Name : Aditya Kulkarni

Class : LY – AIA – 1

Batch : A

Roll No : 2213133

Assignment No. 1

Aim: Introduction to Keras and Tensorflow (Optional – Pytorch). Configure and use google colab and kaggle GPU **Objectives:**

1. To configure anaconda and google colab, kaggle environment
2. To Explore TF/Keras/Pytorch libraries
3. To learn to use GPU/TPU
4. To learn and understand Git **Theory:**

Keras Configuration

1. Setup Environment

2. pip install keras

3. to check whether it has installed properly: python>>import keras

Python>> print keras.__version__

Tensorflow Configuration:

1. pip install tensorflow==2.2.0

2. To verify installation: python>> import tensorflow as tf

If no error then the installation has been completed

Colaboratory Configuration

1. Setup the environment

2. Connect to the Drive

3. Upload the files using: from google.colab import files

files.upload()

Kaggle Configuration:

1. Create a Kaggle Account

2. Create an Authorization token

3. Upload on Colab using the upload code

4. Make a folder and make the Json file executable

5. Get the dataset

6. Copy the API command and run on colab

GitHub Configuration

1. pip install git

2. Set up user profile by: global user.name “name”

Global user.email “mail”

Dataset Attributes:

1. Pregnancies

2. Glucose

3. BloodPressure

4. SkinThickness

5. Insulin

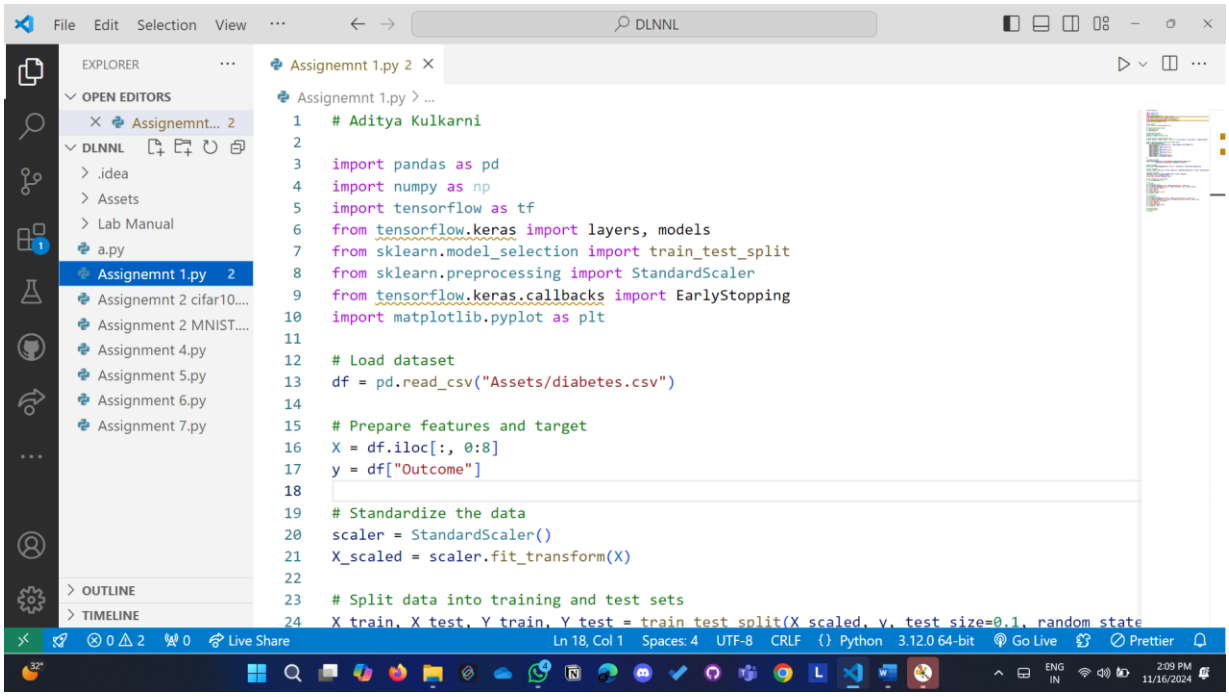
6. BMI

7. DiabetesPedigreeFunction

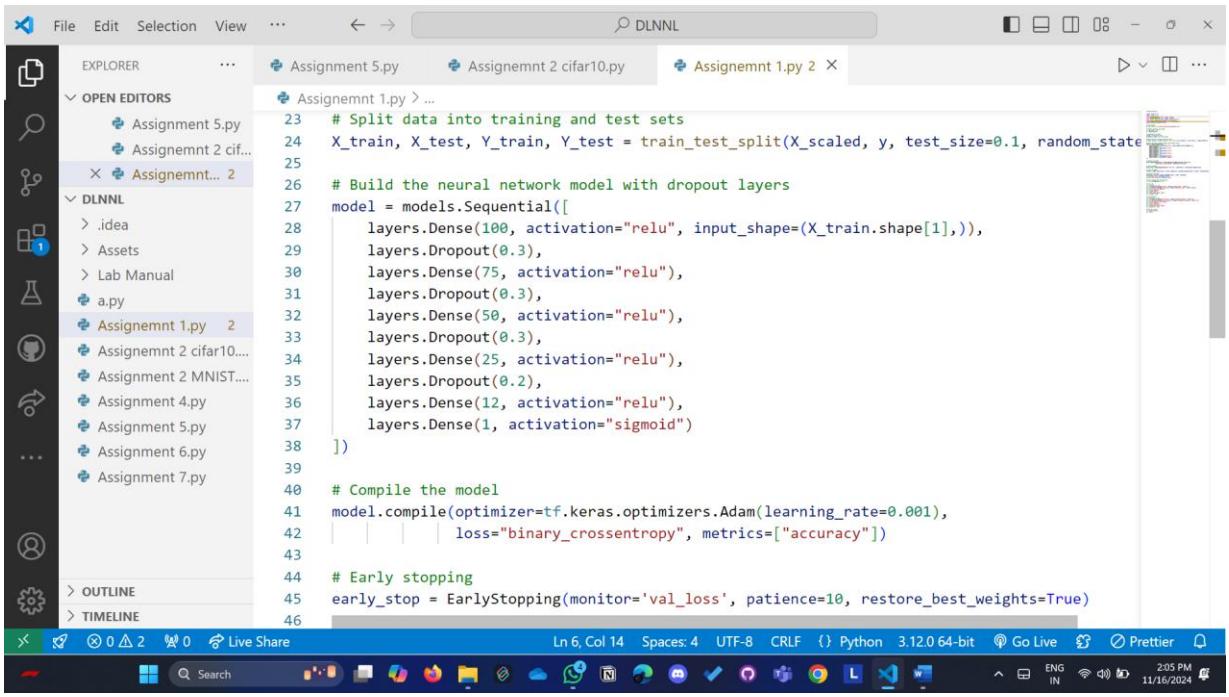
8. Age

9. Outcome

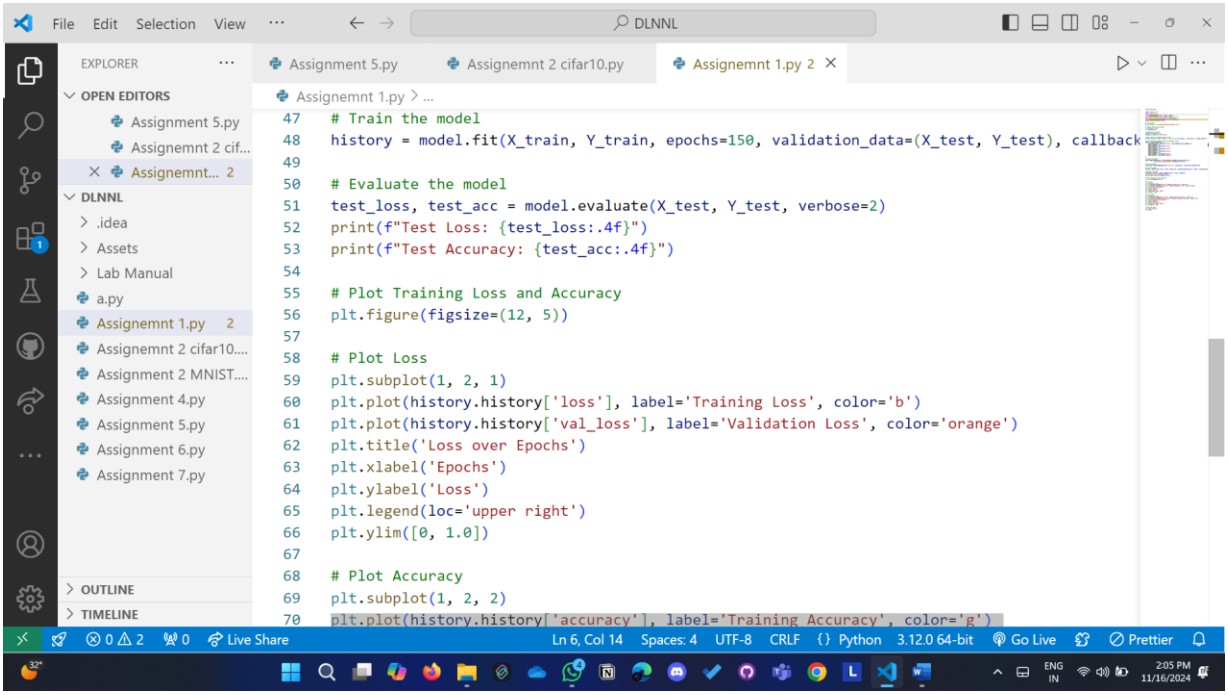
Code:

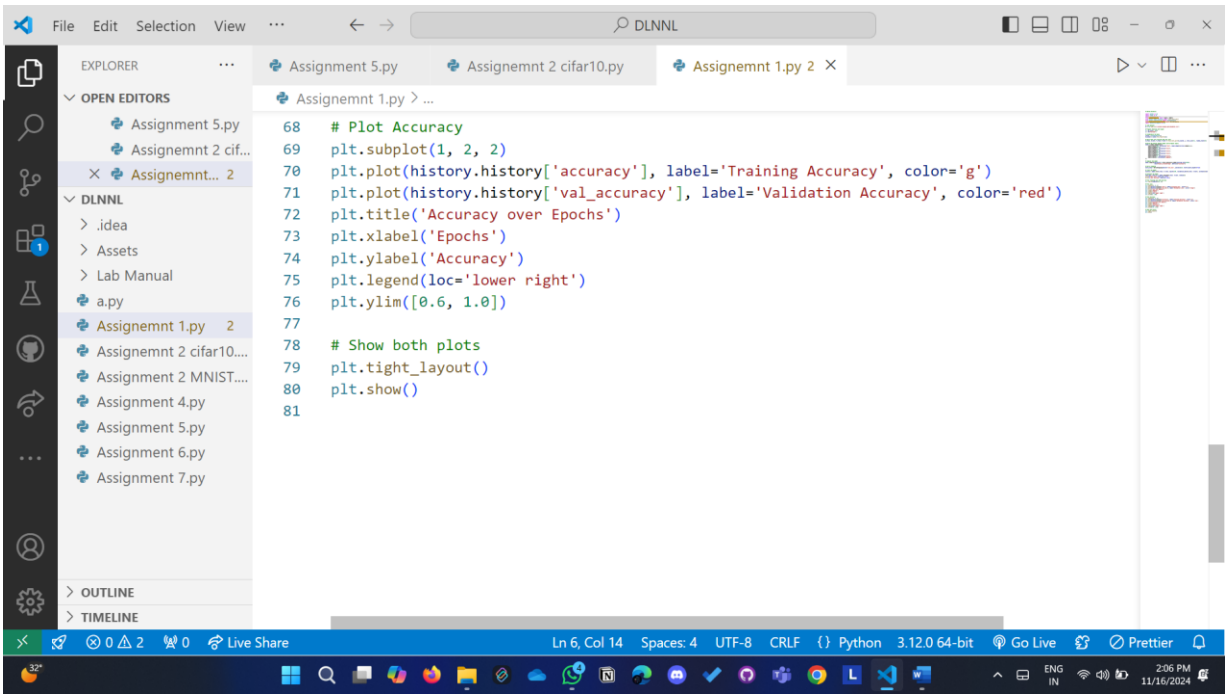


```python
# Aditya Kulkarni

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("Assets/diabetes.csv")

# Prepare features and target
X = df.iloc[:, 0:8]
y = df["Outcome"]

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, y, test_size=0.1, random_state
```



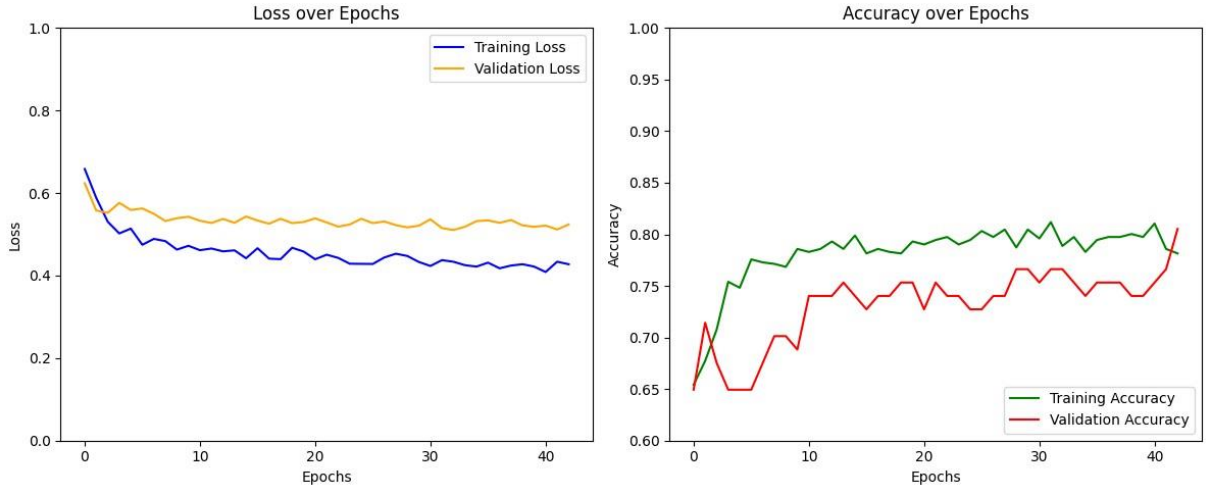
```python
# Split data into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, y, test_size=0.1, random_state

# Build the neural network model with dropout layers
model = models.Sequential([
    layers.Dense(100, activation="relu", input_shape=(X_train.shape[1],)),
    layers.Dropout(0.3),
    layers.Dense(75, activation="relu"),
    layers.Dropout(0.3),
    layers.Dense(50, activation="relu"),
    layers.Dropout(0.3),
    layers.Dense(25, activation="relu"),
    layers.Dropout(0.2),
    layers.Dense(12, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss="binary_crossentropy", metrics=["accuracy"])

# Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```


File Edit Selection View
DLNNL
EXPLORER
Assignment 5.py
Assignemnt 2 cifar10.py
Assignemnt 1.py 2
OPEN EDITORS
Assignment 5.py
Assignemnt 2 cif...
Assignemnt... 2
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py 2
Assignemnt 2 cifar10....
Assignment 2 MNIST....
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
Assignemnt 1.py > ...
47 # Train the model
48 history = model.fit(X_train, Y_train, epochs=150, validation_data=(X_test, Y_test), callback
49
50 # Evaluate the model
51 test_loss, test_acc = model.evaluate(X_test, Y_test, verbose=2)
52 print(f"Test Loss: {test_loss:.4f}")
53 print(f"Test Accuracy: {test_acc:.4f}")
54
55 # Plot Training Loss and Accuracy
56 plt.figure(figsize=(12, 5))
57
58 # Plot Loss
59 plt.subplot(1, 2, 1)
60 plt.plot(history.history['loss'], label='Training Loss', color='b')
61 plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
62 plt.title('Loss over Epochs')
63 plt.xlabel('Epochs')
64 plt.ylabel('Loss')
65 plt.legend(loc='upper right')
66 plt.ylim([0, 1.0])
67
68 # Plot Accuracy
69 plt.subplot(1, 2, 2)
70 plt.plot(history.history['accuracy'], label='Training Accuracy', color='g')
Ln 6, Col 14 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Go Live Prettier
Live Share
2:05 PM
11/16/2024


File Edit Selection View
DLNNL
EXPLORER
Assignment 5.py
Assignemnt 2 cifar10.py
Assignemnt 1.py 2
OPEN EDITORS
Assignment 5.py
Assignemnt 2 cif...
Assignemnt... 2
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py 2
Assignemnt 2 cifar10....
Assignment 2 MNIST....
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
Assignemnt 1.py > ...
68 # Plot Accuracy
69 plt.subplot(1, 2, 2)
70 plt.plot(history.history['accuracy'], label='Training Accuracy', color='g')
71 plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='red')
72 plt.title('Accuracy over Epochs')
73 plt.xlabel('Epochs')
74 plt.ylabel('Accuracy')
75 plt.legend(loc='lower right')
76 plt.ylim([0.6, 1.0])
77
78 # Show both plots
79 plt.tight_layout()
80 plt.show()
81
Ln 6, Col 14 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Go Live Prettier
Live Share
2:06 PM
11/16/2024

Results :

```
Epoch 1/150
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
22/22 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.6205 - loss: 0.6724 - val_accuracy: 0.6494 - val_loss: 0.6238
Epoch 2/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6452 - loss: 0.6180 - val_accuracy: 0.7143 - val_loss: 0.5576
Epoch 3/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6956 - loss: 0.5491 - val_accuracy: 0.6753 - val_loss: 0.5521
Epoch 4/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7461 - loss: 0.4958 - val_accuracy: 0.6494 - val_loss: 0.5762
Epoch 5/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7499 - loss: 0.5038 - val_accuracy: 0.6494 - val_loss: 0.5590
Epoch 6/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7907 - loss: 0.4510 - val_accuracy: 0.6494 - val_loss: 0.5627
Epoch 7/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7822 - loss: 0.4490 - val_accuracy: 0.6753 - val_loss: 0.5497
Epoch 8/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7677 - loss: 0.5037 - val_accuracy: 0.7013 - val_loss: 0.5322
Epoch 9/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7784 - loss: 0.4494 - val_accuracy: 0.7013 - val_loss: 0.5391
Epoch 10/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7776 - loss: 0.4798 - val_accuracy: 0.6883 - val_loss: 0.5426

22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7825 - loss: 0.4454 - val_accuracy: 0.7532 - val_loss: 0.5336
Epoch 37/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8190 - loss: 0.3848 - val_accuracy: 0.7532 - val_loss: 0.5279
Epoch 38/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8012 - loss: 0.4402 - val_accuracy: 0.7532 - val_loss: 0.5346
Epoch 39/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8240 - loss: 0.3866 - val_accuracy: 0.7403 - val_loss: 0.5215
Epoch 40/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7850 - loss: 0.4282 - val_accuracy: 0.7403 - val_loss: 0.5182
Epoch 41/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8230 - loss: 0.3836 - val_accuracy: 0.7532 - val_loss: 0.5211
Epoch 42/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8025 - loss: 0.4217 - val_accuracy: 0.7662 - val_loss: 0.5117
Epoch 43/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7989 - loss: 0.4090 - val_accuracy: 0.8052 - val_loss: 0.5238
3/3 - 0s - 8ms/step - accuracy: 0.7662 - loss: 0.5105
Test Loss: 0.5105
Test Accuracy: 0.7662
```



Conclusion:

Thus we have understood the configuration steps of Google Colab, tensorflow, etc and learned to use tensorflow and create a model to predict the chance of diabetes or not.

Name : Aditya Kulkarni

Class : LY – AIA – 1

Batch : A

Roll No : 2213133

Assignment No. 2

Aim: Develop multi class classifier using deep multilayer perceptron (Keras/tensorflow/pytorch) for MNIST hand recognition dataset and CIFAR10. Fine the parameters for better accuracy.

- Develop application with GUI to upload input to the system
- Test the model **Objectives:**
- Learn Deep Neural Network modeling
- Learn to develop and deploy models **Theory:**

Standardisation

This is one of the most use type of scalar in data preprocessing . This is known as z-score . This re distribute the data in such a way that mean (μ) = 0 and standard deviation (σ) =1 . Here is the below formula for calculation

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation } (x)}$$

Normalization:

Normalization scales the feature between 0.0 & 1.0, retaining their proportional range to each other.

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

The range of normal distribution is [-1,1] with mean =0.

Data Splitting

Train Test Split is one of the important steps in Machine Learning. It is very important because your model needs to be evaluated before it has been deployed. And that evaluation needs to be done on unseen data because when it is deployed, all incoming data is unseen.

The main idea behind the train test split is to convert original data set into 2 parts

- train
- test where train consists of training data and training labels and test consists of testing data and testing labels.

The easiest way to do it is by using *scikit-learn*, which has a built-in function *train_test_split*

Data Cleaning

Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted.

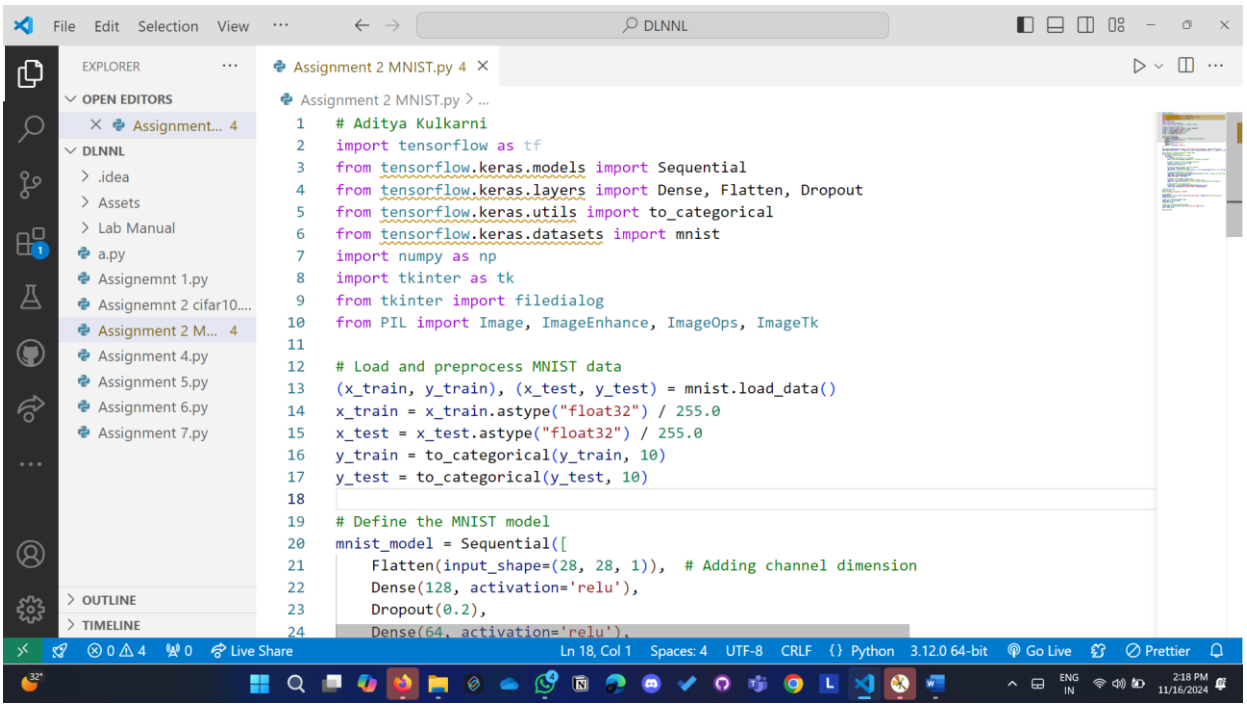
This data is usually not necessary or helpful when it comes to analyzing data because it may hinder the process or provide inaccurate results. There are several methods for cleaning data depending on how it is stored along with the answers being sought.

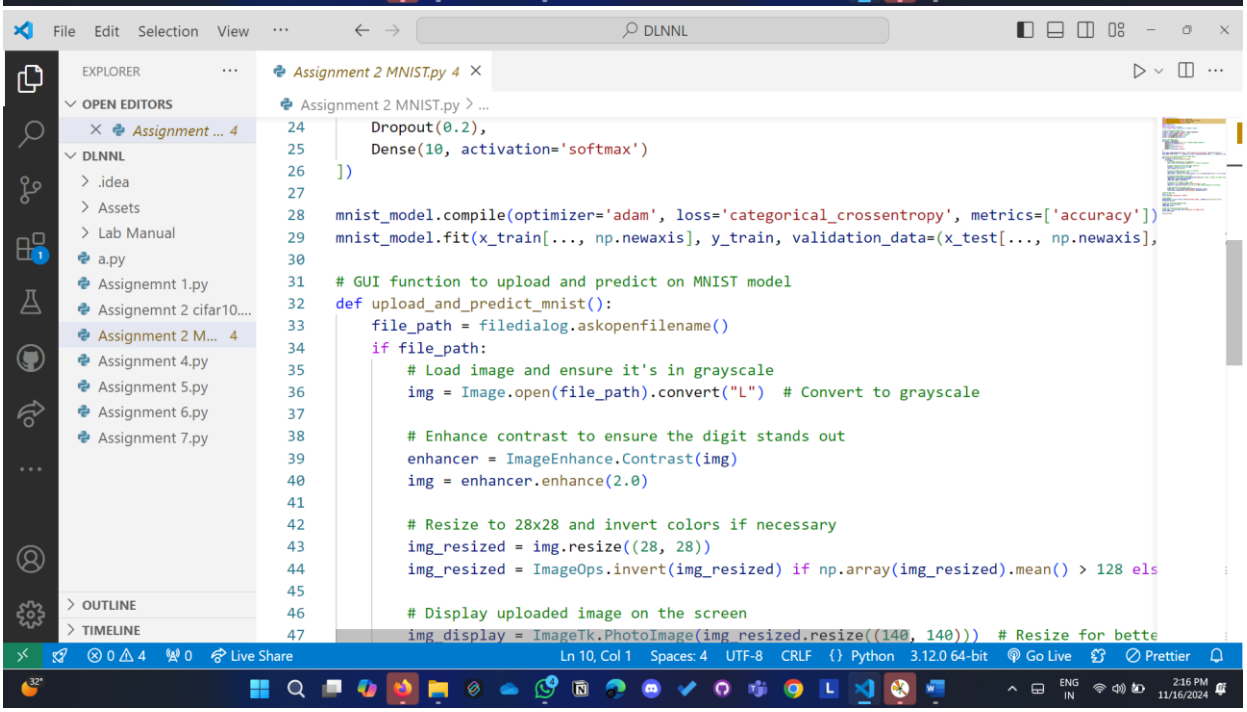
Data cleaning is not simply about erasing information to make space for new data, but rather finding a way to maximize a data set's accuracy without necessarily deleting information.

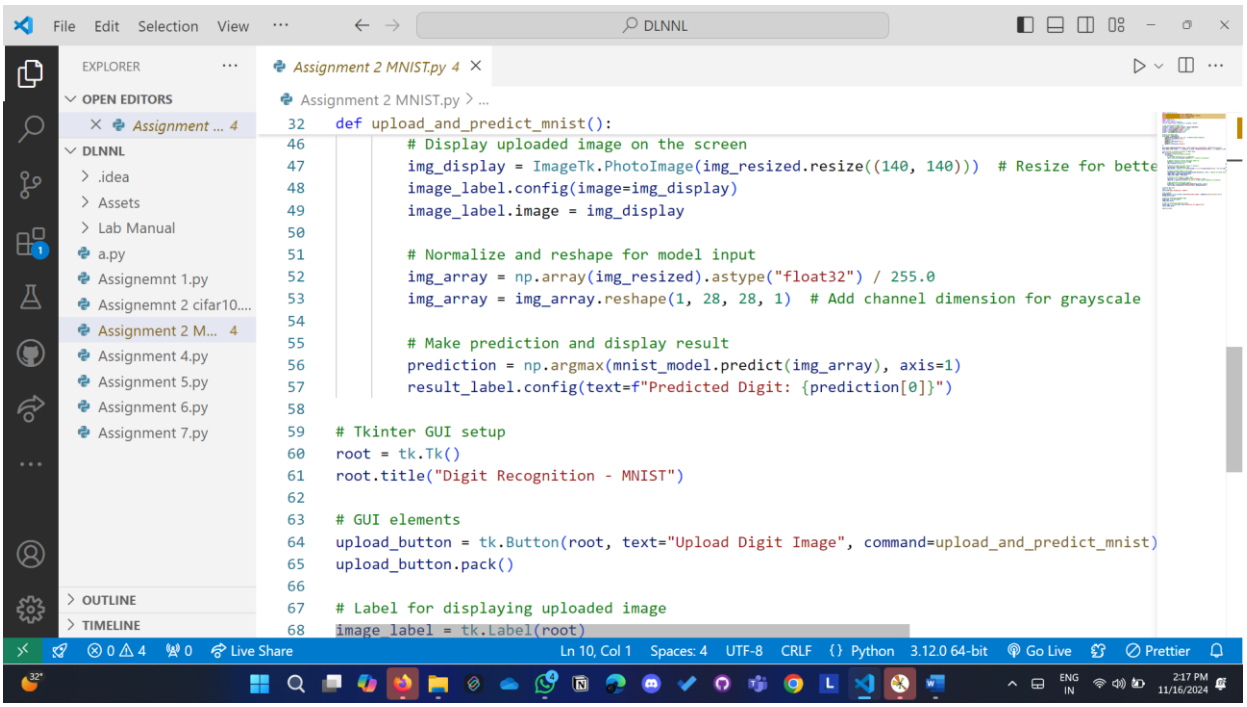
For one, data cleaning includes more actions than removing data, such as fixing spelling and syntax errors, standardizing data sets, and correcting mistakes such as empty fields, missing codes, and identifying duplicate data points. Data cleaning is considered a foundational element of the data science basics, as it plays an important role in the analytical process and uncovering reliable answers.

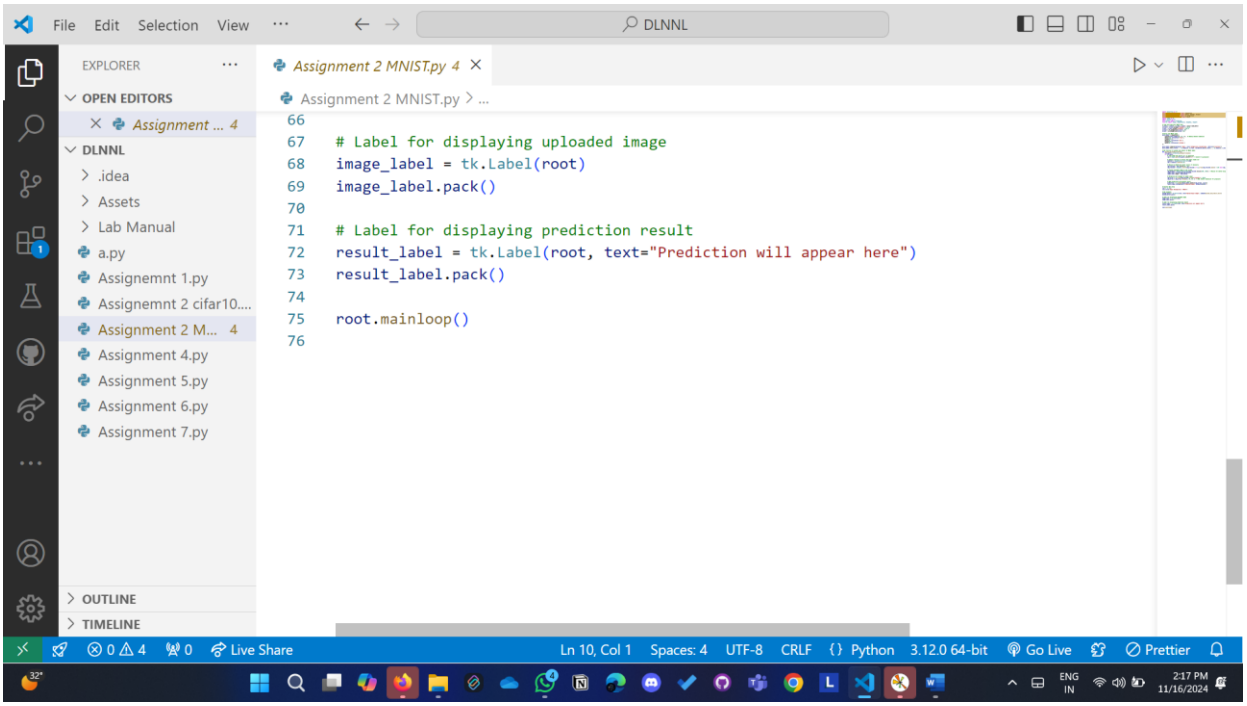
Code :

- MNIST


File Edit Selection View
DLNNL
EXPLORER
OPEN EDITORS
Assignment... 4
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 M... 4
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
Assignment 2 MNIST.py 4
Assignment 2 MNIST.py > ...
1 # Aditya Kulkarni
2 import tensorflow as tf
3 from tensorflow.keras.models import Sequential
4 from tensorflow.keras.layers import Dense, Flatten, Dropout
5 from tensorflow.keras.utils import to_categorical
6 from tensorflow.keras.datasets import mnist
7 import numpy as np
8 import tkinter as tk
9 from tkinter import filedialog
10 from PIL import Image, ImageEnhance, ImageOps, ImageTk
11
12 # Load and preprocess MNIST data
13 (x_train, y_train), (x_test, y_test) = mnist.load_data()
14 x_train = x_train.astype("float32") / 255.0
15 x_test = x_test.astype("float32") / 255.0
16 y_train = to_categorical(y_train, 10)
17 y_test = to_categorical(y_test, 10)
18
19 # Define the MNIST model
20 mnist_model = Sequential([
21 Flatten(input_shape=(28, 28, 1)), # Adding channel dimension
22 Dense(128, activation='relu'),
23 Dropout(0.2),
24 Dense(64, activation='relu'),
Ln 18, Col 1 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Go Live Prettier
2:18 PM 11/16/2024

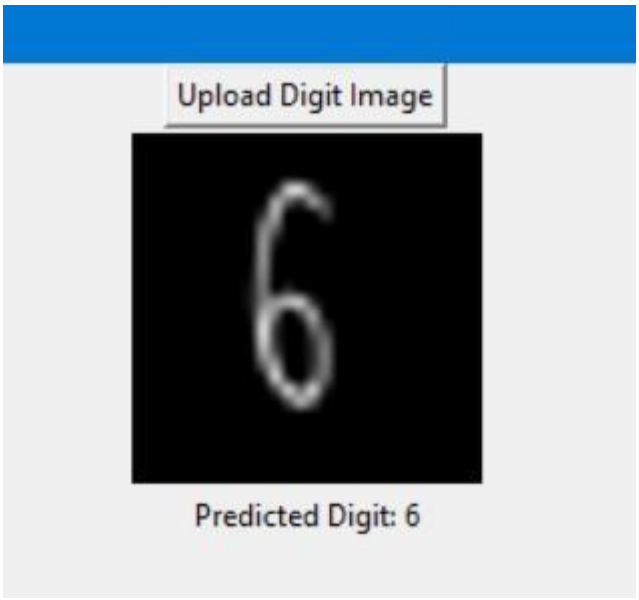

File Edit Selection View
DLNNL
EXPLORER
OPEN EDITORS
Assignment ... 4
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 M... 4
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
Assignment 2 MNIST.py 4
Assignment 2 MNIST.py > ...
24 Dropout(0.2),
25 Dense(10, activation='softmax')
26])
27
28 mnist_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
29 mnist_model.fit(x_train[..., np.newaxis], y_train, validation_data=(x_test[..., np.newaxis],
30
31 # GUI function to upload and predict on MNIST model
32 def upload_and_predict_mnist():
33 file_path = filedialog.askopenfilename()
34 if file_path:
35 # Load image and ensure it's in grayscale
36 img = Image.open(file_path).convert("L") # Convert to grayscale
37
38 # Enhance contrast to ensure the digit stands out
39 enhancer = ImageEnhance.Contrast(img)
40 img = enhancer.enhance(2.0)
41
42 # Resize to 28x28 and invert colors if necessary
43 img_resized = img.resize((28, 28))
44 img_resized = ImageOps.invert(img_resized) if np.array(img_resized).mean() > 128 els
45
46 # Display uploaded image on the screen
47 img_display = ImageTk.PhotoImage(img_resized.resize((140, 140))) # Resize for bette
Ln 10, Col 1 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Go Live Prettier
2:16 PM 11/16/2024


File Edit Selection View
DLNNL
EXPLORER
Assignment 2 MNIST.py 4
OPEN EDITORS
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 M... 4
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
Assignment 2 MNIST.py > ...
32 def upload_and_predict_mnist():
46 # Display uploaded image on the screen
47 img_display = ImageTk.PhotoImage(img_resized.resize((140, 140))) # Resize for bette
48 image_label.config(image=img_display)
49 image_label.image = img_display
50
51 # Normalize and reshape for model input
52 img_array = np.array(img_resized).astype("float32") / 255.0
53 img_array = img_array.reshape(1, 28, 28, 1) # Add channel dimension for grayscale
54
55 # Make prediction and display result
56 prediction = np.argmax(mnist_model.predict(img_array), axis=1)
57 result_label.config(text=f"Predicted Digit: {prediction[0]}")
58
59 # Tkinter GUI setup
60 root = tk.Tk()
61 root.title("Digit Recognition - MNIST")
62
63 # GUI elements
64 upload_button = tk.Button(root, text="Upload Digit Image", command=upload_and_predict_mnist)
65 upload_button.pack()
66
67 # Label for displaying uploaded image
68 image_label = tk.Label(root)
Ln 10, Col 1 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Go Live Prettier
Live Share
2:17 PM 11/16/2024

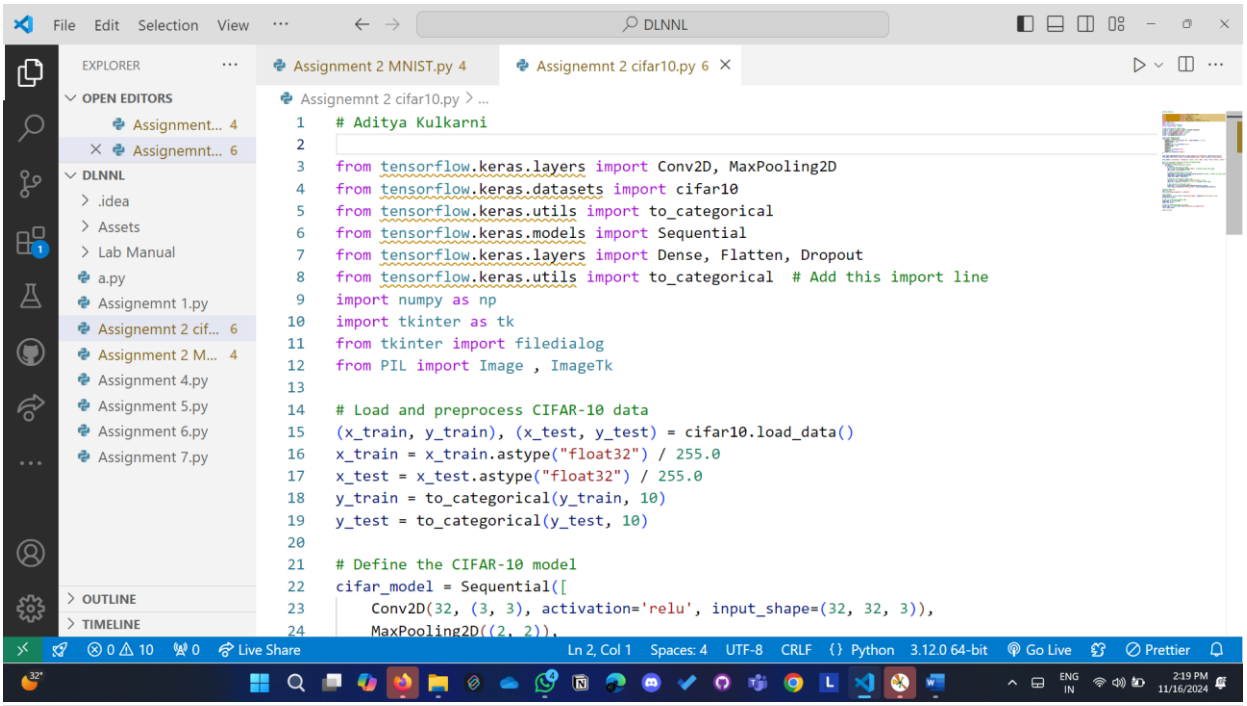

File Edit Selection View
DLNNL
EXPLORER
Assignment 2 MNIST.py 4
OPEN EDITORS
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 M... 4
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
Assignment 2 MNIST.py > ...
66
67 # Label for displaying uploaded image
68 image_label = tk.Label(root)
69 image_label.pack()
70
71 # Label for displaying prediction result
72 result_label = tk.Label(root, text="Prediction will appear here")
73 result_label.pack()
74
75 root.mainloop()
76
Ln 10, Col 1 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Go Live Prettier
Live Share
2:17 PM 11/16/2024

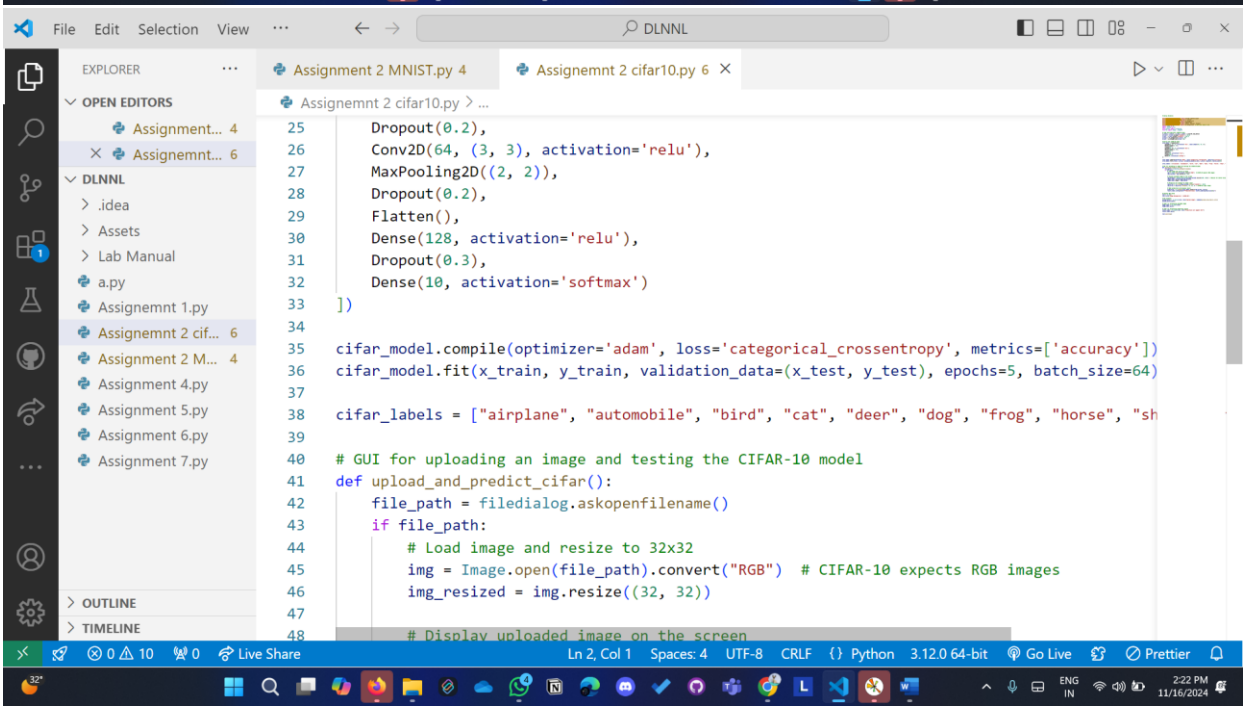
Result :

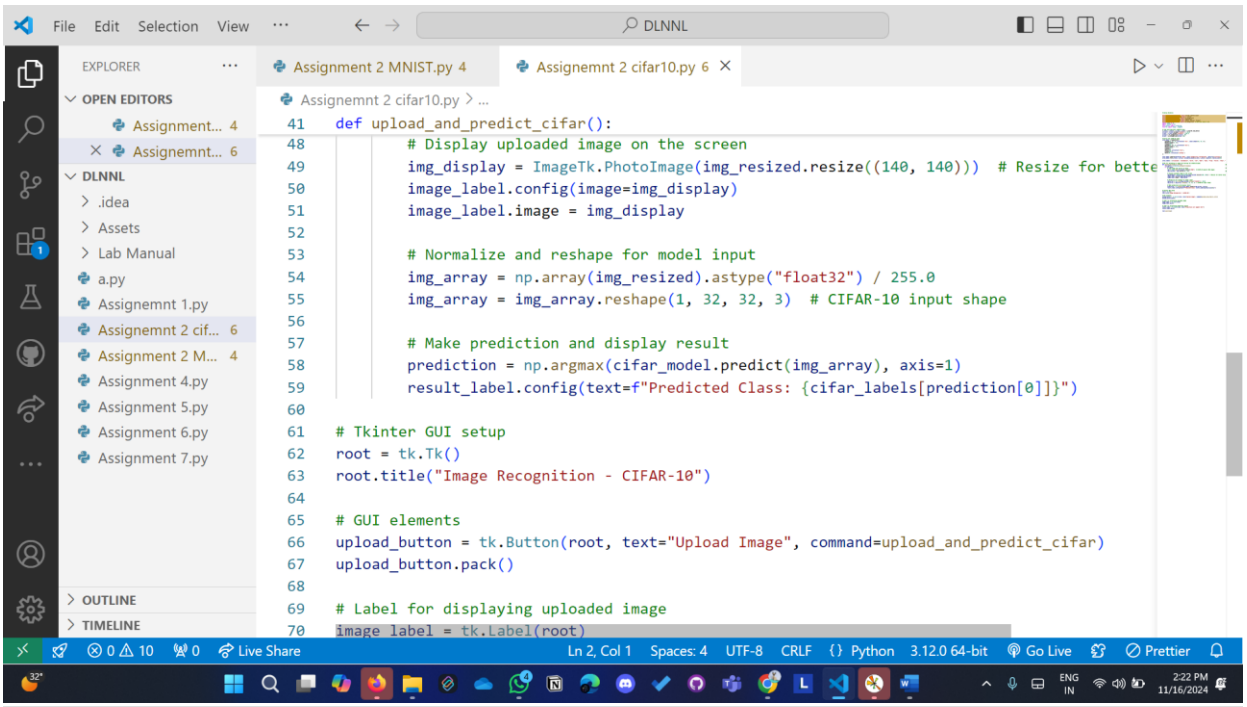
```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7570 - loss: 0.7800 - val_accuracy: 0.9511 - val_lo
ss: 0.1599
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9393 - loss: 0.2070 - val_accuracy: 0.9640 - val_lo
ss: 0.1139
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9569 - loss: 0.1491 - val_accuracy: 0.9707 - val_lo
ss: 0.0958
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9644 - loss: 0.1229 - val_accuracy: 0.9712 - val_lo
ss: 0.0899
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9673 - loss: 0.1074 - val_accuracy: 0.9759 - val_lo
ss: 0.0779
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9728 - loss: 0.0886 - val_accuracy: 0.9769 - val_lo
ss: 0.0744
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9748 - loss: 0.0834 - val_accuracy: 0.9773 - val_lo
ss: 0.0743
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9751 - loss: 0.0764 - val_accuracy: 0.9783 - val_lo
ss: 0.0714
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9773 - loss: 0.0702 - val_accuracy: 0.9804 - val_lo
ss: 0.0671
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9795 - loss: 0.0634 - val_accuracy: 0.9794 - val_lo
ss: 0.0700
```

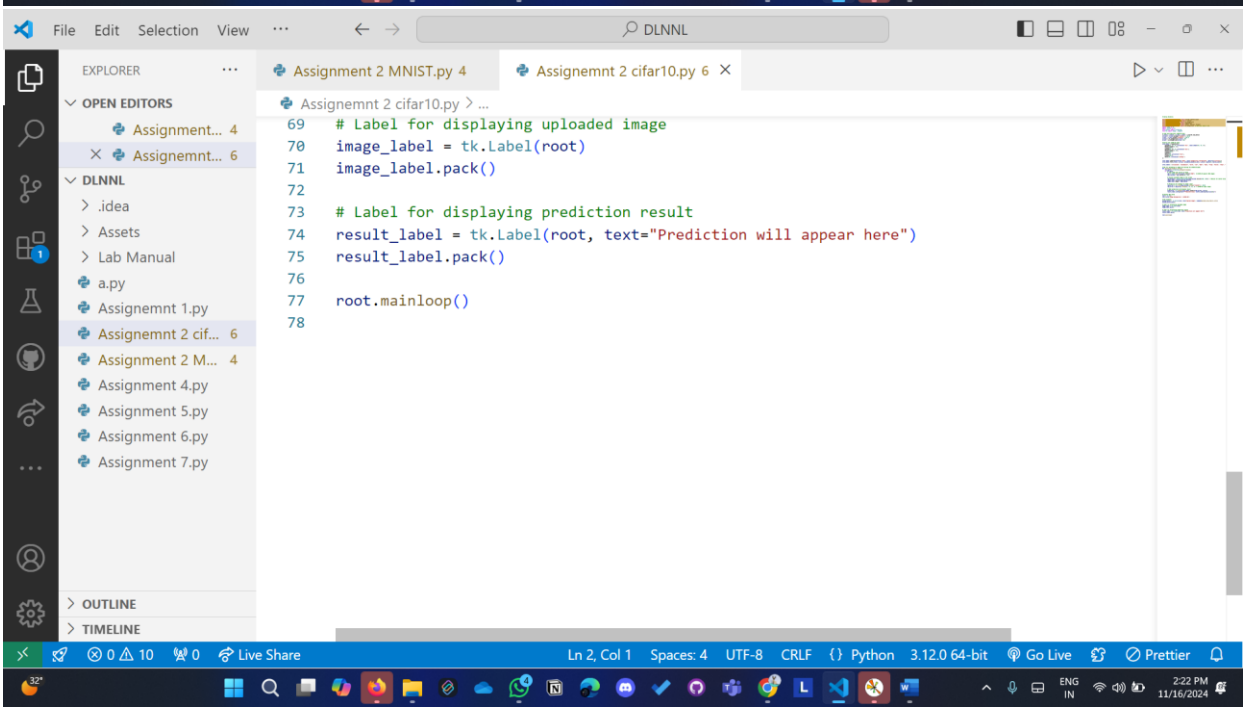


- Cifar 10


File Edit Selection View
DLNNL
EXPLORER
Assignment 2 MNIST.py 4
Assignemnt 2 cifar10.py 6
OPEN EDITORS
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cif... 6
Assignment 2 M... 4
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
# Aditya Kulkarni
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.utils import to_categorical # Add this import line
import numpy as np
import tkinter as tk
from tkinter import filedialog
from PIL import Image , ImageTk
# Load and preprocess CIFAR-10 data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# Define the CIFAR-10 model
cifar_model = Sequential([
Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
MaxPooling2D((2, 2)),
Live Share
Ln 2, Col 1
Spaces: 4
UTF-8
CRLF
Python
3.12.0 64-bit
Go Live
Prettier
2:19 PM
11/16/2024


File Edit Selection View
DLNNL
EXPLORER
Assignment 2 MNIST.py 4
Assignemnt 2 cifar10.py 6
OPEN EDITORS
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cif... 6
Assignment 2 M... 4
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
Dropout(0.2),
Conv2D(64, (3, 3), activation='relu'),
MaxPooling2D((2, 2)),
Dropout(0.2),
Flatten(),
Dense(128, activation='relu'),
Dropout(0.3),
Dense(10, activation='softmax')
])
cifar_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
cifar_model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=5, batch_size=64)
cifar_labels = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "sh
# GUI for uploading an image and testing the CIFAR-10 model
def upload_and_predict_cifar():
file_path = filedialog.askopenfilename()
if file_path:
# Load image and resize to 32x32
img = Image.open(file_path).convert("RGB") # CIFAR-10 expects RGB images
img_resized = img.resize((32, 32))
# Display uploaded image on the screen
Live Share
Ln 2, Col 1
Spaces: 4
UTF-8
CRLF
Python
3.12.0 64-bit
Go Live
Prettier
2:22 PM
11/16/2024


File Edit Selection View
DLNNL
EXPLORER
Assignment 2 MNIST.py 4
Assignemnt 2 cifar10.py 6
Assignemnt 2 cifar10.py > ...
OPEN EDITORS
Assignment... 4
Assignemnt... 6
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cif... 6
Assignment 2 M... 4
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
41 def upload_and_predict_cifar():
48 # Display uploaded image on the screen
49 img_display = ImageTk.PhotoImage(img_resized.resize((140, 140))) # Resize for bette
50 image_label.config(image=img_display)
51 image_label.image = img_display
52
53 # Normalize and reshape for model input
54 img_array = np.array(img_resized).astype("float32") / 255.0
55 img_array = img_array.reshape(1, 32, 32, 3) # CIFAR-10 input shape
56
57 # Make prediction and display result
58 prediction = np.argmax(cifar_model.predict(img_array), axis=1)
59 result_label.config(text=f"Predicted Class: {cifar_labels[prediction[0]]}")
60
61 # Tkinter GUI setup
62 root = tk.Tk()
63 root.title("Image Recognition - CIFAR-10")
64
65 # GUI elements
66 upload_button = tk.Button(root, text="Upload Image", command=upload_and_predict_cifar)
67 upload_button.pack()
68
69 # Label for displaying uploaded image
70 image_label = tk.Label(root)
Ln 2, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.0 64-bit Go Live Prettier
Live Share
0 10 0
2:22 PM 11/16/2024


File Edit Selection View
DLNNL
EXPLORER
Assignment 2 MNIST.py 4
Assignemnt 2 cifar10.py 6
Assignemnt 2 cifar10.py > ...
OPEN EDITORS
Assignment... 4
Assignemnt... 6
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cif... 6
Assignment 2 M... 4
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
69 # Label for displaying uploaded image
70 image_label = tk.Label(root)
71 image_label.pack()
72
73 # Label for displaying prediction result
74 result_label = tk.Label(root, text="Prediction will appear here")
75 result_label.pack()
76
77 root.mainloop()
78
Ln 2, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.0 64-bit Go Live Prettier
Live Share
0 10 0
2:22 PM 11/16/2024

Result :

```
Epoch 1/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 23ms/step - accuracy: 0.3208 - loss: 1.8374 - val_accuracy: 0.5454 - val_
loss: 1.2883
Epoch 2/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 23ms/step - accuracy: 0.5241 - loss: 1.3276 - val_accuracy: 0.6004 - val_
loss: 1.1321
Epoch 3/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 23ms/step - accuracy: 0.5827 - loss: 1.1748 - val_accuracy: 0.6395 - val_
loss: 1.0315
Epoch 4/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 22ms/step - accuracy: 0.6164 - loss: 1.0863 - val_accuracy: 0.6675 - val_
loss: 0.9626
Epoch 5/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 19s 25ms/step - accuracy: 0.6410 - loss: 1.0142 - val_accuracy: 0.6690 - val_
loss: 0.9603
Epoch 46/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.8193 - loss: 0.5043 - val_accuracy: 0.7445 - val_
loss: 0.8015
Epoch 47/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 19s 24ms/step - accuracy: 0.8168 - loss: 0.5019 - val_accuracy: 0.7414 - val_
loss: 0.8077
Epoch 48/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.8151 - loss: 0.5134 - val_accuracy: 0.7436 - val_
loss: 0.8071
Epoch 49/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.8216 - loss: 0.5000 - val_accuracy: 0.7385 - val_
loss: 0.8157
Epoch 50/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 25ms/step - accuracy: 0.8230 - loss: 0.4872 - val_accuracy: 0.7448 - val_
loss: 0.8142
```

Conclusion:

Thus, we have understood the syntax and basic model creation in TensorFlow for 2 different task.

We have also learned how to create a GUI using services to do so.

Name : Aditya Kulkarni

Class : LY – AIA – 1

Batch : A

Roll No : 2213133

Assignment No. 3

Aim: Develop classification model for cat-dogs dataset using CNN model. Analyze the model accuracy and generate classification report.

- Analyze the result with and without regularization/dropout
- Develop an application and test the user given inputs **Objectives:**

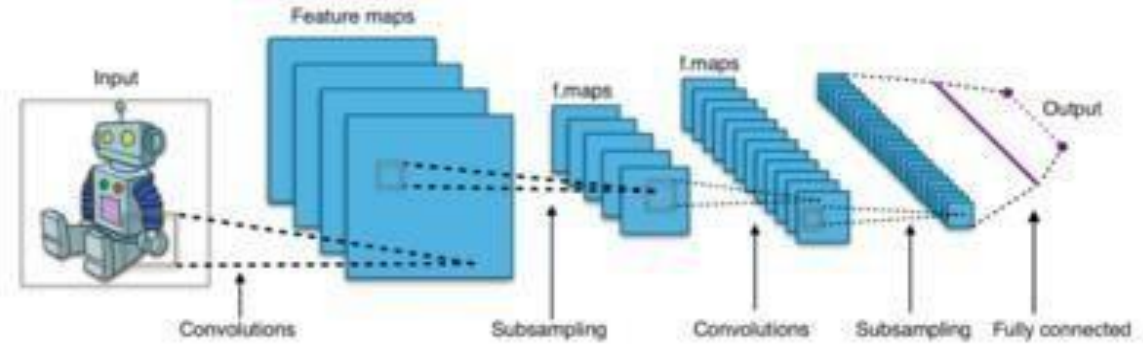
1. To learn about classification

2. To learn CNN

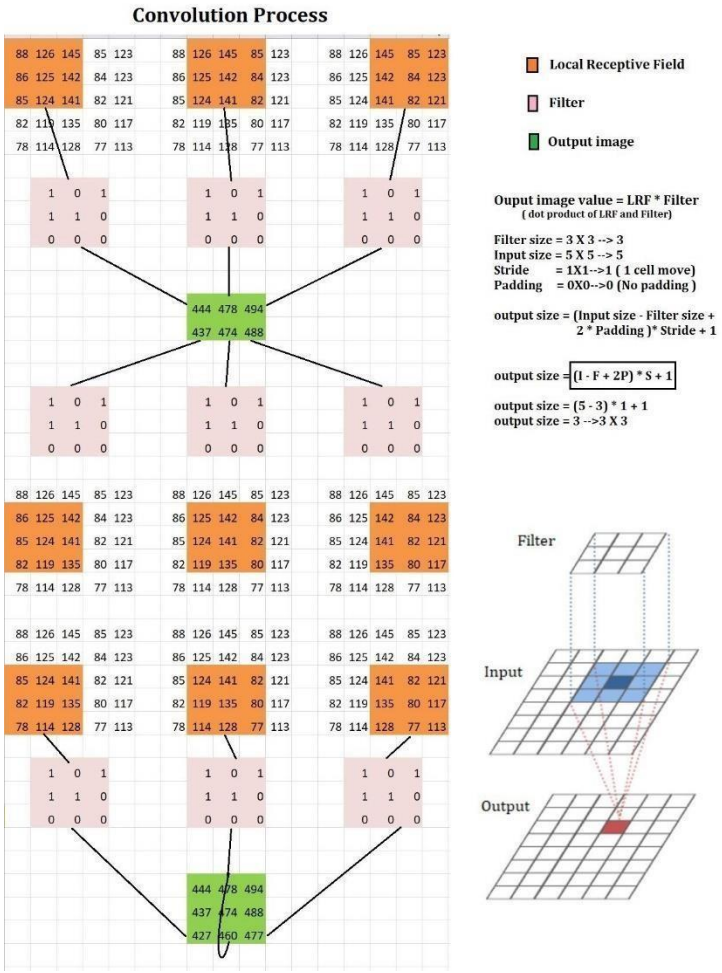
3. To demonstrate and analyse the results **Theory:**

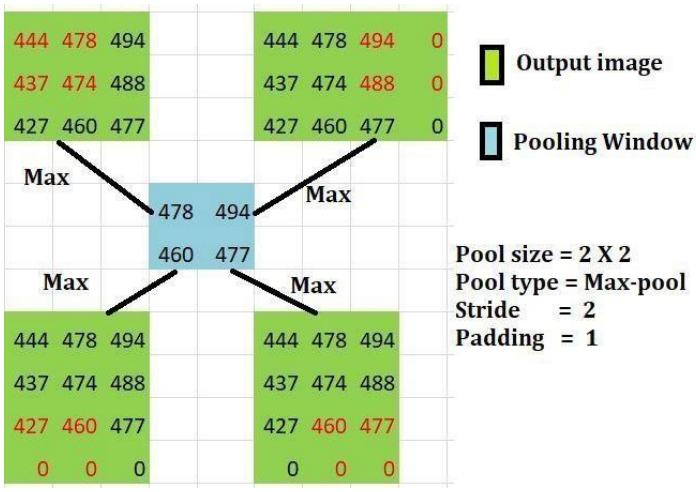
A **convolutional neural network** (**CNN**, or **ConvNet**) is a class of deep neural networks, most commonly applied to analysing visual imagery. They are also known as **shift invariant** or **space invariant artificial neural networks** (**SIANN**), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

CNNs are regularized versions of multilayer perceptron. Multilayer perceptron usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.



Mathematics





Batch Normalization

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-**batch**. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

Regularisation

This technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting. A simple relation for linear regression looks like this. Here Y represents the learned relation and β represents the coefficient estimates for different variables or predictors(X). $Y \approx \beta 0 + \beta 1X1 + \beta 2X2 + \ldots + \beta pXp$

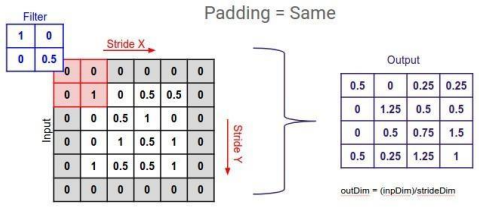
The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

Now, this will adjust the coefficients based on your training data. If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.

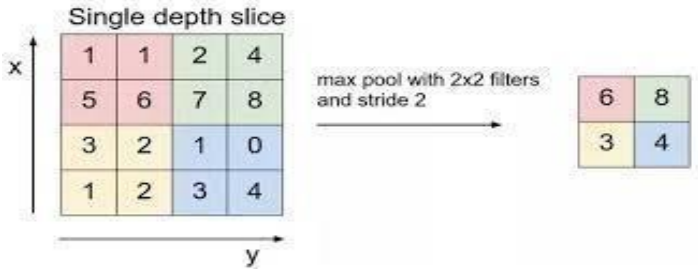
Padding

Padding is a term relevant to convolutional neural networks as it refers to the amount of pixels added to an image when it is being processed by the kernel of a **CNN**. For example, if the **padding** in a **CNN** is set to zero, then every pixel value that is added will be of value zero.

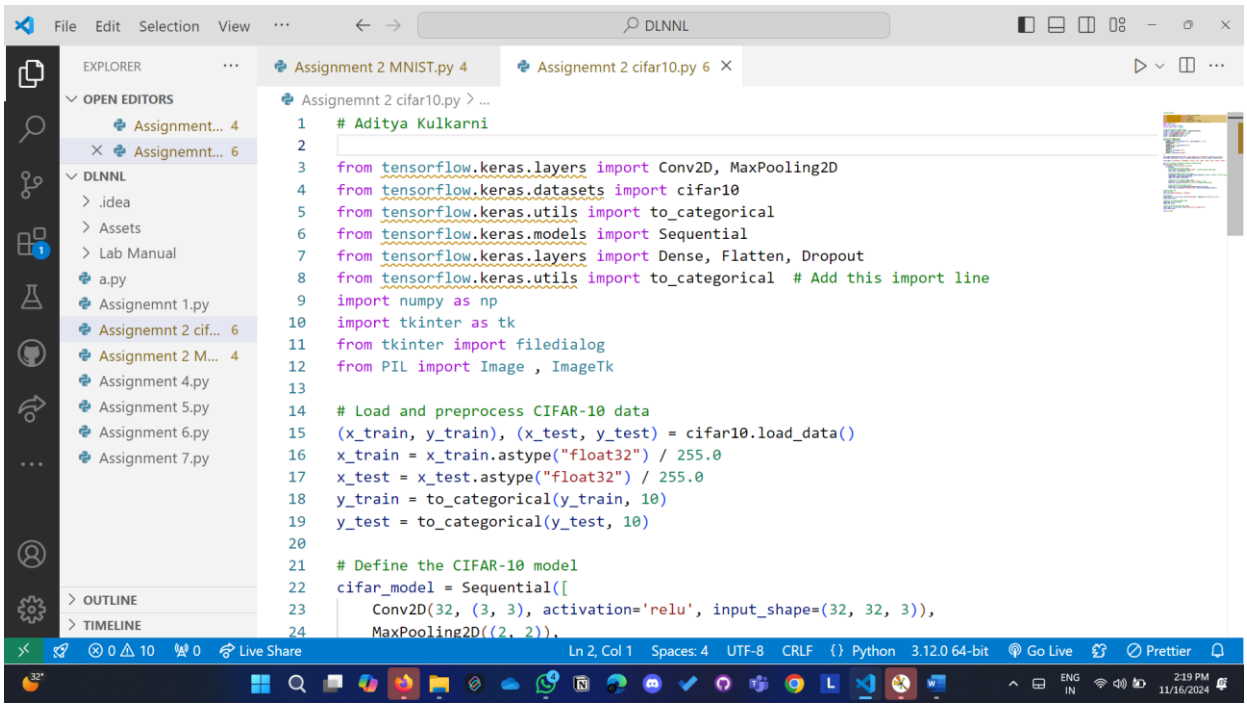


Strides

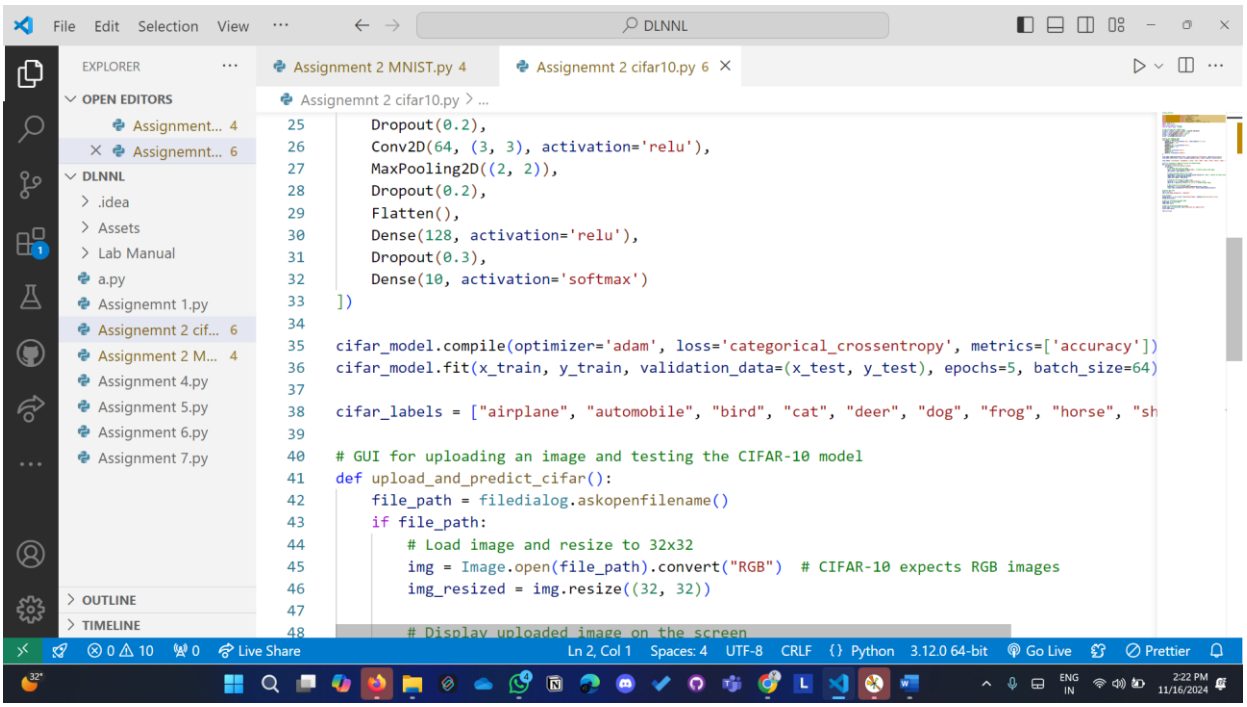
Stride is the number of pixels shifts over the input matrix. When the **stride** is 1 then we move the filters to 1 pixel at a time. When the **stride** is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a **stride** of 2.



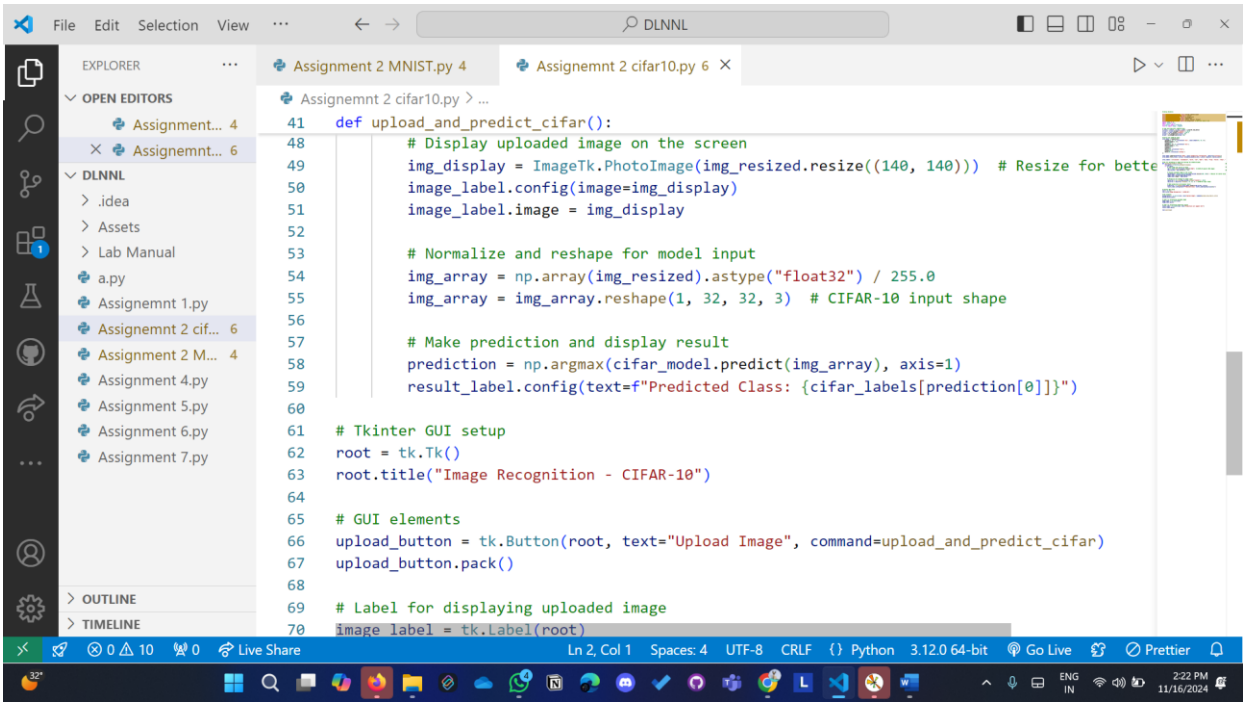
Code:

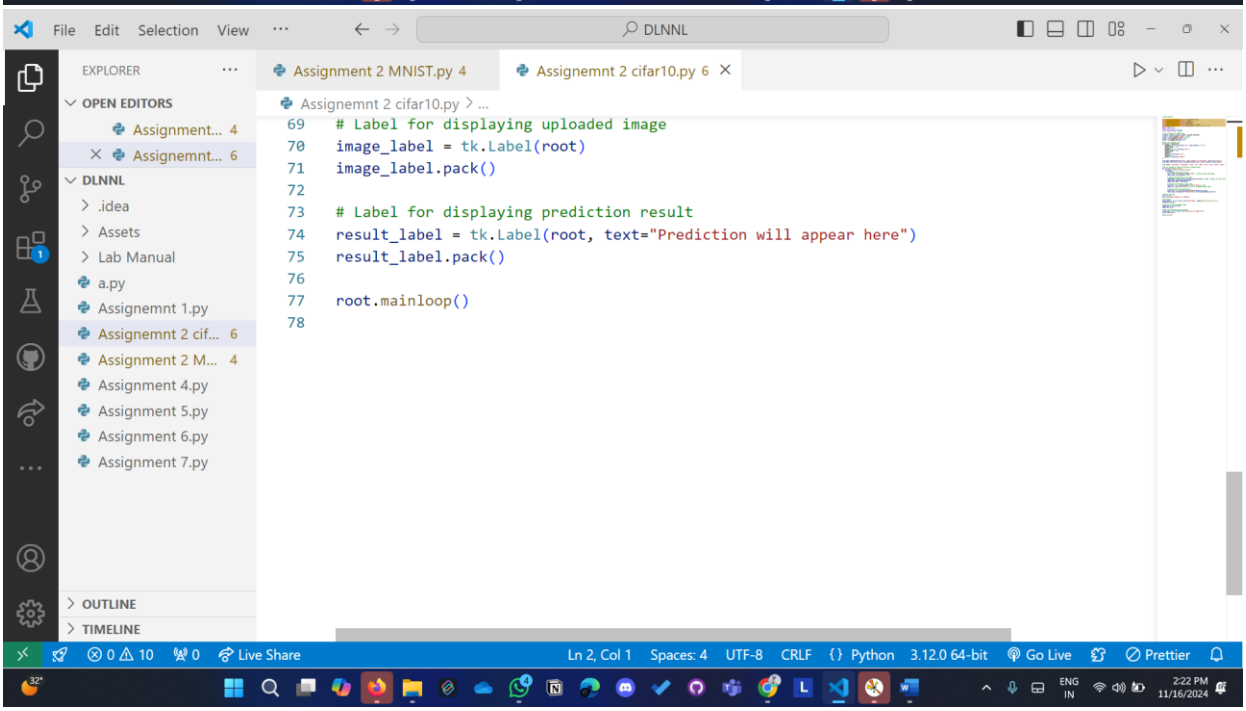


```
# Aditya Kulkarni

from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.utils import to_categorical  # Add this import line
import numpy as np
import tkinter as tk
from tkinter import filedialog
from PIL import Image , ImageTk

# Load and preprocess CIFAR-10 data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Define the CIFAR-10 model
cifar_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
```



```
    Dropout(0.2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])

cifar_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
cifar_model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=5, batch_size=64)

cifar_labels = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "sh

# GUI for uploading an image and testing the CIFAR-10 model
def upload_and_predict_cifar():
    file_path = filedialog.askopenfilename()
    if file_path:
        # Load image and resize to 32x32
        img = Image.open(file_path).convert("RGB")  # CIFAR-10 expects RGB images
        img_resized = img.resize((32, 32))

        # Display uploaded image on the screen
```


File Edit Selection View
DLNNL
EXPLORER
OPEN EDITORS
Assignment... 4
Assignemnt... 6
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cif... 6
Assignment 2 M... 4
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
Assignment 2 MNIST.py 4
Assignemnt 2 cifar10.py 6
Assignemnt 2 cifar10.py > ...
41 def upload_and_predict_cifar():
48 # Display uploaded image on the screen
49 img_display = ImageTk.PhotoImage(img_resized.resize((140, 140))) # Resize for bette
50 image_label.config(image=img_display)
51 image_label.image = img_display
52
53 # Normalize and reshape for model input
54 img_array = np.array(img_resized).astype("float32") / 255.0
55 img_array = img_array.reshape(1, 32, 32, 3) # CIFAR-10 input shape
56
57 # Make prediction and display result
58 prediction = np.argmax(cifar_model.predict(img_array), axis=1)
59 result_label.config(text=f"Predicted Class: {cifar_labels[prediction[0]]}")
60
61 # Tkinter GUI setup
62 root = tk.Tk()
63 root.title("Image Recognition - CIFAR-10")
64
65 # GUI elements
66 upload_button = tk.Button(root, text="Upload Image", command=upload_and_predict_cifar)
67 upload_button.pack()
68
69 # Label for displaying uploaded image
70 image_label = tk.Label(root)
Live Share
Ln 2, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.0 64-bit
Go Live
Prettier
2:22 PM
11/16/2024

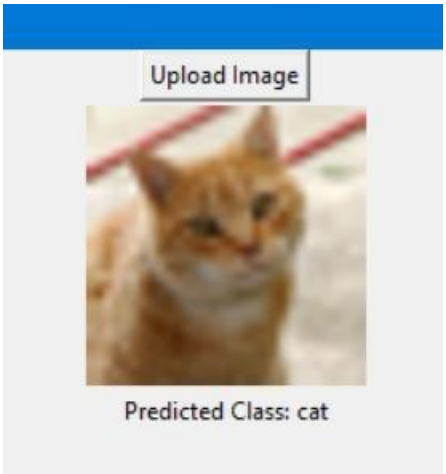

File Edit Selection View
DLNNL
EXPLORER
OPEN EDITORS
Assignment... 4
Assignemnt... 6
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cif... 6
Assignment 2 M... 4
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
Assignment 2 MNIST.py 4
Assignemnt 2 cifar10.py 6
Assignemnt 2 cifar10.py > ...
69 # Label for displaying uploaded image
70 image_label = tk.Label(root)
71 image_label.pack()
72
73 # Label for displaying prediction result
74 result_label = tk.Label(root, text="Prediction will appear here")
75 result_label.pack()
76
77 root.mainloop()
78
Live Share
Ln 2, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.0 64-bit
Go Live
Prettier
2:22 PM
11/16/2024

Results:

```
Epoch 1/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 23ms/step - accuracy: 0.3208 - loss: 1.8374 - val_accuracy: 0.5454 - val_
loss: 1.2883
Epoch 2/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 23ms/step - accuracy: 0.5241 - loss: 1.3276 - val_accuracy: 0.6004 - val_
loss: 1.1321
Epoch 3/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 23ms/step - accuracy: 0.5827 - loss: 1.1748 - val_accuracy: 0.6395 - val_
loss: 1.0315
Epoch 4/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 22ms/step - accuracy: 0.6164 - loss: 1.0863 - val_accuracy: 0.6675 - val_
loss: 0.9626
Epoch 5/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 19s 25ms/step - accuracy: 0.6410 - loss: 1.0142 - val_accuracy: 0.6690 - val_
loss: 0.9603
```

```
Epoch 46/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.8193 - loss: 0.5043 - val_accuracy: 0.7445 - val_
loss: 0.8015
Epoch 47/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 19s 24ms/step - accuracy: 0.8168 - loss: 0.5019 - val_accuracy: 0.7414 - val_
loss: 0.8077
Epoch 48/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.8151 - loss: 0.5134 - val_accuracy: 0.7436 - val_
loss: 0.8071
Epoch 49/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.8216 - loss: 0.5000 - val_accuracy: 0.7385 - val_
loss: 0.8157
Epoch 50/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 25ms/step - accuracy: 0.8230 - loss: 0.4872 - val_accuracy: 0.7448 - val_
loss: 0.8142
```

GUI



Conclusion:

Thus, we have understood how and where CNN is used and how it is programmed in TensorFlow.

We have also been able to polish GUI creation skills even further.

Name : Aditya Kulkarni
Class : LY – AIA – 1
Batch : A
Roll No : 2213133

Assignment 4

Aim: Develop face recognition system using CNN. Create a dataset of minimum 20 students from your class. Check and validate the accuracy of the model.

 Apply dimensionality reduction on input image and plot the change in accuracy of system.

Objectives:

1. To learn Data set creation

2. To learn data normalization

Theory:

Dataset creation steps

1. Articulate the problem early.
2. Establish data collection mechanisms.
3. Format data to make it consistent.
4. Reduce data.
5. Complete data cleaning.
6. Decompose data.
7. Rescale data.
8. Discretize data.

Image Augmentation

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. Image data augmentation is used to expand the training dataset in order to improve the performance and ability of the model to generalize.

The intent is to expand the training dataset with new, plausible examples. This means, variations of the training set image that are likely to be seen by the model. For example, a horizontal flip of a picture of a cat may make sense, because the photo could have been taken from the left or right. A vertical flip of the photo of a cat does not make sense and would probably not be appropriate given that the model is very unlikely to see a photo of an upside-down cat.

Libraries for image augmentation

There are a lot of image augmentations packages

- skimage
- opencv
- imgaug
- Albumentations

- Augmentor

- Keras(ImageDataGenerator class)

Fit_generator, validate_generator, predict_generator

fit_generator

```
fit_generator(
   generator, steps_per_epoch=None, epochs=1, verbose=1, callbacks=None,
   validation_data=None, validation_steps=None, validation_freq=1,
   class_weight=None, max_queue_size=10, workers=1, use_multiprocessing=False,
   shuffle=True, initial_epoch=0
)
```

Fits the model on data yielded batch-by-batch by a Python generator.

predict_generator

```
predict_generator(
   generator, steps=None, callbacks=None, max_queue_size=10, workers=1,
   use_multiprocessing=False, verbose=0
)
```

Generates predictions for the input samples from a data generator.

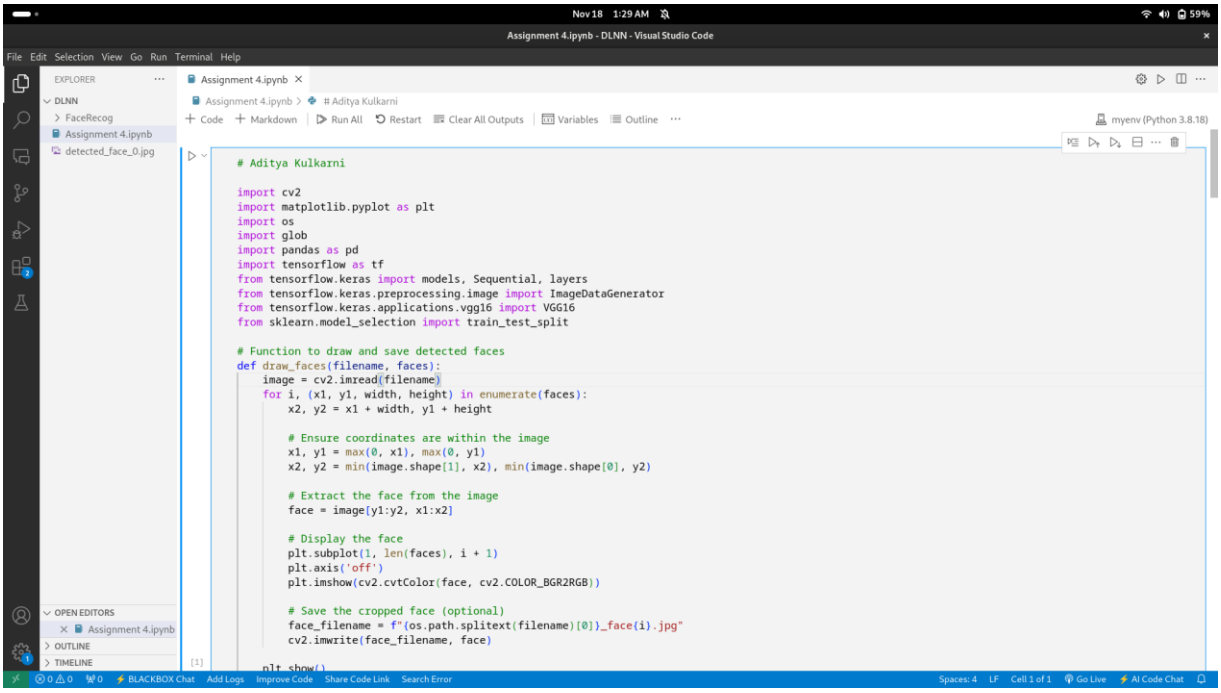
evaluate_generator

```
evaluate_generator(
   generator, steps=None, callbacks=None, max_queue_size=10, workers=1,
   use_multiprocessing=False, verbose=0
)
```

Evaluates the model on a data generator.

Code:

MTCNN



```
# Aditya Kulkarni

import cv2
import matplotlib.pyplot as plt
import os
import glob
import pandas as pd
import tensorflow as tf
from tensorflow.keras import models, Sequential, layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.vgg16 import VGG16
from sklearn.model_selection import train_test_split

# Function to draw and save detected faces
def draw_faces(filename, faces):
    image = cv2.imread(filename)
    for i, (x1, y1, width, height) in enumerate(faces):
        x2, y2 = x1 + width, y1 + height

        # Ensure coordinates are within the image
        x1, y1 = max(0, x1), max(0, y1)
        x2, y2 = min(image.shape[1], x2), min(image.shape[0], y2)

        # Extract the face from the image
        face = image[y1:y2, x1:x2]

        # Display the face
        plt.subplot(1, len(faces), i + 1)
        plt.axis('off')
        plt.imshow(cv2.cvtColor(face, cv2.COLOR_BGR2RGB))

        # Save the cropped face (optional)
        face_filename = f"{os.path.splitext(filename)[0]}_face{i}.jpg"
        cv2.imwrite(face_filename, face)

    plt.show()

# Load images and detect faces
path = glob.glob("FaceRecog/*.jpg")
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

for img in path:
    filename = img
    image = cv2.imread(filename)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
    if len(faces) > 0:
        draw_faces(filename, faces)

# Prepare dataset
file_names = os.listdir("FaceRecog")
NameArray = []
for name in file_names:
    category = name.split('.')[0]
    if category in ['girl1', 'ujjwal', 'jack','mike']:
        NameArray.append(category)
    else:
        NameArray.append('unknown')  # Handle unexpected categories

train = pd.DataFrame({
    'filename': file_names,
    'category': NameArray
})

# Filter out 'unknown' categories if any
train = train[train['category'] != 'unknown']

# Split dataset into training and validation sets
train_df, validate_df = train_test_split(train, test_size=0.2, random_state=0)
train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)
```

Nov 18 1:32 AM 59%

Assignment 4.ipynb - DLNN - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

DLNN

FaceRecog

Assignment 4.ipynb

detected_face_0.jpg

OPEN EDITORS

Assignment 4.ipynb

OUTLINE

TIMELINE

Assignment 4.ipynb

Assignment 4.ipynb > # Aditya Kulkarni

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...

myenv (Python 3.8.18)

```
# Split dataset into training and validation sets
train_df, validate_df = train_test_split(train, test_size=0.2, random_state=0)
train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)

# Data augmentation
training = ImageDataGenerator(
    rotation_range=5,
    rescale=1.0 / 255,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
)

trainingdata = training.flow_from_dataframe(
    train_df,
    "FaceRecog",
    x_col='filename',
    y_col='category',
    target_size=(224, 224),
    class_mode='categorical'
)

validation = ImageDataGenerator(rescale=1.0 / 255)

validationdata = validation.flow_from_dataframe(
    validate_df,
    "FaceRecog",
    x_col='filename',
    y_col='category',
    target_size=(224, 224),
    class_mode='categorical'
)

# Define the model
```

[1]

0 0 0 BLACKBOX Chat Add Logs Improve Code Share Code Link Search Error

Spaces: 4 LF Cell 1 of 1 Go Live AI Code Chat

Nov 18 1:33 AM 59%

Assignment 4.ipynb - DLNN - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

DLNN

FaceRecog

Assignment 4.ipynb

detected_face_0.jpg

OPEN EDITORS

Assignment 4.ipynb

OUTLINE

TIMELINE

Assignment 4.ipynb

Assignment 4.ipynb > # Aditya Kulkarni

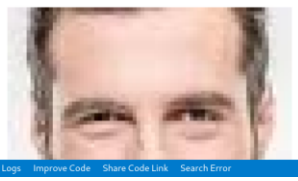
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...

myenv (Python 3.8.18)

```
# Define the model
base = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base.trainable = False

model = Sequential()
model.add(base)
model.add(layers.Flatten())
model.add(layers.Dense(400, activation='relu'))
model.add(layers.Dense(len(trainingdata.class_indices), activation='softmax'))  # Dynamically set output layer size

# Compile and train the model
model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(trainingdata, validation_data=validationdata, epochs=30)

# Evaluate the model
_, validation_acc = model.evaluate(validationdata, verbose=0)
print(f'Validation Accuracy: {validation_acc}')
```

[1] ✓ 30.6s Python

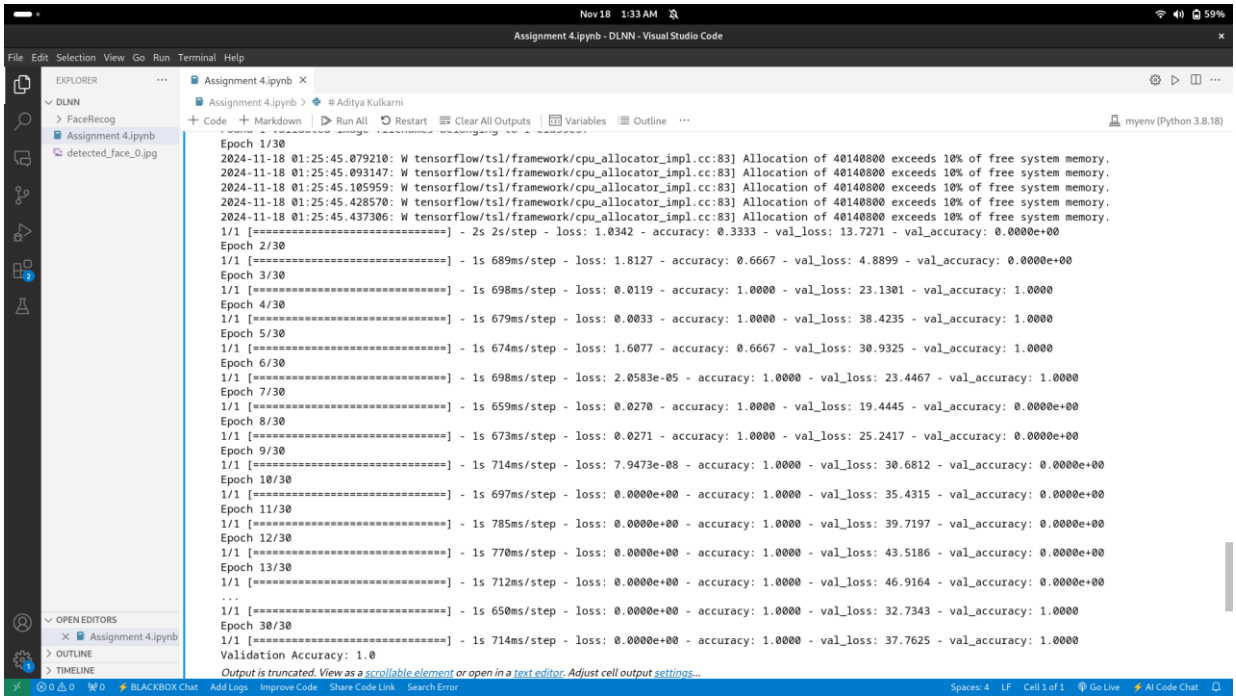
```
2024-11-18 01:25:39.465877: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers on your machine, GPU will not be used.
2024-11-18 01:25:39.847145: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers on your machine, GPU will not be used.
2024-11-18 01:25:39.848541: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in p
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-11-18 01:25:41.064284: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
```



0 0 0 BLACKBOX Chat Add Logs Improve Code Share Code Link Search Error

Spaces: 4 LF Cell 1 of 1 Go Live AI Code Chat

Results:



Conclusion:

Thus, we have understood how to create Face recognition system is used and how it is programmed in TensorFlow.

Plus we have also learned to use various face detection algorithms

Name : Aditya Kulkarni

Class : LY – AIA – 1

Batch : A

Roll No : 2213133

Assignment No. 5

Aim: Write a program to demonstrate the change in accuracy/loss/convergence time with change in optimizers like stochastic gradient descent, adam, adagrad, RMSprop and Nadam for any suitable application **Objectives:**

1. To learn optimization algorithms 2. To

learn and understand hyperparameters

Theory:

SGD, Adam, RMSprop, Nadam

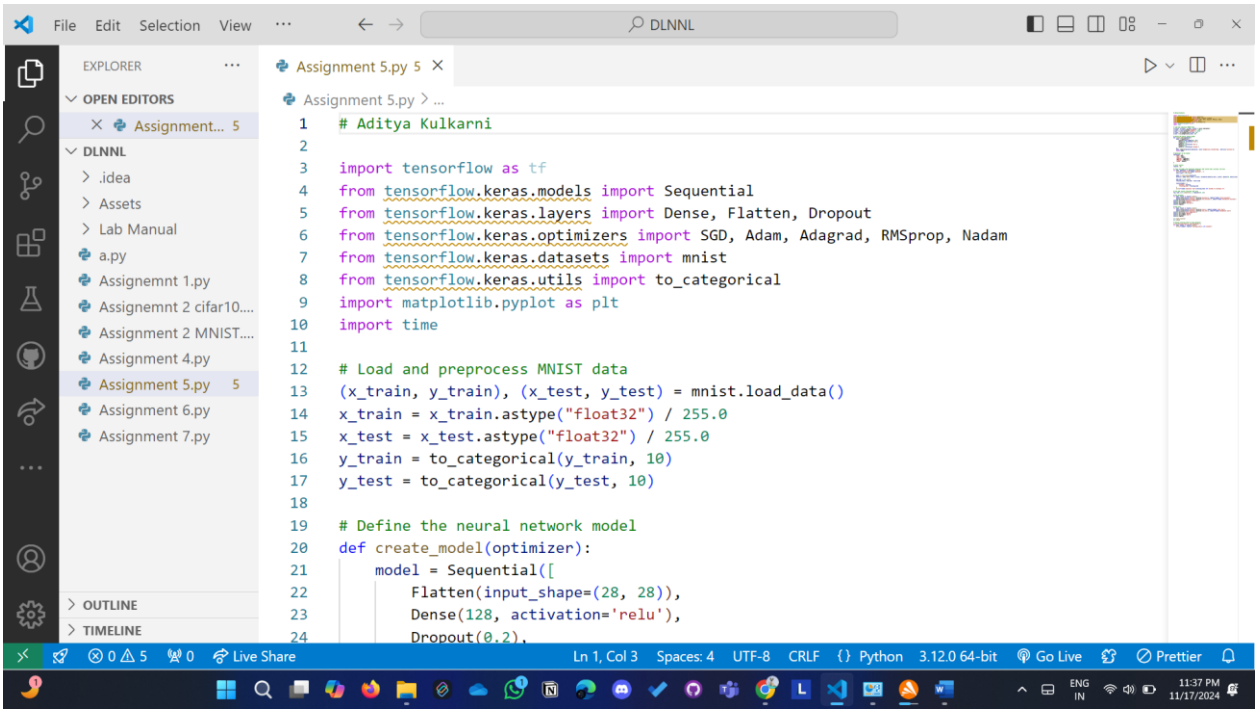
The word '*stochastic*'means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets gets big.

Adam is a replacement optimization algorithm for stochastic gradient descent for training **deep learning** models. **Adam** combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

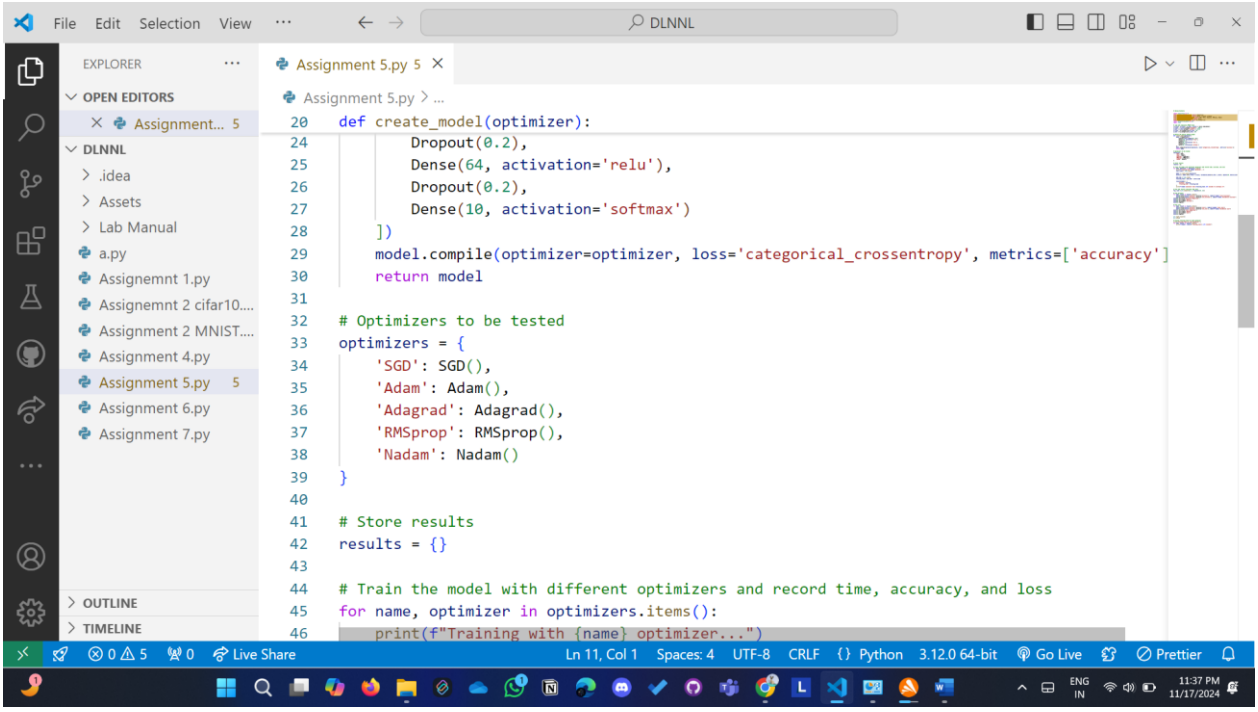
The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated. The following equations show how the gradients are calculated for the RMSprop and gradient descent with momentum. The value of momentum is denoted by beta and is usually set to 0.9.

Nadam combines NAG and Adam. Nadam is employed for noisy gradients or for gradients with high curvatures. The learning process is accelerated by summing up the exponential decay of the moving averages for the previous and current gradient

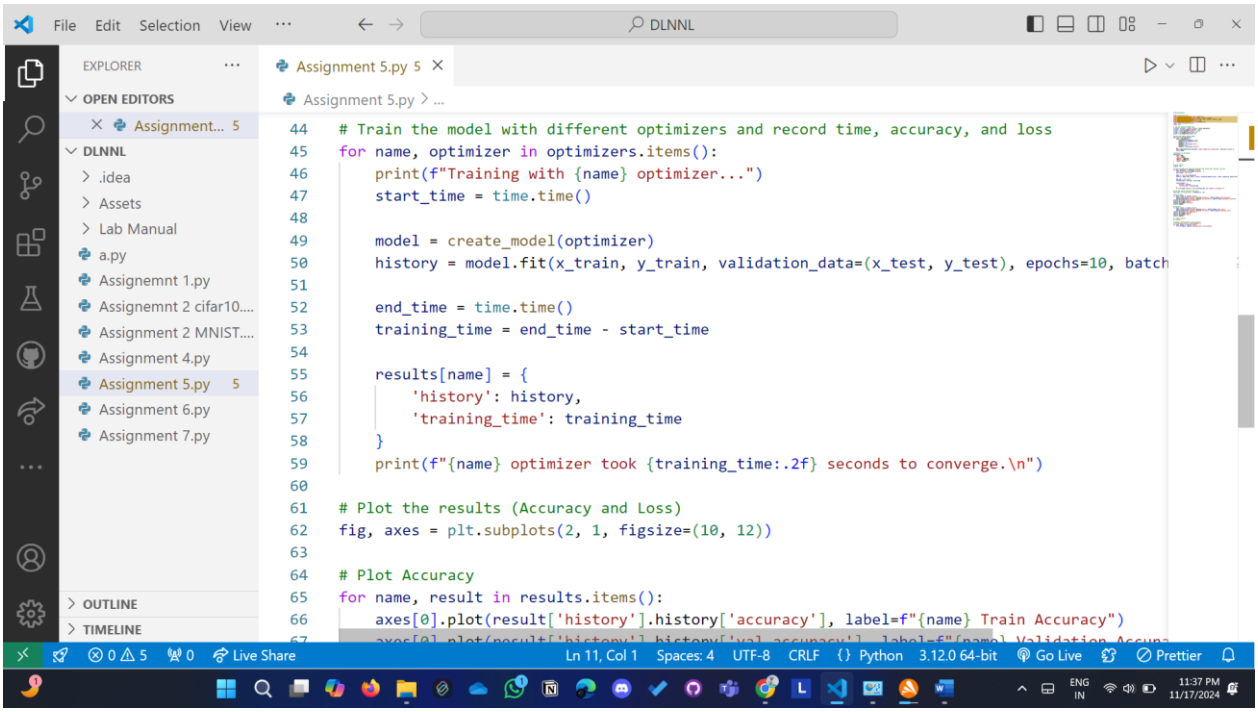
Code:

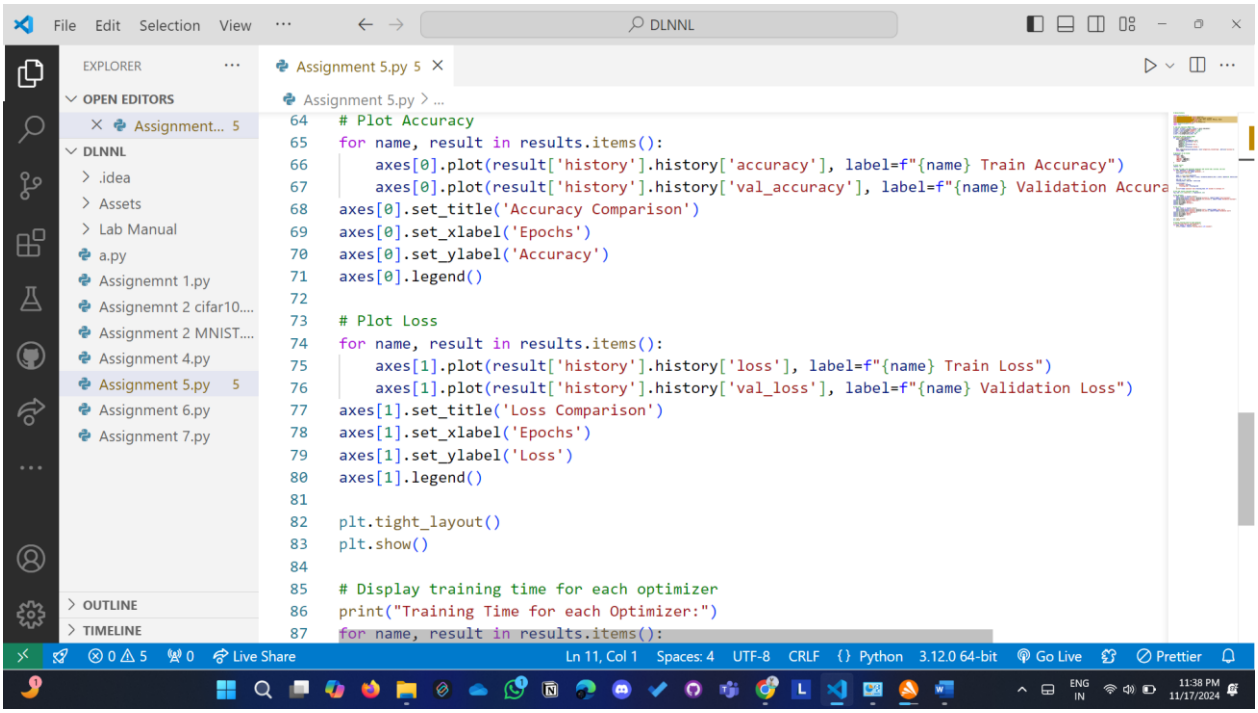


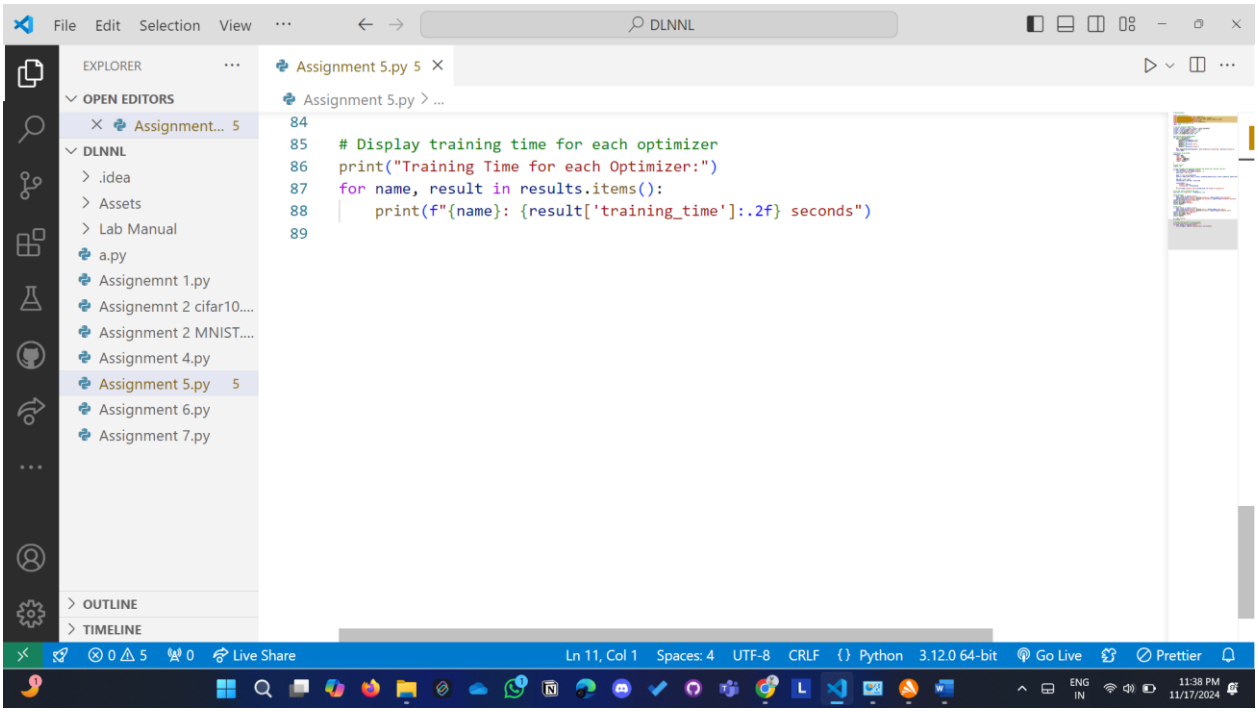
```python
# Aditya Kulkarni

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import SGD, Adam, Adagrad, RMSprop, Nadam
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import time

# Load and preprocess MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Define the neural network model
def create_model(optimizer):
    model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(128, activation='relu'),
        Dropout(0.2),
```



```python
    def create_model(optimizer):
        Dropout(0.2),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Optimizers to be tested
optimizers = {
    'SGD': SGD(),
    'Adam': Adam(),
    'Adagrad': Adagrad(),
    'RMSprop': RMSprop(),
    'Nadam': Nadam()
}

# Store results
results = {}

# Train the model with different optimizers and record time, accuracy, and loss
for name, optimizer in optimizers.items():
    print(f"Training with {name} optimizer...")
```

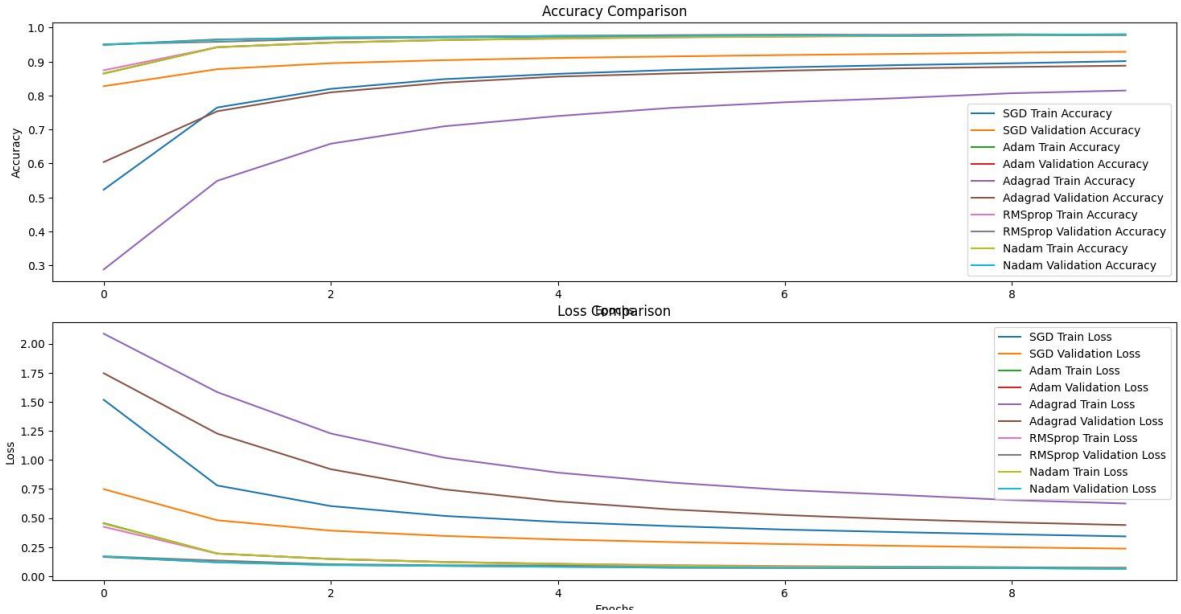

File Edit Selection View
DLNNL
EXPLORER
Assignment 5.py 5
OPEN EDITORS
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 MNIST....
Assignment 4.py
Assignment 5.py 5
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
# Train the model with different optimizers and record time, accuracy, and loss
for name, optimizer in optimizers.items():
print(f"Training with {name} optimizer...")
start_time = time.time()
model = create_model(optimizer)
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, batch
end_time = time.time()
training_time = end_time - start_time
results[name] = {
'history': history,
'training_time': training_time
}
print(f"{name} optimizer took {training_time:.2f} seconds to converge.\n")
# Plot the results (Accuracy and Loss)
fig, axes = plt.subplots(2, 1, figsize=(10, 12))
# Plot Accuracy
for name, result in results.items():
axes[0].plot(result['history'].history['accuracy'], label=f"{name} Train Accuracy")
Live Share
Ln 11, Col 1
Spaces: 4
UTF-8
CRLF
Python
3.12.0 64-bit
Go Live
Prettier
11:37 PM
11/17/2024


File Edit Selection View
DLNNL
EXPLORER
Assignment 5.py 5
OPEN EDITORS
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 MNIST....
Assignment 4.py
Assignment 5.py 5
Assignment 6.py
Assignment 7.py
OUTLINE
TIMELINE
# Plot Accuracy
for name, result in results.items():
axes[0].plot(result['history'].history['accuracy'], label=f"{name} Train Accuracy")
axes[0].plot(result['history'].history['val_accuracy'], label=f"{name} Validation Accura
axes[0].set_title('Accuracy Comparison')
axes[0].set_xlabel('Epochs')
axes[0].set_ylabel('Accuracy')
axes[0].legend()
# Plot Loss
for name, result in results.items():
axes[1].plot(result['history'].history['loss'], label=f"{name} Train Loss")
axes[1].plot(result['history'].history['val_loss'], label=f"{name} Validation Loss")
axes[1].set_title('Loss Comparison')
axes[1].set_xlabel('Epochs')
axes[1].set_ylabel('Loss')
axes[1].legend()
plt.tight_layout()
plt.show()
# Display training time for each optimizer
print("Training Time for each Optimizer:")
for name, result in results.items():
Live Share
Ln 11, Col 1
Spaces: 4
UTF-8
CRLF
Python
3.12.0 64-bit
Go Live
Prettier
11:38 PM
11/17/2024



Result :

```
SGD optimizer took 13.52 seconds to converge.

Training with Adam optimizer...
Adam optimizer took 15.86 seconds to converge.

Training with Adagrad optimizer...
Adagrad optimizer took 14.59 seconds to converge.

Training with RMSprop optimizer...
RMSprop optimizer took 15.14 seconds to converge.

Training with Nadam optimizer...
Nadam optimizer took 16.63 seconds to converge.

Training Time for each Optimizer:
SGD: 13.52 seconds
Adam: 15.86 seconds
Adagrad: 14.59 seconds
RMSprop: 15.14 seconds
Nadam: 16.63 seconds
PS D:\2213133>
```



Conclusion:

Thus, we have understood the difference in performance of various optimisation algorithms

Name : Aditya Kulkarni

Class : LY – AIA – 1

Batch : A

Roll No : 2213133

Assignment No. 6

Aim: Apply transfer learning with pre-trained VGG16 model on assignment 3 and analyze the result.

Objectives:

1. To learn pre-trained models

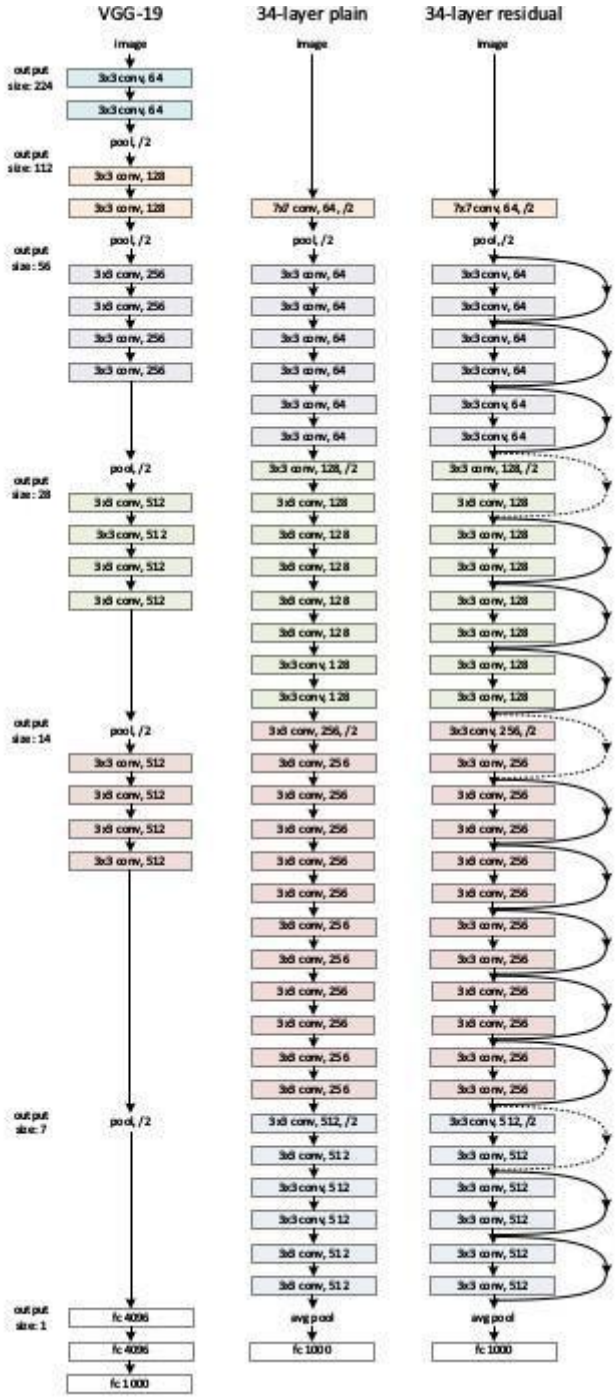
2. To learn transfer learning **Theory:**

Transfer learning

Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. This area of research bears some relation to the long history of psychological literature on transfer of learning, although formal ties between the two fields are limited. From the practical standpoint, reusing or transferring information from previously learned tasks for the learning of new tasks has the potential to significantly improve the sample efficiency of a reinforcement learning agent.

ResNet 50

- Use 3*3 filters mostly.
- Down sampling with CNN layers with stride 2.
- Global average pooling layer and a 1000-way fully-connected layer with Softmax in the end.



Plain VGG and VGG with Residual Blocks

There are two kinds of residual connections:

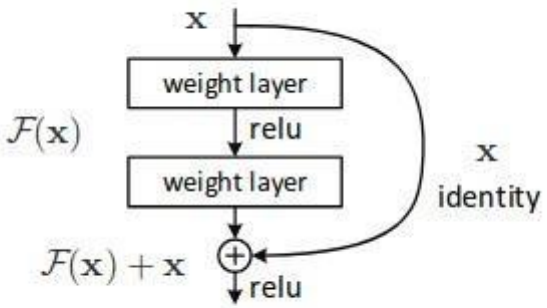


Figure 2. Residual learning: a building block.

Residual block

1. The identity shortcuts (x) can be directly used when the input and output are of the same dimensions.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \qquad (1)$$

Residual block function when input and output dimensions are same

2. When the dimensions change, A) The shortcut still performs identity mapping, with extra zero entries padded with the increased dimension. B) The projection shortcut is used to match the dimension (done by 1*1 conv) using the following formula

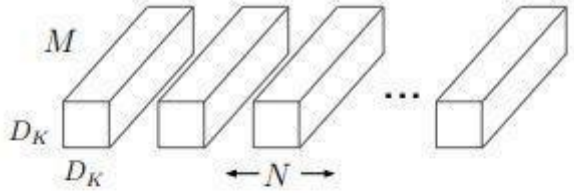
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}. \qquad (2)$$

Residual block function when the input and output dimensions are not same.

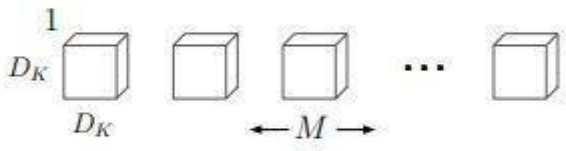
The first case adds no extra parameters, the second one adds in the form of W_{s}

MOBILENET

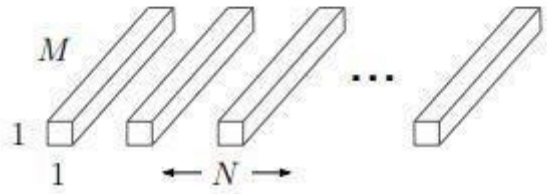
MobileNet is an efficient and portable CNN architecture that is used in real world applications. MobileNets primarily use **depthwise seperable convolutions** in place of the standard convolutions used in earlier architectures to build lighter models.MobileNets introduce two new global hyperparameters(width multiplier and resolution multiplier) that allow model developers to trade off **latency** or **accuracy** for speed and low size depending on their requirements.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



Architecture

MobileNets are built on depthwise seperable convolution layers.Each depthwise seperable convolution layer consists of a depthwise convolution and a pointwise convolution.Counting depthwise and pointwise convolutions as seperate layers, a MobileNet has 28 layers.A standard MobileNet has 4.2 million parameters which can be further reduced by tuning the width multiplier hyperparameter appropriately.
The size of the input image is 224 × 224 × 3.

The detailed architecture of a MobileNet is given below :

TYPE	STRIDE	KERNEL SHAPE	INPUT SIZE
Conv	2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw	1	3 × 3 × 32	112 × 112 × 32
Conv	1	1 × 1 × 32 × 64	112 × 112 × 32

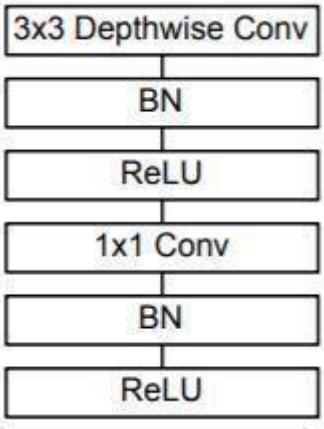
Conv dw	2	3 × 3 × 64	112 × 112 × 64
Conv	1	1 × 1 × 64 × 128	56 × 56 × 64
TYPE	**STRIDE**	**KERNEL SHAPE**	**INPUT SIZE**
Conv dw	1	3 × 3 × 128	56 × 56 × 128
Conv	1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw	2	3 × 3 × 128	56 × 56 × 128
Conv	1	1 × 1 × 128 × 256	56 × 56 × 128
Conv dw	1	3 × 3 × 256	28 × 28 × 256
Conv	1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw	2	3 × 3 × 256	28 × 28 × 256
Conv	1	1 × 1 × 256 × 512	14 × 14 × 256
Conv dw	1	3 × 3 × 512	14 × 14 × 512
Conv	1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw	1	3 × 3 × 512	14 × 14 × 512
Conv	1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw	1	3 × 3 × 512	14 × 14 × 512
Conv	1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw	1	3 × 3 × 512	14 × 14 × 512
Conv	1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw	1	3 × 3 × 512	14 × 14 × 512
Conv	1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw	2	3 × 3 × 512	14 × 14 × 512
Conv	1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw	2	3 × 3 × 1024	7 × 7 × 1024
Conv	1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool	1	Pool 7 × 7	7 × 7 × 1024
Fully Connected	1	1024 × 1000	1 × 1 × 1024
Softmax	1	Classifier	1 × 1 × 1000

Standard Convoltion layer :

A single standard convolution unit (denoted by **Conv** in the table above) looks like this :

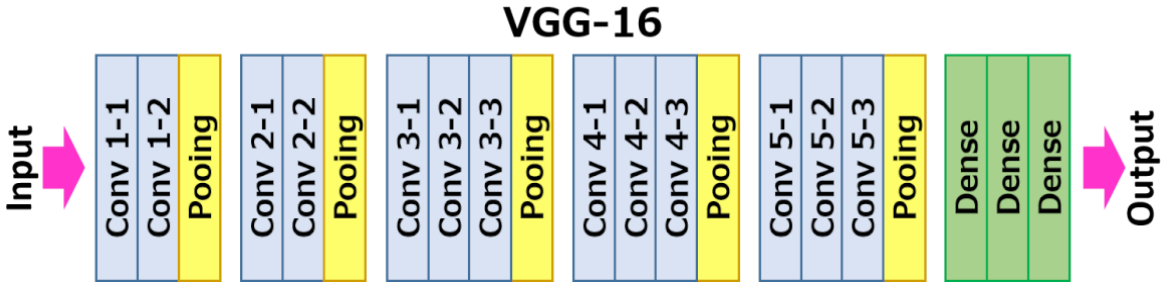
Depthwise seperable Convolution layer

A single Depthwise seperable convolution unit (denoted by **Conv dw** in the table above) looks like this :

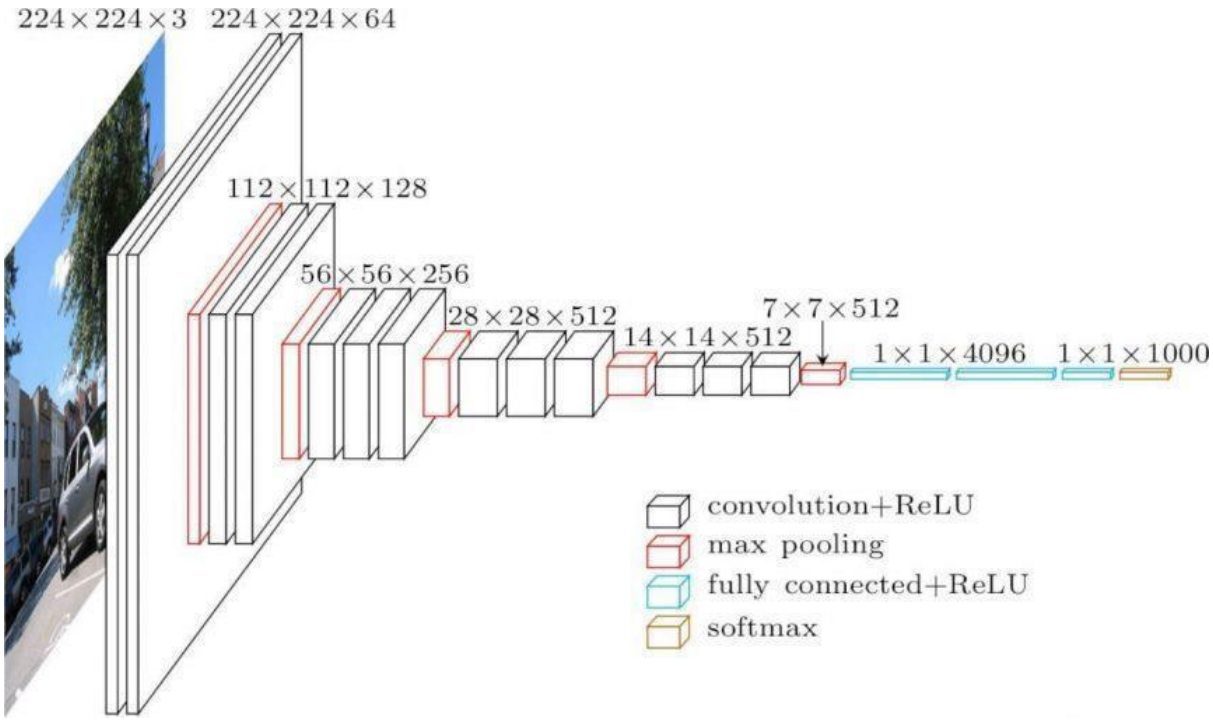


VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the
University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU’s.



The architecture depicted below is VGG16.

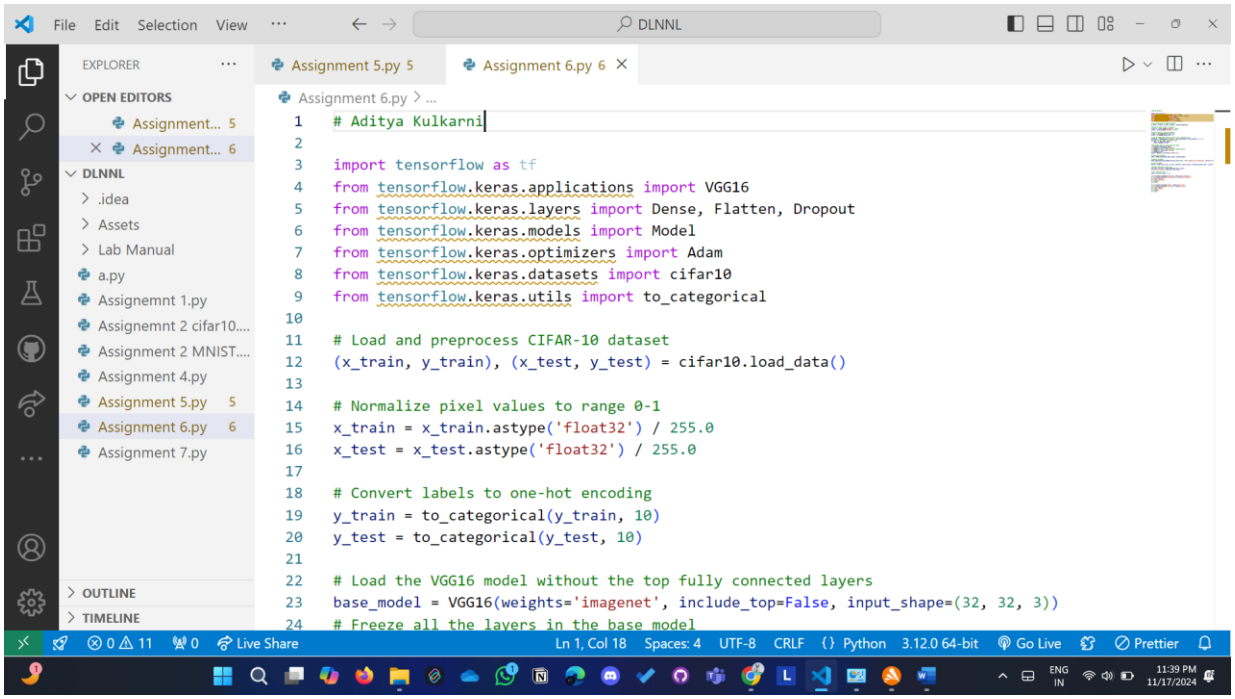


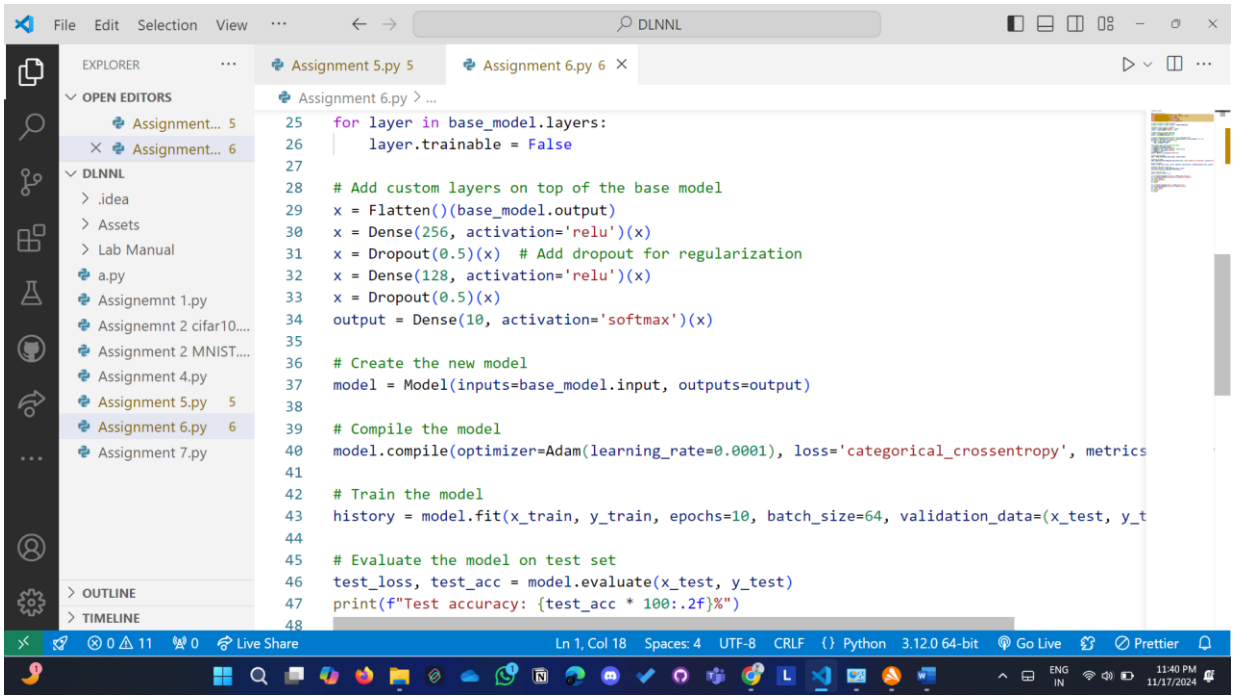
VGG16 Architecture

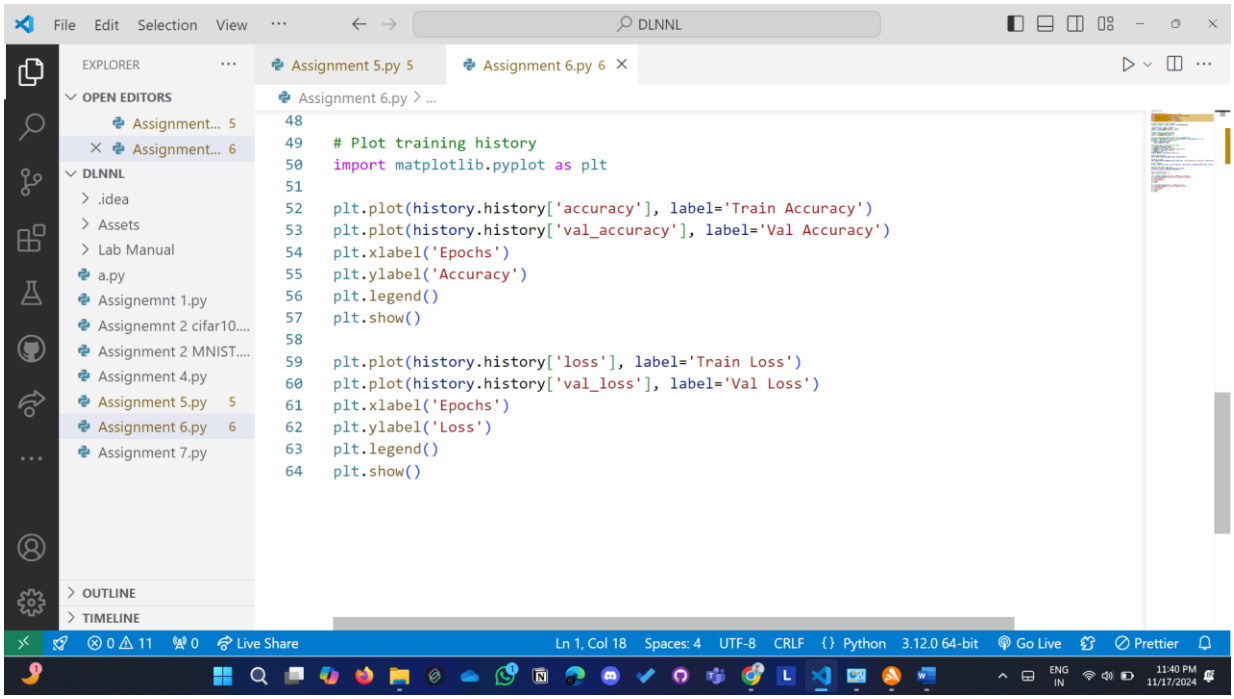
The input to cov1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the softmax layer. The configuration of the fully connected layers is the same in all networks.

Code:


File Edit Selection View
DLNNL
EXPLORER
OPEN EDITORS
Assignment... 5
Assignment... 6
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 MNIST....
Assignment 4.py
Assignment 5.py 5
Assignment 6.py 6
Assignment 7.py
OUTLINE
TIMELINE
Assignment 5.py 5
Assignment 6.py 6
Assignment 6.py > ...
1 # Aditya Kulkarni
2
3 import tensorflow as tf
4 from tensorflow.keras.applications import VGG16
5 from tensorflow.keras.layers import Dense, Flatten, Dropout
6 from tensorflow.keras.models import Model
7 from tensorflow.keras.optimizers import Adam
8 from tensorflow.keras.datasets import cifar10
9 from tensorflow.keras.utils import to_categorical
10
11 # Load and preprocess CIFAR-10 dataset
12 (x_train, y_train), (x_test, y_test) = cifar10.load_data()
13
14 # Normalize pixel values to range 0-1
15 x_train = x_train.astype('float32') / 255.0
16 x_test = x_test.astype('float32') / 255.0
17
18 # Convert labels to one-hot encoding
19 y_train = to_categorical(y_train, 10)
20 y_test = to_categorical(y_test, 10)
21
22 # Load the VGG16 model without the top fully connected layers
23 base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
24 # Freeze all the layers in the base model
Live Share
Ln 1, Col 18 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Go Live Prettier
11:39 PM 11/17/2024


File Edit Selection View
DLNNL
EXPLORER
OPEN EDITORS
Assignment... 5
Assignment... 6
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 MNIST....
Assignment 4.py
Assignment 5.py 5
Assignment 6.py 6
Assignment 7.py
OUTLINE
TIMELINE
Assignment 5.py 5
Assignment 6.py 6
Assignment 6.py > ...
25 for layer in base_model.layers:
26 layer.trainable = False
27
28 # Add custom layers on top of the base model
29 x = Flatten()(base_model.output)
30 x = Dense(256, activation='relu')(x)
31 x = Dropout(0.5)(x) # Add dropout for regularization
32 x = Dense(128, activation='relu')(x)
33 x = Dropout(0.5)(x)
34 output = Dense(10, activation='softmax')(x)
35
36 # Create the new model
37 model = Model(inputs=base_model.input, outputs=output)
38
39 # Compile the model
40 model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics
41
42 # Train the model
43 history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_data=(x_test, y_t
44
45 # Evaluate the model on test set
46 test_loss, test_acc = model.evaluate(x_test, y_test)
47 print(f"Test accuracy: {test_acc * 100:.2f}%")
48
Live Share
Ln 1, Col 18 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Go Live Prettier
11:40 PM 11/17/2024



Results:

```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 153s 193ms/step - accuracy: 0.1656 - loss: 2.2899 - val_accuracy: 0.4158 - va
l_loss: 1.7431
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 137s 176ms/step - accuracy: 0.3503 - loss: 1.8248 - val_accuracy: 0.4777 - va
l_loss: 1.5285
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 142s 182ms/step - accuracy: 0.4184 - loss: 1.6451 - val_accuracy: 0.5083 - va
l_loss: 1.4364
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 147s 188ms/step - accuracy: 0.4540 - loss: 1.5644 - val_accuracy: 0.5246 - va
l_loss: 1.3763
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 149s 191ms/step - accuracy: 0.4794 - loss: 1.4928 - val_accuracy: 0.5363 - va
l_loss: 1.3411
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 127s 163ms/step - accuracy: 0.4914 - loss: 1.4495 - val_accuracy: 0.5441 - va
l_loss: 1.3080
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 123s 158ms/step - accuracy: 0.5060 - loss: 1.4209 - val_accuracy: 0.5498 - va
l_loss: 1.2847
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 121s 155ms/step - accuracy: 0.5207 - loss: 1.3800 - val_accuracy: 0.5587 - va
l_loss: 1.2660
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 122s 155ms/step - accuracy: 0.5289 - loss: 1.3700 - val_accuracy: 0.5612 - va
l_loss: 1.2483
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 122s 156ms/step - accuracy: 0.5342 - loss: 1.3474 - val_accuracy: 0.5686 - va
l_loss: 1.2317
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 80ms/step - accuracy: 0.5642 - loss: 1.2305
Test accuracy: 56.86%
```

Conclusion:

Thus, we have understood the importance of Transfer learning and also the usage of transfer learning in tensorflow.

Name : Aditya Kulkarni

Class : LY – AIA – 1

Batch : A

Roll No : 2213133

Assignment No. 7

Aim: Develop RNN model for Cryptocurrency pricing prediction or text sentiment analysis

Objectives:

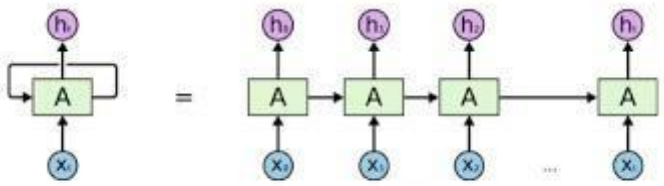
1. To learn RNN 2. To learn and implement LSTM

Theory:

RNN

A **recurrent neural network** (**RNN**) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition[4] or speech recognition.

The term “recurrent neural network” is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that cannot be unrolled.



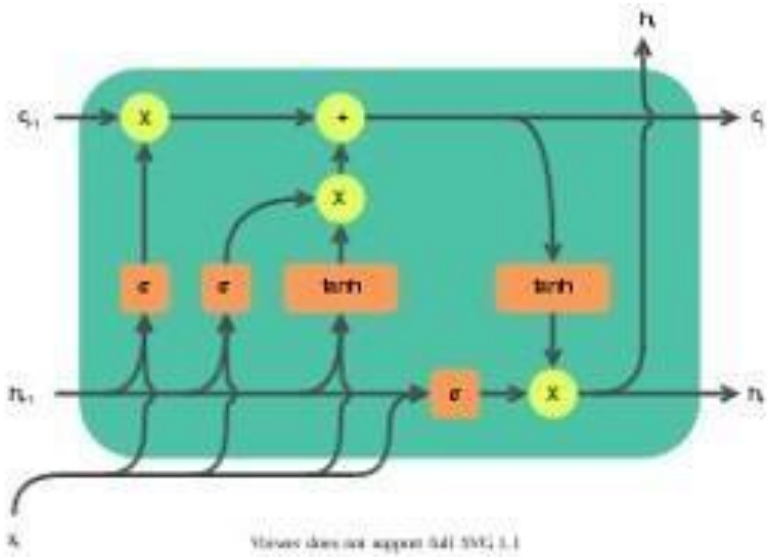
An unrolled recurrent neural network.

LSTM

Long short-term memory (**LSTM**) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or IDSs (intrusion detection systems).

A common LSTM unit is composed of a **cell**, an **input gate**, an **output gate** and a **forget gate**. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series.
LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.



BPTT Backpropagation refers to

two things:

- The mathematical method used to calculate derivatives and an application of the derivative chain rule.
- The training algorithm for updating network weights to minimize error. It is this latter understanding of backpropagation that we are using here.

The goal of the backpropagation training algorithm is to modify the weights of a neural network in order to minimize the error of the network outputs compared to some expected output in response to corresponding inputs.

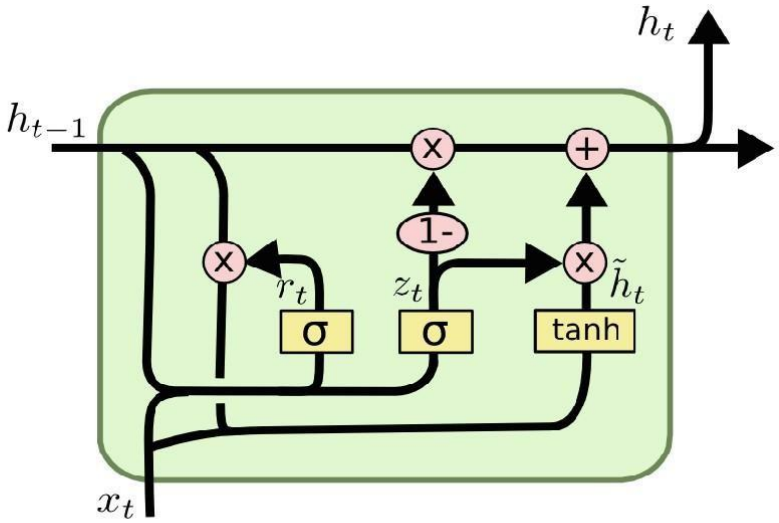
It is a supervised learning algorithm that allows the network to be corrected with regard to the specific errors made.

The general algorithm is as follows:

1. Present a training input pattern and propagate it through the network to get an output.
2. Compare the predicted outputs to the expected outputs and calculate the error.
3. Calculate the derivatives of the error with respect to the network weights.
4. Adjust the weights to minimize the error.
5. Repeat.

GRU

Gated recurrent units (**GRU**s) are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho et al. The GRU is like a long short-term memory (LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate. GRU's performance on certain tasks of polyphonic music modeling, speech signal modeling and natural language processing was found to be similar to that of LSTM. GRUs have been shown to exhibit better performance on certain smaller and less frequent datasets.



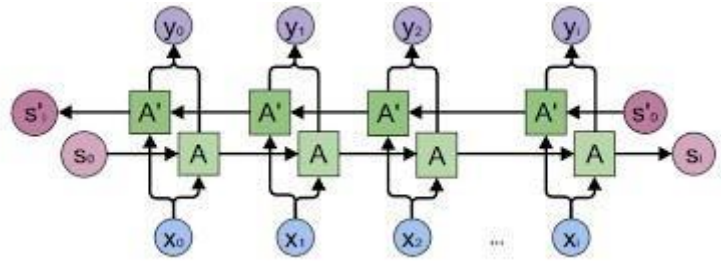
Bi directional RNN

Bidirectional recurrent neural networks(RNN) are really just putting two independent RNNs together. The input sequence is fed in normal time order for one network, and in reverse time order for another. The outputs of the two networks are usually concatenated at each time step, though there are other options, e.g. summation.

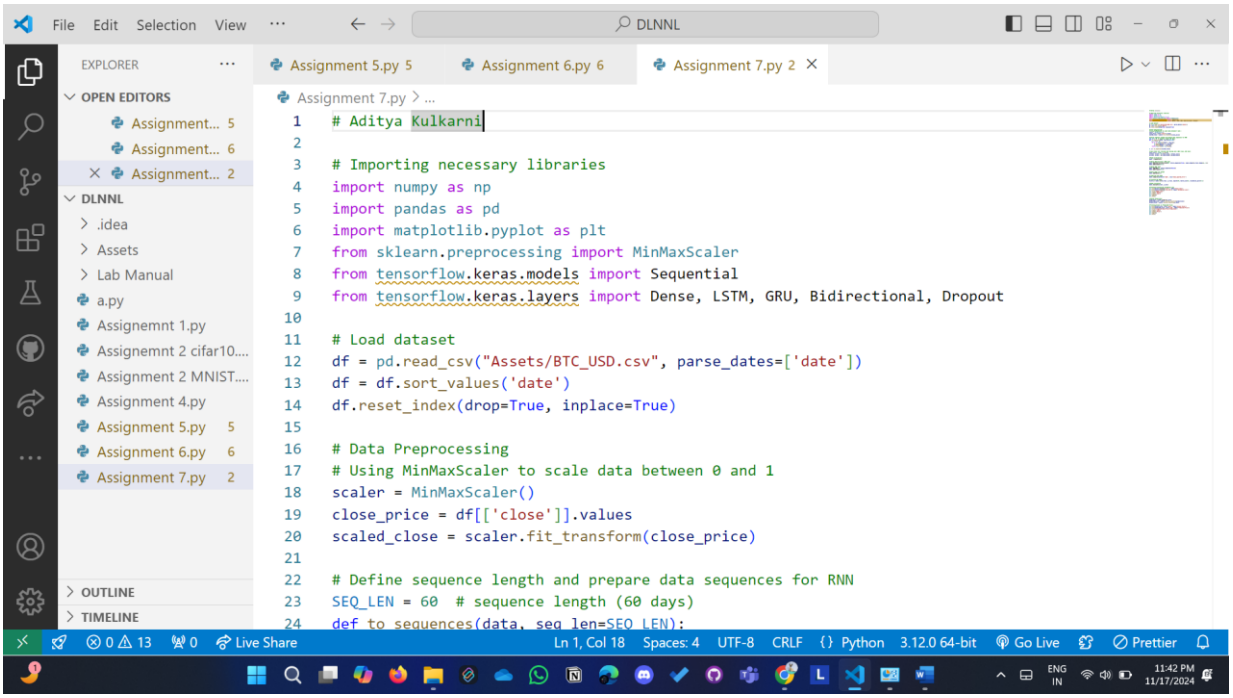
This structure allows the networks to have both backward and forward information about the sequence at every time step. The concept seems easy enough. But when it comes to actually implementing a neural network which utilizes bidirectional structure, confusion arises…

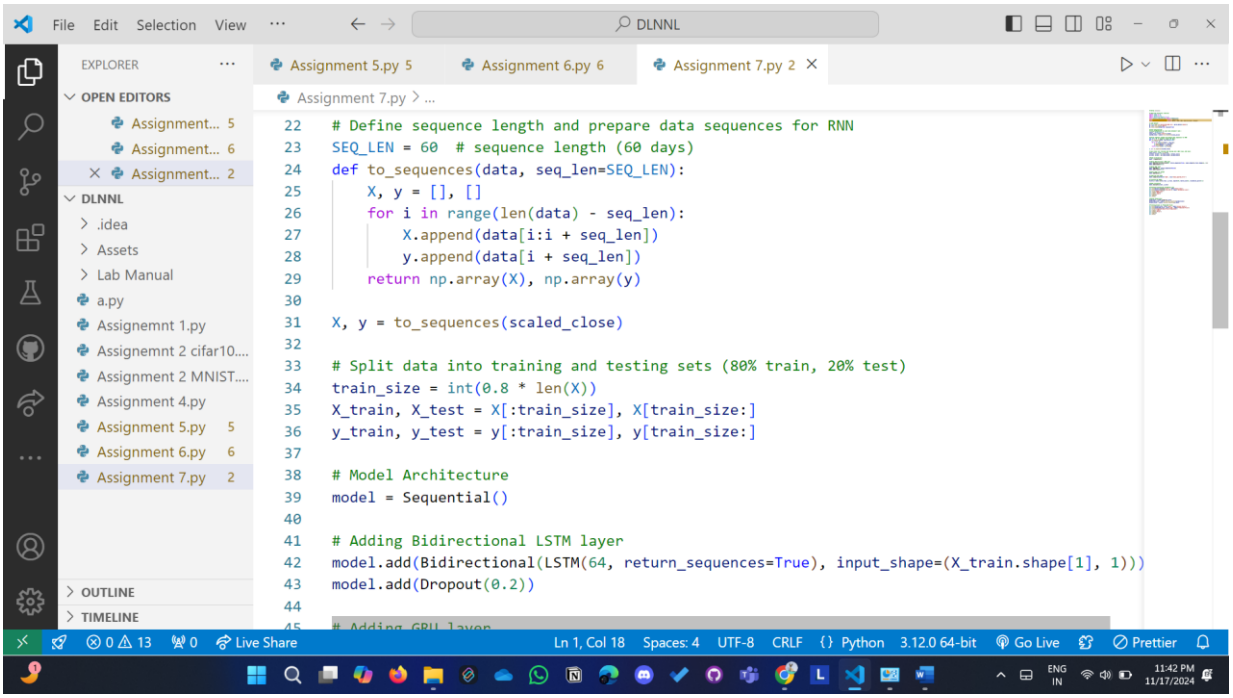
The first confusion is about **the way to forward the outputs of a bidirectional RNN to a dense neural network**.

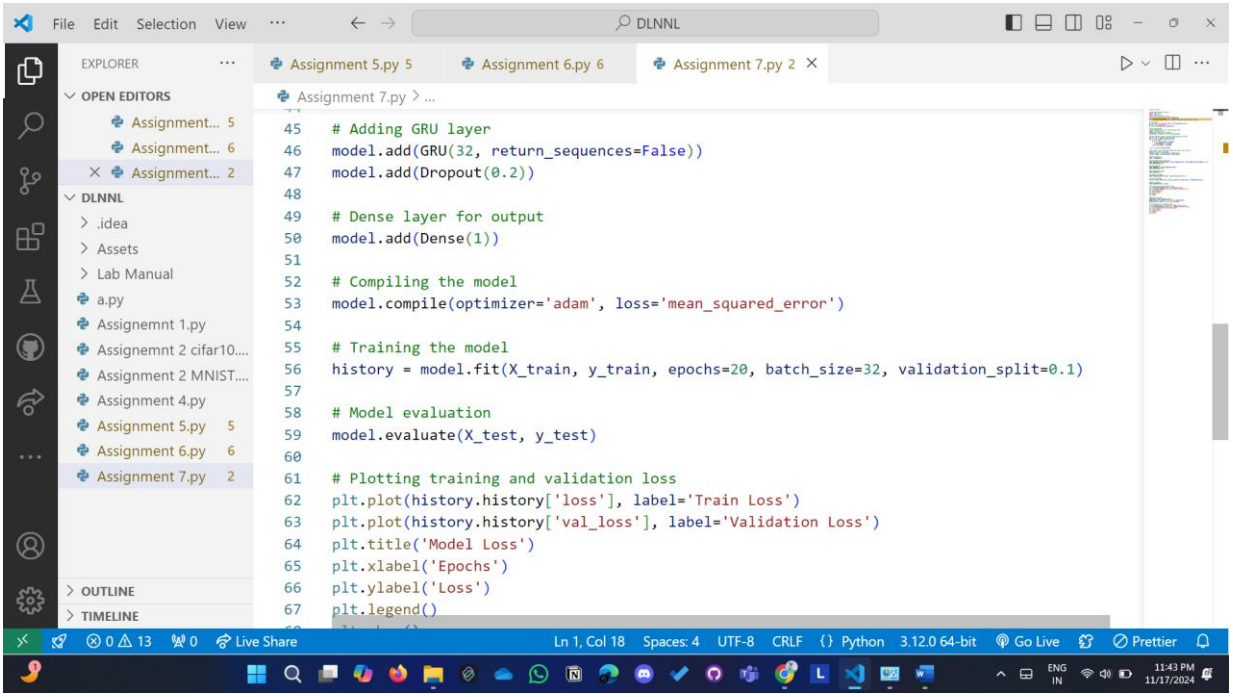
The second confusion is about the **returned hidden states**. In seq2seq models, we'll want hidden states from the encoder to initialize the hidden states of the decoder. Intuitively, if we can only choose hidden states at one time step(as in PyTorch), we'd want the one at which the RNN just consumed the last input in the sequence. But **if** the hidden states of time step *n* (the last one) are returned, as before, we'll have the hidden states of the reversed RNN with only one step of inputs seen.

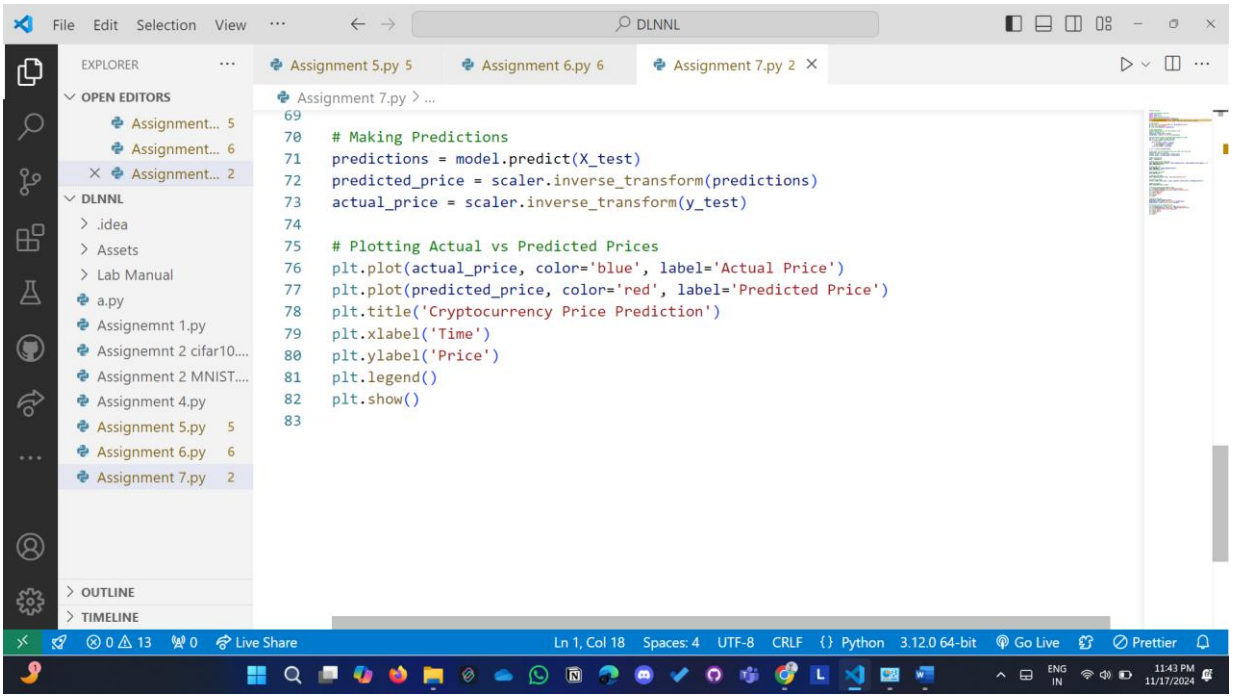


Code:




File Edit Selection View
DLNNL
EXPLORER
Assignment 5.py 5
Assignment 6.py 6
Assignment 7.py 2
OPEN EDITORS
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 MNIST....
Assignment 4.py
OUTLINE
TIMELINE
# Define sequence length and prepare data sequences for RNN
SEQ_LEN = 60 # sequence length (60 days)
def to_sequences(data, seq_len=SEQ_LEN):
X, y = [], []
for i in range(len(data) - seq_len):
X.append(data[i:i + seq_len])
y.append(data[i + seq_len])
return np.array(X), np.array(y)
X, y = to_sequences(scaled_close)
# Split data into training and testing sets (80% train, 20% test)
train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
# Model Architecture
model = Sequential()
# Adding Bidirectional LSTM layer
model.add(Bidirectional(LSTM(64, return_sequences=True), input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
Live Share
Ln 1, Col 18
Spaces: 4
UTF-8
CRLF
Python
3.12.0 64-bit
Go Live
Prettier
11:42 PM
11/17/2024

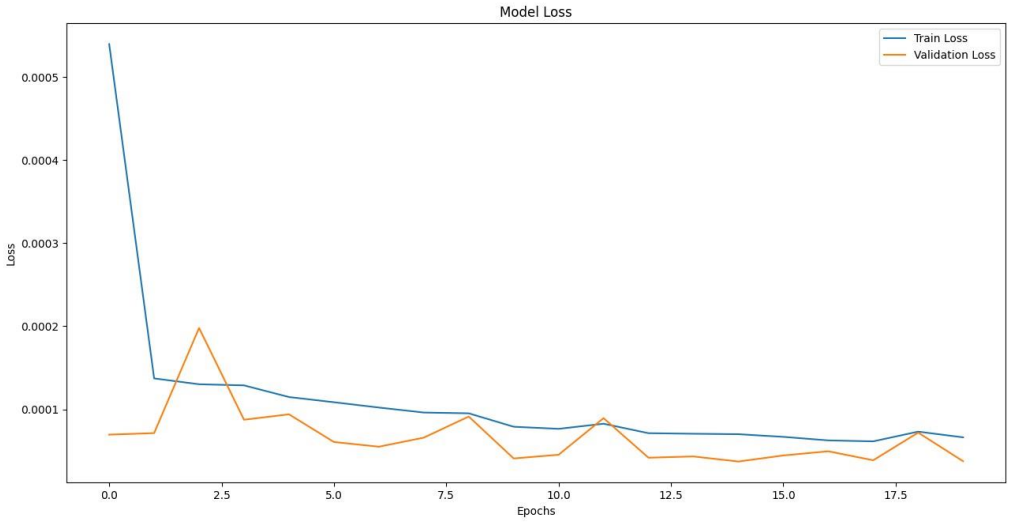

File Edit Selection View
DLNNL
EXPLORER
Assignment 5.py 5
Assignment 6.py 6
Assignment 7.py 2
OPEN EDITORS
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 MNIST....
Assignment 4.py
OUTLINE
TIMELINE
# Adding GRU layer
model.add(GRU(32, return_sequences=False))
model.add(Dropout(0.2))
# Dense layer for output
model.add(Dense(1))
# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error')
# Training the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.1)
# Model evaluation
model.evaluate(X_test, y_test)
# Plotting training and validation loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
Live Share
Ln 1, Col 18
Spaces: 4
UTF-8
CRLF
Python
3.12.0 64-bit
Go Live
Prettier
11:43 PM
11/17/2024

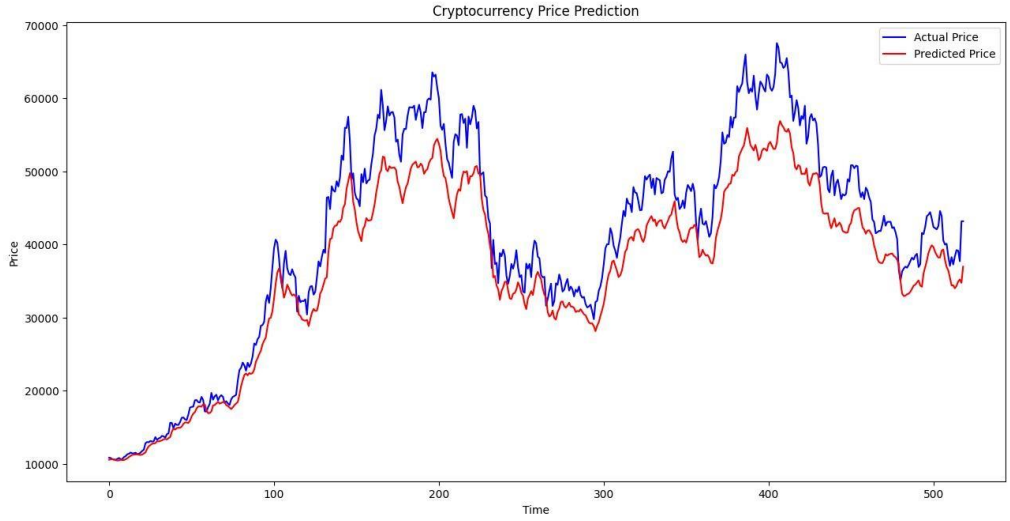


Results:

```
Epoch 1/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 7s 42ms/step - loss: 0.0013 - val_loss: 6.9518e-05
Epoch 2/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - loss: 1.5053e-04 - val_loss: 7.1398e-05
Epoch 3/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 3s 42ms/step - loss: 1.2944e-04 - val_loss: 1.9792e-04
Epoch 4/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - loss: 1.4095e-04 - val_loss: 8.7458e-05
Epoch 5/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - loss: 1.0291e-04 - val_loss: 9.4002e-05
```

```
Epoch 16/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - loss: 6.7753e-05 - val_loss: 4.4484e-05
Epoch 17/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - loss: 6.9390e-05 - val_loss: 4.9524e-05
Epoch 18/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - loss: 5.9826e-05 - val_loss: 3.8700e-05
Epoch 19/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 7.6771e-05 - val_loss: 7.2000e-05
Epoch 20/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 2s 33ms/step - loss: 6.3870e-05 - val_loss: 3.7543e-05
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0043
```





Conclusion:

Thus, we have learned how to code LSTMs and understood How Recurrent neural networks work

Name : Aditya Kulkarni

Class : LY-AIA-1

Roll No : 2213133

Assignment No. 8

Aim: Develop an autoencoder to encode and decode the image. Analyze the results.

a) Develop AE for MNIST dataset

b) Use output of AE as input to CNN

Objectives:

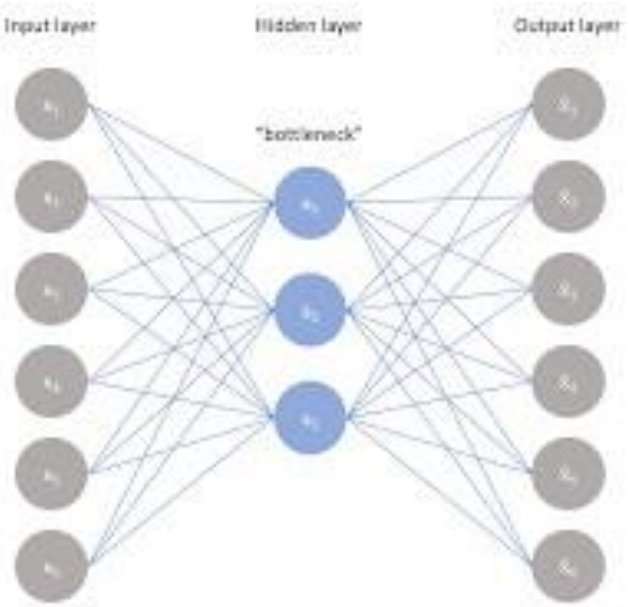
1. To learn AE 2. To

implement AE

Theory:

Autoencoders

An **autoencoder** is a type of artificial neural network used to learn efficient data coding in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal “noise”. Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input, hence its name. Several variants exist to the basic model, with the aim of forcing the learned representations of the input to assume useful properties. Examples are the regularized autoencoders (*Sparse*, *Denoising* and *Contractive* autoencoders), proven effective in learning representations for subsequent classification tasks, and *Variational* autoencoders, with their recent applications as generative models. Autoencoders are effectively used for solving many applied problems, from face recognition to acquiring the semantic meaning of words.



There are, basically, 7 types of autoencoders:

- Denoising autoencoder
- Sparse Autoencoder
- Deep Autoencoder
- Contractive Autoencoder
- Undercomplete Autoencoder
- Convolutional Autoencoder
- Variational Autoencoder

1) Denoising Autoencoder

Denoising autoencoders create a corrupted copy of the input by introducing some noise. This helps to avoid the autoencoders to copy the input to the output without learning features about the data. These autoencoders take a partially corrupted input while training to recover the original undistorted input. The model learns a vector field for mapping the input data towards a lower dimensional manifold which describes the natural data to cancel out the added noise.

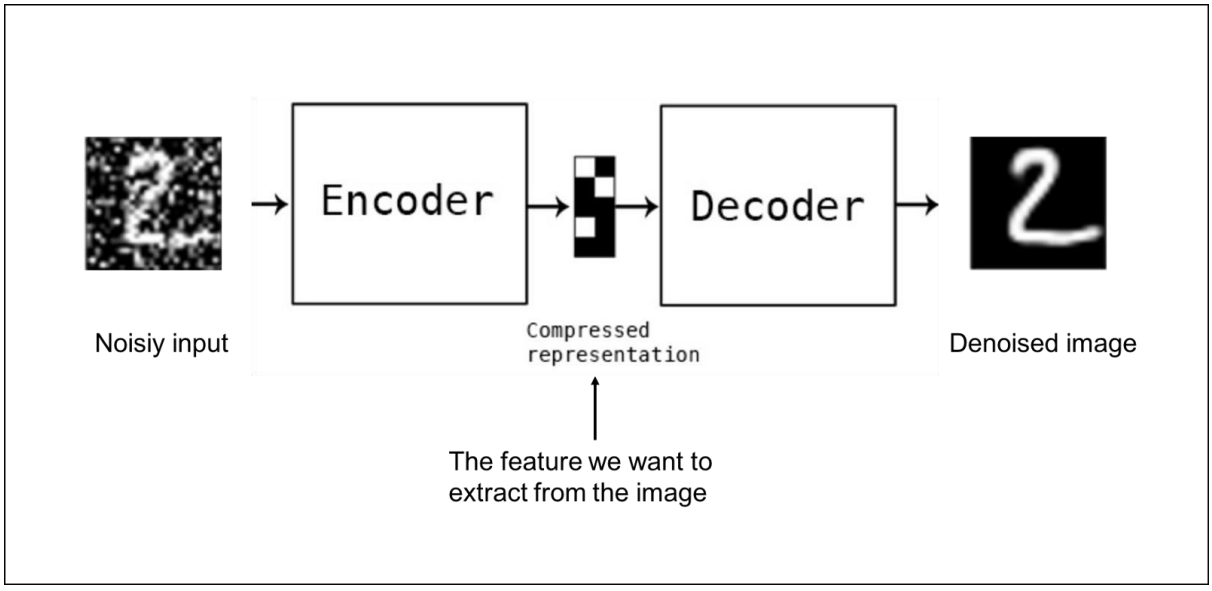
Advantages-

- It was introduced to achieve good representation. Such a representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input.
- Corruption of the input can be done randomly by making some of the input as zero. Remaining nodes copy the input to the noised input.

- Minimizes the loss function between the output node and the corrupted input.
- Setting up a single-thread denoising autoencoder is easy.

Drawbacks-

- To train an autoencoder to denoise data, it is necessary to perform preliminary stochastic mapping in order to corrupt the data and use as input.
- This model isn't able to develop a mapping which memorizes the training data because our input and target output are no longer the same.



2) Sparse Autoencoder

Sparse autoencoders have hidden nodes greater than input nodes. They can still discover important features from the data. A generic sparse autoencoder is visualized where the obscurity of a node corresponds with the level of activation. Sparsity constraint is introduced on the hidden layer. This is to prevent output layer copy input data. Sparsity may be obtained by additional terms in the loss function during the training process, either by comparing the probability distribution of the hidden unit activations with some low desired value,or by manually zeroing all but the strongest hidden unit activations. Some of the most powerful AIs in the 2010s involved sparse autoencoders stacked inside of deep neural networks.

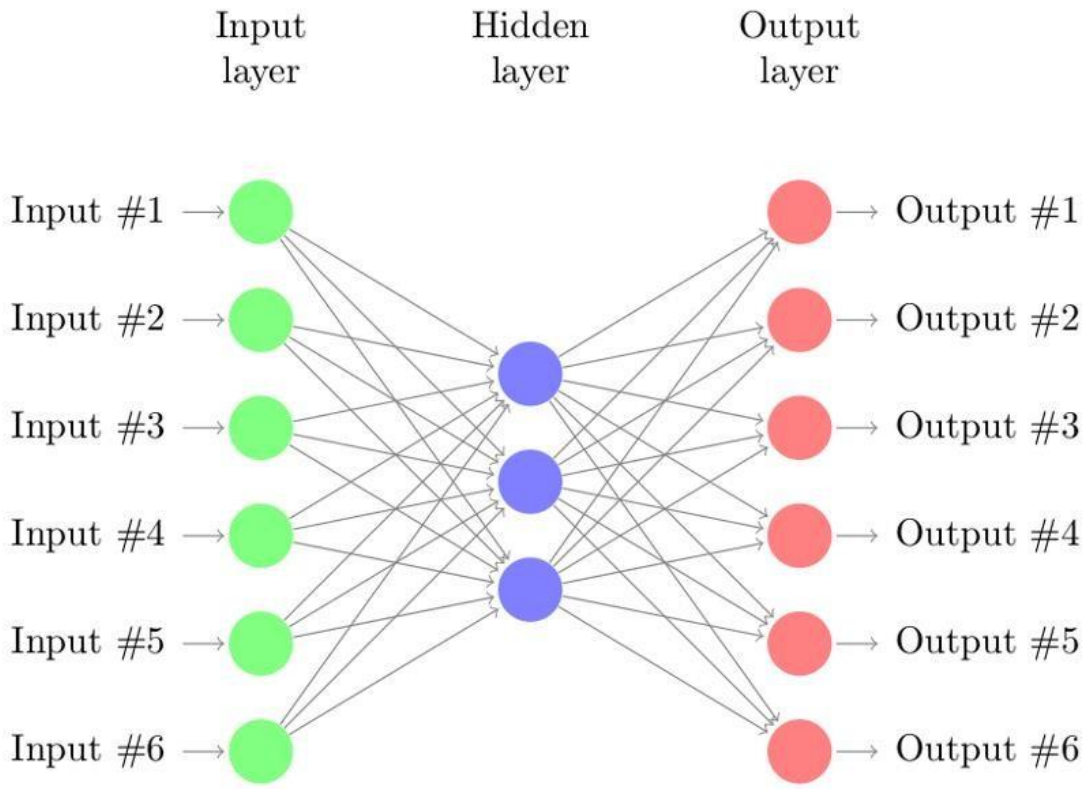
Advantages-

- Sparse autoencoders have a sparsity penalty, a value close to zero but not exactly zero. Sparsity penalty is applied on the hidden layer in addition to the reconstruction error. This prevents overfitting.

- They take the highest activation values in the hidden layer and zero out the rest of the hidden nodes. This prevents autoencoders to use all of the hidden nodes at a time and forcing only a reduced number of hidden nodes to be used.

Drawbacks-

- For it to be working, it's essential that the individual nodes of a trained model which activate are data dependent, and that different inputs will result in activations of different nodes through the network.



3) Deep Autoencoder

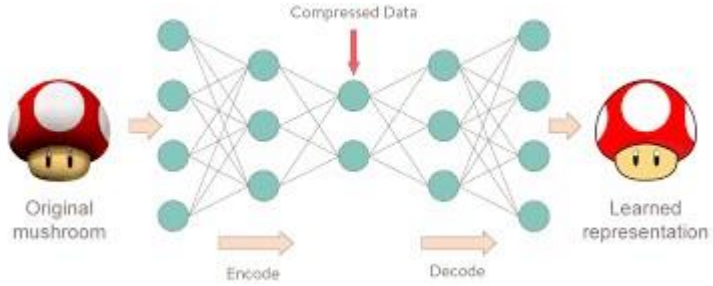
Deep Autoencoders consist of two identical deep belief networks, oOne network for encoding and another for decoding. Typically deep autoencoders have 4 to 5 layers for encoding and the next 4 to 5 layers for decoding. We use unsupervised layer by layer pre-training for this model. The layers are Restricted Boltzmann Machines which are the building blocks of deep-belief networks. Processing the benchmark dataset MNIST, a deep autoencoder would use binary transformations after each RBM. Deep autoencoders are useful in topic modeling, or statistically modeling abstract topics that are distributed across a collection of documents. They are also capable of compressing images into 30 number vectors.

Advantages-

- Deep autoencoders can be used for other types of datasets with real-valued data, on which you would use Gaussian rectified transformations for the RBMs instead.
- Final encoding layer is compact and fast.

Drawbacks-

- Chances of overfitting to occur since there's more parameters than input data.
- Training the data maybe a nuance since at the stage of the decoder's backpropagation, the learning rate should be lowered or made slower depending on whether binary or continuous data is being handled.

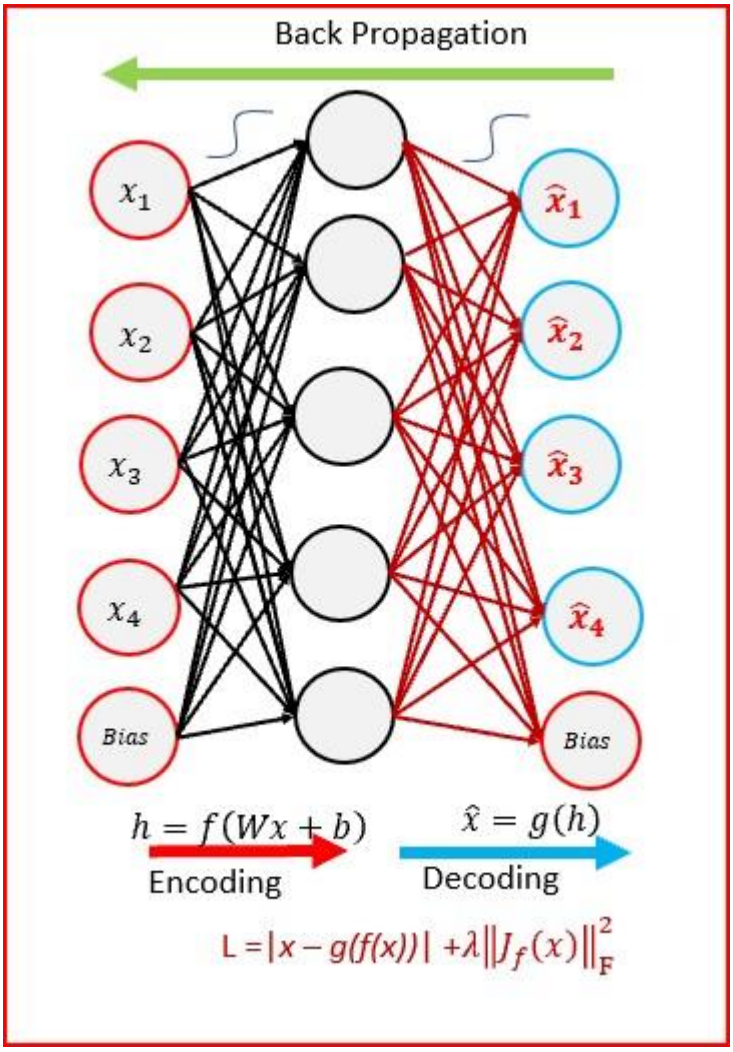


4) Contractive Autoencoder

The objective of a contractive autoencoder is to have a robust learned representation which is less sensitive to small variation in the data. Robustness of the representation for the data is done by applying a penalty term to the loss function. Contractive autoencoder is another regularization technique just like sparse and denoising autoencoders. However, this regularizer corresponds to the Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input. Frobenius norm of the Jacobian matrix for the hidden layer is calculated with respect to input and it is basically the sum of square of all elements.

Advantages-

- Contractive autoencoder is a better choice than denoising autoencoder to learn useful feature extraction.
- This model learns an encoding in which similar inputs have similar encodings. Hence, we're forcing the model to learn how to contract a neighborhood of inputs into a smaller neighborhood of outputs.



5) Undercomplete Autoencoder

The objective of undercomplete autoencoder is to capture the most important features present in the data. Undercomplete autoencoders have a smaller dimension for hidden layer compared to the input layer. This helps to obtain important features from the data. It minimizes the loss function by penalizing the g(f(x)) for being different from the input x.

Advantages-

- Undercomplete autoencoders do not need any regularization as they maximize the probability of data rather than copying the input to the output.

Drawbacks-

- Using an overparameterized model due to lack of sufficient training data can create overfitting.

6) Convolutional Autoencoder

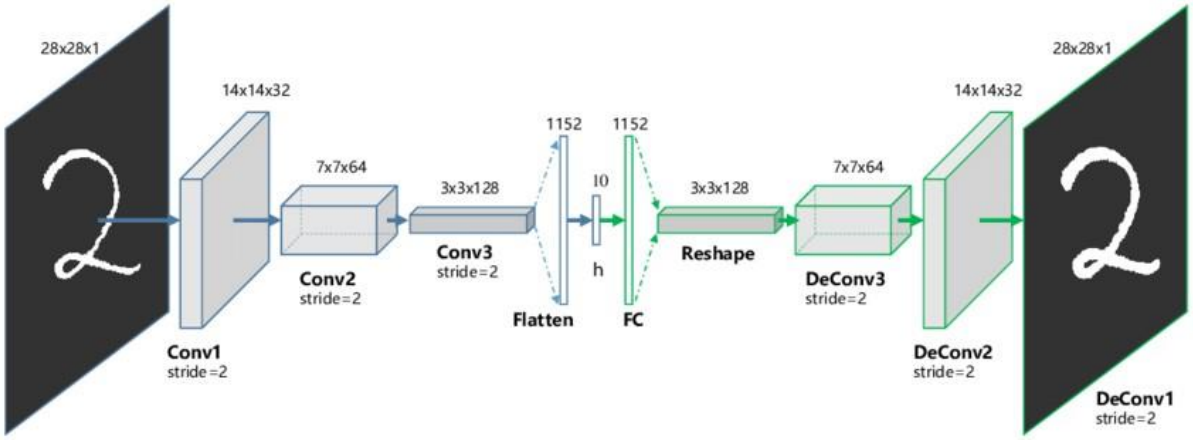
Autoencoders in their traditional formulation does not take into account the fact that a signal can be seen as a sum of other signals. Convolutional Autoencoders use the convolution operator to exploit this observation. They learn to encode the input in a set of simple signals and then try to reconstruct the input from them, modify the geometry or the reflectance of the image. They are the state-of-art tools for unsupervised learning of convolutional filters. Once these filters have been learned, they can be applied to any input in order to extract features. These features, then, can be used to do any task that requires a compact representation of the input, like classification.

Advantages-

- Due to their convolutional nature, they scale well to realistic-sized high dimensional images.
- Can remove noise from picture or reconstruct missing parts.

Drawbacks-

- The reconstruction of the input image is often blurry and of lower quality due to compression during which information is lost.



7) Variational Autoencoder

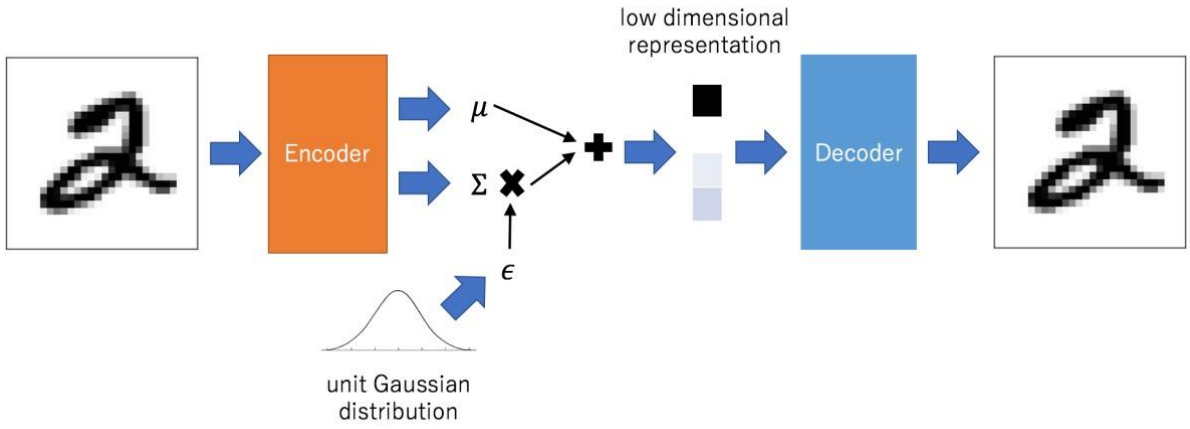
Variational autoencoder models make strong assumptions concerning the distribution of latent variables. They use a variational approach for latent representation learning, which results in an additional loss component and a specific estimator for the training algorithm called the Stochastic Gradient Variational Bayes estimator. It assumes that the data is generated by a directed graphical model and that the encoder is learning an approximation to the posterior distribution where Φ and θ denote the parameters of the encoder (recognition model) and decoder (generative model) respectively. The probability distribution of the latent vector of a variational autoencoder typically matches that of the training data much closer than a standard autoencoder.

Advantages-

- It gives significant control over how we want to model our latent distribution unlike the other models.
- After training you can just sample from the distribution followed by decoding and generating new data.

Drawbacks-

- When training the model, there is a need to calculate the relationship of each parameter in the network with respect to the final output loss using a technique known as backpropagation. Hence, the sampling process requires some extra attention.



Applications

Autoencoders work by compressing the input into a latent space representation and then reconstructing the output from this representation. This kind of network is composed of two parts:

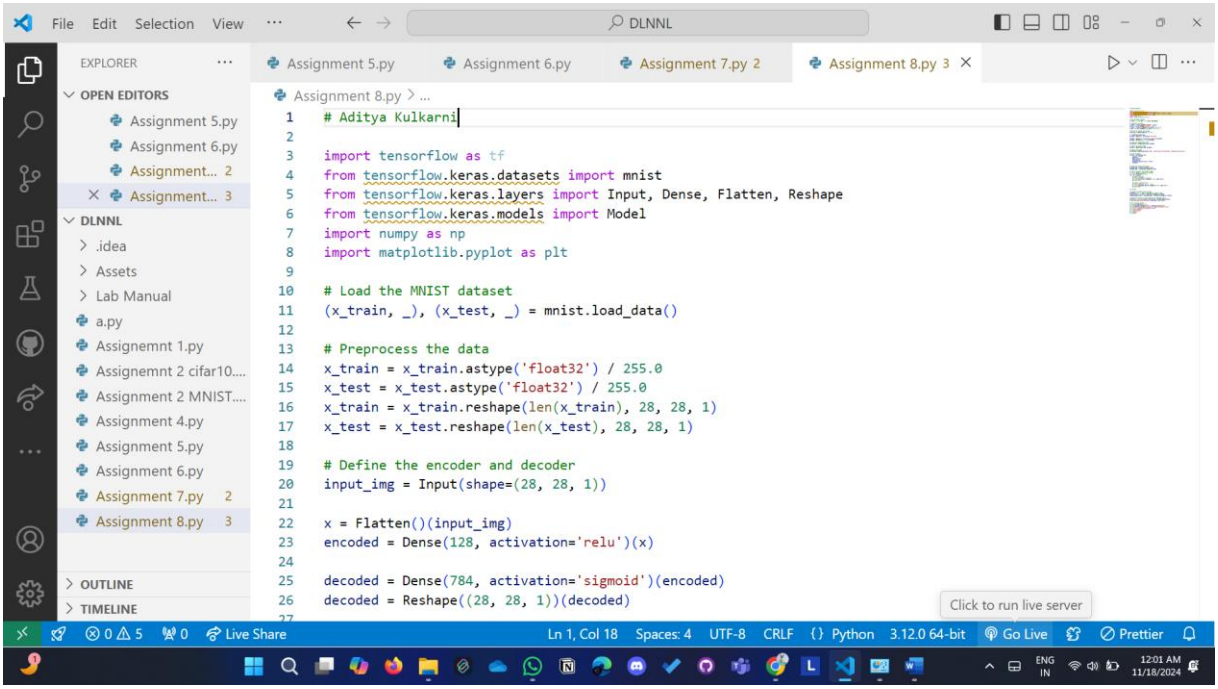
1. Encoder: This is the part of the network that compresses the input into a latent-space representation. It can be represented by an encoding function h=f(x).
2. Decoder: This part aims to reconstruct the input from the latent space representation. It can be represented by a decoding function r=g(h).

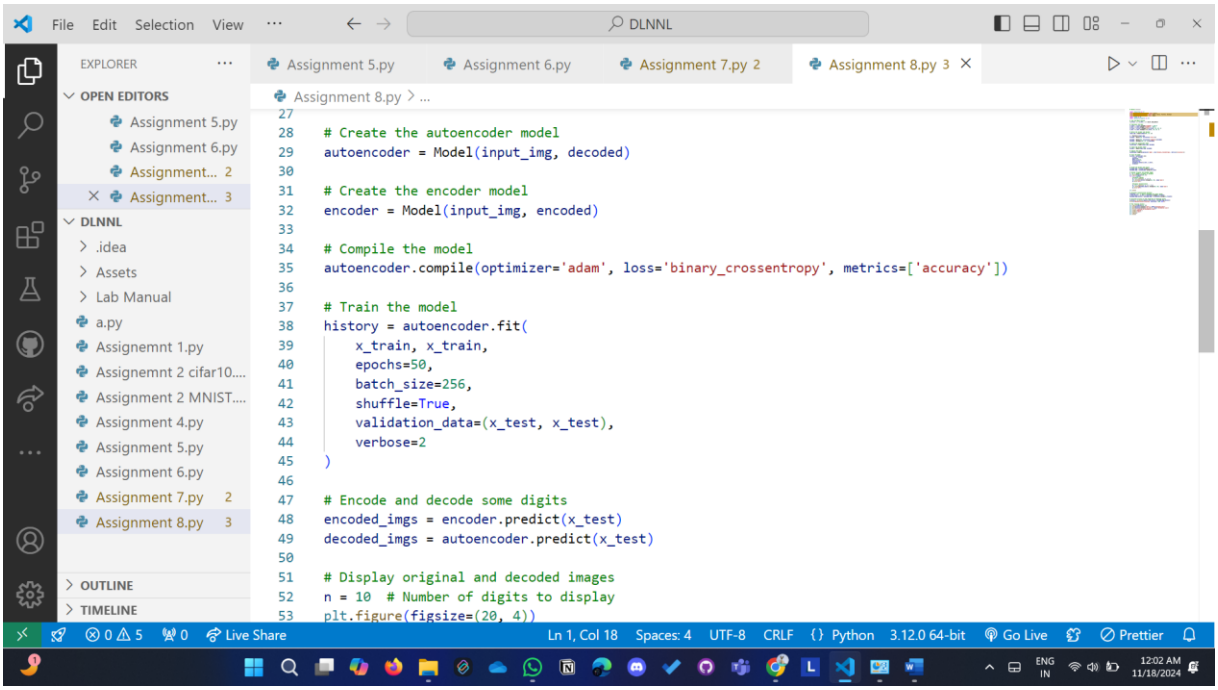
If the only purpose of autoencoders was to copy the input to the output, they would be useless. We hope that by training the autoencoder to copy the input to the output, the latent representation will take on useful properties. This can be achieved by creating constraints on the copying task. If the autoencoder is given too much capacity, it can learn to perform the copying task without extracting any useful information about the distribution of the data. This can also occur if the dimension of the latent representation is the same as the input, and in the overcomplete case, where the dimension of the latent representation is greater than the input. In these cases, even a linear encoder and linear decoder can learn to copy the input to the output without learning anything useful about the data

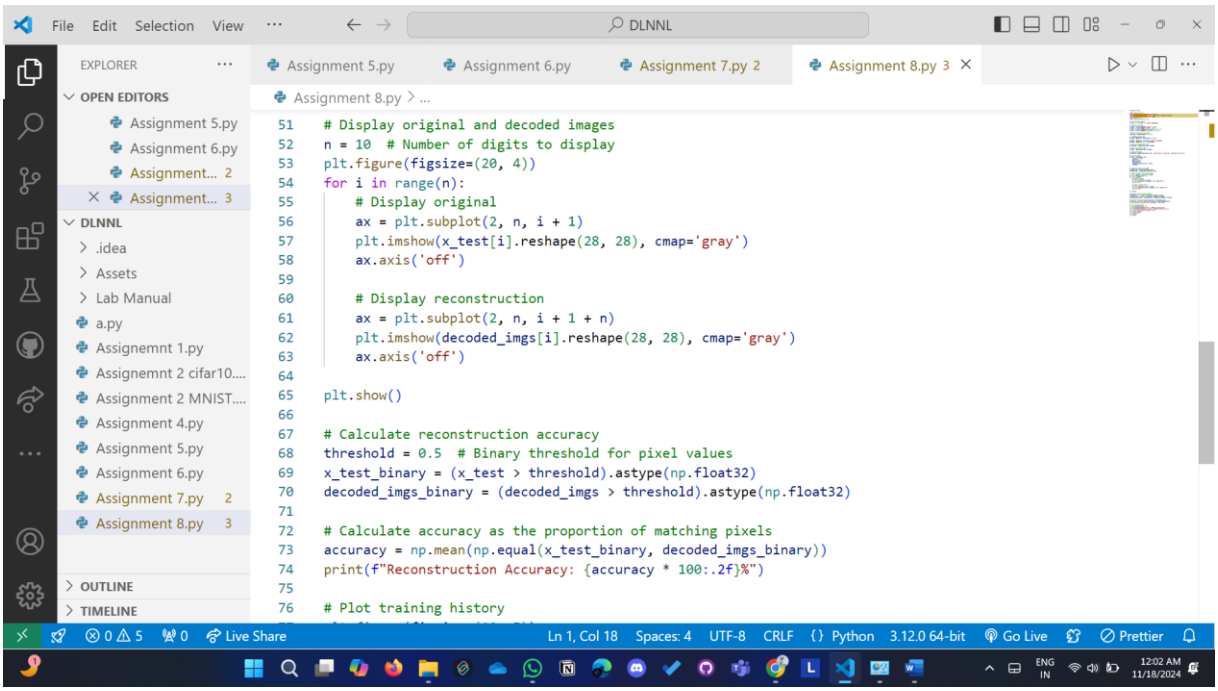
distribution. Ideally, one could train any architecture of autoencoder successfully, choosing the code dimension and the capacity of the encoder and decoder based on the complexity of distribution to be modeled.

Autoencoders are learned automatically from data examples. It means that it is easy to train specialized instances of the algorithm that will perform well on a specific type of input and that it does not require any new engineering, only the appropriate training data. However, autoencoders will do a poor job for image compression. As the autoencoder is trained on a given set of data, it will achieve reasonable compression results on data similar to the training set used but will be poor general-purpose image compressors. Autoencoders are trained to preserve as much information as possible when an input is run through the encoder and then the decoder, but are also trained to make the new representation have various nice properties.

Code:



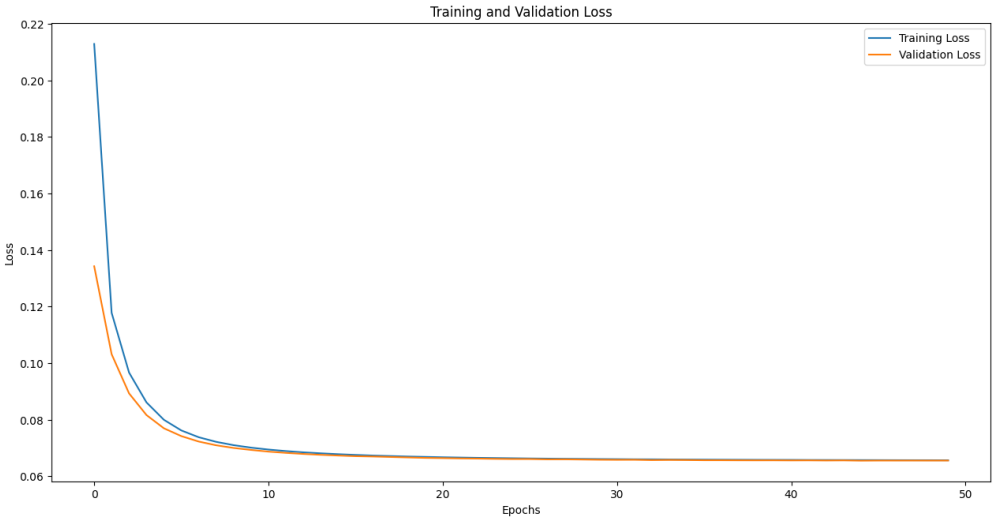

File Edit Selection View
DLNNL
EXPLORER
Assignment 5.py
Assignment 6.py
Assignment 7.py 2
Assignment 8.py 3
OPEN EDITORS
Assignment 5.py
Assignment 6.py
Assignment... 2
Assignment... 3
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 MNIST....
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py 2
Assignment 8.py 3
OUTLINE
TIMELINE
Assignment 8.py > ...
# Create the autoencoder model
autoencoder = Model(input_img, decoded)
# Create the encoder model
encoder = Model(input_img, encoded)
# Compile the model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = autoencoder.fit(
x_train, x_train,
epochs=50,
batch_size=256,
shuffle=True,
validation_data=(x_test, x_test),
verbose=2
)
# Encode and decode some digits
encoded_imgs = encoder.predict(x_test)
decoded_imgs = autoencoder.predict(x_test)
# Display original and decoded images
n = 10 # Number of digits to display
plt.figure(figsize=(20, 4))
Live Share
Ln 1, Col 18
Spaces: 4
UTF-8
CRLF
Python
3.12.0 64-bit
Go Live
Prettier
ENG
IN
12:02 AM
11/18/2024


File Edit Selection View
DLNNL
EXPLORER
Assignment 5.py
Assignment 6.py
Assignment 7.py 2
Assignment 8.py 3
OPEN EDITORS
Assignment 5.py
Assignment 6.py
Assignment... 2
Assignment... 3
DLNNL
.idea
Assets
Lab Manual
a.py
Assignemnt 1.py
Assignemnt 2 cifar10....
Assignment 2 MNIST....
Assignment 4.py
Assignment 5.py
Assignment 6.py
Assignment 7.py 2
Assignment 8.py 3
OUTLINE
TIMELINE
Assignment 8.py > ...
# Display original and decoded images
n = 10 # Number of digits to display
plt.figure(figsize=(20, 4))
for i in range(n):
# Display original
ax = plt.subplot(2, n, i + 1)
plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
ax.axis('off')
# Display reconstruction
ax = plt.subplot(2, n, i + 1 + n)
plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
ax.axis('off')
plt.show()
# Calculate reconstruction accuracy
threshold = 0.5 # Binary threshold for pixel values
x_test_binary = (x_test > threshold).astype(np.float32)
decoded_imgs_binary = (decoded_imgs > threshold).astype(np.float32)
# Calculate accuracy as the proportion of matching pixels
accuracy = np.mean(np.equal(x_test_binary, decoded_imgs_binary))
print(f"Reconstruction Accuracy: {accuracy * 100:.2f}%")
# Plot training history
Live Share
Ln 1, Col 18
Spaces: 4
UTF-8
CRLF
Python
3.12.0 64-bit
Go Live
Prettier
ENG
IN
12:02 AM
11/18/2024

```
# Calculate reconstruction accuracy
threshold = 0.5  # Binary threshold for pixel values
x_test_binary = (x_test > threshold).astype(np.float32)
decoded_imgs_binary = (decoded_imgs > threshold).astype(np.float32)

# Calculate accuracy as the proportion of matching pixels
accuracy = np.mean(np.equal(x_test_binary, decoded_imgs_binary))
print(f"Reconstruction Accuracy: {accuracy * 100:.2f}%")

# Plot training history
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Results:

```
235/235 - 2s - 7ms/step - accuracy: 0.8155 - loss: 0.0658 - val_accuracy: 0.8145 - val_loss: 0.0656
Epoch 40/50
235/235 - 2s - 7ms/step - accuracy: 0.8155 - loss: 0.0658 - val_accuracy: 0.8145 - val_loss: 0.0656
Epoch 41/50
235/235 - 2s - 7ms/step - accuracy: 0.8155 - loss: 0.0658 - val_accuracy: 0.8145 - val_loss: 0.0656
Epoch 42/50
235/235 - 2s - 10ms/step - accuracy: 0.8155 - loss: 0.0657 - val_accuracy: 0.8145 - val_loss: 0.0656
Epoch 43/50
235/235 - 2s - 7ms/step - accuracy: 0.8155 - loss: 0.0657 - val_accuracy: 0.8145 - val_loss: 0.0655
Epoch 44/50
235/235 - 2s - 8ms/step - accuracy: 0.8155 - loss: 0.0657 - val_accuracy: 0.8145 - val_loss: 0.0656
Epoch 45/50
235/235 - 2s - 9ms/step - accuracy: 0.8155 - loss: 0.0657 - val_accuracy: 0.8145 - val_loss: 0.0655
Epoch 46/50
235/235 - 2s - 10ms/step - accuracy: 0.8155 - loss: 0.0657 - val_accuracy: 0.8145 - val_loss: 0.0655
Epoch 47/50
235/235 - 2s - 8ms/step - accuracy: 0.8155 - loss: 0.0657 - val_accuracy: 0.8145 - val_loss: 0.0655
Epoch 48/50
235/235 - 2s - 8ms/step - accuracy: 0.8155 - loss: 0.0657 - val_accuracy: 0.8145 - val_loss: 0.0655
Epoch 49/50
235/235 - 2s - 8ms/step - accuracy: 0.8155 - loss: 0.0656 - val_accuracy: 0.8145 - val_loss: 0.0655
Epoch 50/50
235/235 - 2s - 7ms/step - accuracy: 0.8155 - loss: 0.0656 - val_accuracy: 0.8145 - val_loss: 0.0655
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 889us/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Reconstruction Accuracy: 99.08%
PS D:\DLNNL>
```



Conclusion:

Thus, we have understood how autoencoders work and how to program them