**Convolution Neural Networks (CNN)**

**Instructions:**

Please share your answers filled in-line in the word document. Submit code separately wherever applicable.

Please ensure you update all the details:

**Name: Vishvash C Batch ID:** 23012024

**Topic: Convolutional Neural Networks**

**Guidelines:**

**1. An assignment submission is considered complete only when the correct and executable code(s) and documentation explaining the method and results are submitted. Failing to submit either of those will be considered an invalid submission and not a correct submission.**

**2. Ensure that you submit your assignments correctly and in full. Resubmission is not allowed.**

**3. Post the submission you can evaluate your work by referring to the keys provided. (will be available only post the submission).**

**Hints:**

1. **Business Problem**
   1. **What is the business objective?**
   2. **Are there any constraints?**
2. **Work on each feature of the dataset to create a data dictionary.**
3. **Data Pre-processing**

**3.1 Data Cleaning, Feature Engineering, etc.**

**3.2 Outlier treatment if applicable.**

1. **Model Building**
   1. **Build a convolution neural network model.**
   2. **Train and test the model.**
   3. **Briefly explain the model output in the documentation.**
2. **Write about the benefits/impact of the solution - in what way does the business (client) benefit from the solution provided?**
3. **Use Tensorflow for this assignment. Depending on your system configuration, either Tensorflow GPU or Tensorflow CPU versions.**

**Problem Statement: -**

1. Build a CNN model on the CIFAR-10 dataset by applying a few regularization techniques like dropout and data augmentation. Download the data set from a library called tensorflow. Finally do the deployment on streamlit application
2. Find out the differences between the Convnet filter and the Maxpool layers.

**Convolutional layer:** In a convolutional layer, filters (also called kernels) convolve across the input data to extract features through element-wise multiplications and summations.

**Maxpool layer:** In a maxpooling layer, the input is divided into rectangular pooling regions, and the maximum value within each region is taken as the output. It reduces the spatial dimensions of the input, helping to decrease computational complexity and control overfitting.

1. If the input of an image is 64x64x3 which has been convolved by 10 5x5 filters with stride 1 and padding 2:
2. How many activation maps are obtained?

The number of filters used in the convolutional layer. Here, it's 10.

1. What is the size of the activation maps?

Size of activation maps: To find the size, we use this formula: (input size + 2 \* padding - filter size) / stride + 1.

For example, if the input size is 64, padding is 2, filter size is 5, and stride is 1, the activation map size will be 64x64.

1. How many parameters are calculated?

To find out how many values our model needs to learn, we use a simple formula. For each filter in the convolutional layer, we multiply the width, height, and number of channels of the input data. Then, we add 1 for a special value called the bias. After that, we multiply this result by the total number of filters we have. In this case, since we have 10 filters and the input image has 3 channels (red, green, blue), the calculation becomes: (5 \* 5 \* 3 + 1) \* 10 = 760

1. What are the different techniques that need to be applied to overcome the issue of overfitting? Provide brief explanations of how these techniques address the issue.

**Dropout:** During training, randomly ignore some connections between neurons to prevent the model from relying too much on specific features.

**Data augmentation:** Create variations of the training data by applying transformations like rotations or flips, which helps the model generalize better to new, unseen data.

**L2 Regularization:** Add a penalty term to the loss function, encouraging the model to find simpler patterns and avoid overly complex solutions.

**Early stopping:** Stop training the model when the performance on a validation dataset starts to get worse, to prevent it from becoming too specialized to the training data.

**Model complexity reduction:** Simplify the model by reducing the number of layers, neurons, or parameters, especially if the dataset is not large enough to support a more complex model.

**Code:**

'''

The CIFAR-10 dataset is a popular benchmark dataset in the field of computer vision. It stands for the Canadian Institute for Advanced Research (CIFAR), which funded the collection of the dataset. CIFAR-10 consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The dataset is split into 50,000 training images and 10,000 test images.

Here are some key details about the CIFAR-10 dataset:

Classes: CIFAR-10 contains images across 10 different classes, with each class representing a specific object or category. The classes are:

Airplane

Automobile

Bird

Cat

Deer

Dog

Frog

Horse

Ship

Truck

Image Size: Each image in the CIFAR-10 dataset is of size 32x32 pixels. These are relatively small images compared to many other datasets, making CIFAR-10 suitable for quick experimentation and prototyping.

Color Channels: CIFAR-10 images are RGB (Red, Green, Blue) color images, meaning they have three color channels. Each pixel in the image is represented by three values, one for each color channel, ranging from 0 to 255.

Training and Test Split: The dataset is divided into training and test sets. The training set consists of 50,000 images, while the test set contains 10,000 images. This split ensures that models are trained on one set of data and evaluated on another, unseen set to assess their generalization performance.

Challenges: CIFAR-10 poses several challenges to machine learning models due to its relatively small image size, low resolution, and the presence of multiple classes. Models trained on CIFAR-10 must learn to distinguish between various objects with limited visual information, making it a challenging dataset for tasks such as image classification.

Overall, CIFAR-10 serves as a standard benchmark dataset for evaluating the performance of machine learning algorithms, particularly in tasks related to image classification and object recognition. Many research papers and studies in the field of computer vision use CIFAR-10 as a baseline dataset for comparison and evaluation of new techniques and algorithms.

'''

import tensorflow as tf

tf.test.is\_gpu\_available(

cuda\_only=False, min\_cuda\_compute\_capability=None

)

import pandas as pd

from tensorflow.keras import layers

from tensorflow.keras import models

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

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

# Design the Network

model = models.Sequential()

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

model.add(layers.MaxPooling2D((2, 2)))

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

model.add(layers.MaxPooling2D((2, 2)))

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

model.add(layers.Flatten())

model.add(layers.Dropout(0.5)) # Adding dropout layer for regularization

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(10, activation='softmax'))

model.summary()

# Num of Parameters

# [(w\*h\*d)+1]\*k

# w = width; h = height; d = filters from previous layer

# k = current layer filters

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(10, activation='softmax'))

model.summary()

# Load the cifar10 dataset

(train\_images, train\_labels), (test\_images, test\_labels) = cifar10.load\_data()

# reshaping test\_images to 2d

arr\_reshaped = test\_images.reshape(test\_images.shape[0], -1)

# converting reshaped array to dataframe

df = pd.DataFrame(arr\_reshaped)

# selecting 100 rows randomly

df1 = df.sample(n = 100)

# Saving small amount of data for testing

df1.to\_csv('cifar10\_cnn.csv', index=False)

train\_images = train\_images.reshape((50000, 32, 32, 3))

train\_images = train\_images.astype('float32') / 255

test\_images = test\_images.reshape((10000, 32, 32, 3))

test\_images = test\_images.astype('float32') / 255

train\_labels = to\_categorical(train\_labels)

test\_labels = to\_categorical(test\_labels)

# Split training data into training and validation sets

x\_train, x\_val, y\_train, y\_val = train\_test\_split(train\_images, train\_labels, test\_size=0.2, random\_state=42)

# Data augmentation

datagen = tf.keras.preprocessing.image.ImageDataGenerator(

rotation\_range=15,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

horizontal\_flip=True,

fill\_mode='nearest'

)

datagen.fit(x\_train)

# Compile the model

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

# Train the model with data augmentation

model.fit(datagen.flow(x\_train, y\_train, batch\_size=64), epochs=5, validation\_data=(x\_val, y\_val))

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

# model.fit(train\_images, train\_labels, epochs=5, batch\_size=64)

# test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2)

# test\_acc

model\_json = model.to\_json()

with open("model.json", "w") as json\_file:

json\_file.write(model\_json)

# serialize weights to HDF5

model.save\_weights("model.weights.h5")

#######################

# Testing on new data

from tensorflow.keras.models import model\_from\_json

import pandas as pd

# from keras import model\_from\_json

# opening and store file in a variable

json\_file = open('model.json', 'r')

loaded\_model\_json = json\_file.read()

json\_file.close()

# use Keras model\_from\_json to make a loaded model

loaded\_model = model\_from\_json(loaded\_model\_json)

# load weights into new model

loaded\_model.load\_weights("model.weights.h5")

print("Loaded Model from disk")

# compile and evaluate loaded model

# loaded\_model.compile(loss='categorical\_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

test1 = pd.read\_csv("cifar10\_cnn.csv")

arr\_img = test1.to\_numpy()

test\_pred = arr\_img.reshape((len(arr\_img), 32, 32, 3))

test\_pred = test\_pred.astype('float32') / 255

predictions = pd.DataFrame(loaded\_model.predict(test\_pred))

**Output:**

model.fit(datagen.flow(x\_train, y\_train, batch\_size=64), epochs=5, validation\_data=(x\_val, y\_val))

Epoch 1/5

C:\Users\Lenovo\anaconda3\envs\python\_10\lib\site-packages\keras\src\trainers\data\_adapters\py\_dataset\_adapter.py:120: UserWarning: Your `PyDataset` class should call `super().\_\_init\_\_(\*\*kwargs)` in its constructor. `\*\*kwargs` can include `workers`, `use\_multiprocessing`, `max\_queue\_size`. Do not pass these arguments to `fit()`, as they will be ignored.

self.\_warn\_if\_super\_not\_called()

625/625 ━━━━━━━━━━━━━━━━━━━━ 65s 99ms/step - accuracy: 0.2618 - loss: 1.9509 - val\_accuracy: 0.4824 - val\_loss: 1.4414

Epoch 2/5

625/625 ━━━━━━━━━━━━━━━━━━━━ 63s 100ms/step - accuracy: 0.4529 - loss: 1.5008 - val\_accuracy: 0.5322 - val\_loss: 1.3093

Epoch 3/5

625/625 ━━━━━━━━━━━━━━━━━━━━ 62s 98ms/step - accuracy: 0.5011 - loss: 1.3832 - val\_accuracy: 0.5972 - val\_loss: 1.1316

Epoch 4/5

625/625 ━━━━━━━━━━━━━━━━━━━━ 66s 104ms/step - accuracy: 0.5410 - loss: 1.2812 - val\_accuracy: 0.6126 - val\_loss: 1.0943

Epoch 5/5

625/625 ━━━━━━━━━━━━━━━━━━━━ 77s 122ms/step - **accuracy: 0.5582 -** loss: 1.2378 - val\_accuracy: 0.6233 - val\_loss: 1.0628

Out[38]: <keras.src.callbacks.history.History at 0x1c1668ddff0>

predictions = pd.DataFrame(loaded\_model.predict(test\_pred))

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step

predictions

Out[62]:

0 1 2 ... 7 8 9

0 0.405616 0.003016 0.079367 ... 0.004266 0.427278 0.009926

1 0.097147 0.086218 0.019673 ... 0.081302 0.059564 0.514793

2 0.015059 0.073830 0.017713 ... 0.138812 0.028263 0.210453

3 0.000910 0.001999 0.014016 ... 0.021721 0.000685 0.004500

4 0.004750 0.002077 0.077029 ... 0.063572 0.002274 0.006010

.. ... ... ... ... ... ... ...

95 0.119509 0.153529 0.021240 ... 0.026850 0.296408 0.283851

96 0.227436 0.002151 0.416751 ... 0.021432 0.076911 0.038839

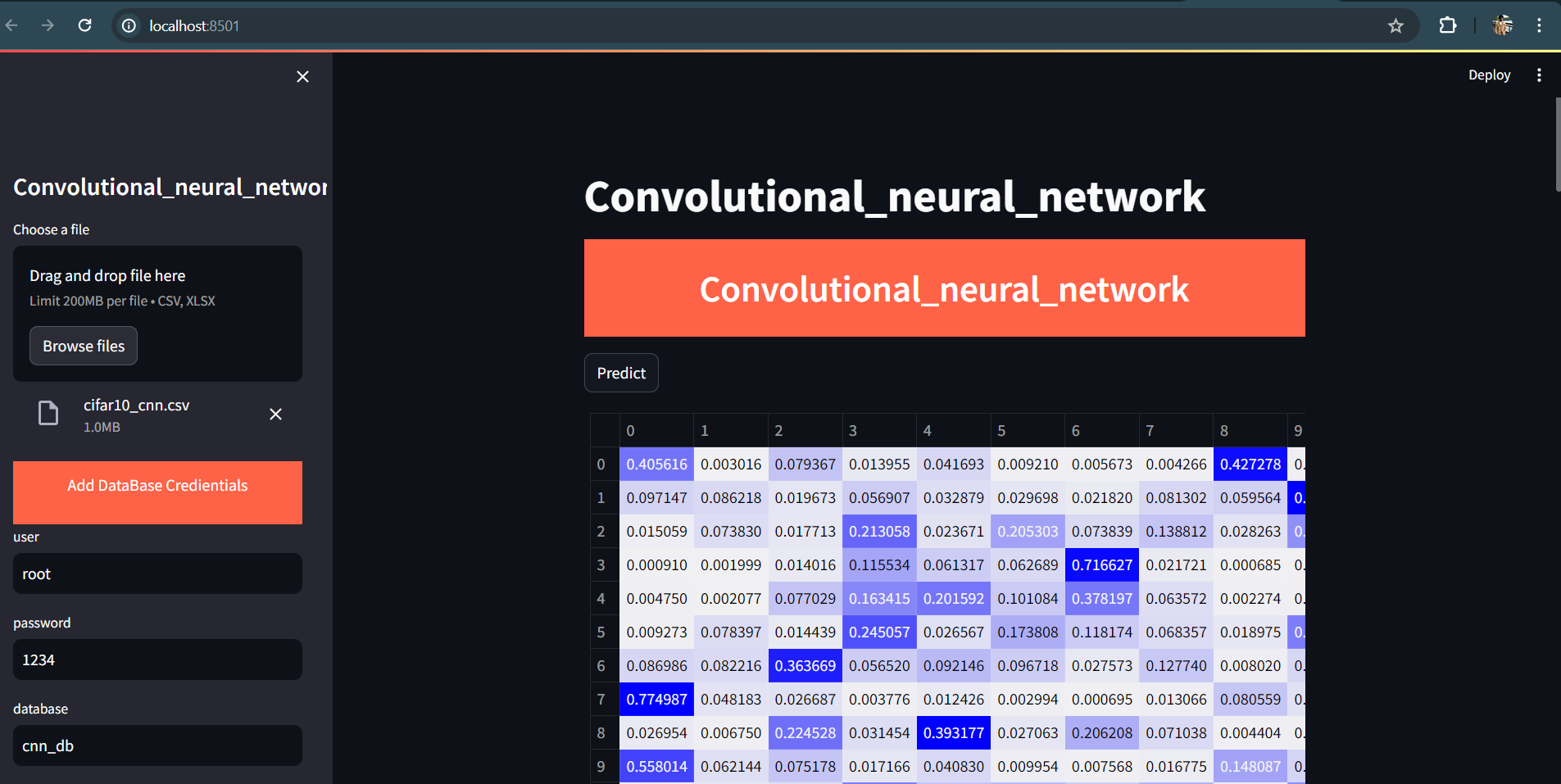
97 0.009658 0.000522 0.030402 ... 0.203891 0.004618 0.007171

98 0.004135 0.699140 0.000794 ... 0.003418 0.005093 0.282059

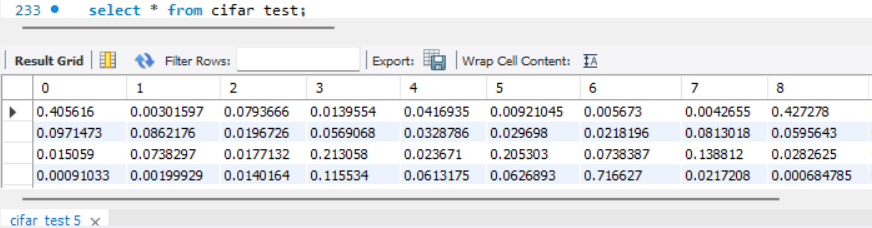
99 0.032110 0.070602 0.090654 ... 0.156205 0.028789 0.124529

[100 rows x 10 columns]

**Deployment of CNN model through streamlit**

****

**Saving predicted values in MySQL for monitoring**

****