# Model:

* Baseline: The baseline embeds the title into a 768d vector using pre-trained SBert. And the image is passed through pre-trained resnet18 (passed through until the pooling layer before the fc) to get a 512d vector. Both these vectors are multiplied with weight matrices to equalise their dimensions and are concatenated. The concatenated vector is passed through a 3 layer fc to get an output(single output.
* Visual attention: Here the image is passed through 4 layers of the resnet18 to get a 512\*196 dimension output. Here there are 196 spatial locations and the idea of visual attention is to assign an attention weight to each of them. The mechanism is taken from [this paper of hierarchical co attention for vqa](https://arxiv.org/pdf/1606.00061.pdf). There they apply attention to both image and text but we focus on only the image part. ( we apply the image attention part from equations in section 3.3 of parallel co-attention from the paper).

For gossipcop we submitted an ensemble of the baseline and visual attention(bagging).

And we fine tune the baseline trained on gossipcop to politifact and submit this finetuned baseline.

# Model analysis:

We use Integrated gradients to assign attributions to the input. An attribution indicates how important a particular pixel of an image or a token(in the text) to the model prediction. Here a negative attribution indicates the token/pixel is inclined to fake news prediction and a positive one is to real news prediction.

Correlation condition :

***We want to check if the model has captured on the correlations in the dataset. Let us take a word in the title of news like ‘marriage’. Now we check x=freq(marriage in fake news)/no\_of\_fake news and y=freq(marriage in real news)/no\_of\_real\_news from the entire dataset.***

***If the word ‘marriage’ receives a negative attribution and x>y(‘marriage’ more important to fake news than real news) then we can say that the model has captured the correlation in the dataset.***

***Similarly if ‘marriage’ received a positive attribution and y>x we can conclude the same.***

We ran our model on the test set and noted down maximum positive attribution token for true positives and minimum negative attributions token for true negatives. Now if the above condition satisfies for the majority of tokens in both true positives and true negatives we can conclude that the model captured the correlation from the dataset. And we confirmed this.

Note : The model analysis was done on a different baseline. It is similar to the submitted baseline but has just one layer in the fc net.

# Exploratory Data Analysis:

EDA is an important part of any Data Science / Machine Learning Project. We try to capture different features from the title and text of the news headline. Some of the features extracted are as follows:

**Length of the Text**

**Length of the Title**

**Number of Characters in Title / Text**

# number of charcters

df\_eda['length\_title'] = df\_eda['cleaned\_title'].apply(len)

**Sentiment Score:**

# function to calculate sentiment score of title/text of a news

def senti\_score(text):

sia = SentimentIntensityAnalyzer()

sentiment\_score = (sia.polarity\_scores(text))['compound']

return sentiment\_score

**PoS Tags in Each Title/Text**

#unique pos tags

def unique\_pos(edit):

a = TextBlob(edit)

pos\_tags = a.tags

tags = []

for tag in pos\_tags:

tags.append(tag[1])

return len(list(set(tags)))

**No of Unique PoS tags**

# get the actual words associated with the tags

noun\_list = []

adjective\_list = []

adverb\_list = []

verb\_list = []

for title in titles:

adj = []

advb = []

verb = []

c = TextBlob(title)

pos\_tags = c.tags

for tag in pos\_tags:

if tag[1] in adjective\_tags:

adj.append(tag[0])

elif tag[1] in adverb\_tags:

advb.append(tag[0])

elif tag[1] in verb\_tags:

verb.append(tag[0])

noun\_list.append(list(set(c.noun\_phrases)))

adjective\_list.append(list(set(adj)))

adverb\_list.append(list(set(advb)))

verb\_list.append(list(set(verb)))

**Count of different PoS tags in a text**

# getting the count of each noun,adj,adv,verb tags

no\_of\_noun = []

no\_of\_adj = []

no\_of\_adv = []

no\_of\_verb = []

for i in range(len(noun\_list)):

a = len(noun\_list[i])

no\_of\_noun.append(a)

for i in range(len(adjective\_list)):

a = len(adjective\_list[i])

no\_of\_adj.append(a)

for i in range(len(adverb\_list)):

a = len(adverb\_list[i])

no\_of\_adv.append(a)

for i in range(len(verb\_list)):

a = len(verb\_list[i])

no\_of\_verb.append(a)

Most Common N-grams in text

# function to calculate n-grams

import re

from itertools import tee, islice

def ngrams(lst, n):

tlst = lst

while True:

a, b = tee(tlst)

l = tuple(islice(a, n))

if len(l) == n:

yield l

next(b)

tlst = b

else:

break

we find that the fake-news have comparatively high count of celebrity names in the top bi-grams as real-news.

Word-Cloud

#wordcloud

def corpus(df):

words=list(df['cleaned\_title'].apply(lambda x: word\_tokenize(x)))

all\_words=[]

for wordlist in words:

all\_words+=wordlist

return all\_words

def corpus\_text(df):

words=list(df['cleaned\_text'].apply(lambda x: word\_tokenize(x)))

all\_words=[]

for wordlist in words:

all\_words+=wordlist

# all\_words.remove("said '")

# all\_words.remove('one')

# all\_words.remove('time')

# all\_words.remove('like')

# all\_words.remove('new')

# print(all\_words[15])

return all\_words

def display\_wordcloud(all\_words):

most\_common=FreqDist(all\_words).most\_common(100)

wordcloud=WordCloud(background\_color='white').generate(str(most\_common))

fig=plt.figure(figsize=(10,10),facecolor='white')

plt.imshow(wordcloud,interpolation='bilinear')

plt.axis('off')

plt.title('Common words',fontsize=50)

plt.show()

Tf-Idf Vectorization

# stemming and lemmatization for tf-idf

def preprocess(headline):

st = PorterStemmer()

# Stemming

a = " ".join([st.stem(word) for word in headline.split()])

# Lemmatization

a = " ".join([Word(word).lemmatize() for word in a.split()])

return a

df\_eda['cleaned\_title']=df\_eda['cleaned\_title'].apply(preprocess)

df\_eda['cleaned\_text']=df\_eda['cleaned\_text'].apply(preprocess)

vectorizer = TfidfVectorizer()

tfidf\_title = vectorizer.fit\_transform(df\_eda['cleaned\_title'])

and others.

After performing feature extraction, we plot graphs of these different features with target variable. We plot separate graphs with real and fake news to check if we can spot any correlations between the feature and target variable. But we observe that the values of different features change arbitrarily among a fixed (real or fake) type of news.

As we are not able to form a strong hypothesis based on statistical features we proceed with deep-learning approaches. As a starting model we also try a tf-idf based classification model.