Preprocessing

Downloading the dataset from kaggle

```
!pip install -q kaggle
from google.colab import files
from IPython.display import clear_output

clear_output()
# Upload your kaggle.json file
files.upload()

!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d harisudhan411/phishing-and-legitimate-
urls
!unzip /content/phishing-and-legitimate-urls.zip -d new_data_urls

clear_output()
```

Tokenizing each URL, padding them to max length of 256, and splitting the data into train,test,validation sets.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import torch
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, GlobalMaxPooling1D,
Embedding, Dense, Dropout, Flatten

df = pd.read_csv('/content/new_data_urls/new_data_urls.csv')

tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(df['url'])
sequences = tokenizer.texts_to_sequences(df['url'])

x = pad_sequences(sequences, maxlen=256, padding='post',
truncating='post')
y = df['status'].values

x_train, x_temp, y_train, y_temp = train_test_split(x, y,
```

```
test_size=0.4, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp,
test_size=0.5, random_state=42)
```

Training

Using a CNN for efficient pattern recognition in the URLs. Training over 10 epochs.

```
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index)+1,
output_dim=50, input_length=256))
model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

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

history = model.fit(x_train, y_train, epochs=10,
validation_data=(x_val, y_val), batch_size=32)
```

```
Epoch 1/10
15413/15413 [==============================] - 116s 7ms/step - loss:
0.1609 - accuracy: 0.9418 - val_loss: 0.1131 - val_accuracy: 0.9596
Epoch 2/10
15413/15413 [==============================] - 91s 6ms/step - loss:
0.1157 - accuracy: 0.9597 - val_loss: 0.1033 - val_accuracy: 0.9624
Epoch 3/10
15413/15413 [==============================] - 83s 5ms/step - loss:
0.1057 - accuracy: 0.9633 - val_loss: 0.0960 - val_accuracy: 0.9655
Epoch 4/10
15413/15413 [==============================] - 83s 5ms/step - loss:
0.0995 - accuracy: 0.9656 - val_loss: 0.0935 - val_accuracy: 0.9665
Epoch 5/10
15413/15413 [==============================] - 82s 5ms/step - loss:
0.0956 - accuracy: 0.9668 - val_loss: 0.0910 - val_accuracy: 0.9681
Epoch 6/10
15413/15413 [==============================] - 81s 5ms/step - loss:
0.0925 - accuracy: 0.9681 - val_loss: 0.0914 - val_accuracy: 0.9687
Epoch 7/10
15413/15413 [==============================] - 104s 7ms/step - loss:
0.0893 - accuracy: 0.9690 - val_loss: 0.0890 - val_accuracy: 0.9698
Epoch 8/10
15413/15413 [==============================] - 85s 6ms/step - loss:
0.0870 - accuracy: 0.9696 - val_loss: 0.0890 - val_accuracy: 0.9698
```

```
Epoch 9/10
15413/15413 [==============================] - 82s 5ms/step - loss:
0.0854 - accuracy: 0.9703 - val_loss: 0.0879 - val_accuracy: 0.9701
Epoch 10/10
15413/15413 [==============================] - 84s 5ms/step - loss:
0.0843 - accuracy: 0.9708 - val_loss: 0.0888 - val_accuracy: 0.9704
```

Results

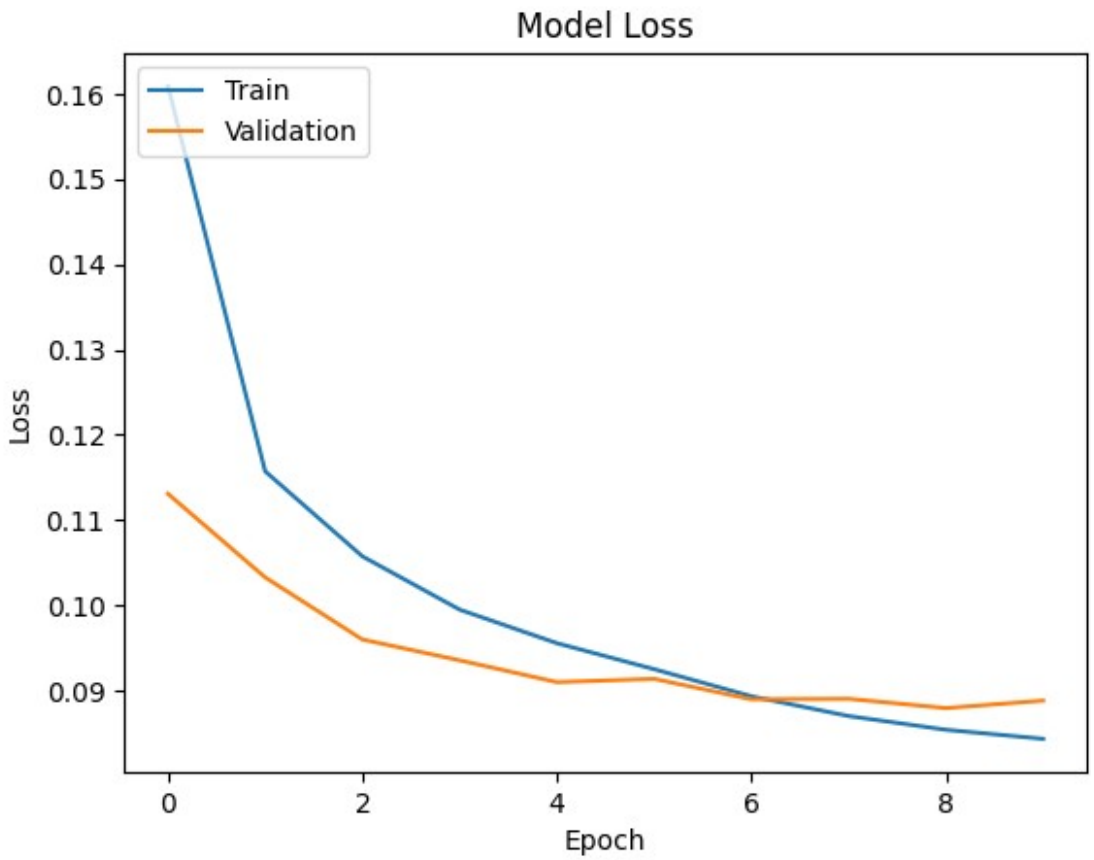
```
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import numpy as np

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')

loss, accuracy = model.evaluate(x_test, y_test)

clear_output()

plt.show()
print('Test Accuracy:', accuracy)
```




```
Test Accuracy: 0.9710465669631958
```

Downloading

Downloading the tokenizer dictionary

```
import json
from google.colab import files

word_index = tokenizer.word_index
with open('word_index.json', 'w') as f:
    json.dump(word_index, f)

files.download('word_index.json')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

Convert the model to tesnorflowjs compatible models

```
import keras
!pip install tensorflowjs
import tensorflowjs as tfjs
from pathlib import Path
from google.colab import drive
clear_output()
'''
**********************************************************************
**********************

If you get an error with jax import, go into jax_conversion.py then
delete the PolyShape declaration
and change the poly_shape import to from jax.experimental.jax2tf
import PolyShape
AND
delete the line that says PolyShape = shape_poly

**********************************************************************
********************** '''

# change this to the model you are saving
model.save('/content/isPhishing')

# first one is path to the .pb file, second one is the model path
!tensorflowjs_converter --input_format keras
/content/isPhishing/variables/saved_model.pb /content/isPhishing

def importModel():
    model = keras.models.load_model('/content/isPhishing') # where to
load the model from
    tfjs.converters.save_keras_model(model, "TFJSisPhishing") # where
to save the model to
importModel()

drive.mount('/content/drive')
# change the first path name to model you are saving
!cp -r '/content/TFJSisPhishing' '/content/drive/My Drive/models'
```