206,  543.1509],
           [ 972.9252,  432.1348, 1084.3429,  599.9109],
           [1178.8203,  483.0518, 1207.9275,  526.8611],
           [ 928.6594,  351.2741, 1027.0938,  445.9928],
           [1148.1388,  248.4578, 1298.6729,  619.1220],
           [ 806.9293,  383.5974,  927.6639,  583.2614],
           [1184.6785,  483.6467, 1206.5623,  518.6398],
           [ 928.0013,  402.2428, 1055.2094,  682.7582],
           [1177.0301,  483.4050, 1205.2040,  520.0626],
```



```
          [1177.9301,  492.4050, 1205.8940,  528.0626],
          [ 795.9549,  438.3310,  912.4966,  558.4469],
          [1182.7083,  497.9762, 1204.9606,  525.6568],
          [ 929.5096,  366.9445, 1041.1884,  543.0920],
          [1521.7596,  602.9040, 1597.0552,  630.3387],
          [ 770.5167,  504.3984, 1049.3455,  681.6665],
          [ 386.7892,  435.2683, 1089.5959,  668.9307],
          [1782.7791,  704.2247, 1842.4552,  764.6519],
          [ 350.5378,  437.2085,  483.5045,  680.3541],
          [ 940.6858,  425.4261, 1053.9584,  565.6411],
          [1192.0833,  479.0542, 1253.0463,  570.6131],
          [1201.1323,  486.0803, 1250.7799,  566.1834],
          [ 420.6662,  341.7614, 1061.7148,  632.9488],
          [1203.5992,  484.4836, 1267.4409,  605.7438],
          [ 640.4301,  203.6466,  753.3429,  671.9545]],
         grad_fn=<StackBackward0>),
  'labels': tensor([ 1,  1,  1,  1,  1, 37,  1, 37, 62, 62, 37, 39, 62, 15, 62, 62, 34, 62,
          62, 37, 62, 15, 62,  1, 44, 62, 62, 62, 47, 62, 16, 62, 37, 62, 15, 15,
          15, 53, 15,  1, 43, 39, 62, 43, 62]),
  'scores': tensor([0.9998, 0.9998, 0.9998, 0.9997, 0.9993, 0.9628, 0.8721, 0.8368, 0.8039,
          0.7754, 0.5970, 0.5537, 0.5425, 0.5032, 0.4730, 0.3431, 0.3246, 0.3053,
          0.2679, 0.2108, 0.2053, 0.1895, 0.1618, 0.1466, 0.1383, 0.1331, 0.1305,
          0.1283, 0.1064, 0.0995, 0.0917, 0.0902, 0.0869, 0.0832, 0.0803, 0.0801,
          0.0789, 0.0658, 0.0654, 0.0615, 0.0596, 0.0567, 0.0559, 0.0543, 0.0504],
         grad_fn=<IndexBackward0>)}]
```

holiday_preds[0]["boxes"] = holiday_preds[0]["boxes"][holiday_preds[0]["scores"] > 0.8]
holiday_preds[0]["labels"] = holiday_preds[0]["labels"][holiday_preds[0]["scores"] > 0.8]
holiday_preds[0]["scores"] = holiday_preds[0]["scores"][holiday_preds[0]["scores"] > 0.8]

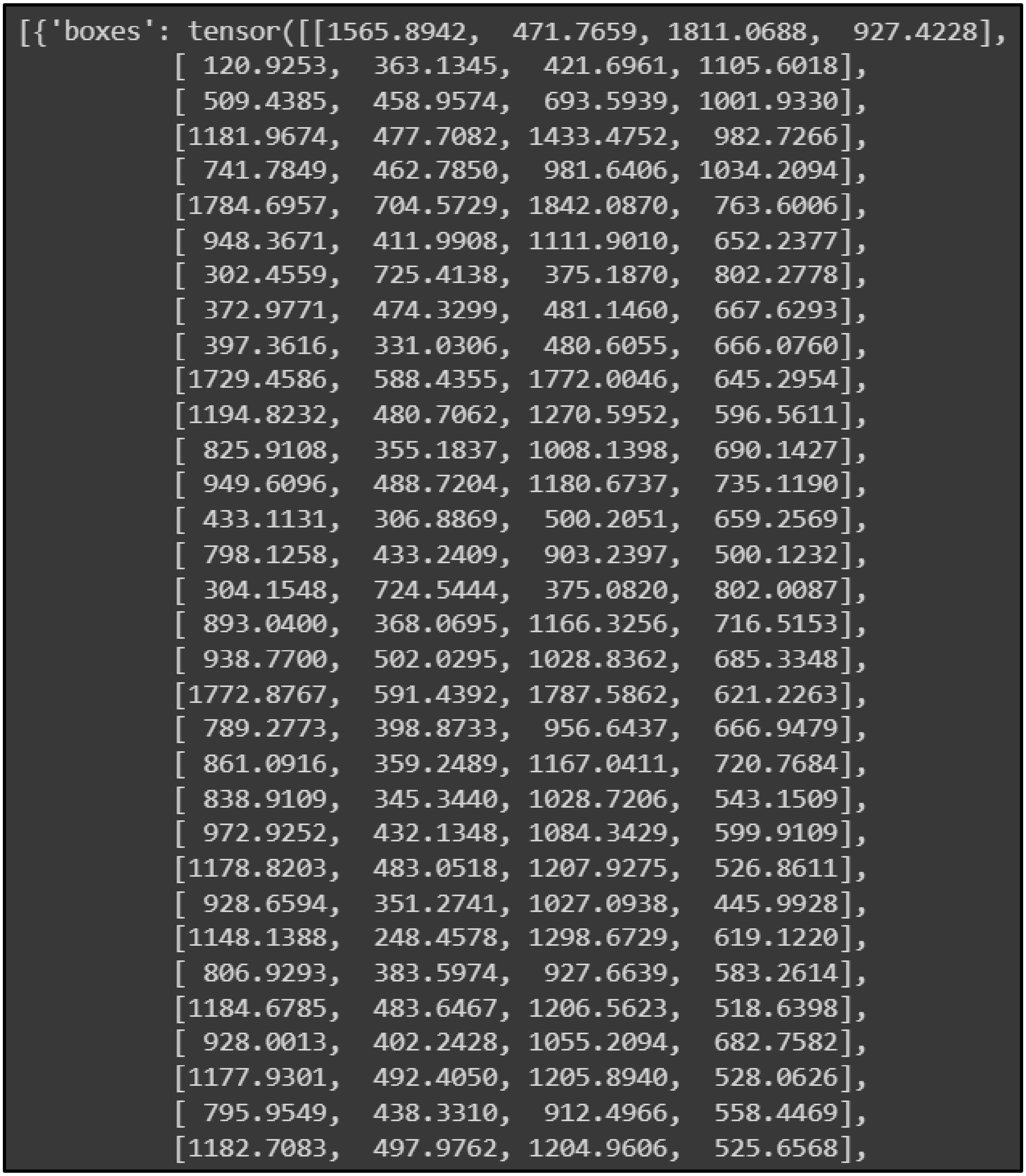
holiday_preds



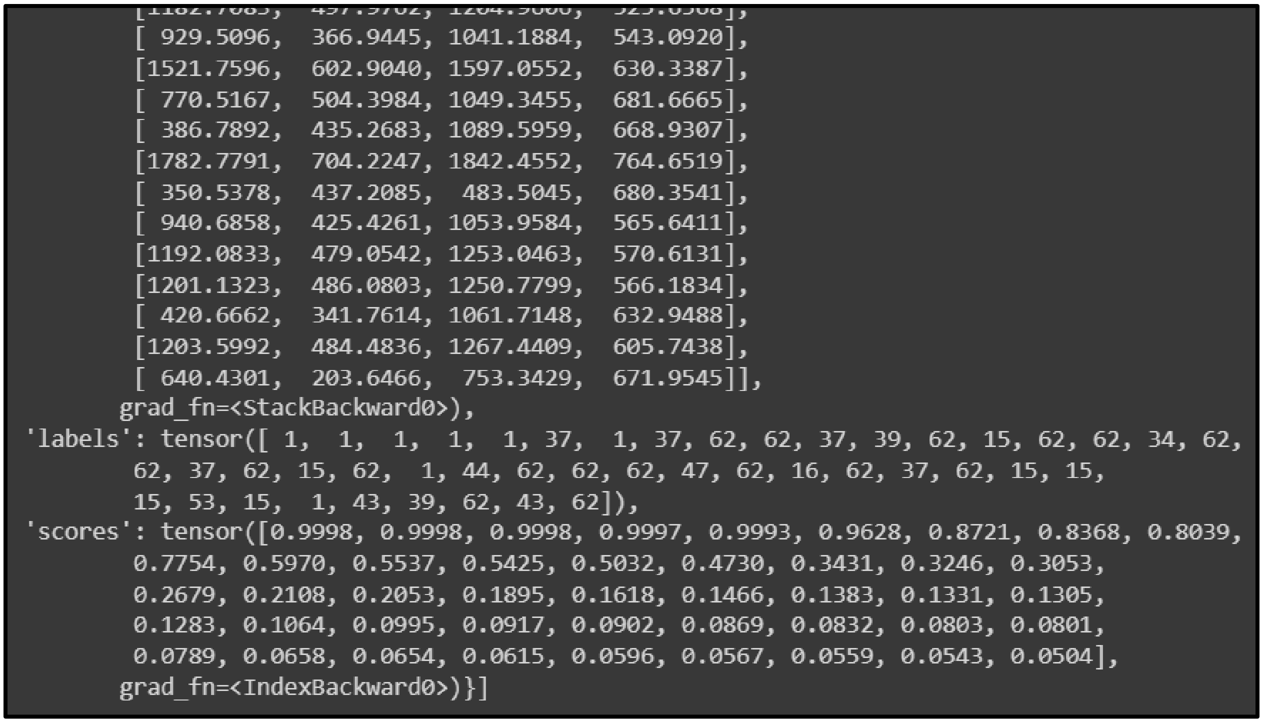
```
[{'boxes': tensor([[1565.8942,  471.7659, 1811.0688,  927.4228],
          [ 120.9253,  363.1345,  421.6961, 1105.6018],
          [ 509.4385,  458.9574,  693.5939, 1001.9330],
          [1181.9674,  477.7082, 1433.4752,  982.7266],
          [ 741.7849,  462.7850,  981.6406, 1034.2094],
          [1784.6957,  704.5729, 1842.0870,  763.6006],
          [ 948.3671,  411.9908, 1111.9010,  652.2377],
          [ 302.4559,  725.4138,  375.1870,  802.2778],
          [ 372.9771,  474.3299,  481.1460,  667.6293]],
         grad_fn=<IndexBackward0>),
  'labels': tensor([ 1,  1,  1,  1,  1, 37,  1, 37, 62]),
  'scores': tensor([0.9998, 0.9998, 0.9998, 0.9997, 0.9993, 0.9628, 0.8721, 0.8368, 0.8039],
         grad_fn=<IndexBackward0>)}]
```

kids_preds = object_detection_model(kids_playing_tensor_float)

kids_preds

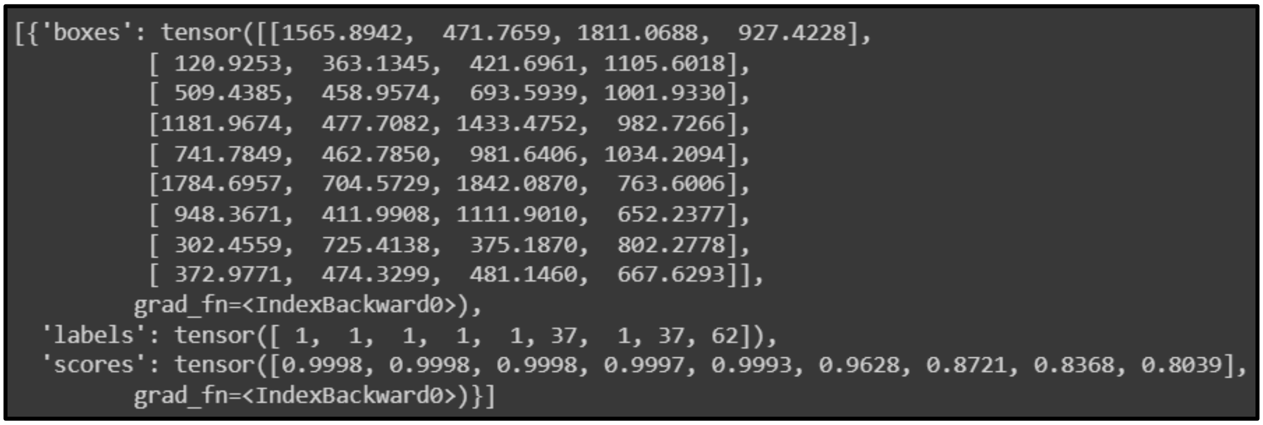


```
[{'boxes': tensor([[1565.8942,  471.7659, 1811.0688,  927.4228],
          [ 120.9253,  363.1345,  421.6961, 1105.6018],
          [ 509.4385,  458.9574,  693.5939, 1001.9330],
          [1181.9674,  477.7082, 1433.4752,  982.7266],
          [ 741.7849,  462.7850,  981.6406, 1034.2094],
          [1784.6957,  704.5729, 1842.0870,  763.6006],
          [ 948.3671,  411.9908, 1111.9010,  652.2377],
          [ 302.4559,  725.4138,  375.1870,  802.2778],
          [ 372.9771,  474.3299,  481.1460,  667.6293],
          [ 397.3616,  331.0306,  480.6055,  666.0760],
          [1729.4586,  588.4355, 1772.0046,  645.2954],
          [1194.8232,  480.7062, 1270.5952,  596.5611],
          [ 825.9108,  355.1837, 1008.1398,  690.1427],
          [ 949.6096,  488.7204, 1180.6737,  735.1190],
          [ 433.1131,  306.8869,  500.2051,  659.2569],
          [ 798.1258,  433.2409,  903.2397,  500.1232],
          [ 304.1548,  724.5444,  375.0820,  802.0087],
          [ 893.0400,  368.0695, 1166.3256,  716.5153],
          [ 938.7700,  502.0295, 1028.8362,  685.3348],
          [1772.8767,  591.4392, 1787.5862,  621.2263],
          [ 789.2773,  398.8733,  956.6437,  666.9479],
          [ 861.0916,  359.2489, 1167.0411,  720.7684],
          [ 838.9109,  345.3440, 1028.7206,  543.1509],
          [ 972.9252,  432.1348, 1084.3429,  599.9109],
          [1178.8203,  483.0518, 1207.9275,  526.8611],
          [ 928.6594,  351.2741, 1027.0938,  445.9928],
          [1148.1388,  248.4578, 1298.6729,  619.1220],
          [ 806.9293,  383.5974,  927.6639,  583.2614],
          [1184.6785,  483.6467, 1206.5623,  518.6398],
          [ 928.0013,  402.2428, 1055.2094,  682.7582],
          [1177.9301,  492.4050, 1205.8940,  528.0626],
          [ 795.9549,  438.3310,  912.4966,  558.4469],
          [1182.7083,  497.9762, 1204.9606,  525.6568],
```



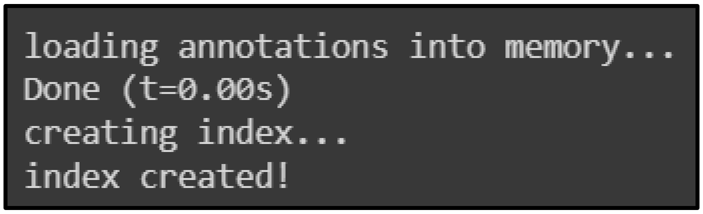
```
        [1182.7083,  497.9762, 1204.9066,  525.6568],
        [ 929.5096,  366.9445, 1041.1884,  543.0920],
        [1521.7596,  602.9040, 1597.0552,  630.3387],
        [ 770.5167,  504.3984, 1049.3455,  681.6665],
        [ 386.7892,  435.2683, 1089.5959,  668.9307],
        [1782.7791,  704.2247, 1842.4552,  764.6519],
        [ 350.5378,  437.2085,  483.5045,  680.3541],
        [ 940.6858,  425.4261, 1053.9584,  565.6411],
        [1192.0833,  479.0542, 1253.0463,  570.6131],
        [1201.1323,  486.0803, 1250.7799,  566.1834],
        [ 420.6662,  341.7614, 1061.7148,  632.9488],
        [1203.5992,  484.4836, 1267.4409,  605.7438],
        [ 640.4301,  203.6466,  753.3429,  671.9545]],
       grad_fn=<StackBackward0>),
 'labels': tensor([ 1,  1,  1,  1,  1, 37,  1, 37, 62, 62, 37, 39, 62, 15, 62, 62, 34, 62,
        62, 37, 62, 15, 62,  1, 44, 62, 62, 62, 47, 62, 16, 62, 37, 62, 15, 15,
        15, 53, 15,  1, 43, 39, 62, 43, 62]),
 'scores': tensor([0.9998, 0.9998, 0.9998, 0.9997, 0.9993, 0.9628, 0.8721, 0.8368, 0.8039,
        0.7754, 0.5970, 0.5537, 0.5425, 0.5032, 0.4730, 0.3431, 0.3246, 0.3053,
        0.2679, 0.2108, 0.2053, 0.1895, 0.1618, 0.1466, 0.1383, 0.1331, 0.1305,
        0.1283, 0.1064, 0.0995, 0.0917, 0.0902, 0.0869, 0.0832, 0.0803, 0.0801,
        0.0789, 0.0658, 0.0654, 0.0615, 0.0596, 0.0567, 0.0559, 0.0543, 0.0504],
       grad_fn=<IndexBackward0>)}]
```

```
kids_preds[0]["boxes"] = kids_preds[0]["boxes"][kids_preds[0]["scores"] > 0.8]
kids_preds[0]["labels"] = kids_preds[0]["labels"][kids_preds[0]["scores"] > 0.8]
kids_preds[0]["scores"] = kids_preds[0]["scores"][kids_preds[0]["scores"] > 0.8]

kids_preds
```



```
[{'boxes': tensor([[1565.8942,  471.7659, 1811.0688,  927.4228],
          [ 120.9253,  363.1345,  421.6961, 1105.6018],
          [ 509.4385,  458.9574,  693.5939, 1001.9330],
          [1181.9674,  477.7082, 1433.4752,  982.7266],
          [ 741.7849,  462.7850,  981.6406, 1034.2094],
          [1784.6957,  704.5729, 1842.0870,  763.6006],
          [ 948.3671,  411.9908, 1111.9010,  652.2377],
          [ 302.4559,  725.4138,  375.1870,  802.2778],
          [ 372.9771,  474.3299,  481.1460,  667.6293]],
         grad_fn=<IndexBackward0>),
  'labels': tensor([ 1,  1,  1,  1,  1, 37,  1, 37, 62]),
  'scores': tensor([0.9998, 0.9998, 0.9998, 0.9997, 0.9993, 0.9628, 0.8721, 0.8368, 0.8039],
         grad_fn=<IndexBackward0>)}]
```

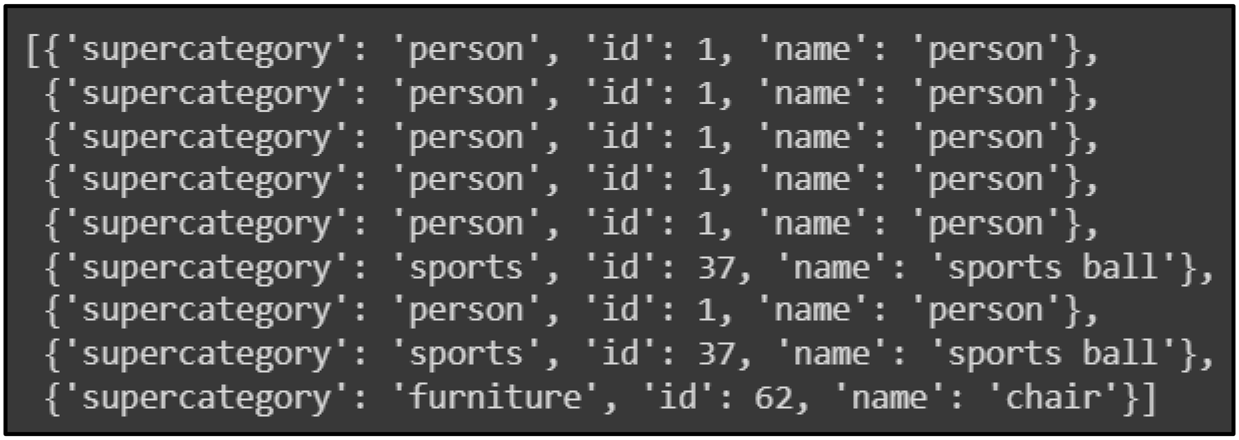
```
from pycocotools.coco import COCO

annFile='/content/instances_val2017.json'

coco=COCO(annFile)
```



```
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
```

holiday_labels = coco.loadCats(holiday_preds[0]["labels"].numpy())

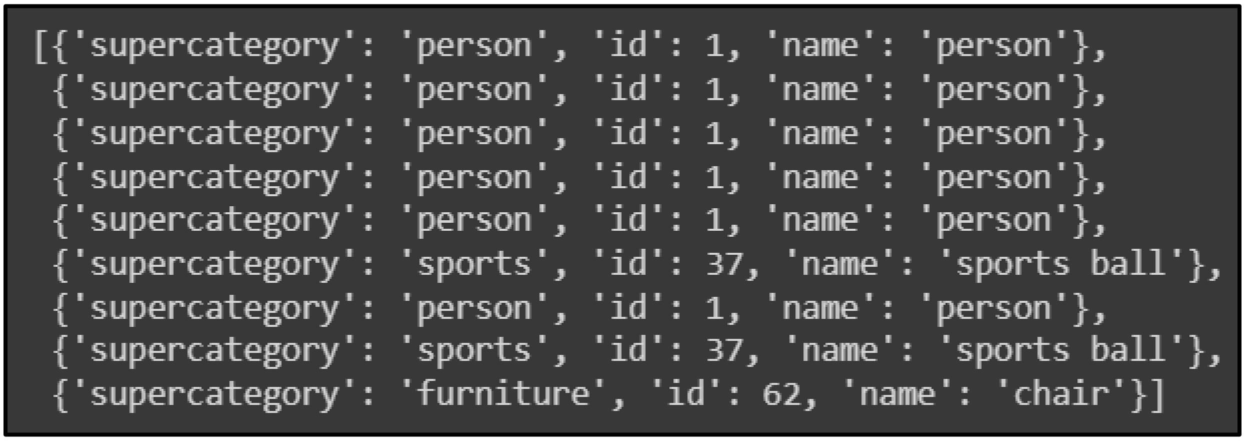
holiday_labels



```
[{'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'sports', 'id': 37, 'name': 'sports ball'},
 {'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'sports', 'id': 37, 'name': 'sports ball'},
 {'supercategory': 'furniture', 'id': 62, 'name': 'chair'}]
```

kids_labels = coco.loadCats(kids_preds[0]["labels"].numpy())

kids_labels



```
[{'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'sports', 'id': 37, 'name': 'sports ball'},
 {'supercategory': 'person', 'id': 1, 'name': 'person'},
 {'supercategory': 'sports', 'id': 37, 'name': 'sports ball'},
 {'supercategory': 'furniture', 'id': 62, 'name': 'chair'}]
```

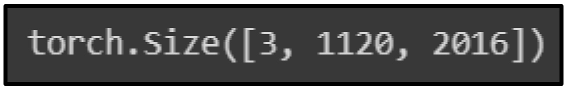
from torchvision.utils import draw_bounding_boxes

holiday_annot_labels = ["{}-{:.2f}".format(label["name"], prob) for label, prob in zip(holiday_labels, holiday_preds[0]["scores"].detach().numpy())]

holiday_output = draw_bounding_boxes(image=holiday_tensor_int[0],

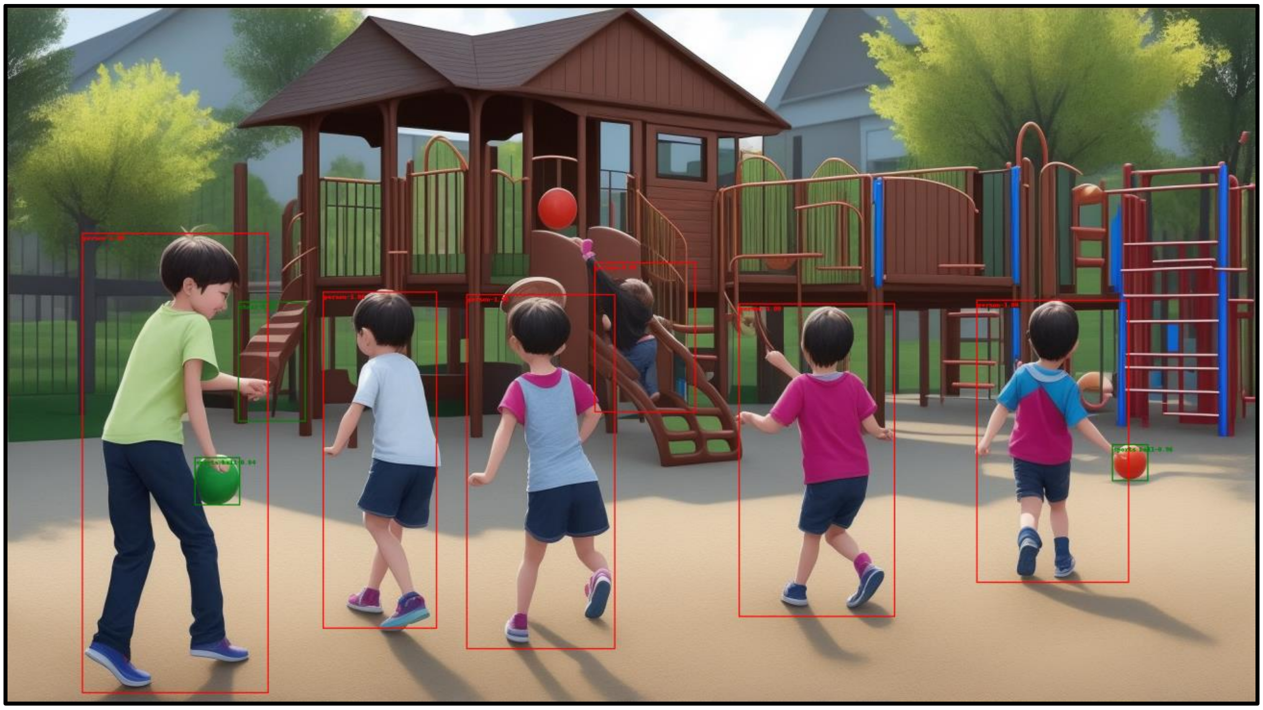
```
                        boxes=holiday_preds[0]["boxes"],
                        labels=holiday_annot_labels,
                        colors=["red" if label["name"]=="person" else "green" for label in
holiday_labels],
                        width=2
                       )
```

```
holiday_output.shape
```



```
torch.Size([3, 1120, 2016])
```

```
from torchvision.transforms.functional import to_pil_image

to_pil_image(holiday_output)
```



```
from torchvision.utils import draw_bounding_boxes

kids_annot_labels = ["{}-{:.2f}".format(label["name"], prob) for label, prob in zip(kids_labels,
kids_preds[0]["scores"].detach().numpy())]

kids_output = draw_bounding_boxes(image=kids_playing_tensor_int[0],
                        boxes=kids_preds[0]["boxes"],
                        labels=kids_annot_labels,
                        colors=["red" if label["name"]=="person" else "green" for label in kids_labels],
                        width=2,
                        font_size=16,
                        fill=True
                       )

to_pil_image(kids_output)
```