# Deep Learning And Artificial Intelligence with TensorFlow Final Project

## Topic: Tagging of Stack Overflow questions using Deep Learning

---

**Project team:**

Dhaval Khamar

Fernando Avalos Garcia

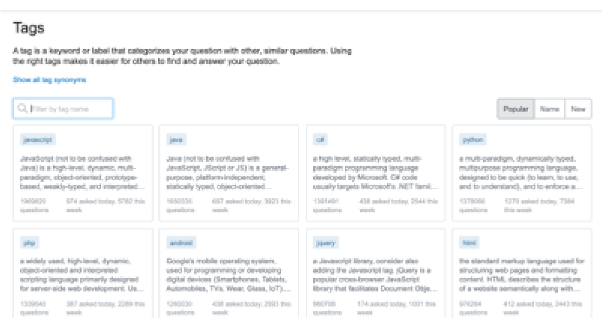
Susan Thomas

# Introduction

Stack Overflow is a question and answer site for professional and enthusiat programmers on a wide range of topics on computer progra
As mentioned on their website, Stack Overflow has over 16.5 million questions and is a great resource for programmers.

Every question on Stack Overflow is marked with relevant tags. There is usually atleast one tag per post.


Tags are useful for users to find similar questions and for programming experts to find and answer open questions. Below are the curren
popular tags on Stack Overflow.




# Problem statement

To reduce the effort for Stack Overflow's content managers to manually tag each question, Deep Learning can be utilized to automaticall
new questions.

**Dataset**

The dataset used in this project was sourced from Stack Exchange.

[License]()

[Dataset]()

The dataset contains 76364 Stack Overflow questions, answers and tags. Each question has between 1 and 5 tags. There are 100 unique
in the dataset.

# Upgrade libraries to latest versions

```
!pip install -U tables
!pip install -U numpy
```

```
Requirement already up-to-date: tables in /usr/local/lib/python3.6/dist-packages (3.6.1)
Requirement already satisfied, skipping upgrade: numexpr>=2.6.2 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied, skipping upgrade: numpy>=1.9.3 in /usr/local/lib/python3.6/dist-packages (from
Requirement already up-to-date: numpy in /usr/local/lib/python3.6/dist-packages (1.18.2)
```

## Importing libraries

```
import pandas as pd
import tensorflow as tf
import numpy as np
import matplotlib as plt
import re
import os
import datetime
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from sklearn.metrics import f1_score, precision_score, recall_score, confusion_matrix
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split

# Load the TensorBoard notebook extension
%load_ext tensorboard
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# Mount Google drive to Colab to use dataset
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

```
# Hyperparameters

EMBEDDING_DIM = 256
MAX_WORDS = 500
BATCH_SIZE = 512
EPOCHS = 20
RNN_EPOCHS = 100
THRESHOLD = 0.5
```

```
df = pd.read_hdf('/content/gdrive/My Drive/Colab Notebooks/TensorFlow project/auto_tagging_data_v2.h5')
```

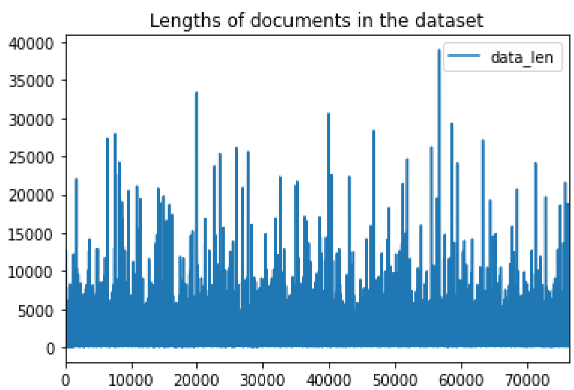
```
df.head()
```

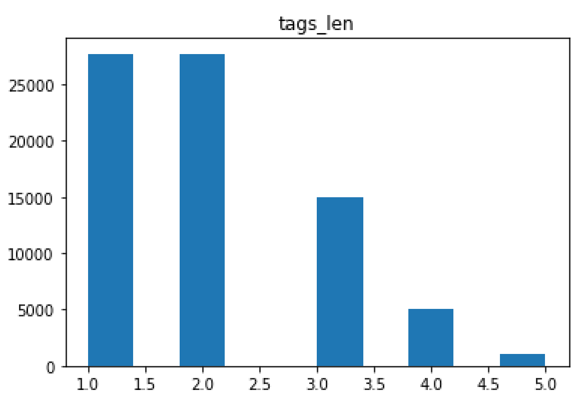
| | Id | Title | Body | Tags |
|---|---|---|---|---|
| **0** | 6 | The Two Cultures: statistics vs. machine learn... | <p>Last year, I read a blog post from <a href=... | [machine-learning] |
| **1** | 21 | Forecasting demographic census | <p>What are some of the ways to forecast demog... | [forecasting] |
| **2** | 22 | Bayesian and frequentist reasoning in plain En... | <p>How would you describe in plain English the... | [bayesian] |
| **3** | 31 | What is the meaning of p | <p>After taking a statistics | [hypothesis-testing, t-test, |

```
# Combining title and body of question to create data 'documents'
df['data'] = df['Title'] + ' ' + df['Body']
```

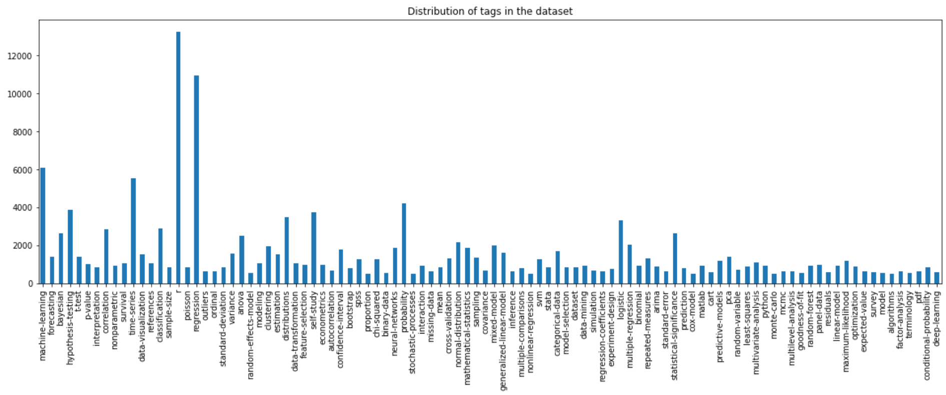
## Dataset Exploration

```
# Getting lengths of each document (question + answer) and number of tags per document
df['data_len'] = df['data'].str.len()
df['tags_len'] = df['Tags'].str.len()

plt1 = df.plot(y=['data_len'], title='Lengths of documents in the dataset')

plt2 = df.hist(column=['tags_len'], grid=False)
```




```
# Finding distribution of tags in the dataset
all_tags = df['Tags'].values.tolist()
tags_count = {}
for tags in all_tags:
  for t in tags:
    if t in tags_count.keys():
      tags_count[t] += 1
    else:
      tags_count.update({t: 0})
tags_count

df_tags = pd.DataFrame.from_dict(tags_count, orient='index')
plt3 = df_tags.plot(kind='bar', figsize=(20,6), legend=False, title='Distribution of tags in the dataset')
```




# Data preprocessing

```
STOPWORDS = set(stopwords.words("english"))

def clean_text(data):
  # Convert to lowercase
  data = data.lower()

  # Remove HTML tags
  tags = re.compile(r'<[^>]+>')
  data = re.sub(tags, '', data)

  # Remove string control characters
  data = re.sub(r'[\n\t\']', ' ', data)

  # Remove punctuation
  data = re.sub(r'[^a-z]', ' ', data)

  # Remove stopwords
  data = ' '.join(word for word in data.split() if word not in STOPWORDS)

  # Remove words of length 2
  data = ' '.join(word for word in data.split() if len(word) > 2)

  # Remove whitespaces
  data = ' '.join(data.split())

  return data

df['data_clean'] = df['data'].apply(lambda x: clean_text(x))


df['data_clean']
```

```
0        two cultures statistics machine learning last ...
1        forecasting demographic census ways forecast d...
2        bayesian frequentist reasoning plain english w...
3        meaning values values statistical tests taking...
4        examples teaching correlation mean causation o...
                                ...
76360    interpretation global test value interaction t...
76361    testing linear model used fit linear model dat...
76362    know simple validation result statistically si...
76363    kendall conditional independence test statisti...
76364    heteroskedasticity regression model testing he...
Name: data_clean, Length: 76365, dtype: object
```

## Word tokenization and Encoding

Tokenizer is a utility in the Keras module to encode text documents into integers. It assigns a number to each unique word in the dataset converts words to their integer form.

For example:

Document 1: I have a dog

Document 2: My dog has a bone

```
docs = ['i have a dog', 'my dog has a bone']
example = pd.DataFrame(docs, columns=['text'])
t = Tokenizer()
t.fit_on_texts(example['text'])
encoded = t.texts_to_sequences(example['text'])
print(t.word_index)
print(encoded)
```

```
{'a': 1, 'dog': 2, 'i': 3, 'have': 4, 'my': 5, 'has': 6, 'bone': 7}

[[3, 4, 1, 2], [5, 2, 6, 1, 7]]
```

```
# Prepare tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['data_clean'])
vocab_size = len(tokenizer.word_index) + 1

# Integer encode the documents
encoded_docs = tokenizer.texts_to_sequences(df['data_clean'])

# Pad documents to a max length of 500 words
max_length = 500
padded_data = pad_sequences(encoded_docs, maxlen=MAX_WORDS, padding='post')
```

```
padded_data.shape
```

```
(76365, 500)
```

```
vocab_size
```

```
81139
```

## Creating Target variable

The target vector is created using MultiLabelBinarizer from sklearn. The 100 tags are given fixed positions in a 100-long vector. The outp vector is a 100-long vector with ones at positions where the corresponding tag for the document is present and zero otherwise.

```
mlb = MultiLabelBinarizer()
target = mlb.fit_transform(df['Tags'])
```

## Creating Train-Test data

Use sklearn's train_test_split to randomly split the dataset into 80% train and 20% test sets.

```
x_train, x_test, y_train, y_test = train_test_split(padded_data, target,
                                                    test_size=0.2,
                                                    random_state=99)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(61092, 500)
(15273, 500)
(61092, 100)
(15273, 100)
```

# Creating the model

## Model Architecture

- **Embedding Layer**

> Indented block An embedding layer stores one vector per word. When called, it converts the sequences of word indices to sequences of vectors. These vectors are trainable. After training (on enough data), words with similar meanings often have similar vectors.

- **Dropout**

> Introduce a 15% dropout to prevent overfitting

- **Conv1D**

> 128 filters of size 5 applied to capture essential features

- **GlobalMaxPooling**

> Applies max-pooling over each document

- **Dense**

> Single fully-connected layer to 100 output nodes

```
def create_cnn_model():
  model = Sequential()

  model.add(layers.Embedding(vocab_size, EMBEDDING_DIM, input_length=MAX_WORDS))
  model.add(layers.Dropout(0.15))
  model.add(layers.Conv1D(128, 5, activation='relu', padding='same', strides=1))
  model.add(layers.GlobalMaxPooling1D())
  model.add(layers.Dense(100, activation='sigmoid'))

  return model
```

# Training the model

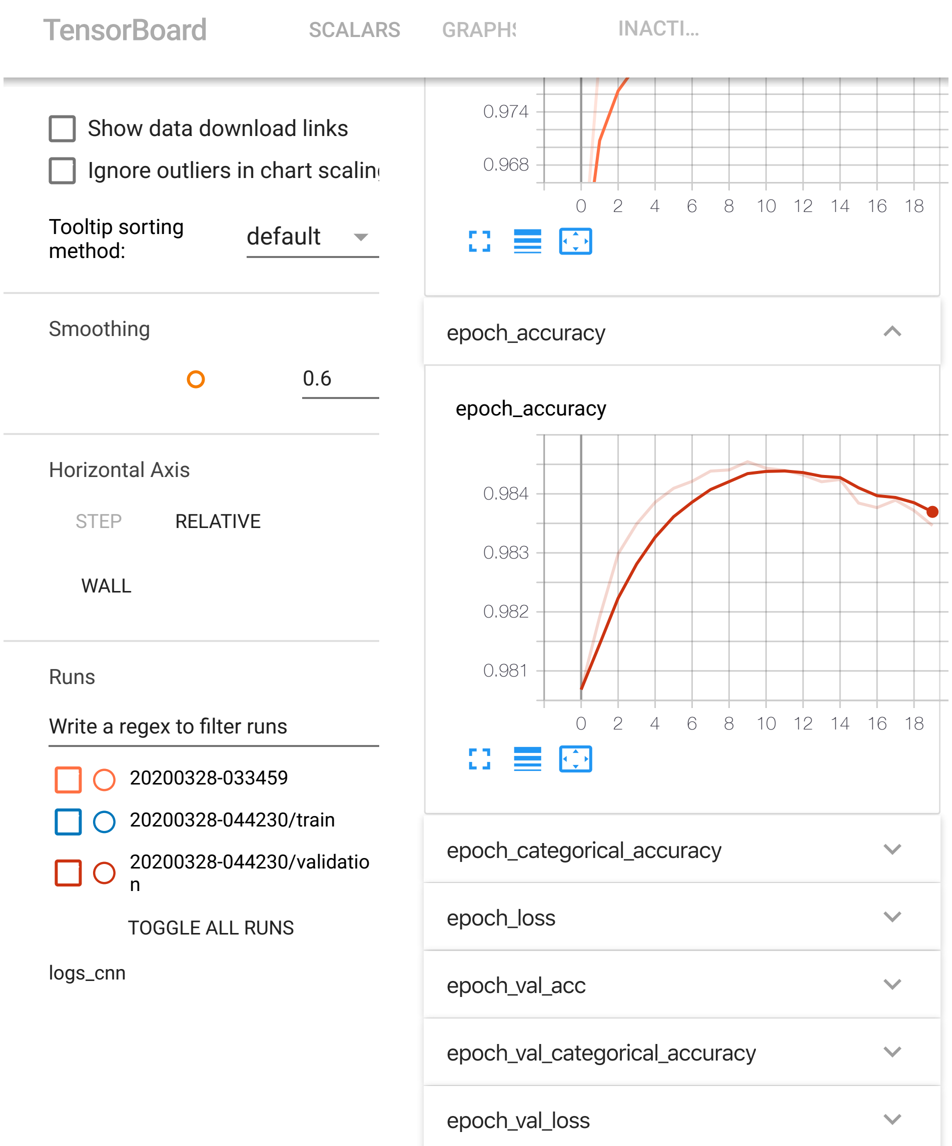
- The model was trained with binary_crossentropy loss function since a multi-label classification problem resembles a collection of b classification problems.
- The 'accuracy' metric is not a good indicator of model correctness since the target vector is a sparse matrix and predicted vector w have many values close to zero.
- The 'categorical_accuracy' metric is more reliable that the right labels were identified.
- Keras has a utility to invoke Tensorboard in its callbacks.

```
def train_cnn_model():
  model = create_cnn_model()
  print(model.summary())
  model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['categorical_accuracy', 'accuracy'])

  logdir = os.path.join("logs_cnn", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
  tensorboard = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

  model.fit(x_train,
          y_train,
          epochs=EPOCHS,
          validation_split=0.1,
          batch_size=BATCH_SIZE,
          callbacks=[tensorboard])

  return model

model = train_cnn_model()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 500, 256)          20771584  
_________________________________________________________________
dropout (Dropout)            (None, 500, 256)          0         
_________________________________________________________________
conv1d (Conv1D)              (None, 500, 128)          163968    
_________________________________________________________________
global_max_pooling1d (Global (None, 128)               0         
_________________________________________________________________
dense (Dense)                (None, 100)               12900     
=================================================================
Total params: 20,948,452
Trainable params: 20,948,452
Non-trainable params: 0
_________________________________________________________________
None
Train on 54982 samples, validate on 6110 samples
Epoch 1/20
54982/54982 [==============================] - 41s 739us/sample - loss: 0.1790 - categorical_accuracy: 0.0468 -
Epoch 2/20
54982/54982 [==============================] - 37s 677us/sample - loss: 0.0826 - categorical_accuracy: 0.1245 -
Epoch 3/20
54982/54982 [==============================] - 38s 684us/sample - loss: 0.0695 - categorical_accuracy: 0.2692 -
Epoch 4/20
54982/54982 [==============================] - 37s 681us/sample - loss: 0.0610 - categorical_accuracy: 0.3152 -
Epoch 5/20
54982/54982 [==============================] - 38s 684us/sample - loss: 0.0554 - categorical_accuracy: 0.3482 -
Epoch 6/20
54982/54982 [==============================] - 38s 685us/sample - loss: 0.0515 - categorical_accuracy: 0.3752 -
Epoch 7/20
54982/54982 [==============================] - 37s 678us/sample - loss: 0.0485 - categorical_accuracy: 0.3983 -
Epoch 8/20
54982/54982 [==============================] - 37s 673us/sample - loss: 0.0460 - categorical_accuracy: 0.4158 -
Epoch 9/20
54982/54982 [==============================] - 36s 663us/sample - loss: 0.0437 - categorical_accuracy: 0.4325 -
Epoch 10/20
54982/54982 [==============================] - 37s 671us/sample - loss: 0.0414 - categorical_accuracy: 0.4477 -
Epoch 11/20
54982/54982 [==============================] - 37s 674us/sample - loss: 0.0391 - categorical_accuracy: 0.4664 -
Epoch 12/20
54982/54982 [==============================] - 37s 668us/sample - loss: 0.0368 - categorical_accuracy: 0.4810 -
Epoch 13/20
54982/54982 [==============================] - 37s 676us/sample - loss: 0.0345 - categorical_accuracy: 0.4981 -
Epoch 14/20
54982/54982 [==============================] - 37s 671us/sample - loss: 0.0322 - categorical_accuracy: 0.5111 -
Epoch 15/20
54982/54982 [==============================] - 37s 678us/sample - loss: 0.0298 - categorical_accuracy: 0.5248 -
Epoch 16/20
54982/54982 [==============================] - 37s 670us/sample - loss: 0.0276 - categorical_accuracy: 0.5390 -
Epoch 17/20
54982/54982 [==============================] - 37s 671us/sample - loss: 0.0253 - categorical_accuracy: 0.5500 -
Epoch 18/20
54982/54982 [==============================] - 37s 672us/sample - loss: 0.0232 - categorical_accuracy: 0.5601 -
Epoch 19/20
54982/54982 [==============================] - 37s 669us/sample - loss: 0.0213 - categorical_accuracy: 0.5689 -
Epoch 20/20
54982/54982 [==============================] - 37s 670us/sample - loss: 0.0195 - categorical_accuracy: 0.5774 -
```

```
%tensorboard --logdir logs_cnn
```

```
Reusing TensorBoard on port 6006 (pid 1808), started 0:30:37 ago. (Use '!kill 1808' to kill it.)
```




## Getting results and calculating metrics

The predicted results contain 100-long vectors containing values from 0 to 1. If values are greater than THRESHOLD=0.5 (hyperparamete classified as 1.

```
y_pred = model.predict(x_test)

q_pred = (y_pred >= THRESHOLD).astype(int)

# Print f1, precision, and recall scores
print('Precision: ', precision_score(y_test, q_pred , average='micro'))
print('Recall: ', recall_score(y_test, q_pred , average='micro'))
print('F1 score: ', f1_score(y_test, q_pred , average='micro'))
```

```
Precision:  0.5930987649081109
Recall:  0.45687569476230955
F1 score:  0.516150479250928
```

```
sample = 1230

mlb.inverse_transform(q_pred)
```

```
mlb.inverse_transform(q_pred)
exp_tags = {i:tag for i, tag in enumerate(mlb.classes_) if y_test[sample][i] == 1}
print('Expected tags: ', exp_tags)

found_tags = {i:tag for i, tag in enumerate(mlb.classes_) if q_pred[sample][i] == 1}
print('Found tags: ', found_tags)
```

```
Expected tags:  {40: 'logistic'}
Found tags:  {9: 'categorical-data', 40: 'logistic'}
```

## Sample output

```
# Function to prepare a test document index for predictions

def get_tags_for_id(post_index):
  # Get post
  sample_post = df['data'][post_index]
  print('Original post:', sample_post)

  # Clean
  clean_data = clean_text(sample_post)

  # Encode string to integers
  encoded_data = tokenizer.texts_to_sequences([clean_data])
  padded_data = pad_sequences(encoded_data, maxlen=MAX_WORDS, padding='post')

  # Predict
  sample_pred = model.predict(padded_data)

  # Binarize output
  sample_pred = (sample_pred >= THRESHOLD).astype(int)

  print('Real tags: ', df['Tags'][post_index])
  print('Predicted tags: ', mlb.inverse_transform(sample_pred))
```

```
# Helper method to test the model
get_tags_for_id(75300)
```

```
Original post: How to Generate a Cloud of 3 dimensional points distributed according to a gaussian? <p>How can

<p>I know that box muller can generate the following 2d cloud but I would like to know how I can do something s

<p><a href="http://i.stack.imgur.com/IYflU.png" rel="nofollow"><img src="http://i.stack.imgur.com/IYflU.png" al

Real tags:  ['distributions' 'normal-distribution']
Predicted tags:  [('distributions', 'normal-distribution')]
```

# Creating a RNN model

## Model Architecture

- **Embedding Layer**

> Indented block An embedding layer stores one vector per word. When called, it converts the sequences of word indices to sequences of vectors. These vectors are trainable. After training (on enough data), words with similar meanings often have similar vectors.

- **Dropout**

> Introduce a 15% dropout to prevent overfitting

- **Bidirectional LSTM**

> Two hidden layers of opposite direction with 64 hidden layers to better understand the context of the text.

> The architecture of the recurrent neural network is Many-To-One.

> Apply recurrent regularizer L2.

> Apply recurrent dropout.

- **Dense**

> Single fully-connected layer to 100 output nodes

```
def create_rnn_model():

  model = tf.keras.models.Sequential()

  model.add(tf.keras.layers.Embedding(vocab_size, EMBEDDING_DIM, input_length=MAX_WORDS))
  model.add(layers.Dropout(0.15))
  model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(
        64, return_sequences=False,
        recurrent_regularizer=tf.keras.regularizers.l2(0.01),
        recurrent_dropout= 0.01,
        recurrent_initializer='glorot_uniform')))
  model.add(tf.keras.layers.Dense(100, activation='sigmoid'))

  return model
```

## Training the model

- The model was trained with binary_crossentropy loss function since a multi-label classification problem resembles a collection of b classification problems.
- The 'accuracy' metric is not a good indicator of model correctness since the target vector is a sparse matrix and predicted vector w have many values close to zero.
- The 'categorical_accuracy' metric is more reliable that the right labels were identified.
- Keras has a utility to invoke Tensorboard in its callbacks.

```
def train_rnn_model():
  model = create_rnn_model()
  print(model.summary())
  model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['categorical_accuracy', 'accuracy'])

  logdir = os.path.join("logs_rnn", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
  tensorboard = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

  model.fit(x_train,
          y_train,
          epochs=EPOCHS,
          validation_split=0.1,
          batch_size=BATCH_SIZE,
          callbacks=[tensorboard])

  return model

model = train_rnn_model()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 500, 256)          20771584  
_________________________________________________________________
dropout_1 (Dropout)          (None, 500, 256)          0         
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               164352    
_________________________________________________________________
dense_1 (Dense)              (None, 100)               12900     
=================================================================
Total params: 20,948,836
Trainable params: 20,948,836
Non-trainable params: 0
_________________________________________________________________
None
Train on 54982 samples, validate on 6110 samples
Epoch 1/20
54982/54982 [==============================] - 222s 4ms/sample - loss: 0.9494 - categorical_accuracy: 0.0410 -
Epoch 2/20
54982/54982 [==============================] - 214s 4ms/sample - loss: 0.1225 - categorical_accuracy: 0.0545 -
Epoch 3/20
54982/54982 [==============================] - 218s 4ms/sample - loss: 0.0900 - categorical_accuracy: 0.0615 -
Epoch 4/20
54982/54982 [==============================] - 216s 4ms/sample - loss: 0.0863 - categorical_accuracy: 0.0683 -
Epoch 5/20
54982/54982 [==============================] - 214s 4ms/sample - loss: 0.0804 - categorical_accuracy: 0.1203 -
Epoch 6/20
54982/54982 [==============================] - 212s 4ms/sample - loss: 0.0734 - categorical_accuracy: 0.1855 -
Epoch 7/20
54982/54982 [==============================] - 214s 4ms/sample - loss: 0.0662 - categorical_accuracy: 0.2609 -
Epoch 8/20
54982/54982 [==============================] - 213s 4ms/sample - loss: 0.0598 - categorical_accuracy: 0.3184 -
Epoch 9/20
54982/54982 [==============================] - 212s 4ms/sample - loss: 0.0546 - categorical_accuracy: 0.3664 -
Epoch 10/20
54982/54982 [==============================] - 216s 4ms/sample - loss: 0.0502 - categorical_accuracy: 0.4048 -
Epoch 11/20
54982/54982 [==============================] - 212s 4ms/sample - loss: 0.0467 - categorical_accuracy: 0.4358 -
Epoch 12/20
54982/54982 [==============================] - 215s 4ms/sample - loss: 0.0436 - categorical_accuracy: 0.4579 -
Epoch 13/20
54982/54982 [==============================] - 213s 4ms/sample - loss: 0.0409 - categorical_accuracy: 0.4776 -
Epoch 14/20
54982/54982 [==============================] - 212s 4ms/sample - loss: 0.0385 - categorical_accuracy: 0.4932 -
Epoch 15/20
54982/54982 [==============================] - 210s 4ms/sample - loss: 0.0362 - categorical_accuracy: 0.5053 -
Epoch 16/20
54982/54982 [==============================] - 210s 4ms/sample - loss: 0.0342 - categorical_accuracy: 0.5190 -
Epoch 17/20
54982/54982 [==============================] - 212s 4ms/sample - loss: 0.0323 - categorical_accuracy: 0.5294 -
Epoch 18/20
54982/54982 [==============================] - 213s 4ms/sample - loss: 0.0305 - categorical_accuracy: 0.5375 -
Epoch 19/20
54982/54982 [==============================] - 213s 4ms/sample - loss: 0.0288 - categorical_accuracy: 0.5455 -
Epoch 20/20
54982/54982 [==============================] - 217s 4ms/sample - loss: 0.0272 - categorical_accuracy: 0.5523 -
```

## Getting results and calculating metrics

The predicted results contain 100-long vectors containing values from 0 to 1. If values are greater than THRESHOLD=0.5 (hyperparamete
classified as 1.

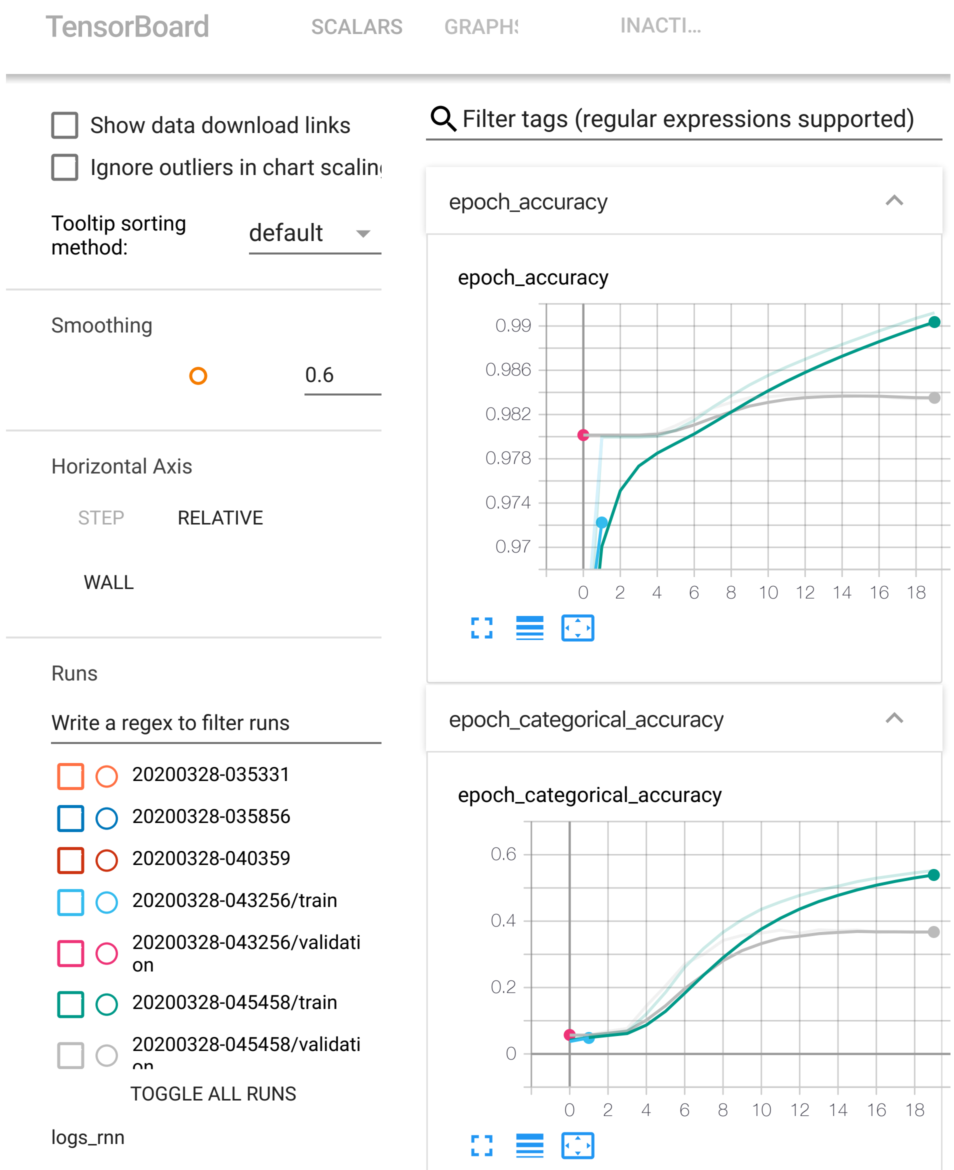
```
y_pred = model.predict(x_test)

q_pred = (y_pred >= THRESHOLD).astype(int)

# Print f1, precision, and recall scores
print('Precision: ', precision_score(y_test, q_pred , average='micro'))
```

```
print('Recall: ', recall_score(y_test, q_pred , average='micro'))
print('F1 score: ', f1_score(y_test, q_pred , average='micro'))
```

```
Precision:  0.6470524017467248
Recall:  0.3875629372915713
F1 score:  0.48476669529301103
```

```
%tensorboard --logdir logs_rnn
```

```
Reusing TensorBoard on port 6007 (pid 4633), started 0:07:02 ago. (Use '!kill 4633' to kill it.)
```




# Results summary

> Comparing the results of the two architectures tested for the project.
>
> Metrics:
>
> > Precision: $\frac{True-Positive}{Actual-Results}$
> >
> > Recall: $\frac{True-Positive}{Predicted-Results}$
> >
> > F1 Score: $\frac{True-Positive+True-Negative}{Total}$

| Model | Precision | Recall | F1 Score |
|---|---|---|---|
| Conv Network | 0.6009195193008011 | **0.4316026940430262** | **0.5023785059177228** |
| RNN Network | **0.6470524017467248** | 0.3875629372915713 | 0.48476669529301103 |