AI Coursework Projects

Devakinandan Palla

List of Projects on Artificial Intelligence:

Module 1 - Artificial Neural Networks (ANN)

Implementation of an Artificial Neural Network (ANN) model to predict whether a patient will experience a death event based on various clinical features in the dataset

```
import tensorflow as tf
from tensorflow import keras

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# 1. Import and EDA

data = pd.read_csv("/content/Heart_failure_clinical_records_dataset.csv")
data.head()
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets
0	75.0	0	582	0	20	1	265000
1	55.0	0	7861	0	38	0	263358
2	65.0	0	146	0	20	0	162000
3	50.0	1	111	0	20	0	210000
4	65.0	1	160	1	20	0	327000

```python
data.isnull().sum().sum()
```

```
0
```

```python
# 2. Train test split and shape
x = data.drop(["DEATH_EVENT"], axis = 1)
y = data["DEATH_EVENT"]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 4

print(x_train.shape)
print(x_test.shape)


# Standard Scaling
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

```
(224, 12)
(75, 12)
```

```python
# 3. Model

classifier = Sequential()
classifier.add(Dense(12, activation = "relu"))
classifier.add(Dropout(0.3))
classifier.add(Dense(7, activation = "relu"))
classifier.add(Dropout(0.3))
classifier.add(Dense(3, activation = "relu"))
classifier.add(Dropout(0.3))
classifier.add(Dense(1, activation = "sigmoid"))
```

```
# 4. Compile

classifier.compile(optimizer = 'Adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
# 5. Fit

# earlystopping = EarlyStopping(monitor = 'val_loss', min_delta = 0.1, patience = 5, verbose

model_history = classifier.fit(x_train, y_train, epochs=5, validation_split=0.2, verbose=1)
```

```
Epoch 1/5
6/6            7s 593ms/step - accuracy: 0.4989 - loss: 0.9297 - val_accuracy: 0.3778 - val_
Epoch 2/5
6/6            4s 7ms/step - accuracy: 0.4571 - loss: 0.9354 - val_accuracy: 0.3778 - val_lo
Epoch 3/5
6/6            0s 6ms/step - accuracy: 0.5157 - loss: 0.8244 - val_accuracy: 0.4222 - val_lo
Epoch 4/5
6/6            0s 7ms/step - accuracy: 0.4659 - loss: 0.8866 - val_accuracy: 0.4222 - val_lo
Epoch 5/5
6/6            0s 7ms/step - accuracy: 0.5052 - loss: 0.7700 - val_accuracy: 0.4444 - val_lo
```

```
model_history.history.keys()
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

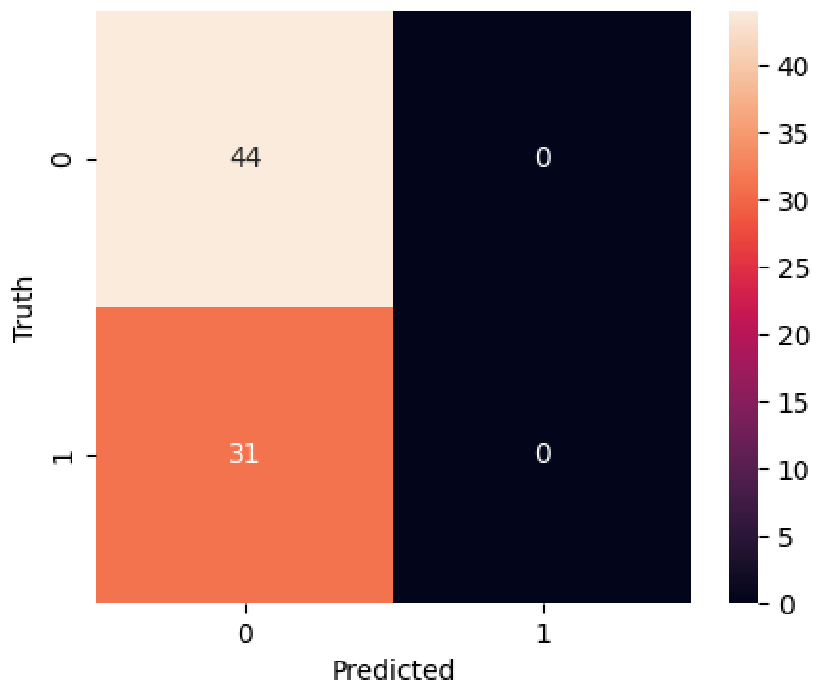
```
# 6. Extract accuracy values from model history
accuracy = model_history.history['accuracy'][-1] * 100
val_accuracy = model_history.history['val_accuracy'][-1] * 100

print("Training Accuracy: {:.2f}%".format(accuracy))
print("Validation Accuracy: {:.2f}%".format(val_accuracy))
```

```
Training Accuracy: 51.40%
Validation Accuracy: 44.44%
```

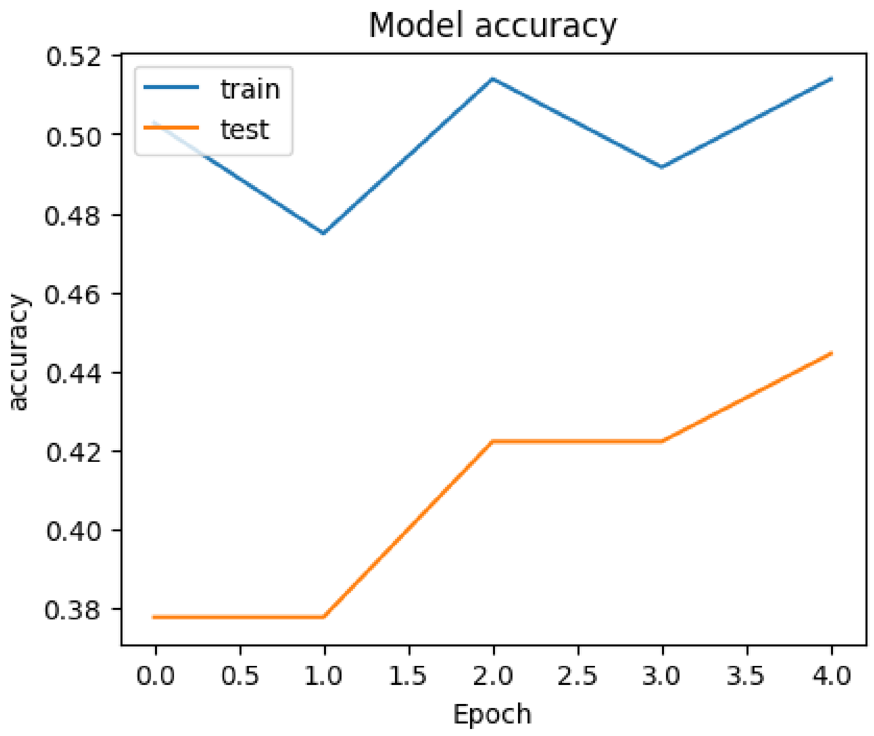
```
# 7. Prediction and Confusion matrix

y_predicted = classifier.predict(x_test)

y_predicted_labels = [np.argmax(i) for i in y_predicted]

cm = tf.math.confusion_matrix(labels = y_test,predictions = y_predicted_labels)

plt.figure(figsize = (5,4))
sns.heatmap(cm,annot = True,fmt = 'd')
plt.xlabel("Predicted")
plt.ylabel("Truth")
```

```
3/3                0s 74ms/step
```

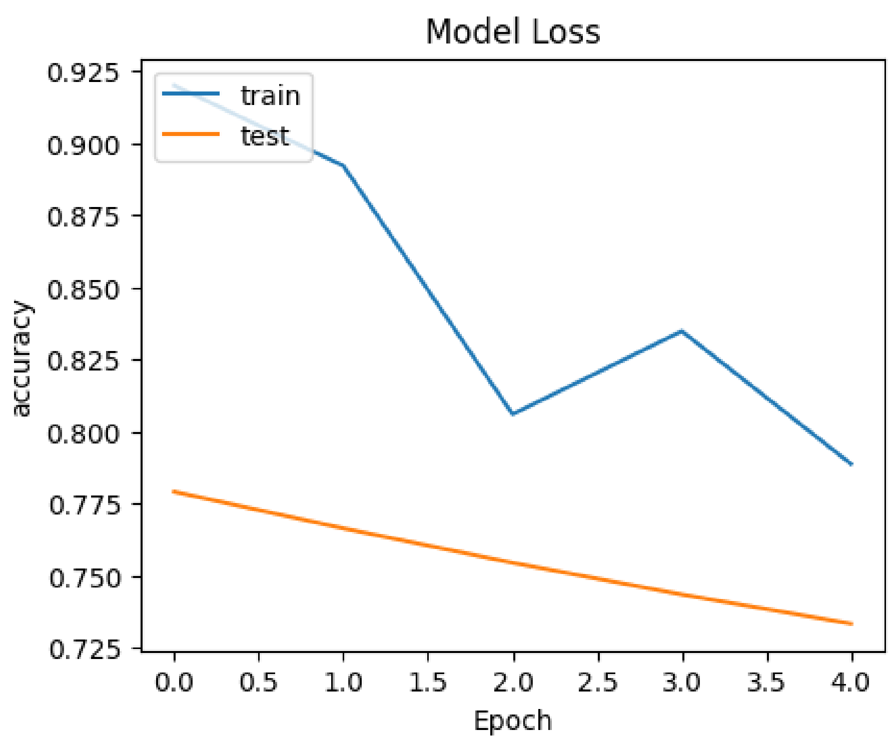
```
Text(33.22222222222222, 0.5, 'Truth')
```



```
# 8. Plottings

plt.figure(figsize = (5,4))
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('accuracy')
plt.legend(['train','test'],loc="upper left")
plt.show()
```



```
plt.figure(figsize = (5,4))
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('accuracy')
plt.legend(['train','test'],loc="upper left")
plt.show()
```



ANN Images

Building a deep learning model that can accurately recognize handwritten digits based on the MNIST dataset

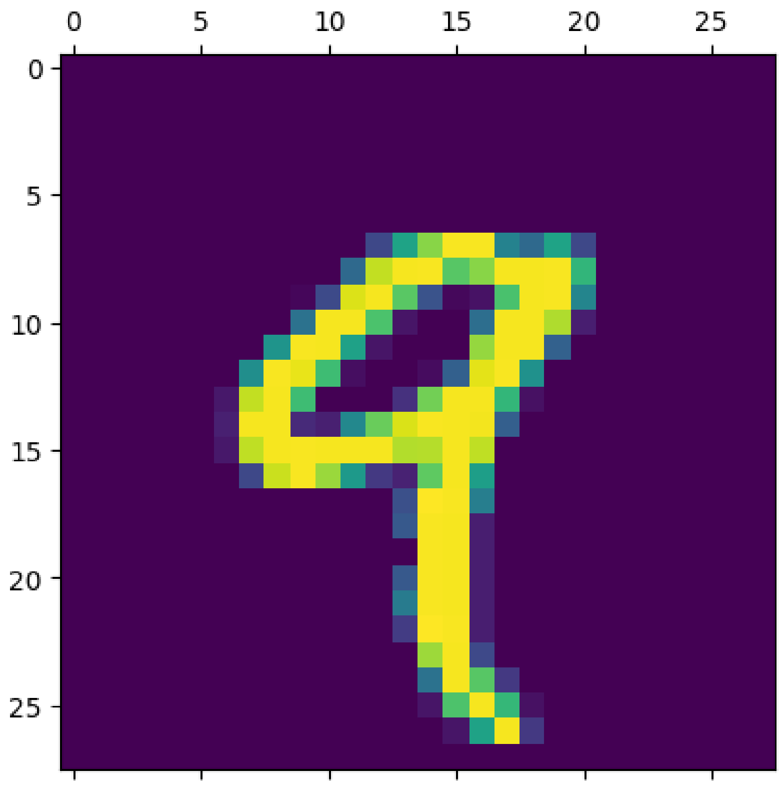
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline
```

```
# Loading the MNIST dataset, which contains handwritten digits, and splits it into training
(X_train, y_train) , (X_test, y_test) = mnist.load_data()
```

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

```
# Exploring Dataset

#1. Checking the size of dataset
print(len(X_train))
print(len(X_test))

#2. Shape of first image
print(X_train[0].shape)

# 3. Label and Image of 5th image
plt.matshow(X_train[4])
print("Label:", y_train[4])
```

```
60000
10000
(28, 28)
Label: 9
```



```python
# Data preprocessing

# Normalizing the pixel values of the images to be between 0 and 1.
X_train=X_train/255
X_test=X_test/255
```

```python
# Using Flatten layer so that we don't have to call .reshape on input dataset
from keras.models import Sequential
from keras.layers import Flatten,Dense
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))

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

model.fit(X_train, y_train, epochs=10)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarnin
  super().__init__(**kwargs)
```

```
Epoch 1/10
1875/1875            4s 2ms/step - accuracy: 0.8756 - loss: 0.4509
Epoch 2/10
1875/1875            3s 2ms/step - accuracy: 0.9620 - loss: 0.1291
Epoch 3/10
1875/1875            6s 2ms/step - accuracy: 0.9746 - loss: 0.0849
Epoch 4/10
1875/1875            3s 1ms/step - accuracy: 0.9811 - loss: 0.0644
Epoch 5/10
1875/1875            3s 2ms/step - accuracy: 0.9853 - loss: 0.0481
Epoch 6/10
1875/1875            6s 2ms/step - accuracy: 0.9886 - loss: 0.0378
Epoch 7/10
1875/1875            5s 3ms/step - accuracy: 0.9896 - loss: 0.0328
Epoch 8/10
1875/1875            8s 2ms/step - accuracy: 0.9917 - loss: 0.0261
Epoch 9/10
1875/1875            3s 2ms/step - accuracy: 0.9938 - loss: 0.0206
Epoch 10/10
1875/1875            5s 2ms/step - accuracy: 0.9946 - loss: 0.0173
```

```
<keras.src.callbacks.history.History at 0x7ec76c3e61d0>
```

```
model.evaluate(X_test, y_test)
```

```
313/313             1s 3ms/step - accuracy: 0.9726 - loss: 0.0975
```

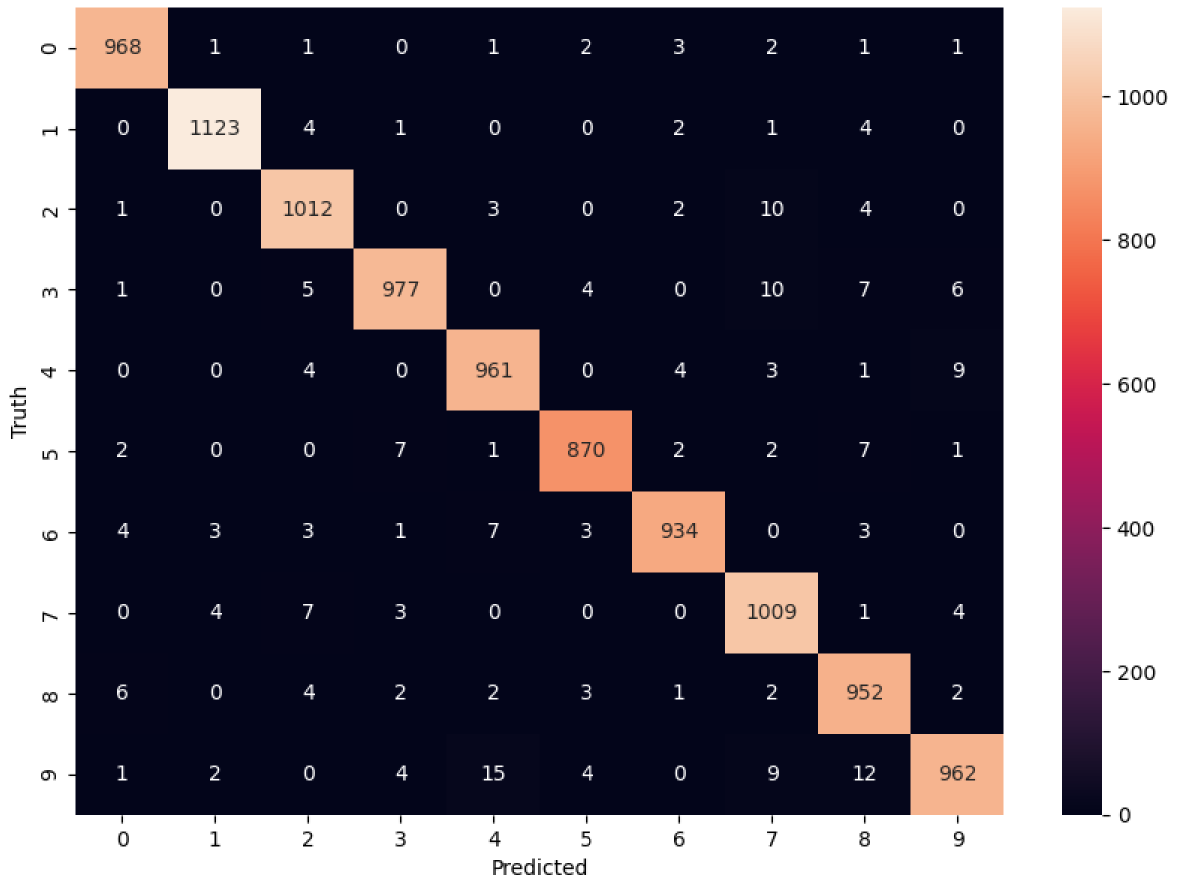
```
[0.08356696367263794, 0.9768000245094299]
```

```
# Evaluating and Visualizing Model Performance
y_predicted = model.predict(X_test)
np.argmax(y_predicted[0])
y_predicted_labels = [np.argmax(i) for i in y_predicted]

cm = tf.math.confusion_matrix(labels=y_test, predictions=y_predicted_labels)

plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTrainer.make_predict_func
```

```
313/313             1s 2ms/step
```

```
Text(95.72222222222221, 0.5, 'Truth')
```



Module 2 - CNN

Implementing a Convolutional Neural Network (CNN) to classify images of brain tumors into four distinct categories

BrainTumour

```
!unzip -u "/content/BrainTumour.zip" -d "/content/BrainTumour"
```

```
unzip:  cannot find or open /content/BrainTumour.zip, /content/BrainTumour.zip.zip or /conte
```

```python
import tensorflow as tf
from tensorflow import keras

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPool2D,Dense,Dropout,Flatten
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam, Adagrad
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.callbacks import EarlyStopping

from sklearn.metrics import confusion_matrix

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# 1. Importing

train_path = '/content/drive/MyDrive/BrainTumour/Training'
test_path = '/content/drive/MyDrive/BrainTumour/Testing'
train_datagen = ImageDataGenerator(validation_split = 0.2,rescale = 1./255)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

```python
# 2. train validation test batches

train_batches = train_datagen.flow_from_directory(
    train_path,target_size = (64,64),
    batch_size = 32,
    subset = 'training',
    class_mode = 'categorical'
)

validation_batches = train_datagen.flow_from_directory(
    train_path,target_size = (64,64),
    batch_size = 32,
    subset = 'validation',
    class_mode = 'categorical'
)

test_batches = test_datagen.flow_from_directory(
    test_path,target_size = (64,64),
    batch_size = 32,
    subset = 'training',
    class_mode = 'categorical'
)
```

```
Found 2297 images belonging to 4 classes.
Found 573 images belonging to 4 classes.
Found 394 images belonging to 4 classes.
```

```
# 3. Model

model = Sequential()
model.add(Conv2D(32,3,activation = 'relu',input_shape = (64,64,3)))
model.add(Dropout(0.25))
model.add(Conv2D(64,3,activation='relu',kernel_regularizer=l2(0.01)))
model.add(MaxPool2D())
model.add(Flatten())
# Hidden Layer
# model -1
model.add(Dense(128,activation = 'relu'))
model.add(Dropout(0.25))
# model-2
model.add(Dense(64,activation = 'relu'))
model.add(Dropout(0.25))
# model - 3
model.add(Dense(32,activation = 'relu'))
model.add(Dropout(0.25))
# output layer
model.add(Dense(4,activation = 'softmax'))
```

```
# 4. Two optimizers (Run whole code with another optimizer)

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# 5. fit

early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.001, patience=5, verbose=0, mo

history = model.fit(x=train_batches, validation_data=validation_batches, epochs=10, verbose=
Epoch 1/10

/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.p
  self._warn_if_super_not_called()

72/72                 808s 10s/step - accuracy: 0.3467 - loss: 1.8245 - val_accuracy: 0.4206 - va
Epoch 2/10
72/72                 12s 157ms/step - accuracy: 0.5367 - loss: 1.2050 - val_accuracy: 0.4311 - v
Epoch 3/10
72/72                 12s 157ms/step - accuracy: 0.6636 - loss: 0.8612 - val_accuracy: 0.4869 - v
Epoch 4/10
72/72                 20s 151ms/step - accuracy: 0.7309 - loss: 0.7505 - val_accuracy: 0.5969 - v
Epoch 5/10
72/72                 12s 153ms/step - accuracy: 0.7583 - loss: 0.6144 - val_accuracy: 0.5654 - v
Epoch 6/10
72/72                 21s 156ms/step - accuracy: 0.8128 - loss: 0.5177 - val_accuracy: 0.5637 - v
Epoch 7/10
72/72                 12s 160ms/step - accuracy: 0.8489 - loss: 0.4061 - val_accuracy: 0.5707 - v
Epoch 8/10
72/72                 20s 151ms/step - accuracy: 0.9017 - loss: 0.3188 - val_accuracy: 0.5620 - v
Epoch 9/10
72/72                 12s 155ms/step - accuracy: 0.8880 - loss: 0.3302 - val_accuracy: 0.6021 - v
```

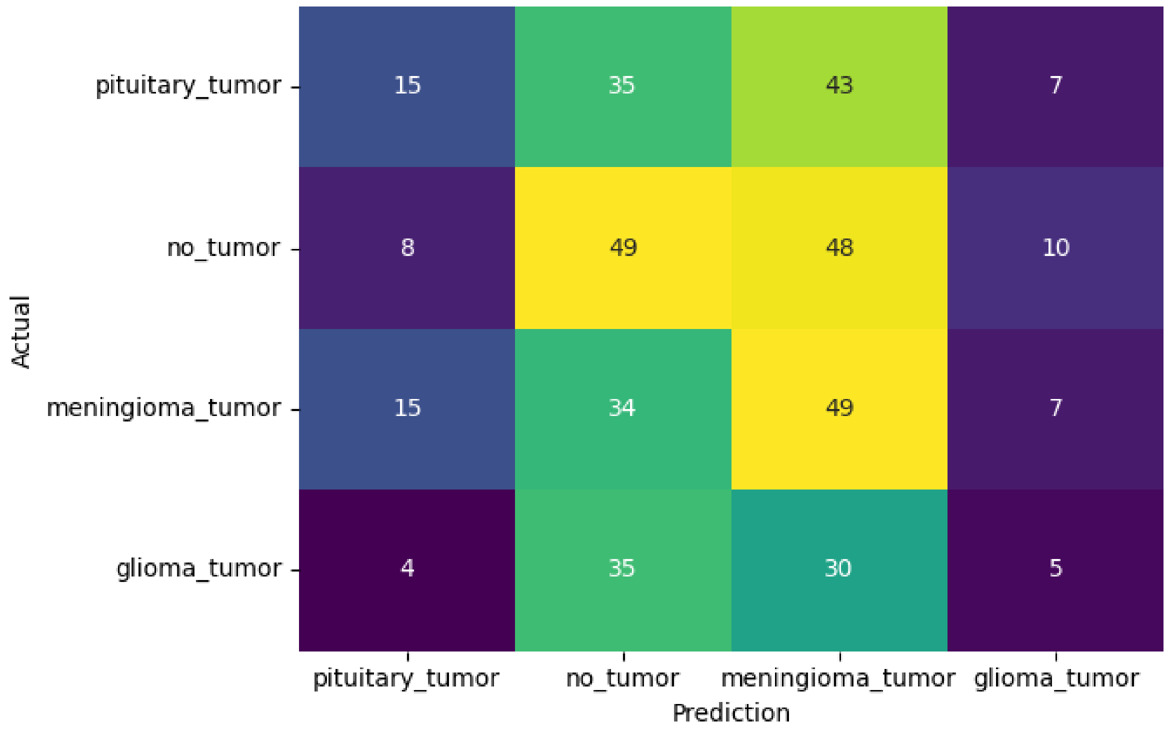
```
# 6. Evalaution

#history.history.keys()

accuracy = history.history['accuracy'][-1] * 100
val_accuracy = history.history['val_accuracy'][-1] * 100

print(f'Training accuracy: {round(accuracy, 2)}%')
print(f'Validation accuracy: {round(val_accuracy, 2)}%')
```

```
Training accuracy: 89.81%
Validation accuracy: 60.21%
```

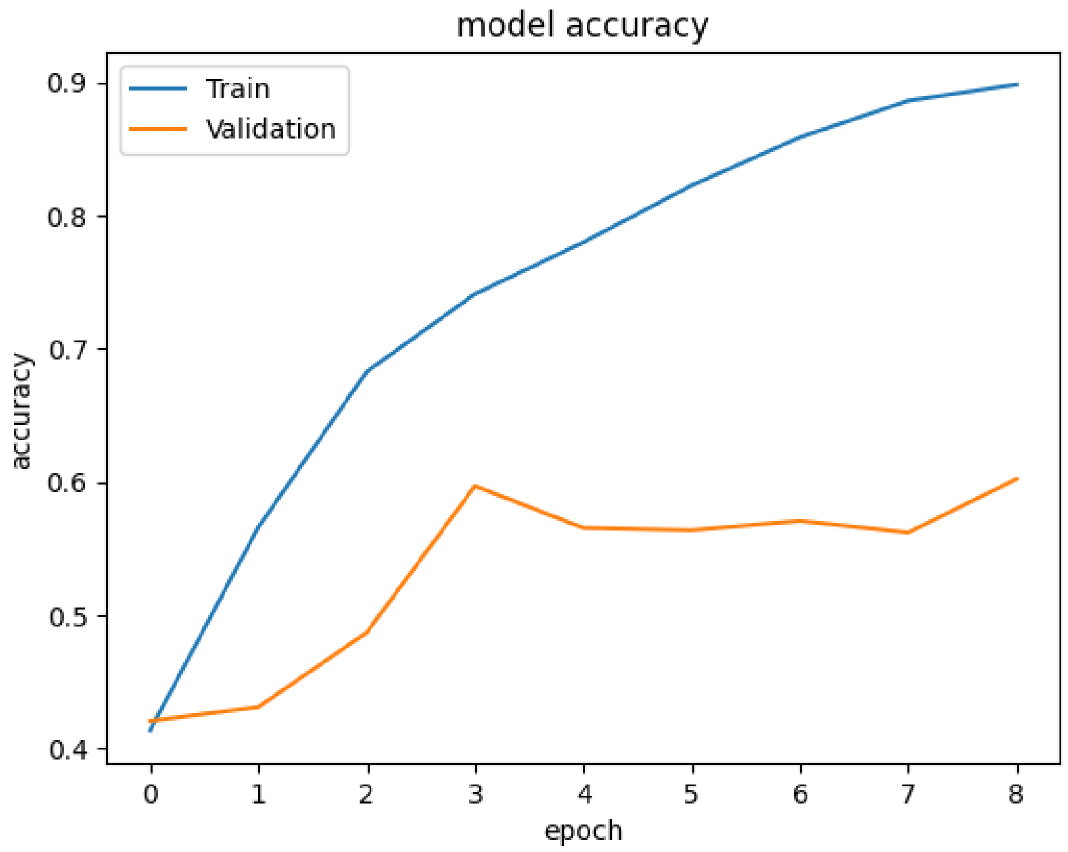
```
# 7. Prediction and confusion matrix

predictions = model.predict(test_batches)

predictions = np.round(predictions)
```

```
cm = confusion_matrix(y_true=test_batches.classes,y_pred=np.argmax(predictions,axis=1))
ax = sns.heatmap(cm, annot=True, xticklabels=["pituitary_tumor",'no_tumor',"meningioma_tumor"
                 yticklabels=["pituitary_tumor",'no_tumor',"meningioma_tumor","glioma_tumor"]
ax.set_xlabel('Prediction')
ax.set_ylabel('Actual')
plt.show()
```

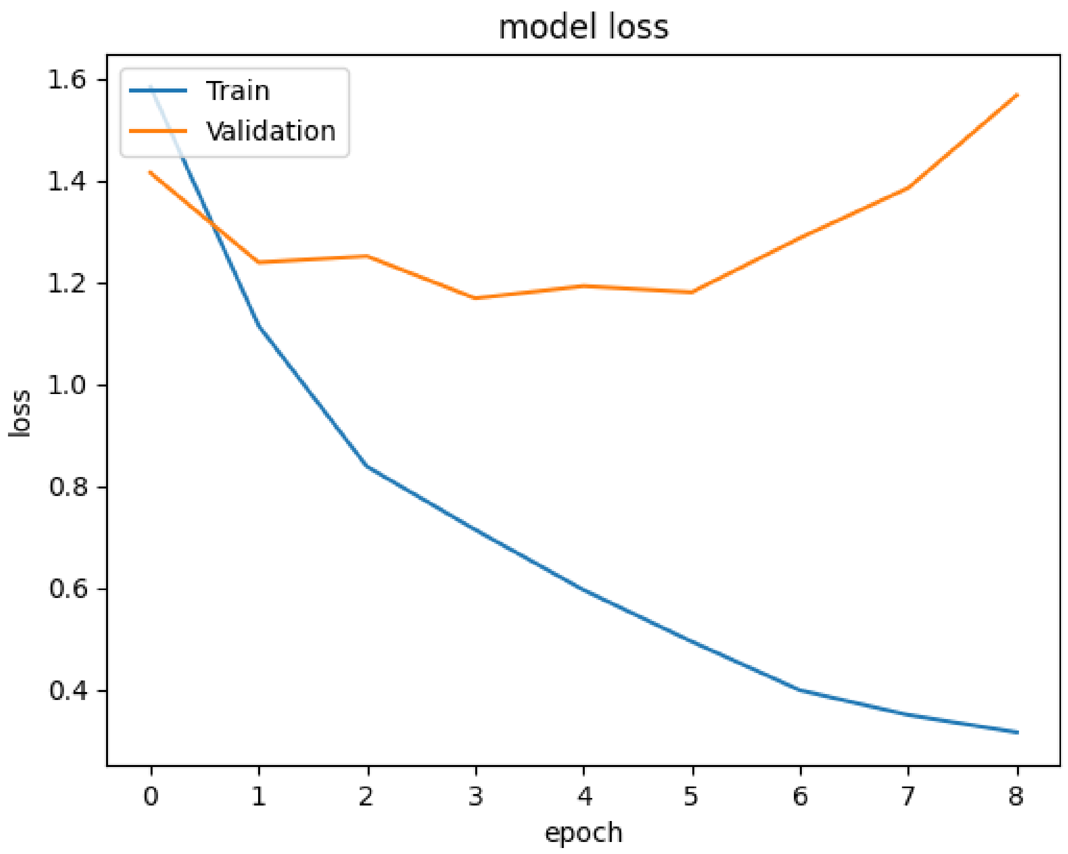
```
13/13             98s 8s/step
```



```
# 8. Summarize history for accuracy

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



Module 3 - RNN

Classification of names as either male or female using a character-level recurrent neural network (RNN)

Gender Classification

```
import numpy as np

from sklearn.model_selection import train_test_split

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Embedding
```

```python
# Dataset
names = ['John', 'Jane', 'Alice', 'Bob', 'Michael', 'Mary', 'David', 'Sarah', 'James', 'Emily
         'William', 'Emma', 'Matthew', 'Olivia', 'Daniel', 'Sophia', 'Christopher', 'Isabella
genders = ['male', 'female', 'female', 'male', 'male', 'female', 'male', 'female', 'male', '
           'male', 'female', 'male', 'female', 'male', 'female', 'male', 'female', 'male', '
```

```python
# Data Preprocessing

# 1. Convert names to numerical representation
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(names)
print("Vocabulary:", tokenizer.word_index)

X = tokenizer.texts_to_sequences(names)
print(X)

# 2. Pad sequences to a fixed length
max_length = max(len(seq) for seq in X)
X = pad_sequences(X, maxlen=max_length, padding='post')
print(X)

X = np.array(X)    # Convert to numpy array

# 3. One-hot encode genders
y = np.array([1 if gender == 'male' else 0 for gender in genders])

# 4. Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(len(tokenizer.word_index))
print(X.shape[1])
```

```
Vocabulary: {'a': 1, 'i': 2, 'e': 3, 'l': 4, 'm': 5, 'h': 6, 'o': 7, 'r': 8, 's': 9, 'n': 10
[[12, 7, 6, 10], [12, 1, 10, 3], [1, 4, 2, 13, 3], [14, 7, 14], [5, 2, 13, 6, 1, 3, 4], [5, 
[[12  7  6 10  0  0  0  0  0  0  0]
 [12  1 10  3  0  0  0  0  0  0  0]
 [ 1  4  2 13  3  0  0  0  0  0  0]
 [14  7 14  0  0  0  0  0  0  0  0]
 [ 5  2 13  6  1  3  4  0  0  0  0]
 [ 5  1  8 18  0  0  0  0  0  0  0]
 [11  1 15  2 11  0  0  0  0  0  0]
 [ 9  1  8  1  6  0  0  0  0  0  0]
 [12  1  5  3  9  0  0  0  0  0  0]
 [ 3  5  2  4 18  0  0  0  0  0  0]
 [16  2  4  4  2  1  5  0  0  0  0]
 [ 3  5  5  1  0  0  0  0  0  0  0]
 [ 5  1 17 17  6  3 16  0  0  0  0]
 [ 7  4  2 15  2  1  0  0  0  0  0]
 [11  1 10  2  3  4  0  0  0  0  0]
 [ 9  7 19  6  2  1  0  0  0  0  0]
 [13  6  8  2  9 17  7 19  6  3  8]
 [ 2  9  1 14  3  4  4  1  0  0  0]
 [ 1 10 11  8  3 16  0  0  0  0  0]
 [ 1 15  1  0  0  0  0  0  0  0  0]]
19
11
```

```
# Simple RNN model
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=10, input_length=X.sh
model.add(SimpleRNN(10))
model.add(Dense(1, activation='sigmoid'))

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

model.fit(X_train, y_train, epochs=10, batch_size=1, verbose=1)
```

```
Epoch 1/10

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: 
  warnings.warn(

16/16            3s 3ms/step - accuracy: 0.6526 - loss: 0.6732
Epoch 2/10
16/16            0s 3ms/step - accuracy: 0.5441 - loss: 0.6660
Epoch 3/10
16/16            0s 3ms/step - accuracy: 0.9269 - loss: 0.5826
Epoch 4/10
16/16            0s 3ms/step - accuracy: 0.8733 - loss: 0.5417
Epoch 5/10
16/16            0s 3ms/step - accuracy: 0.9751 - loss: 0.4394
Epoch 6/10
16/16            0s 3ms/step - accuracy: 0.9053 - loss: 0.3288
Epoch 7/10
```

```
16/16                 0s 4ms/step - accuracy: 1.0000 - loss: 0.2234
Epoch 8/10
16/16                 0s 3ms/step - accuracy: 1.0000 - loss: 0.1863
Epoch 9/10
16/16                 0s 3ms/step - accuracy: 1.0000 - loss: 0.1059
Epoch 10/10
16/16                 0s 6ms/step - accuracy: 1.0000 - loss: 0.0974
```

```
<keras.src.callbacks.history.History at 0x7ec76c34d690>
```

```
# Evaluation and validation
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Test Accuracy: {accuracy * 100:.2f}%')
```

```
Test Accuracy: 25.00%
```

```
# Prediction and Testing

# Define a function to predict gender for a given name
def predict_gender(name):
    name_seq = tokenizer.texts_to_sequences([name])   # Convert names to numerical represent

    max_length = max(len(name_seq) for name_seq in X)     # Pad sequences to a fixed length
    name_seq = pad_sequences(name_seq, maxlen=max_length, padding='post')

    name_seq = np.array(name_seq)     # Convert to numpy array

    prediction = model.predict(name_seq)[0][0]       # Make the prediction

    gender = 'male' if prediction >= 0.5 else 'female'       # Convert prediction to gender

    return gender

# Testing with a random name
random_name = 'Emma'
predicted_gender = predict_gender(random_name)
print(f"The predicted gender for the name '{random_name}' is: {predicted_gender}")
```

```
1/1               0s 265ms/step
The predicted gender for the name 'Emma' is: female
```

Email Classifcation

Implementing a basic Recurrent Neural Network (RNN) for email classification to determine whether an email is spam or non-spam

```
# Sample Dataset :
messages = ['Reminder:Your appointment is tomorrow at 10 AM','Dont forget to submit your repo
            'Thank you for your payment.Your order has been confirmed', 'Your flight reservat
            'Congratulations on your recent promotion! Well Deserved',
            'Congratulations! You have qualified for a cashback reward','Unlock VIP access an
            'Get Free shipping on all orders for a limited time only','Dont miss this chance
            'Exclusive deal alert! Save up to 70% on select items']
            # names
labels = [0,0,0,0,0,1,1,1,1,1]

# gender
# 1 - Non Spam
# 0 - Spam
```

```
# Data Preprocessing

# 1. Convert names to numerical representation
tokenizer = Tokenizer(char_level=False)
tokenizer.fit_on_texts(messages)
print("Vocabulary:", tokenizer.word_index)

X = tokenizer.texts_to_sequences(messages)
print(X)

# 2. Pad sequences to a fixed length
max_length = max(len(seq) for seq in X)
X = pad_sequences(X, maxlen=max_length, padding='post')
print(X)

X = np.array(X)    # Convert to numpy array

# 3. One-hot encode
y = np.array(labels)

# 4. Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
print(len(tokenizer.word_index))
print(X.shape[1])
```

```
Vocabulary: {'your': 1, 'to': 2, 'for': 3, 'on': 4, 'dont': 5, 'you': 6, 'has': 7, 'been': 8
[[14, 1, 15, 16, 17, 18, 19, 20], [5, 21, 2, 22, 1, 23, 24, 25, 26, 27], [28, 6, 3, 1, 29, 1
[[14  1 15 16 17 18 19 20  0  0  0]
 [ 5 21  2 22  1 23 24 25 26 27  0]
 [28  6  3  1 29  1 30  7  8 31  0]
 [ 1 32 33  7  8 34 35  0  0  0  0]
 [ 9  4  1 36 37 38 39  0  0  0  0]
 [ 9  6 40 41  3 10 42 43  0  0  0]
 [44 11 45 12 46 11 13 47 12 48  0]
 [49 50 51  4 52 53  3 10 54 55 13]
 [ 5 56 57 58  2 59  1 60 61 62  0]
 [63 64 65 66 67  2 68  4 69 70  0]]
70
11
```

```
# Simple RNN model
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=10, input_length=X.sl
model.add(SimpleRNN(10))
model.add(Dense(1, activation='sigmoid'))

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

model.fit(X_train, y_train, epochs=10, batch_size=1, verbose=1)
```

```
Epoch 1/10
8/8               3s 3ms/step - accuracy: 0.7053 - loss: 0.6814
Epoch 2/10
8/8               0s 3ms/step - accuracy: 0.9156 - loss: 0.6509
Epoch 3/10
8/8               0s 3ms/step - accuracy: 1.0000 - loss: 0.5855
Epoch 4/10
8/8               0s 3ms/step - accuracy: 1.0000 - loss: 0.5241
Epoch 5/10
8/8               0s 3ms/step - accuracy: 1.0000 - loss: 0.4546
Epoch 6/10
8/8               0s 3ms/step - accuracy: 1.0000 - loss: 0.3736
Epoch 7/10
```

```
8/8              0s 3ms/step - accuracy: 1.0000 - loss: 0.3102
Epoch 8/10
8/8              0s 3ms/step - accuracy: 1.0000 - loss: 0.2188
Epoch 9/10
8/8              0s 3ms/step - accuracy: 1.0000 - loss: 0.1881
Epoch 10/10
8/8              0s 3ms/step - accuracy: 1.0000 - loss: 0.1459

<keras.src.callbacks.history.History at 0x7ec762c747f0>
```

```python
# Evaluation and validation
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Test Accuracy: {accuracy * 100:.2f}%')
```

```
Test Accuracy: 50.00%
```

```python
# Prediction and Testing

# Define a function to classify mail to spam or not spam
def predict_message(message):
    tf.keras.backend.clear_session()     # Clear the TensorFlow session

    message_seq = tokenizer.texts_to_sequences([messages])     # Convert the message to nume

    max_length = max(len(seq) for seq in X)       # Pad sequences to a fixed length
    message_seq = pad_sequences(message_seq, maxlen=max_length, padding='post')

    message_seq = np.array(message_seq)        # Convert to numpy array

    prediction = model.predict(message_seq)[0][0]        # Make the prediction

    labels = 'Non-Spam' if prediction >= 0.5 else 'Spam'        # Convert prediction to clas

    return labels

# Testing with a random message
random_message = 'You need to come to the office hours tomorrow'
predicted_label = predict_message(random_message)
print(f"The message classification for '{random_message}' is: {predicted_label}")
```

```
1/1              0s 108ms/step
The message classification for 'You need to come to the office hours tomorrow' is: Non-Spam
```

Sentiment Analysis

Analyzing the sentiment based on textual statements using Natural Language processing

```python
# Sentiment Analysis

sentiment_message = [
    'I love this product! It works really well.',
    'The customer service was terrible. I will never buy from this company again.',
    'The movie was amazing. I highly recommend it to everyone.',
    'The food was awful. I wouldn\'t eat here again even if it was free.',
    'I had a great experience with this service. I will definitely use it again.',
    'The product arrived damaged and the seller was unresponsive.',
    'The book was disappointing. I expected more from the author.',
    'This restaurant never disappoints. The food is always delicious.',
    'The delivery was delayed and the package arrived damaged.',
    'The hotel room was dirty and the staff were rude.'
]

# Corresponding Sentiment Labels (0 for negative, 1 for positive):
labels_messages = [1, 0, 1, 0, 1, 0, 0, 1, 0, 0]
```

```python
# Data Preprocessing

# 1. Convert names to numerical representation
tokenizer = Tokenizer(char_level=False)
tokenizer.fit_on_texts(sentiment_message)
print("Vocabulary:", tokenizer.word_index)

X = tokenizer.texts_to_sequences(sentiment_message)
print(X)

# 2. Pad sequences to a fixed length
max_length = max(len(seq) for seq in X)
X = pad_sequences(X, maxlen=max_length, padding='post')
print(X)

X = np.array(X)    # Convert to numpy array

# 3. One-hot encode genders
```

```
y = np.array(labels_messages)

# 4. Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(len(tokenizer.word_index))
print(X.shape[1])
```

```
Vocabulary: {'the': 1, 'was': 2, 'i': 3, 'this': 4, 'it': 5, 'again': 6, 'and': 7, 'product'
[[3, 16, 4, 8, 5, 17, 18, 19], [1, 20, 9, 2, 21, 3, 10, 11, 22, 12, 4, 23, 6], [1, 24, 2, 25
[[ 3 16  4  8  5 17 18 19  0  0  0  0  0  0]
 [ 1 20  9  2 21  3 10 11 22 12  4 23  6  0]
 [ 1 24  2 25  3 26 27  5 28 29  0  0  0  0]
 [ 1 13  2 30  3 31 32 33  6 34 35  5  2 36]
 [ 3 37 38 39 40 41  4  9  3 10 42 43  5  6]
 [ 1  8 14 15  7  1 44  2 45  0  0  0  0  0]
 [ 1 46  2 47  3 48 49 12  1 50  0  0  0  0]
 [ 4 51 11 52  1 13 53 54 55  0  0  0  0  0]
 [ 1 56  2 57  7  1 58 14 15  0  0  0  0  0]
 [ 1 59 60  2 61  7  1 62 63 64  0  0  0  0]]
64
14
```

```
# Simple RNN model
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=10, input_length=X.s
model.add(SimpleRNN(10))
model.add(Dense(1, activation='sigmoid'))

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

model.fit(X_train, y_train, epochs=2, batch_size=1, verbose=1)
```

```
Epoch 1/2
8/8              2s 4ms/step - accuracy: 0.8942 - loss: 0.6653
Epoch 2/2
8/8              0s 3ms/step - accuracy: 0.7108 - loss: 0.6371

<keras.src.callbacks.history.History at 0x7ec76c3c78b0>
```

```python
# Evaluation and validation
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Test Accuracy: {accuracy * 100:.2f}%')
```

```
Test Accuracy: 100.00%
```

```python
# Prediction and Testing

# Define a function to perform sentiment analysis
def predict_sentiment_message(sentiment_message):
    tf.keras.backend.clear_session()     # Clear the TensorFlow session

    message_seq = tokenizer.texts_to_sequences([sentiment_message])     # Convert the messag

    max_length = max(len(seq) for seq in X)       # Pad sequences to a fixed length
    message_seq = pad_sequences(message_seq, maxlen=max_length, padding='post')

    message_seq = np.array(message_seq)        # Convert to numpy array

    prediction = model.predict(message_seq)[0][0]        # Make the prediction

    labels = 'Positive' if prediction >= 0.5 else 'Negative'     # Convert prediction to cla

    return labels

# Testing with a random message
random_message = 'I love this product! It works really well.'
predicted_label = predict_message(random_message)
print(f"The message classification for '{random_message}' is: {predicted_label}")
```

```
WARNING:tensorflow:5 out of the last 16 calls to <function TensorFlowTrainer.make_predict_fu

1/1            0s 265ms/step
The message classification for 'I love this product! It works really well.' is: Non-Spam
```

Module 3 - Next Word Prediction

Predicting the next word based on incomplete sentences

```
# import

import numpy as np
import tensorflow as tf
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

with open("/content/IndiaUS.txt") as myfile:
    mytext = myfile.read()  # Read the entire text file as a string
```

```
# Data Preprocessing

# 1. Convert names to numerical representation
tokenizer = Tokenizer()
tokenizer.fit_on_texts([mytext])
num_classes = len(tokenizer.word_index) + 1

input_sequences = []
for line in mytext.split('\n'):
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        input_sequences.append(token_list[:i+1])

# 2. Pad sequences to a fixed length
maxlen = max(len(seq) for seq in input_sequences)
input_sequences = np.array(pad_sequences(input_sequences, maxlen=maxlen, padding='pre'))
# print(input_sequences)

# 3. x, y
X = input_sequences[:, :-1]
y = input_sequences[:, -1]
y = np.array(tf.keras.utils.to_categorical(y, num_classes=num_classes))
```

```python
# LSTM Model

model = Sequential()
model.add(Embedding(num_classes, 100, input_length=maxlen-1))
model.add(LSTM(150))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=59, verbose=1)
```

```
Epoch 1/59

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: 
  warnings.warn(

43/43            4s 9ms/step - accuracy: 0.0434 - loss: 6.2712
Epoch 2/59
43/43            0s 7ms/step - accuracy: 0.0426 - loss: 5.8116
Epoch 3/59
43/43            0s 7ms/step - accuracy: 0.0640 - loss: 5.7218
Epoch 4/59
43/43            0s 6ms/step - accuracy: 0.0536 - loss: 5.6789
Epoch 5/59
43/43            0s 6ms/step - accuracy: 0.0530 - loss: 5.5070
Epoch 6/59
43/43            0s 7ms/step - accuracy: 0.0474 - loss: 5.5270
Epoch 7/59
43/43            0s 7ms/step - accuracy: 0.0552 - loss: 5.3251
Epoch 8/59
43/43            0s 6ms/step - accuracy: 0.0725 - loss: 5.1749
Epoch 9/59
43/43            0s 6ms/step - accuracy: 0.1029 - loss: 4.9420
Epoch 10/59
43/43            0s 7ms/step - accuracy: 0.1180 - loss: 4.6287
Epoch 11/59
43/43            0s 6ms/step - accuracy: 0.1448 - loss: 4.4686
Epoch 12/59
43/43            0s 6ms/step - accuracy: 0.1602 - loss: 4.2695
Epoch 13/59
43/43            0s 7ms/step - accuracy: 0.1679 - loss: 4.0743
Epoch 14/59
```

```
43/43            0s 6ms/step - accuracy: 0.1985 - loss: 3.8813
Epoch 15/59
43/43            0s 7ms/step - accuracy: 0.2411 - loss: 3.5732
Epoch 16/59
43/43            0s 7ms/step - accuracy: 0.2738 - loss: 3.4481
Epoch 17/59
43/43            1s 6ms/step - accuracy: 0.2981 - loss: 3.2945
Epoch 18/59
43/43            0s 6ms/step - accuracy: 0.3328 - loss: 3.0863
Epoch 19/59
43/43            0s 7ms/step - accuracy: 0.4257 - loss: 2.8350
Epoch 20/59
43/43            0s 6ms/step - accuracy: 0.4556 - loss: 2.6780
Epoch 21/59
43/43            0s 6ms/step - accuracy: 0.5218 - loss: 2.4800
Epoch 22/59
43/43            0s 7ms/step - accuracy: 0.5512 - loss: 2.3449
Epoch 23/59
43/43            1s 9ms/step - accuracy: 0.6374 - loss: 2.1380
Epoch 24/59
43/43            1s 9ms/step - accuracy: 0.6626 - loss: 2.0485
Epoch 25/59
43/43            1s 21ms/step - accuracy: 0.7119 - loss: 1.8379
Epoch 26/59
43/43            0s 11ms/step - accuracy: 0.7575 - loss: 1.7240
Epoch 27/59
43/43            1s 22ms/step - accuracy: 0.7941 - loss: 1.6087
Epoch 28/59
43/43            1s 11ms/step - accuracy: 0.8181 - loss: 1.4396
Epoch 29/59
43/43            0s 7ms/step - accuracy: 0.8378 - loss: 1.3609
Epoch 30/59
43/43            0s 7ms/step - accuracy: 0.8785 - loss: 1.2069
Epoch 31/59
43/43            0s 7ms/step - accuracy: 0.8946 - loss: 1.1226
Epoch 32/59
43/43            0s 7ms/step - accuracy: 0.9098 - loss: 1.0002
Epoch 33/59
43/43            0s 6ms/step - accuracy: 0.9241 - loss: 0.9355
Epoch 34/59
43/43            0s 7ms/step - accuracy: 0.9325 - loss: 0.8460
Epoch 35/59
43/43            0s 7ms/step - accuracy: 0.9494 - loss: 0.7949
```

```
Epoch 36/59
43/43           0s 6ms/step - accuracy: 0.9488 - loss: 0.7151
Epoch 37/59
43/43           0s 7ms/step - accuracy: 0.9622 - loss: 0.6858
Epoch 38/59
43/43           1s 7ms/step - accuracy: 0.9574 - loss: 0.6278
Epoch 39/59
43/43           0s 7ms/step - accuracy: 0.9700 - loss: 0.5628
Epoch 40/59
43/43           0s 6ms/step - accuracy: 0.9721 - loss: 0.5222
Epoch 41/59
43/43           0s 7ms/step - accuracy: 0.9770 - loss: 0.4805
Epoch 42/59
43/43           1s 7ms/step - accuracy: 0.9673 - loss: 0.4651
Epoch 43/59
43/43           0s 7ms/step - accuracy: 0.9865 - loss: 0.3991
Epoch 44/59
43/43           0s 7ms/step - accuracy: 0.9820 - loss: 0.4012
Epoch 45/59
43/43           1s 7ms/step - accuracy: 0.9807 - loss: 0.3743
Epoch 46/59
43/43           0s 7ms/step - accuracy: 0.9760 - loss: 0.3526
Epoch 47/59
43/43           0s 7ms/step - accuracy: 0.9878 - loss: 0.3031
Epoch 48/59
43/43           0s 6ms/step - accuracy: 0.9829 - loss: 0.3111
Epoch 49/59
43/43           0s 7ms/step - accuracy: 0.9825 - loss: 0.2679
Epoch 50/59
43/43           0s 7ms/step - accuracy: 0.9842 - loss: 0.2641
Epoch 51/59
43/43           1s 7ms/step - accuracy: 0.9846 - loss: 0.2532
Epoch 52/59
43/43           0s 7ms/step - accuracy: 0.9844 - loss: 0.2373
Epoch 53/59
43/43           0s 7ms/step - accuracy: 0.9840 - loss: 0.2185
Epoch 54/59
43/43           0s 8ms/step - accuracy: 0.9810 - loss: 0.2138
Epoch 55/59
43/43           1s 9ms/step - accuracy: 0.9851 - loss: 0.2074
Epoch 56/59
43/43           0s 9ms/step - accuracy: 0.9857 - loss: 0.1962
Epoch 57/59
```

```
43/43                0s 9ms/step - accuracy: 0.9864 - loss: 0.1768
Epoch 58/59
43/43                1s 10ms/step - accuracy: 0.9911 - loss: 0.1642
Epoch 59/59
43/43                1s 10ms/step - accuracy: 0.9911 - loss: 0.1563
```

```
<keras.src.callbacks.history.History at 0x7ec76156acb0>
```

```
# Prediction

input_text = "Joe biden"
next_words = 6
for _ in range(next_words):
    token_list = tokenizer.texts_to_sequences([input_text])[0]
    token_list = pad_sequences([token_list], maxlen=maxlen-1, padding='pre')
    predicted = np.argmax(model.predict(token_list), axis=-1)
    output_word = ""
    for word, index in tokenizer.word_index.items():
        if index == predicted:
            output_word = word
            break
    input_text += " " + output_word
print(input_text)
```

```
1/1              0s 172ms/step
1/1              0s 17ms/step
1/1              0s 16ms/step
1/1              0s 16ms/step
1/1              0s 16ms/step
1/1              0s 16ms/step
Joe biden and indian prime minister narendra modi
```

Module 4 - Q Learning

Implementation of Q-learning, a reinforcement learning algorithm, to train an agent to navigate through a grid-based environment and reach a goal state while avoiding obstacle

```
# Define the environment
environment = np.array([
    [0, 0, 0, 0, 0],
    [1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 1],
    [0, 0, 0, 0, 3]
])

# Define parameters
num_states = np.prod(environment.shape)
num_actions = 4  # up, down, left, right
learning_rate = 0.1
gamma = 0.9  # discount factor
epsilon = 0.1  # exploration rate
num_episodes = 10

# Initialize Q-table
Q = np.zeros((num_states, num_actions))

# Convert 2D coordinates to 1D index
def state_to_index(state):
    return state[0] * environment.shape[1] + state[1]

# Convert 1D index to 2D coordinates
def index_to_state(index):
    return (index // environment.shape[1], index % environment.shape[1])

# Choose action using epsilon-greedy policy
def choose_action(state):
    if np.random.rand() < epsilon:
        return np.random.randint(num_actions)  # random action
    else:
        return np.argmax(Q[state_to_index(state)])  # greedy action

# Perform Q-learning
for episode in range(num_episodes):
    state = (4, 0)  # starting state
    done = False
    while not done:
        action = choose_action(state)
        next_state = state
        # Move agent to next state
        if action == 0:  # up
            next_state = (max(state[0] - 1, 0), state[1])
        elif action == 1:  # down
            next_state = (min(state[0] + 1, environment.shape[0] - 1), state[1])
        elif action == 2:  # left
            next_state = (state[0], max(state[1] - 1, 0))
        elif action == 3:  # right
            next_state = (state[0], min(state[1] + 1, environment.shape[1] - 1))

        # Reward
        if environment[next_state[0], next_state[1]] == 1:  # obstacle
            reward = -100
        elif environment[next_state[0], next_state[1]] == 3:  # goal
            reward = 100
            done = True
        else:
            reward = -1  # default penalty

        # Update Q-value
        Q[state_to_index(state), action] += learning_rate * (
            reward + gamma * np.max(Q[state_to_index(next_state)]) - Q[state_to_index(state)
        )

        # Move to next state
        state = next_state

# Print learned Q-values
print("Learned Q-values:")
print(Q)
```

```
Learned Q-values:
[[-6.79346521e-01 -1.90090000e+01 -6.79346521e-01 -6.88191453e-01]
 [-6.79346521e-01 -1.00000000e+01 -6.81950582e-01 -7.70867490e-01]
 [-7.72553056e-01 -1.90000000e+01 -7.65391267e-01 -8.06016741e-01]
 [-8.57201536e-01 -1.00000000e+01 -8.15391558e-01 -8.97164524e-01]
 [-9.52191035e-01 -9.77591284e-01 -9.03957593e-01 -9.56179250e-01]
 [-1.90000000e-01 -2.07910000e-01 -1.00000000e+01 -1.00000000e+01]
 [-1.00000000e-01 -1.19368000e-01 -1.00000000e+01 -1.00000000e+01]
 [-1.00000000e-01 -1.90000000e-01 -1.00000000e+01 -1.00000000e+01]
 [-1.00000000e-01 -1.00000000e-01 -1.00000000e+01 -1.44108955e-01]
 [-9.79090783e-01 -9.96952232e-01 -1.00000000e+01 -9.56179250e-01]
 [-1.90000000e+01 -1.02284816e+00 -9.55704705e-01 -9.67029822e-01]
 [-1.00000000e+01 -1.00000000e+01 -1.04286175e+00 -9.54645627e-01]
 [-1.00000000e+01 -1.00000000e+01 -1.03186229e+00 -9.77493096e-01]
 [-1.00000000e+01 -1.00000000e+01 -1.00790737e+00 -9.82914927e-01]
 [-9.76515510e-01 -1.00000000e+01 -1.01033629e+00 -9.56179250e-01]
 [-9.10646767e-01 -9.39216466e-01 -8.64827525e-01 -1.00000000e+01]
 [-1.00000000e-01 -1.00000000e-01 -1.09000000e-01 -1.00000000e+01]
 [-2.45717398e-01 -1.00000000e-01 -1.00000000e+01  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01 -1.00000000e+01  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-8.23988365e-01 -7.72553056e-01 -7.72553056e-01  5.16616042e-02]
 [-1.00000000e+01 -3.94039900e-01 -4.37597723e-01  5.67878087e+00]
 [-1.00000000e+01  1.89840910e+00 -1.99810000e-01  2.73986313e+01]
 [-1.90000000e+01 -1.00000000e-01  2.00044157e+00  6.51321560e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]
```

Module 5 - Autoencoders

Vanilla

Implementation of a vanilla autoencoder using Keras to compress and reconstruct images from the MNIST dataset, which contains grayscale images of handwritten digits.

```python
from keras.models import Model
from keras.layers import Dense, Input
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt

# Load MNIST Dataset
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

# Hyperparameters
batch_size = 128
```