DEEP LEARNING AND AI

USING CNN TO FIND ACCURACIES FOR IMDB DATASET
USING AI TO DETERMINE DOGS IN THE IMAGE DATASET

1. Run CNN with IMDB Dataset in csv format

Step 1: Load IMDB Dataset

```
In [1]: import numpy as np # linear algebra
        import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
In [2]: df=pd.read_csv("../jupyter_notebook_project_1/DataUpload/IMDB Dataset.csv")
        df.head()
```

Out[2]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Step 2: Convert the sentiment value to integer value (0,1) for the algorithm to be able to process.

```
In [4]: # ----- Get labels -----
        y = np.int32(df.sentiment.astype('category').cat.codes.to_numpy())
        # ----- Get number of classes -----
        num_classes = np.unique(y).shape[0]
```

```
In [17]: print(y)

         [1 1 1 ... 0 0 0]
```

Step 3: Preprocessing data

We need to clean this data before we apply an algorithm. We need to:

- Remove br tag
- Remove all single characters
- Substituting multiple space with single space
- Remove prefixed “b”
- Converting to lowercase

```
In [5]: stemmer = WordNetLemmatizer()
        def custom_standardization(text):

            text = re.sub('<br />', ' ', str(text))

            text = re.sub(r'\W', ' ', str(text))

            # remove all single characters
            text = re.sub(r'\s+[a-zA-Z]\s+', ' ', text)

            # substituting multiple spaces with single space
            text = re.sub(r'\s+', ' ', text, flags=re.I)

            # removing prefixed 'b'
            text = re.sub(r'^b\s+', '', text)

            # converting to Lowercase
            text = text.lower()

            # lemmatization
            text = text.split()

            text = [stemmer.lemmatize(word) for word in text]
            text = ' '.join(text)

            return text
            pass
```

The following is the dataset after applied the processing above

```
In [6]: df['Cleaned_Text'] = df.review.apply(custom_standardization)

In [7]: df['Cleaned_Text'].head()

Out[7]: 0    one of the other reviewer ha mentioned that af...
        1    a wonderful little production the filming tech...
        2    i thought this wa wonderful way to spend time ...
        3    basically there a family where little boy jake...
        4    petter mattei love in the time of money is vis...
        Name: Cleaned_Text, dtype: object
```

Step 4: Before we use word embeddings, we need to convert our words into integers first. To do this we define the size of our vocabulary (max_features) and find the most occurring words in our data, to be used in the dictionary. After that, we use a tokenizer to represent each review as an integer vector corresponding to the dictionary. Tokenizer vectorize a text corpus into a list of integers. Each integer maps to a value in a dictionary that encodes the entire corpus, with the keys in the dictionary being the vocabulary terms themselves.

```
In [9]: # ----- Prepare text for embedding -----
        max_features = 10000
        # ----- Get top 10000 most occuring words in list-----
        results = Counter()
        df['Cleaned_Text'].str.split().apply(results.update)
        vocabulary = [key[0] for key in results.most_common(max_features)]

        # ----- Create tokenizer based on your top 10000 words -----
        tokenizer = Tokenizer(num_words=max_features)
        tokenizer.fit_on_texts(vocabulary)
```

Step 5: Convert our dataset into vector, and split them into train and split dataset

```
In [13]: # ----- Convert words to ints and pad -----
         X = tokenizer.texts_to_sequences(df['Cleaned_Text'].values)
         X = pad_sequences(X)


         # ----- Split into Train, Test, Validation sets -----
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
         X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

```
In [15]: X
```

```
Out[15]: array([[   0,    0,    0, ...,  126, 3881,  441],
                [   0,    0,    0, ..., 1923,   69,  223],
                [   0,    0,    0, ...,   61,   15,  206],
                ...,
                [   0,    0,    0, ..., 3699,    2, 5609],
                [   0,    0,    0, ...,   47,  739,   40],
                [   0,    0,    0, ...,  783,    8,   10]], dtype=int32)
```

Step 6: Define and Run CNN model

```
In [10]: # Define and train a model
         output_dim = 16
         max_input_lenght = X.shape[1]
```

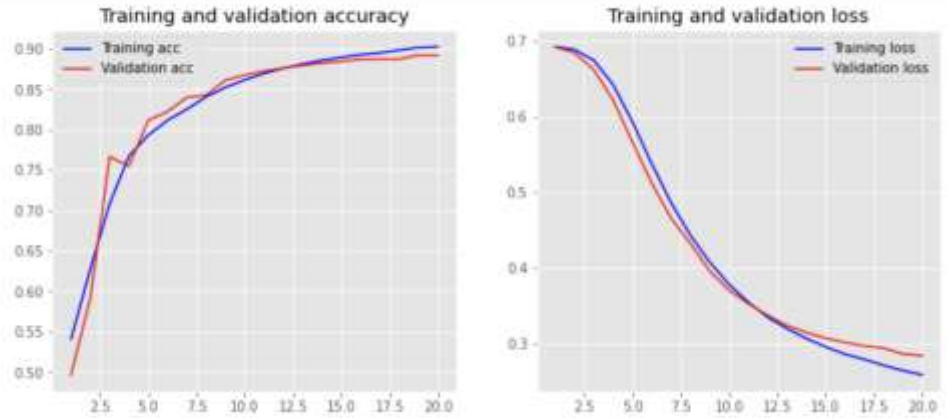
```
In [11]: # ----- Define model -----
         model = tf.keras.Sequential()
         model.add(tf.keras.layers.Embedding(input_dim=max_features,
                                             output_dim=output_dim, input_length=max_input_lenght))
         model.add(tf.keras.layers.Dropout(0.2))
         model.add(tf.keras.layers.GlobalAveragePooling1D())
         model.add(tf.keras.layers.Dense(16, activation='relu'))
         model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

         # ----- Compile model -----
         model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                       optimizer=tf.keras.optimizers.Adam(1e-4), metrics=["accuracy"])
```

Run model:

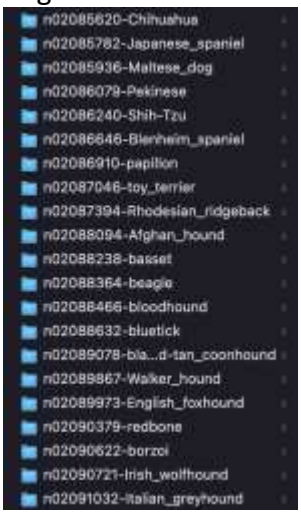
```
Epoch 17/20
4000/4000 [==============================] - 43s 11ms/step - loss: 0.2826 - accuracy: 0.8932 - val_loss: 0.2972 - val
_accuracy: 0.8875
Epoch 18/20
4000/4000 [==============================] - 44s 11ms/step - loss: 0.2749 - accuracy: 0.8968 - val_loss: 0.2944 - val
_accuracy: 0.8875
Epoch 19/20
4000/4000 [==============================] - 42s 11ms/step - loss: 0.2678 - accuracy: 0.8987 - val_loss: 0.2866 - val
_accuracy: 0.8923
Epoch 20/20
4000/4000 [==============================] - 42s 10ms/step - loss: 0.2651 - accuracy: 0.9014 - val_loss: 0.2843 - val
_accuracy: 0.8921
```

Step 7: Evaluate model:

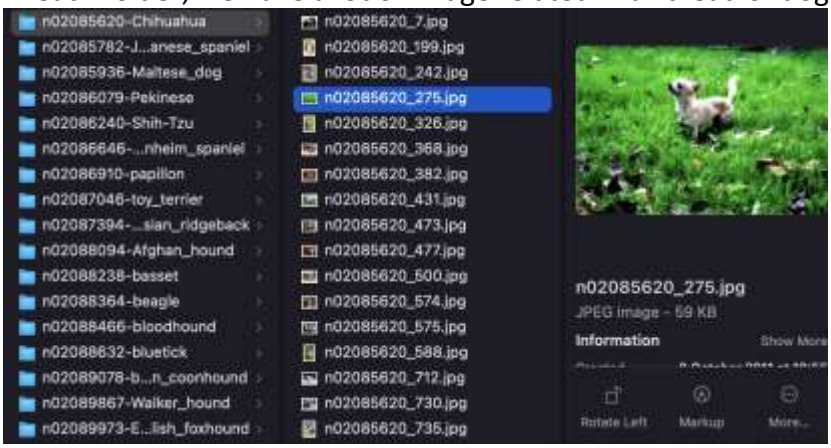
```
In [16]: plot_history(history_1)
```



2. Apply the final project for another dataset.
   In this project, we use the collection of the of the dog. With 120 folders of kind of the dog. And we tried to use CNN to classify and predict this collection.



   In each folder, we have a lot of image related with bread of dog.



   We use the model in the previous project to try to classify the bread of dog. And the training give the good accuracy with 99%

```
Epoch 48/99
----------
train Loss: 0.4465 Acc: 0.8654
val Loss: 0.0342 Acc: 0.9928

Epoch 49/99
----------
train Loss: 0.4692 Acc: 0.8510
val Loss: 0.0319 Acc: 0.9928

Epoch 50/99
----------
train Loss: 0.4438 Acc: 0.8611
val Loss: 0.0310 Acc: 0.9971
```

The following picture is some predicted by this model:

predicted: n02085620-Chihuahua

predicted: n02085782-Japanese_spaniel

predicted: n02085782-Japanese_spaniel

predicted: n02086646-Blenheim_spaniel

predicted: n02088094-Afghan_hound