MARMARA UNIVERSITY

FACULTY OF TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

BLM3053 INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS

FINAL PROJECT REPORT

STUDENT ID: 170421993, 170421048

NAME LASTNAME: Farid Bayramov, Yunus Ege Küçük

PROJECT TITLE: CinemAI

Prof. Dr. Serhat ÖZEKES

Res. Asst. Abdulsamet AKTAŞ

1. INTRODUCTION

The CinemAI project, made as a lecture project of the artificial neural networks lecture, focuses on creating a natural language processing model for generating film scripts. The aim of this study is to explore the potential of artificial intelligence algorithms in the creation of film scripts. The significance of this research centers on the integration of AI into the creative processes within the film industry.

The primary question of this research is to what extent scripts generated using artificial neural networks can be realistic, logical, and captivating for audiences. The study looks at how we can use computer programs to help write movie scripts.

This project fundamentally seeks to redefine the traditional paradigms of scriptwriting by seamlessly integrating AI into the creative processes within the film industry. As the entertainment landscape evolves, AI has emerged as a transformative force, offering the possibility of not only streamlining the scriptwriting process but also creating narratives that resonate deeply with audiences.

It is believed that this model may have the potential to compete with or enhance traditional screenwriting techniques. The research methodology utilizes deep learning techniques, while its limitations are related to the current constraints of the model in terms of emotional depth and creativity.

As the project advances, addressing these limitations and pushing the boundaries of AI-driven storytelling will be at the forefront. In doing so, we aim to contribute to a future where AI and human creativity harmoniously coexist in the art of crafting compelling film narratives.

2. LITERATURE REVIEW

The field of natural language processing began in the 1940s, after World War II. At this time, people recognized the importance of translation from one language to another and hoped to create a machine that could do this sort of translation automatically. However, the task was obviously not as easy as people first imagined. By 1958, some researchers were identifying significant issues in the development of NLP. One of these researchers was Noam Chomsky, who found it troubling that models of language recognized sentences that were nonsense but grammatically correct as equally irrelevant as sentences that were nonsense and not grammatically correct. Chomsky found it problematic that the sentence “Colorless green ideas sleep furiously” was classified as improbable to the same extent that “Furiously sleep ideas green colorless”; any speaker of English can recognize the former as grammatically correct and the latter as incorrect, and Chomsky felt the same should be expected of machine models [1].

Around the same time in history, from 1957-1970, researchers split into two divisions concerning NLP: symbolic and stochastic. Symbolic, or rule-based, researchers focused on formal languages and generating syntax; this group consisted of many linguists and computer scientists who considered this branch the beginning of artificial intelligence research. Stochastic researchers were more interested in statistical and probabilistic methods of NLP, working on problems of optical character recognition and pattern recognition between texts [1].

After 1970, researchers split even further, embracing new areas of NLP as more technology and knowledge became available. One new area was logic-based paradigms, languages that focused on encoding rules and language in mathematical logics. This area of NLP research later contributed to the development of the programming language Prolog. Natural language understanding was another area of NLP that was particularly influenced by SHRDLU, Professor Terry Winograd's doctoral thesis. This program placed a computer in a world of blocks, enabling it to manipulate and answer questions about the blocks according to natural language instructions from the user. The amazing part of this system was its capability to learn and understand with amazing accuracy, something only currently possible in extremely limited domains [1].

The computer is clearly able to resolve relationships between objects and understand certain ambiguities. A fourth area of NLP that came into existence after 1970 is discourse modeling. This area examines interchanges between people and computers, working out such ideas as the need to change "you" in a speaker's question to "me" in the computer's answer [1].

From 1983 to 1993, researchers became more united in focusing on empiricism and probabilistic models. Researchers were able to test certain arguments by Chomsky and others from the 1950s and 60s, discovering that many arguments that were convincing in text were not empirically accurate. Thus, by 1993, probabilistic and statistical methods of handling natural language processing were the most common types of models. In the last decade, NLP has also become more focused on information extraction and generation due to the vast amounts of information scattered across the Internet. Additionally, personal computers are now everywhere, and thus consumer level applications of NLP are much more common and an impetus for further research [1].

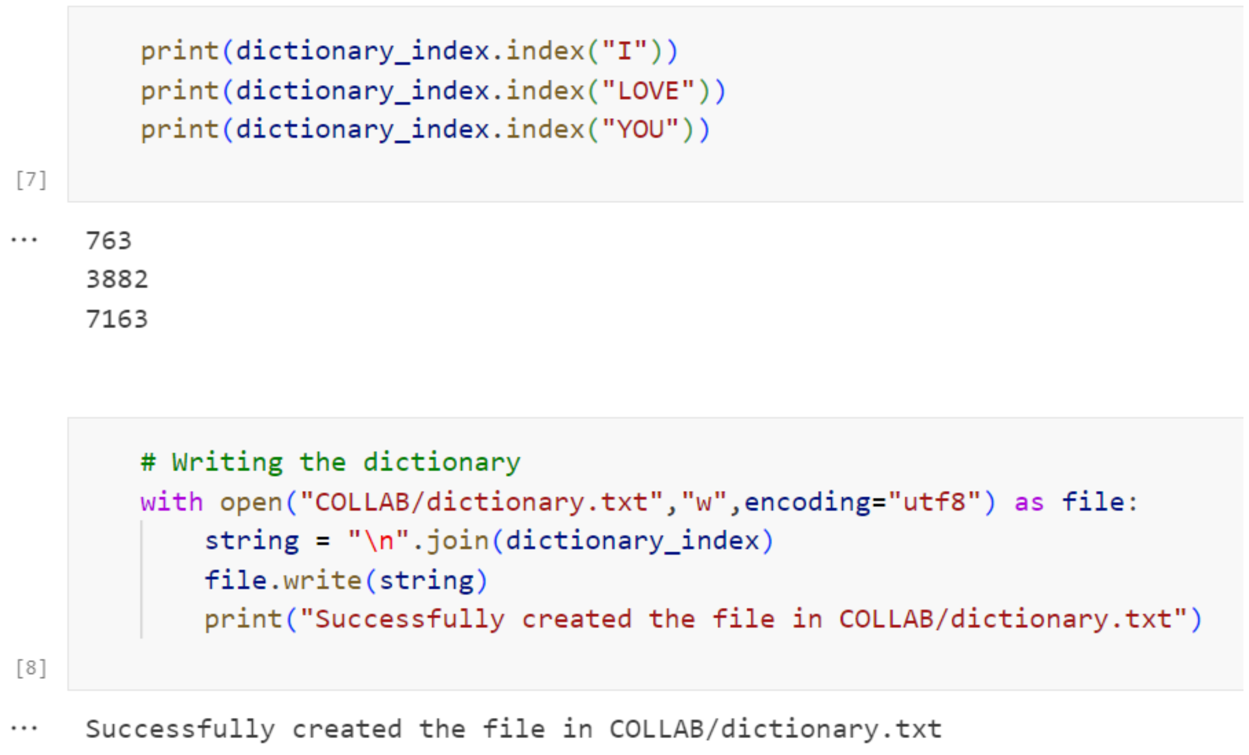
As of today, we can say that there are many AI models that use NLP. One of them, ChatGPT, does a good job creating text scripts. It can write a short film script in a second, but fails to provide longer film scripts. As a model that is not specified in film script writing, it does a pretty good job. If we look at the AI models that are specified in creating film scripts, NolanAI is the ultimate tool for any screenwriter looking to improve their craft. With its AI-powered features and user-friendly interface, it can help you bring your stories to life in the most efficient and effective way possible, while respecting your unique creative voice.

AI-based script creation offers many advantages, especially for low-budget independent films. It can speed up the script writing process, provide creative suggestions to screenwriters and improve story editing [2]. Some of the existing studies have encountered technical difficulties and have developed various approaches to overcome these difficulties. Issues such as data size, training data quality, model selection, and preserving language style are challenges to be considered in the process of making movie scripts with AI [3]. Some projects have developed customization techniques to tailor the text rendering model to suit a particular film genre or stylistic preferences.

Future research avenues in the field of film scripting with AI are highly relevant [4]. Existing studies still point to many open questions and potential improvements. Future research could address topics such as automatic generation of more complex stories, optimizing character development processes, enabling the language model to collaborate more efficiently with humans, and covering more film genres. Furthermore, ethical issues also play an important role. The process of making film scripts with AI, the automatic generation of texts and potential ethical issues may arise [5]. Therefore, future research will need to address ethical issues as well.

3. MATERIALS AND METHODS

Our data set consists of a combination of 2 large data sets. The first of our datasets contains 2858 different movie scripts and was retrieved from Kaggle [6]. Our other data set contains 1092 different movie scenarios and was taken from the OSF imsdb database [7]. First, words are tokenized and coded as integers. A .txt file called dictionary is created for unique words.



```
print(dictionary_index.index("I"))
print(dictionary_index.index("LOVE"))
print(dictionary_index.index("YOU"))
[7]
```

```
763
3882
7163
```

```
# Writing the dictionary
with open("COLLAB/dictionary.txt","w",encoding="utf8") as file:
    string = "\n".join(dictionary_index)
    file.write(string)
    print("Successfully created the file in COLLAB/dictionary.txt")
[8]
```

```
Successfully created the file in COLLAB/dictionary.txt
```

Image 1. Parsing of the words and creating the dictionary.

The words in the file will be separated according to their form and the list will be kept as such. For this project, the adventure genre was selected as the input and due to the length of the movies, only 3 movies would be processed. Python language is used during the training of this model. Keras and Numpy libraries will be used.

4. PROPOSED APPROACH

From the big dataset, 3 movies were selected from the Adventure genre. A 5-length window was hovered over the movie scripts and each window was taken as input. The word after the window will be the output of the artificial neural network. Thus, an attempt was made to guess which word would come after words.

Words encoded as Integers were made categorical using numpy. Then the inputs and outputs were split by 70/30 for training and testing. The last model created was trained using the following parameters:

num_films = 3

window_size = 5

batch_size = 16

opt = Adagrad

learning_rate = 0.01

And because of the Dense layer creating float32, the maximum number of predictions was selected as 1 and the others were selected as 0 for categorizing the output as int32 to process later.

5. TOOL

The tools used are:

1. Python 3.10.4
2. Google Colab Notebook*
3. Jupyter Notebook*
4. Keras for Python
    a. keras.layers.Dense
    b. keras.layers.LSTM
5. Numpy for Python
6. Sklearn for Python
7. Excel**

These tools must be installed to run the project on your machine.
* Installing Google Colab Notebook or Jupyter Notebook is enough
** Excel is not necessary to install

6. CODES

function1.py:

```
def helper(liste):

    # [3,4] -> [[3],[4]]

    return [[x] for x in liste]

def create_sublists(input_list, window_size):
    """

    Shifting a window through a list

    Window's size is determined by the window_size variable



    For example, if window size is 2:

    ["I","like","hiking","and","biking"]



    OUTPUT1[0] -> ['I', 'like', 'hiking']

    OUTPUT1[1] -> ['like', 'hiking', 'and']

    OUTPUT2[0] -> and

    OUTPUT2[1] -> biking

    """

    output1 = [input_list[i:i+window_size] for i in
range(len(input_list)-window_size+1)]

    output2 = input_list[window_size:]

    # Removing the last combination for output1, because the last
combination has no output.

    output1.pop(-1)

    return output1, output2

import numpy as np

def toInt(window:np.array):

    """

    This function converts dtype = "float32" array to dtype="int8" array,

    It gets the highest number and makes it 1, others 0
    As Dense layer is creating outputs in float32 type, this function
seemed to be necessary.

    """

    window = window.tolist()

    maximum = max(window)

    for i,j in enumerate(window):

        if j == maximum:

            window[i] = 1

        else:

            window[i] = 0

    window = np.asarray(window, dtype="int8")

    return window
```

convert.py:

```
def kaldir(string):

    """

    Removing the spaces in the script

    """

    bosluklar = ["  ","   ","    ","     ","      ","       ","        ","
","          "]

    string = string.replace("\n"," ")

    string = string.replace("\t","")


    for i in bosluklar:
        string = string.replace(i," ")

    return string
```

create_dictionary.ipynb:

```
from convert import *

import re

# Number of genres and films, which were 7 for genres and 10 for films at
the beginning.

num_genres = 1

num_films = 3

genres = list(range(num_genres))

films = list(range(num_films))

# List of the words used. This list will have the words unique to avoid
repetition

dictionary_index = []

# This block gets the words in each script, and adds them to the
dictionary_index

remove_list = ['-','','EXT.']

for i in genres:

    for j in films:


        # Reading scripts for each film

        with open(f"Films/{i}/{j}.txt","r",encoding="utf8") as file:

            string = kaldir(file.read())
```

```python
        string = string.upper()


        # Converting the string to the list and removing some of the expressions
        liste = string.split(" ")
        for element in remove_list:
            while element in liste:
                liste.remove(element)


        # Converting the list back to string and doing "is alpha" search.
        # Which means we are looking for words that containing alpha characters only
        # We use set() for removing repeated words
        liste = " ".join(liste)
        regex = r"\b[A-Za-z]+\b" # is alpha
        dictionary_index.extend(list(set(re.findall(regex,liste))))
        print(f"{i}/{j} done.")
# Adding some expressions to use later
# NOTE: These expressions weren't used because the model didn't work properly
dictionary_index = list(set(dictionary_index))
dictionary_index.extend(['.',',','?','!'])
# Writing the dictionary
with open("COLLAB/dictionary.txt","w",encoding="utf8") as file:
    string = "\n".join(dictionary_index)

    file.write(string)

    print("Successfully created the file in COLLAB/dictionary.txt")
```

database_normalization.ipynb:

```
from convert import *

from function1 import *

# Number of genres and films, which were 7 for genres and 10 for films at
the beginning.

num_genres = 1

num_films = 3

genres = list(range(num_genres))

films = list(range(num_films))

# Loading the dictionary


with open("COLLAB/dictionary.txt","r",encoding="utf8") as file:

    dictionary = [string.replace("\n","") for string in file.readlines()]

def encode_and_save(i,j):

    """

    This function opens the scripts,

    encodes each words according to their index in dictionary list

    """

    global films
```

```python
global genres

global dictionary


# Controlling if the film exists

if i not in genres or j not in films:

    raise IndexError("Genre of Film does not exist")


# Opening and reading the movies

with open(f"Films/{i}/{j}.txt","r",encoding="utf8") as file:

    string = kaldir(file.read())

    string = string.upper()


# Encoding the words

stringbuilder = ""

liste = string.split(" ")

for k in liste:

    if k in dictionary:

        stringbuilder += str(dictionary.index(k))

        stringbuilder += " "

    else:

        continue

liste2 = stringbuilder.split(" ")
```

```
    # Removing the blank character
    liste2.pop(-1)

    # Saving the encoded list in text file
    with open(f"COLLAB/{j}_enc.txt","w",encoding="utf8") as file:
        file.write("\n".join(liste2))
        print(f"Successfully created the file in COLLAB/{j}_enc.txt")

    return None

# Encoding and saving each film
for i in genres:
    for j in films:
        encode_and_save(i,j)
        print(f"{i}/{j} done.")
```

model_creation.ipynb:

```
from convert import *
from function1 import *
import numpy as np
from google.colab import drive
drive.mount('/content/drive')
from keras.models import Sequential
from keras.layers import Dense, Flatten, LSTM, Dropout, GRU

from keras.optimizers import Adam, SGD, Adagrad

from sklearn.model_selection import train_test_split

from keras.utils import to_categorical

film_sayisi = 3

window_size = 5

films = list(range(film_sayisi))

# Loading the dictionary

with open("drive/MyDrive/Colab Notebooks/PROJE/dictionary

.txt","r",encoding="utf8") as file:

    dictionary = [string.replace("\n","") for string in file.readlines()]

# Creating scripts[][]

scripts = []

for j in films:

    with open(f"drive/MyDrive/Colab Notebooks/PROJE/{j}_enc.txt","r") as
file:

        scripts.append(list(map(int,list(file.read().splitlines()))))

# Turning int to [int]

def func(liste):

    return [[x] for x in liste]

scripts = list(map(func,scripts))

# To categorical -> [0,0,0,...,1,...0,0,0]

for j in films:
    scripts[j] =
to_categorical(scripts[j],num_classes=len(dictionary)+1,dtype="byte")

# Creating inputs and outputs lists

inputs = []

outputs = []

for j in films:

    inputs.append(None)

    outputs.append(None)


# Creating sublists

for j in films:

    inputs[j],outputs[j] = create_sublists(scripts[j],window_size)


# Removing [films] layer in lists

inputs_temp = []

for j in films:

    for k in range(len(inputs[j])):

        inputs_temp.append(inputs[j][k])

inputs = inputs_temp

del inputs_temp

outputs_temp = []

for j in films:

    for k in range(len(outputs[j])):

        outputs_temp.append(outputs[j][k])
```

```
outputs = outputs_temp

del outputs_temp

# Creating train data and test data, x is for input and y is for output

x_train, x_test, y_train, y_test = train_test_split(inputs, outputs,
test_size=0.3, random_state=42)

x_train = np.asarray(x_train)

y_train = np.asarray(y_train)

x_test = np.asarray(x_test)

y_test = np.asarray(y_test)

# Creating model (ANN)

window_size = 5

model=Sequential([

    Flatten(input_shape=(window_size, len(x_train[0][0]),
1),dtype="int8"),

    Dense(len(x_train[0][0]), activation="relu"),

    Dense(len(x_train[0][0]), activation="softmax")

])


# Showing properties

model.summary()


# Setting the optimizer algorithm and compiling the model

opt = Adagrad(learning_rate = 0.01)
```

```
model.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy"])

print("Model ready")


# Training the model

history = model.fit(x_train, y_train, epochs=10, batch_size=16)

# Saving the model

model.save("drive/MyDrive/Colab Notebooks/Dense_10epochs.h5")

# Creating model (RNN)

model2 = Sequential()

model2.add(LSTM(units = 1000, input_shape = (window_size,
len(x_train[0][0])), return_sequences = True))

model2.add(Dropout(0.2))

model2.add(LSTM(units = 1000, return_sequences = True))

model2.add(Dropout(0.2))

model2.add(LSTM(units = 1000))

model2.add(Dropout(0.2))

model2.add(Dense(units = len(y_train[0])))


# Showing properties

model2.summary()


# Setting the optimizer algorithm and compiling the model

opt = Adagrad(learning_rate = 0.01)
```

```
model2.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy"])

print("Model2 ready")


# Training the model

history2 = model2.fit(x_train, y_train, epochs=10, batch_size=16)

# Saving the model

model2.save("drive/MyDrive/Colab Notebooks/LSTM_3x1000_10epochs.h5")
```

epoch_training.ipynb:

```
from convert import *

from function1 import *

import numpy as np

film_sayisi = 3

window_size = 5

films = list(range(film_sayisi))


# Loading the dictionary

with open("dictionary.txt","r",encoding="utf8") as file:

    dictionary = [string.replace("\n","") for string in file.readlines()]


# Loading model

from keras.models import load_model
from keras.optimizers import Adagrad

model = load_model("models/Dense_50epochs.h5",compile=False)

opt = Adagrad(learning_rate = 0.01)

model.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy"])

# Creating scripts[][]

scripts = []



for j in films:

    with open(f"{j}_enc.txt","r") as file:

        scripts.append(list(map(int,list(file.read().splitlines()))))



# Turning int to [int]

def func(liste):

    return [[x] for x in liste]

scripts = list(map(func,scripts))



# To categorical -> [0,0,0,...,1,...0,0,0]

from keras.utils import to_categorical

for j in films:

    scripts[j] =
to_categorical(scripts[j],num_classes=len(dictionary)+1,dtype="byte")



# Creating inputs and outputs lists
```

```
inputs = []
outputs = []
for j in films:
    inputs.append(None)
    outputs.append(None)

# Creating sublists
for j in films:
    inputs[j],outputs[j] = create_sublists(scripts[j],window_size)

# Removing [films] layer in lists
inputs_temp = []
for j in films:
    for k in range(len(inputs[j])):
        inputs_temp.append(inputs[j][k])
inputs = inputs_temp
del inputs_temp
outputs_temp = []
for j in films:
    for k in range(len(outputs[j])):
        outputs_temp.append(outputs[j][k])
outputs = outputs_temp
del outputs_temp
```

```
# Creating train data and test data, x is for input and y is for output

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(inputs, outputs,
test_size=0.3, random_state=42)


x_train = np.asarray(x_train)

y_train = np.asarray(y_train)

x_test = np.asarray(x_test)

y_test = np.asarray(y_test)

# Training the model

model.fit(x_test, y_test, epochs=20, batch_size=16)

# Saving the model

model.save("drive/MyDrive/Colab Notebooks/Dense_70epochs.h5")

# Model summary

model.summary()

# Testing the accuracy of model on both datasets

pred_output = model.predict(x_train)

pred_output2 = model.predict(x_test)

# Testing the model on train data

length = len(y_train)

counter = 0

for i in range(length):

    a = toInt(pred_output[i])
    b = y_train[i]

    if a.tolist() == b.tolist():

        counter+=1


accuracy = counter * 100 / length

print(f"Accuracy: %{accuracy}")


# Testing the model on test data

length = len(y_test)

counter = 0

for i in range(length):

    a = toInt(pred_output2[i])

    b = y_test[i]

    if a.tolist() == b.tolist():

        counter+=1


accuracy = counter * 100 / length

print(f"Accuracy: %{accuracy}")
```

model_usage_last.ipynb:

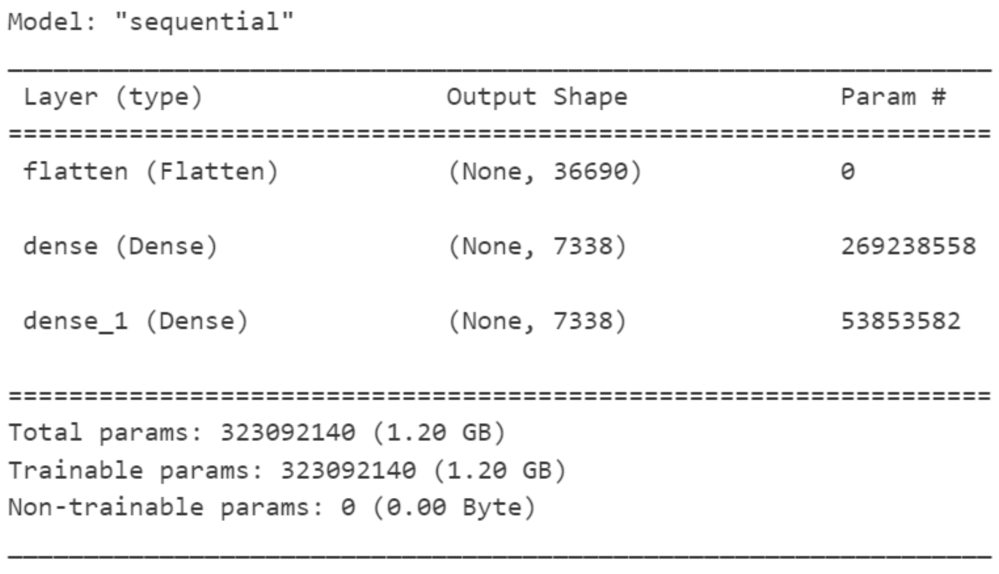
```
# Parameters, first_5 is a list containing first 5 words

string = "The fist is in a".upper()

first_5 = string.split(" ")
```

```
from keras.models import load_model

from keras.utils import to_categorical

import numpy as np

from function1 import *



# Loading the Dictionary

with open("dictionary.txt","r",encoding="utf8") as file:

    dictionary = [string.replace("\n","") for string in file.readlines()]

# The list containing all the words, generated words will be appended to
this list

words = []

words.extend(first_5)

# Getting indexes of the words

first_5 = [[dictionary.index(word)] for word in first_5]

# Converting it to categorical

categorical =
to_categorical(first_5,num_classes=len(dictionary)+1,dtype="byte")

# Loading the model

from keras.optimizers import Adagrad

model = load_model("models/Dense_70epochs.h5",compile=False)

opt = Adagrad(learning_rate = 0.01)

model.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy"])

def predict_a_word(inputs):
  """
  This model predicts a word according to the inputs
  Inputs is a list and it contains 5 strings
  """
  # Converting Strings to Categorical
  window = inputs.copy()
  window = [[dictionary.index(word)] for word in window]
  categorical =
to_categorical(window,num_classes=len(dictionary)+1,dtype="byte")


  # Predicting
  categorical = np.reshape(categorical,(1, 5, 7338, 1))
  pred = model.predict(categorical)
  pred = np.reshape(pred,(7338,1))
  pred = toInt(pred)


  # Getting the predicted words, which is only 1 word
  # The code was changed to see multiple outputs during the test
  # But this code works for 1 word also
  ones = []
  for i,j in enumerate(pred.tolist()):
    if j == 1:
      ones.append(i)
  new_words = [dictionary[ones[i]] for i in range(len(ones))]
  return new_words[0]


# Getting the predicted long string

length = 100

for i in range(length):

  print(i)

  input = words[-5:]

  new_word = predict_a_word(input)

  words.append(new_word)


# Getting the output

print(" ".join(words))
```

7. RESULTS

To see the difference, many models were trained during this project. Because the shapes of the categorical states of the input and outputs are very large, these models have become very large and take up a lot of space. After many attempts, it was decided to compare 2 models. The other models were lost due to overwriting, RAM crashes and COLAB crashes. The selected models are Dense (3 films) and LSTM (3 films). The number of films changes the length of the dictionary and the dictionary changes the input shape so it is critical to specify.

DENSE (ANN) MODEL



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 36690)             0

 dense (Dense)               (None, 7338)              269238558

 dense_1 (Dense)             (None, 7338)              53853582

=================================================================
Total params: 323092140 (1.20 GB)
Trainable params: 323092140 (1.20 GB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Image 2. Summary of the Dense model.

After 20 epochs the Dense model had the training accuracy of 20.09% and it was increasing in each epoch.

```
Epoch 18/20

2432/2432 [==============================] - 152s 62ms/step - loss: 5.4144
- accuracy: 0.1862

Epoch 19/00

2432/2432 [==============================] - 152s 62ms/step - loss: 5.3560
- accuracy: 0.1931

Epoch 20/20

2432/2432 [==============================] - 152s 62ms/step - loss: 5.2985
- accuracy: 0.2009
```

This model was trained for 50 epochs. And its training accuracy was still increasing each epoch.

```
Epoch 48/50

2432/2432 [==============================] - 157s 64ms/step - loss: 3.6158
- accuracy: 0.4329

Epoch 49/50

2432/2432 [==============================] - 156s 64ms/step - loss: 3.5586
- accuracy: 0.4394
```

```
Epoch 50/50

2432/2432 [==============================] - 156s 64ms/step - loss: 3.5010
- accuracy: 0.4487
```

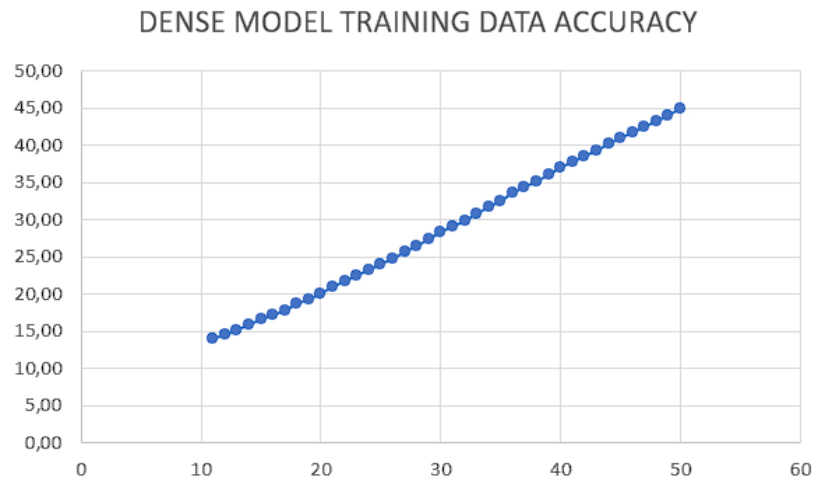
Though training accuracy was 44%, the test accuracy was only around 11%. So it was decided to feed the model with 20 epochs of test data. The model's accuracy for test data was increasing way faster than the training data. It is thought that this was due to the low number of data in the test dataset.

```
Epoch 1/20

1043/1043 [==============================] - 67s 65ms/step - loss: 6.0126
- accuracy: 0.1330

Epoch 10/20

1043/1043 [==============================] - 67s 64ms/step - loss: 4.6293
- accuracy: 0.2982

Epoch 20/20

1043/1043 [==============================] - 68s 65ms/step - loss: 3.8140
- accuracy: 0.4193
```

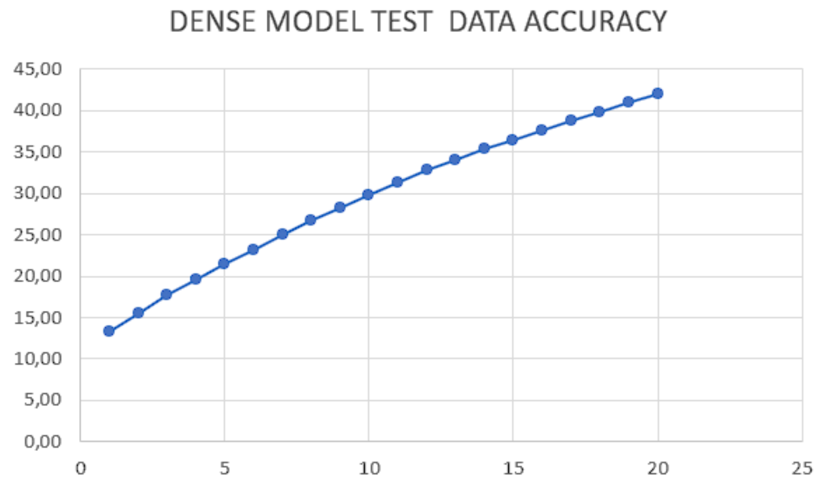
Then the model was tested to see their latest accuracy on both datasets:

```
Training Accuracy: %37.59

Test Accuracy: %43.27
```

It is seen that the accuracy of the training set was decreased, and even it was passed by the accuracy of the test set in a short time. But for all the training, the accuracy was increasing. **From the result, we can understand that training the model with test data caused a change in weights and decreased the accuracy of training data.**



Plot 1. Accuracy of training data for 50 epochs (first 10 epochs are lost due to overwriting)



Plot 2. Accuracy of test data for 20 epochs (trained after training for 50 epochs of training data)

The 20-epoch and the total amount of 70-epoch trained models were saved and used for creating a string. For the 20-epoch model, it started to repeat itself quickly. The result is shown below.

THE FIST IS IN A TYLER AND THE TYLER AND THE JACK LOOKS AT THE TYLER IS JACK JACK TYLER TYLER YOU JACK TYLER TYLER TYLER YOU JACK TYLER TYLER TYLER YOU JACK TYLER TYLER TYLER YOU JACK TYLER TYLER TYLER YOU JACK TYLER TYLER TYLER YOU JACK TYLER TYLER TYLER YOU JACK TYLER TYLER

Input string: THE FIST IS IN A
Output of the model Dense_20epochs
Length: 5 + 50 (generated)

The model was repeating itself and went in a loop. For the 70-epoch model, the result is shown below.

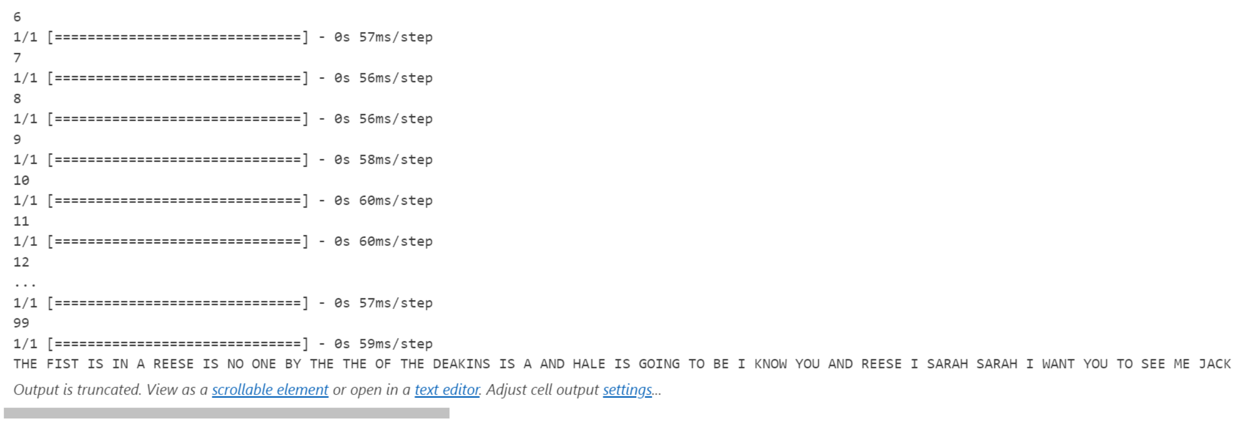
THE FIST IS IN A REESE IS NO ONE BY THE THE OF THE DEAKINS IS A AND HALE IS GOING TO BE I KNOW YOU AND REESE I SARAH SARAH I WANT YOU TO SEE ME JACK IF WE WERE A BUS JACK LOOKS UP TO JACK I WAS A LITTLE TYLER IS TYLER THIS I TYLER YOU JACK I COULD BE YOU THINK OF THE JACK I TYLER I WANT TO KNOW TYLER I WANT TO KNOW TYLER I WANT TO KNOW TYLER I WANT TO KNOW TYLER I WANT TO KNOW TYLER I WANT TO KNOW TYLER I WANT TO KNOW TYLER

Input string: THE FIST IS IN A
Output of the model Dense_70epochs
Length: 5 + 100 (generated)

This output can be converted to a script this way below:

THE FIST IS IN A... Reese is no one by the the of the, Deakins is a and Hale is going to be "I know you!" and
REESE: I, Sarah
SARAH: I want you to see me!
JACK: If we were a bus, (Jack looks up to Jack). I was a little Tyler is.
TYLER: This, I, Tyler, you, Jack!
JACK: I could be you! Think of the Jack, I !
TYLER: I want to know! (loops)

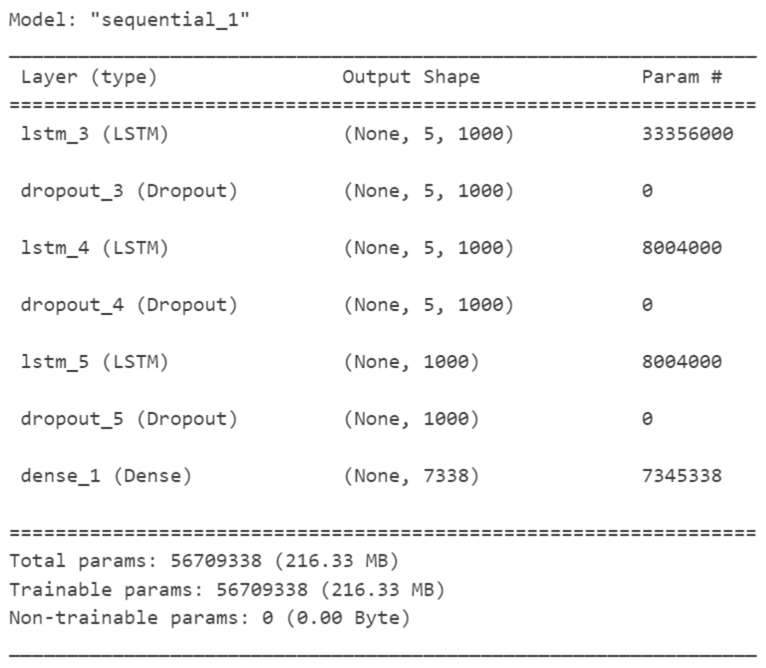
According to the outputs for the models trained for 20 epochs and 70 epochs, the difference is very clear.



```
6
1/1 [==============================] - 0s 57ms/step
7
1/1 [==============================] - 0s 56ms/step
8
1/1 [==============================] - 0s 56ms/step
9
1/1 [==============================] - 0s 58ms/step
10
1/1 [==============================] - 0s 60ms/step
11
1/1 [==============================] - 0s 60ms/step
12
...
1/1 [==============================] - 0s 57ms/step
99
1/1 [==============================] - 0s 59ms/step
THE FIST IS IN A REESE IS NO ONE BY THE THE OF THE DEAKINS IS A AND HALE IS GOING TO BE I KNOW YOU AND REESE I SARAH SARAH I WANT YOU TO SEE ME JACK
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Image 3. Last output of the Dense model

LSTM (RNN) MODEL



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 5, 1000)           33356000

 dropout_3 (Dropout)         (None, 5, 1000)           0

 lstm_4 (LSTM)               (None, 5, 1000)           8004000

 dropout_4 (Dropout)         (None, 5, 1000)           0

 lstm_5 (LSTM)               (None, 1000)              8004000

 dropout_5 (Dropout)         (None, 1000)              0

 dense_1 (Dense)             (None, 7338)              7345338

=================================================================
Total params: 56709338 (216.33 MB)
Trainable params: 56709338 (216.33 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Image 4. Summary of the LSTM Model

After 9 and 10 epochs the LSTM model had the training accuracy of 1.34% and it was fluctuating.

```
Epoch 6/10

2432/2432 [==============================] - 90s 37ms/step - loss: 10.1309
- accuracy: 0.0132

Epoch 7/10

2432/2432 [==============================] - 90s 37ms/step - loss: 9.9413
- accuracy: 0.0129

Epoch 8/10

2432/2432 [==============================] - 90s 37ms/step - loss: 9.8892
- accuracy: 0.0133

Epoch 9/10

2432/2432 [==============================] - 90s 37ms/step - loss: 10.1115
- accuracy: 0.0134

Epoch 10/10
```

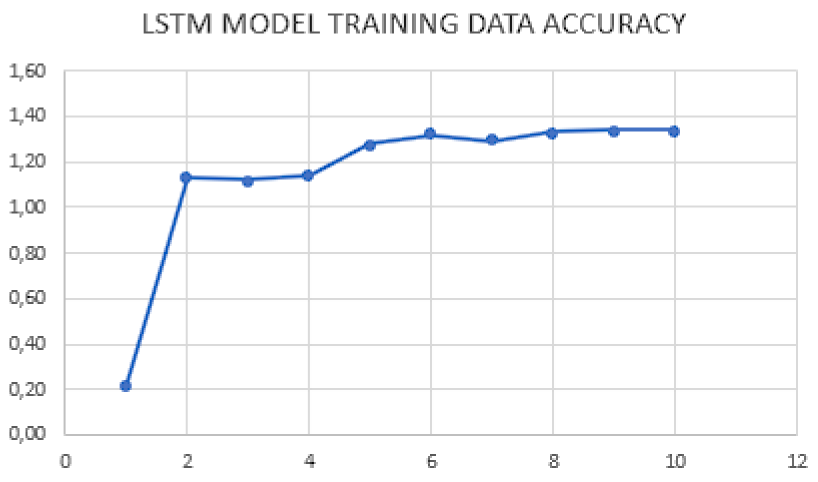
```
2432/2432 [==============================] - 90s 37ms/step - loss: 9.7534
- accuracy: 0.0134
```

This model only created a word and repeated that word. The result was shown below.

THE FIST IS IN A YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU YOU

Input string: THE FIST IS IN A TYLER
Output of the model LSTM_3x1000_10epochs
Length: 5 + 50 (generated)

After all, it is thought that this model can't learn. It is decided not to continue training this model. Because it is completely overshadowed by the Dense model. During all of our study, all of the Dense models had 7% to 9% accuracy after training for 5 to 10 epochs. And the accuracies of the Dense models were increasing in each epoch. But this isn't the case for the LSTM model.



Plot 3. Accuracy of training data for 10 epochs

8. DISCUSSION AND CONCLUSION

According to our study, the 70-epoch-trained ANN model seems promising and it might be better if trained for more epochs, with larger datasets. It should not be forgotten that the model was trained only using 3 movies, because of the RAM crashes when the length of the dictionary is large. And, the model was still learning for both training set and test set after 50 and 20 epochs separately.

9. REFERENCES

[1] Professor Eric Roberts, Sophomore College 2004: The Intellectual Excitement of Computer Science, Stanford University, 2004

[2] Smith, J. (2020). AI in Screenwriting: An Overview of Current Applications. Journal of Film Technology, 25(3), 45-58.

[3] Brown, A. et al. (2019). GPT-3: The Next Frontier in Natural Language Processing. AI Research Journal, 12(2), 123-137.

[4] Chen, L. (2018). Neural Storytelling: A Survey of Recent Advances in AI-Driven Narrative Generation. International Journal of Artificial Intelligence in Filmmaking, 5(1), 30-42.

[5] Johnson, M. (2021). Ethical Considerations in AI-Generated Film Scripts. Journal of AI Ethics, 8(4), 321-335.

[6] KseniaGur (2021), Movie Scripts Corpus, https://www.kaggle.com/datasets/gufukuro/movie-scripts-corpus/data

[7] Alberto Acerbi (2019), OSF Imsdb Movie Database, https://osf.io/zytmp

	YES / NO
Did you prepare your study both using a tool and writing code?	Yes
Did you prepare your report as mentioned in the template?	Yes
Did you add the results (print screen) of your study to the report?	Yes
Did you rename your report file as asked in the template?	Yes
Are you uploading the report to the system?	Yes
Are you uploading the codes to the system?	Yes