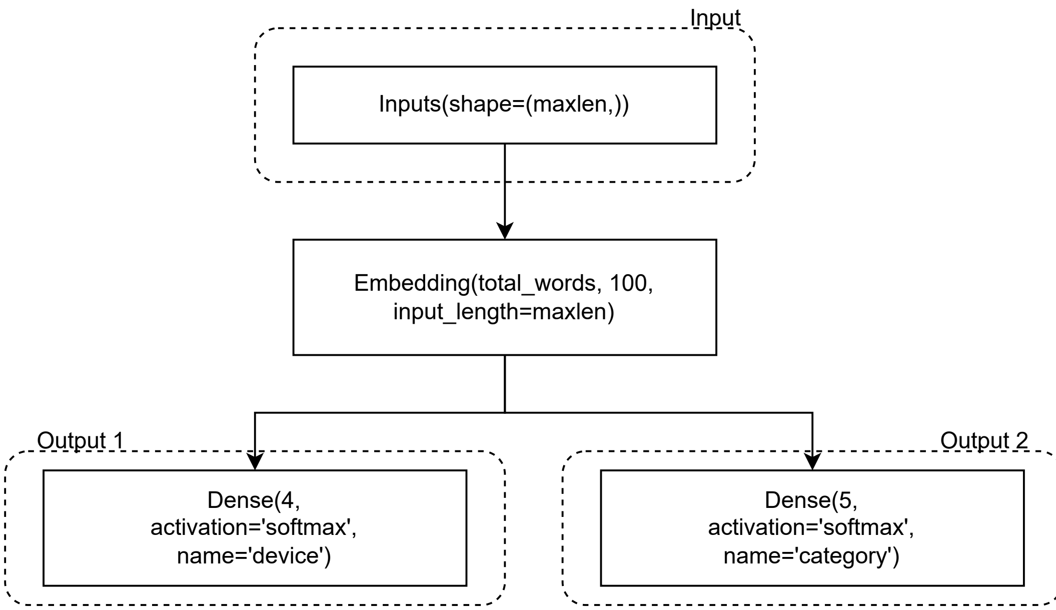
MODEL ARCHITECTURE



[1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/google-devices-q-and-a-dataset/All About Google Devices.xlsx
/kaggle/input/google-devices-q-and-a-dataset/All About Google Devices - Sheet1.csv
```

Open the dataframe

[2]:

```
df = pd.read_csv('/kaggle/input/google-devices-q-and-a-dataset/All About Google Devices - Sheet1.csv')
df.head()
```

[2]:

	Question	Device	Category
0	Can I use my Pixel as a hotspot to share inter...	Pixel Phones	Connectivity
1	What are the pros and cons of using Chrome OS ...	Chromebooks	Operating System
2	How can I set up parental controls on Google A...	Google Assistant	Security
3	Does the Nest Doorbell offer night vision reco...	Nest Devices	Security
4	Can I customize the ambient light color and in...	Google Home	Personalization

[3]:

```
df['Device'].value_counts()
```

[3]:
```
df['Device'].value_counts()
```

[3]:
```
Device
Google Assistant & Nest Devices    70
Google Wifi                        59
Nest Devices & Security            48
Pixel Fold                         48
Pixel Phones                       47
                                   ..
Chromebooks & Travel                1
Pixel Phones & Home Care            1
Pixel Phones & Arts & Culture       1
Nest Devices & Smart Home           1
Pixel Phones & Home Improvement     1
Name: count, Length: 132, dtype: int64
```

Cleaning the feature

[4]:
```
import re
df['Question'] = df['Question'].apply(lambda x: x.lower())
df['Question'] = df['Question'].apply(lambda x: re.findall(r'[\w]+', x))
```

[5]:
```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

stop_words = set(stopwords.words('english'))
ps = PorterStemmer()
df['Question'] = df['Question'].apply(lambda x: [i for i in x if i not in stop_words])
df['Question'] = df['Question'].apply(lambda x: [ps.stem(i) for i in x])
df['Question'] = df['Question'].apply(lambda x: " ".join(x))
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

[6]:
```
label = {'Device':{},
         'Category':{}}
```

[6]:
```
label = {'Device':{},
         'Category':{}}

label1_index = df['Device'].value_counts().index
label1_values = df['Device'].value_counts().values

for index, values in zip(label1_index, label1_values):
    label['Device'][index] = values

label2_index = df['Category'].value_counts().index
label2_values = df['Category'].value_counts().values

for index, values in zip(label2_index, label2_values):
    label['Category'][index] = values
```

[7]:
```
df['Device Counts'] = df['Device'].apply(lambda x: label['Device'][x])
df['Category Counts'] = df['Category'].apply(lambda x: label['Category'][x])
```

[8]:
```
df = df[df['Device Counts'] > 25]
df = df[df['Category Counts'] > 15]
```

Preprocess the label

[9]:
```
target = {'Device':{},
          'Category':{}}

index = 0
for i in df['Device'].drop_duplicates():
    target['Device'][i] = index
    index += 1

index = 0
for i in df['Category'].drop_duplicates():
    target['Category'][i] = index
```

Preprocess the label

[9]:

```
target = {'Device':{},
          'Category':{}}

index = 0
for i in df['Device'].drop_duplicates():
    target['Device'][i] = index
    index += 1

index = 0
for i in df['Category'].drop_duplicates():
    target['Category'][i] = index
    index += 1

df['Device'] = df['Device'].replace(target['Device'])
df['Category'] = df['Category'].replace(target['Category'])
```

[10]:

```
target['Category']
```

[10]:

```
{'Smart Home & Sustainability': 0,
 'Communication & Convenience': 1,
 'Connectivity & Guest Management': 2,
 'Connectivity & Digital Wellbeing': 3,
 'Communication & Accessibility': 4}
```

Split the data

[11]:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(df['Question'], df[['Device', 'Category']],
                                                    random_state=20,
                                                    test_size=0.2)
y_train, z_train = y_train['Device'], y_train['Category']
y_test, z_test = y_test['Device'], y_test['Category']
```

Split the data

[11]:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(df['Question'], df[['Device', 'Category']],
                                                    random_state=20,
                                                    test_size=0.2)
y_train, z_train = y_train['Device'], y_train['Category']
y_test, z_test = y_test['Device'], y_test['Category']
```

+ Code + Markdown

Preprocess the feature

[12]:

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(oov_token='<OOV>')
tokenizer.fit_on_texts(x_train)

train_sequences = tokenizer.texts_to_sequences(x_train)
test_sequences = tokenizer.texts_to_sequences(x_test)

total_words = len(tokenizer.word_index) + 1
maxlen = max([len(x) for x in train_sequences])

train_sequences = pad_sequences(train_sequences, maxlen=maxlen)
test_sequences = pad_sequences(test_sequences, maxlen=maxlen)
```

Prepare the model

[28]:

```
import tensorflow as tf
```

Prepare the model

[28]:

```
import tensorflow as tf

inputs = tf.keras.Input(shape=(maxlen,))
embedding = tf.keras.layers.Embedding(total_words, 100, input_length=maxlen)(inputs)
x = tf.keras.layers.GlobalAveragePooling1D()(embedding)
x = tf.keras.layers.Dense(64, activation='relu')(x)
out1 = tf.keras.layers.Dense(4, activation='softmax', name='device')(x)
out2 = tf.keras.layers.Dense(5, activation='softmax', name='category')(x)
model = tf.keras.Model(inputs=inputs, outputs=[out1, out2])

model.compile(loss={'device':'sparse_categorical_crossentropy',
                    'category':'sparse_categorical_crossentropy'},
              optimizer='adam',
              metrics={'device':'accuracy',
                       'category':'accuracy'})
```

[29]:

```
class myCallback(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs={}):
        if logs.get('loss') < 0.05:
            self.model.stop_training = True
```

[30]:

```
callback = myCallback()
history = model.fit(x=train_sequences,
                    y=[y_train, z_train],
                    epochs=100,
                    validation_data=(test_sequences, [y_test, z_test]),
                    callbacks=callback)
```

```
Epoch 1/100
4/4 [==============================] - 2s 263ms/step - loss: 2.9896 - device_loss: 1.3829 - category_loss: 1.6067 - device_accuracy: 0.3235 - category_accuracy: 0.4118 - val_lo
ss: 2.9703 - val_device_loss: 1.3754 - val_category_loss: 1.5948 - val_device_accuracy: 0.4231 - val_category_accuracy: 0.7308
Epoch 2/100
4/4 [==============================] - 1s 169ms/step - loss: 2.9626 - device_loss: 1.3698 - category_loss: 1.5928 - device_accuracy: 0.5392 - category_accuracy: 0.6863 - val_lo
ss: 2.9427 - val_device_loss: 1.3632 - val_category_loss: 1.5795 - val_device_accuracy: 0.4615 - val_category_accuracy: 0.8846
Epoch 3/100
4/4 [==============================] - 1s 191ms/step - loss: 2.9298 - device_loss: 1.3524 - category_loss: 1.5775 - device_accuracy: 0.6275 - category_accuracy: 0.7255 - val_lo
```

[30]:

```
callback = myCallback()
history = model.fit(x=train_sequences,
                    y=[y_train, z_train],
                    epochs=100,
                    validation_data=(test_sequences, [y_test, z_test]),
                    callbacks=callback)
```

```
Epoch 1/100
4/4 [==============================] - 2s 263ms/step - loss: 2.9896 - device_loss: 1.3829 - category_loss: 1.6067 - device_accuracy: 0.3235 - category_accuracy: 0.4118 - val_lo
ss: 2.9703 - val_device_loss: 1.3754 - val_category_loss: 1.5948 - val_device_accuracy: 0.4231 - val_category_accuracy: 0.7308
Epoch 2/100
4/4 [==============================] - 1s 169ms/step - loss: 2.9626 - device_loss: 1.3698 - category_loss: 1.5928 - device_accuracy: 0.5392 - category_accuracy: 0.6863 - val_lo
ss: 2.9427 - val_device_loss: 1.3632 - val_category_loss: 1.5795 - val_device_accuracy: 0.4615 - val_category_accuracy: 0.8846
Epoch 3/100
4/4 [==============================] - 1s 191ms/step - loss: 2.9298 - device_loss: 1.3524 - category_loss: 1.5775 - device_accuracy: 0.6275 - category_accuracy: 0.7255 - val_lo
ss: 2.9084 - val_device_loss: 1.3469 - val_category_loss: 1.5614 - val_device_accuracy: 0.4615 - val_category_accuracy: 0.8846
Epoch 4/100
4/4 [==============================] - 0s 145ms/step - loss: 2.8897 - device_loss: 1.3303 - category_loss: 1.5594 - device_accuracy: 0.6275 - category_accuracy: 0.8039 - val_lo
ss: 2.8680 - val_device_loss: 1.3276 - val_category_loss: 1.5404 - val_device_accuracy: 0.4615 - val_category_accuracy: 0.8846
Epoch 5/100
4/4 [==============================] - 1s 222ms/step - loss: 2.8411 - device_loss: 1.3033 - category_loss: 1.5378 - device_accuracy: 0.6275 - category_accuracy: 0.8627 - val_lo
ss: 2.8210 - val_device_loss: 1.3052 - val_category_loss: 1.5158 - val_device_accuracy: 0.4615 - val_category_accuracy: 0.9231
Epoch 6/100
4/4 [==============================] - 0s 83ms/step - loss: 2.7834 - device_loss: 1.2714 - category_loss: 1.5120 - device_accuracy: 0.6569 - category_accuracy: 0.8922 - val_los
s: 2.7655 - val_device_loss: 1.2783 - val_category_loss: 1.4871 - val_device_accuracy: 0.4615 - val_category_accuracy: 0.8846
Epoch 7/100
4/4 [==============================] - 1s 211ms/step - loss: 2.7152 - device_loss: 1.2341 - category_loss: 1.4810 - device_accuracy: 0.6961 - category_accuracy: 0.9118 - val_lo
ss: 2.6982 - val_device_loss: 1.2456 - val_category_loss: 1.4526 - val_device_accuracy: 0.6538 - val_category_accuracy: 0.8846
Epoch 8/100
4/4 [==============================] - 1s 124ms/step - loss: 2.6354 - device_loss: 1.1905 - category_loss: 1.4449 - device_accuracy: 0.7843 - category_accuracy: 0.9314 - val_lo
ss: 2.6207 - val_device_loss: 1.2079 - val_category_loss: 1.4128 - val_device_accuracy: 0.8077 - val_category_accuracy: 0.8846
Epoch 9/100
4/4 [==============================] - 1s 104ms/step - loss: 2.5421 - device_loss: 1.1401 - category_loss: 1.4021 - device_accuracy: 0.8824 - category_accuracy: 0.9412 - val_lo
ss: 2.5300 - val_device_loss: 1.1635 - val_category_loss: 1.3665 - val_device_accuracy: 0.8462 - val_category_accuracy: 0.8846
Epoch 10/100
4/4 [==============================] - 0s 15ms/step - loss: 2.4348 - device_loss: 1.0823 - category_loss: 1.3525 - device_accuracy: 0.9118 - category_accuracy: 0.9510 - val_los
s: 2.4251 - val_device_loss: 1.1120 - val_category_loss: 1.3130 - val_device_accuracy: 0.8846 - val_category_accuracy: 0.9231
Epoch 11/100
4/4 [==============================] - 1s 126ms/step - loss: 2.3108 - device_loss: 1.0170 - category_loss: 1.2939 - device_accuracy: 0.9216 - category_accuracy: 0.9608 - val_lo
ss: 2.3091 - val_device_loss: 1.0554 - val_category_loss: 1.2537 - val_device_accuracy: 0.9231 - val_category_accuracy: 0.9231
Epoch 12/100
4/4 [==============================] - 0s 78ms/step - loss: 2.1737 - device_loss: 0.9449 - category_loss: 1.2288 - device_accuracy: 0.9412 - category_accuracy: 0.9706 - val_los
s: 2.1834 - val_device_loss: 0.9959 - val_category_loss: 1.1875 - val_device_accuracy: 0.9231 - val_category_accuracy: 0.9231
Epoch 13/100
4/4 [==============================] - 1s 191ms/step - loss: 2.0285 - device_loss: 0.8707 - category_loss: 1.1578 - device_accuracy: 0.9608 - category_accuracy: 0.9608 - val_lo
ss: 2.0532 - val_device_loss: 0.9352 - val_category_loss: 1.1180 - val_device_accuracy: 0.9231 - val_category_accuracy: 0.9231
Epoch 14/100
4/4 [==============================] - 0s 80ms/step - loss: 1.8729 - device_loss: 0.7927 - category_loss: 1.0802 - device_accuracy: 0.9608 - category_accuracy: 0.9608 - val_los
s: 1.9079 - val_device_loss: 0.8682 - val_category_loss: 1.0397 - val_device_accuracy: 0.9231 - val_category_accuracy: 0.8846
Epoch 15/100
4/4 [==============================] - 0s 145ms/step - loss: 1.7068 - device_loss: 0.7112 - category_loss: 0.9955 - device_accuracy: 0.9706 - category_accuracy: 0.9510 - val_lo
ss: 1.7475 - val_device_loss: 0.7937 - val_category_loss: 0.9538 - val_device_accuracy: 0.9231 - val_category_accuracy: 0.8846
Epoch 16/100
4/4 [==============================] - 0s 15ms/step - loss: 1.5379 - device_loss: 0.6306 - category_loss: 0.9073 - device_accuracy: 0.9706 - category_accuracy: 0.9510 - val_los
s: 1.5808 - val_device_loss: 0.7165 - val_category_loss: 0.8643 - val_device_accuracy: 0.9231 - val_category_accuracy: 0.9231
Epoch 17/100
4/4 [==============================] - 0s 91ms/step - loss: 1.3690 - device_loss: 0.5519 - category_loss: 0.8172 - device_accuracy: 0.9706 - category_accuracy: 0.9608 - val_los
s: 1.4227 - val_device_loss: 0.6444 - val_category_loss: 0.7783 - val_device_accuracy: 0.9231 - val_category_accuracy: 0.9231
```

NLP Next Level Draft saved

File Edit View Run Add-ons Help

Run All

Save Version

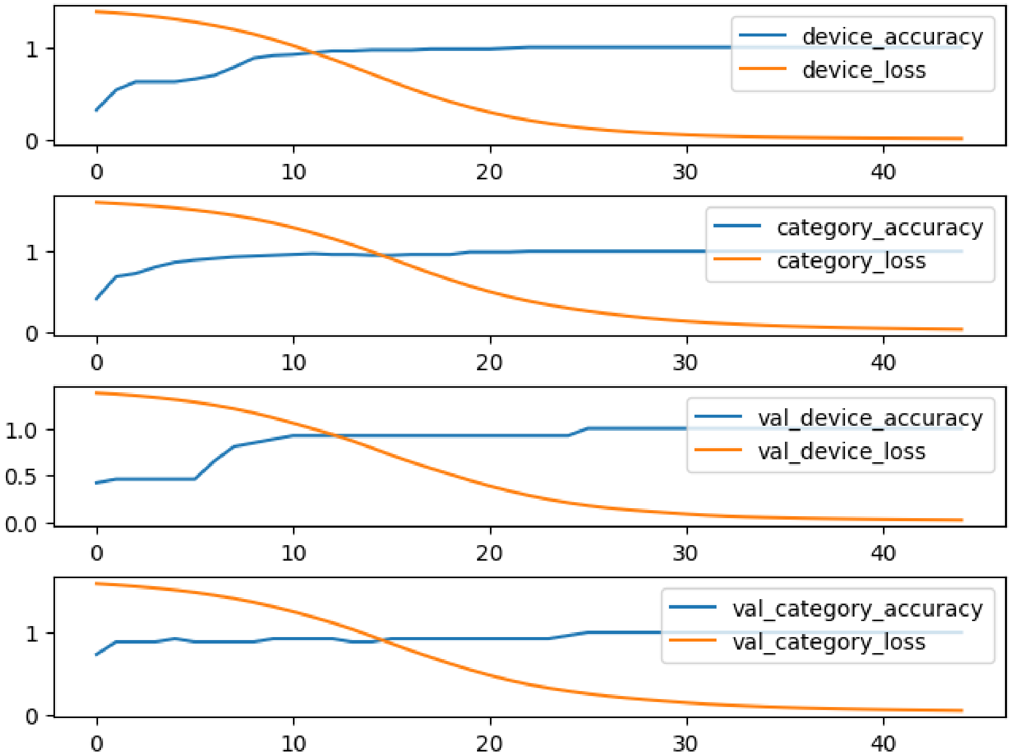
Plot the training result

```
import matplotlib.pyplot as plt

plot_names = [['device_accuracy', 'device_loss'],
              ['category_accuracy', 'category_loss'],
              ['val_device_accuracy', 'val_device_loss'],
              ['val_category_accuracy', 'val_category_loss'

fig, axs = plt.subplots(4, layout='constrained')

for i, j in zip(range(4), plot_names):
    axs[i].plot(history.history[j[0]])
    axs[i].plot(history.history[j[1]])
    axs[i].legend([j[0], j[1]], loc='upper right')
```



+ Code + Markdown

[44]: