IMAGE CAPTIONING

Group Name: F

Group Members:

First name	Last Name	Student number

Submission date: ***[Insert submission date here]***

Contents

1.0 Abstract

This project seeks to apply Machine Learning and Deep Learning techniques to generate a short contextual description of the image.

In more precise terms we are looking at data from Common Objects in Context website which contains a comprehensive dataset with images and its associated captions on which we employed machine learning and deep learning techniques to generated automated short description of the image.

Successful completion of this project will demonstrate our ability to process dataset consisting of image and captions, generate a model based on the dataset and try to predict the context of the image in textual format.

2.0 Introduction

2.1 General Introduction

In this project, we are trying to create a model that can infer context from an image and generate short description of an image in a natural language format. The model will input an image and try to predict the contextual information of an image in a natural language format by generating a simple sentence from the image.

The below images are the few images, where the model have tried to get the contextual information of an image.

a man standing on a beach flying a kite

a man sitting at a table with a pizza

a man is riding a skateboard down a street

a stop sign that has been vandalized with graffiti

2.2 Simple use case of the project

In 2014, according to Mary Meeker's annual Internet Trends report, people uploaded an average of 1.8 billion digital images every single day. That's 657 billion photos per year. Another way to think about it: Every two minutes, humans take more photos than ever existed in total 150 years ago

With such a huge influx of images, comes a cumbersome task to identify the images correctly, searching the images and indexing the images. It is also difficult task to get contextual information from the images as for the computer, the image is an RGB presentation of the data represented in byte format.

By creating a model that could identify context information from an image, we can make the process of searching and indexing the images much faster. The ability for a computer to identify contextual information from an image will help it further in video processing, automation and many other endeavors.

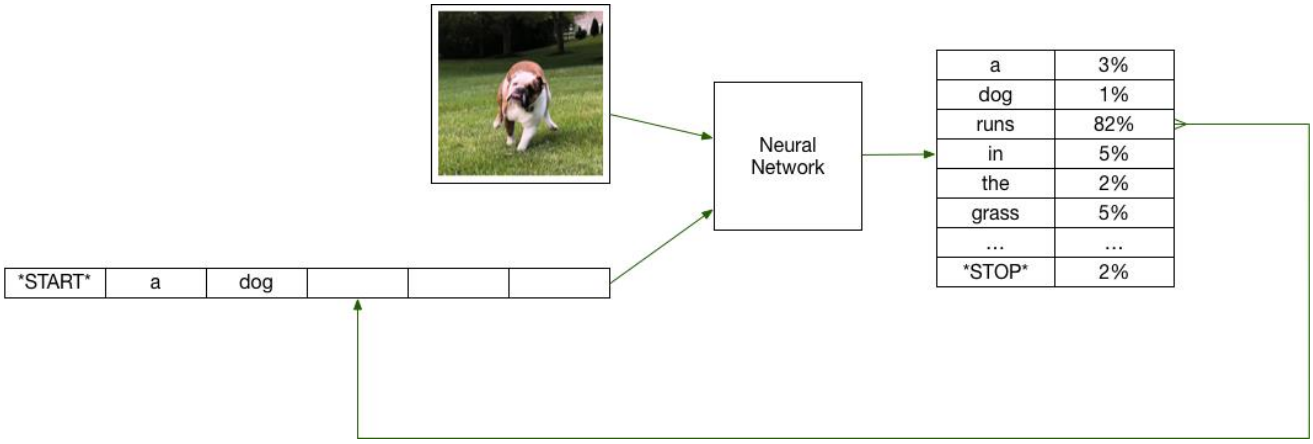
Getting contextual image and generating description of an image can help us to store the images in a structured manner as images are considered a form of unstructured data. Generating description of an image in a natural language format can help us to bring structure to an otherwise unstructured series of images.

We generally use natural language to search for an image on the internet and generating automatic description for an image, will help us to identify and index the image without needing an human intervention to categorize and describe the image.

3.0 Methods

3.1 General context of the project

As outlined in the project title, the objective of this project is to generate via a plethora of statistical models and algorithms, get the contextual information of an image in the form of natural human readable format.



The image above describes the general process of training and predicting the contextual information of an image in the form of natural language. The model is calculating the probability of a given word with respect to the features from the images and all the previous words in the caption.

This model is a word level model, wherein the model is calculating the probability of each word given the previous group of words and the features extracted from the image. Each caption is denoted by a tag for the start of the sentence and the tag for the end of the sentence. The model will always start with the starting tags and keep on predicting the next word for an image until it reaches the end tag.

For a model, the image is used for feature extraction and it works as a sort of image segmentation which helps identify different objects in an image. The model learns the real contextual information from the captions while in training and along with the feature extracted from the image help to create a natural language description of an image.

3.2 Context and Setting of the Study

Below of the hardware and software and libraries used for the training

Hardware

- 4 GB Nvidia GTX 1070Ti
- 32 GB SSD Ram
- Intel i7 9th Gen Processor

Software

- Windows 10
- Jupyter Notebook (used for writing code)
- Anaconda Environment (Environment for data science with essential libraries)

Libraries and Programming Language

- TensorFlow and KERAS (used for modeling)
- Scikit Learn (used for data processing)
- NLTK (used for model evaluation)
- Pandas and NumPy (for data preprocessing and data modification)
- Matplotlib (plotting charts)

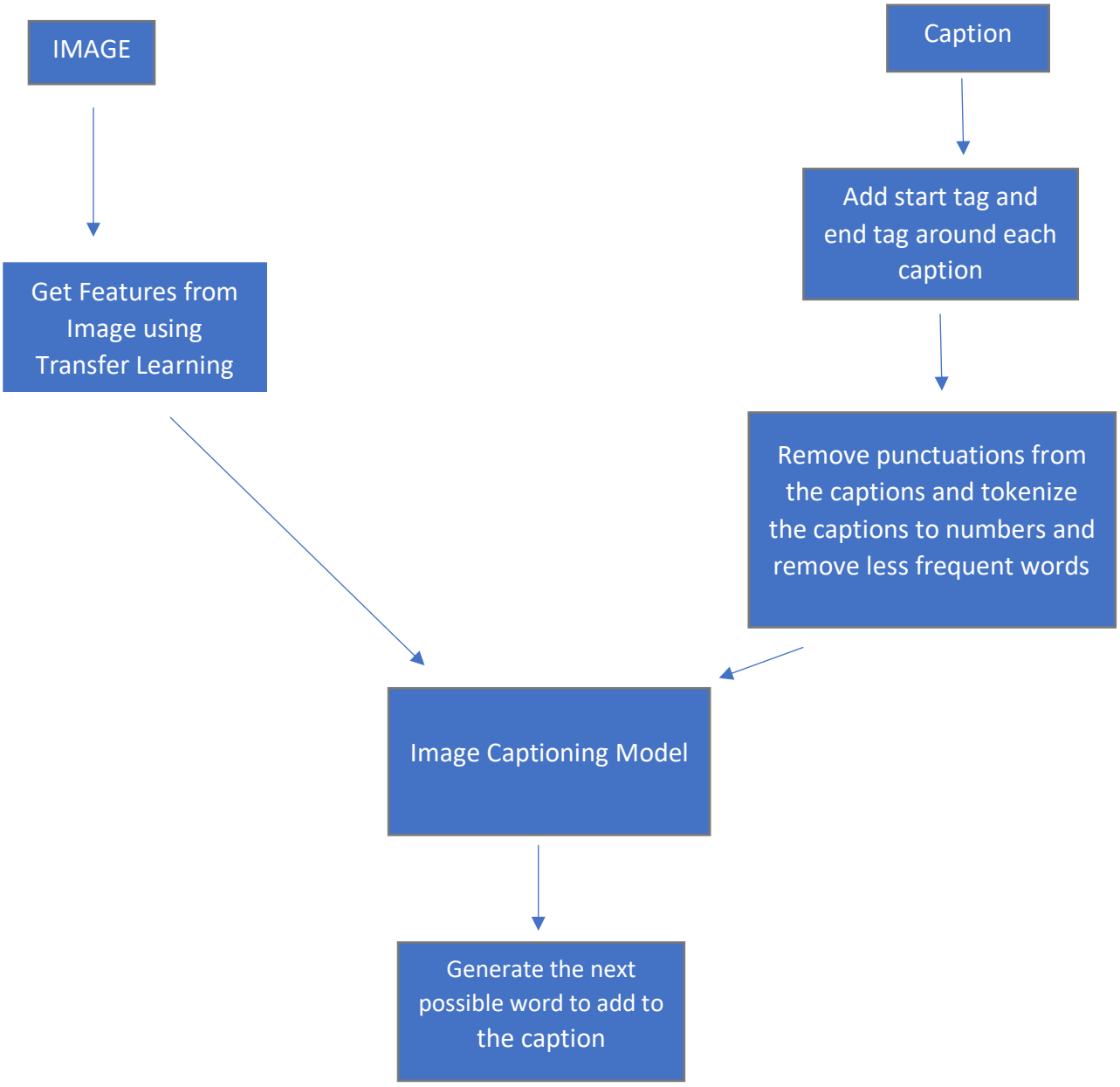
3.3 Literature and Dataset Overview

The project is based on the paper *"Deep Visual-Semantic Alignments for Generating Image Descriptions"* by Andrej Karpathy and Li Fei Fei from the Department of Computer Science at Stanford University.

The dataset is from the Common Objects in Context dataset which contains 14GB worth of images each containing at most 5 different captions for the images. The dataset is an opensource dataset and is free to download from the website.

3.4 Design of the Process

The design of the process is a following:



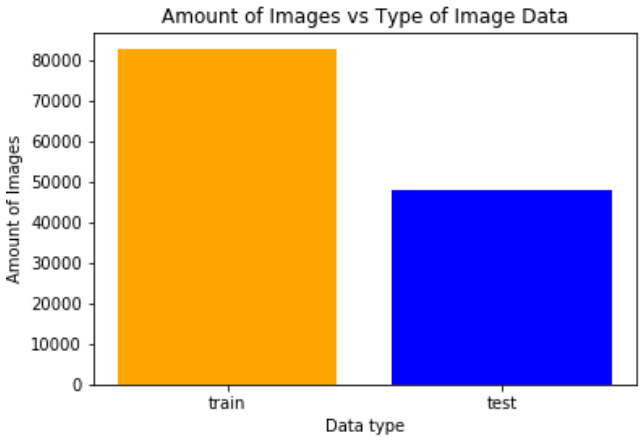
The above image outlines the whole process for our training and testing. We are going to rerun the whole process until we generate the caption or until we train the model. The whole process is described in the next few sections of this report.

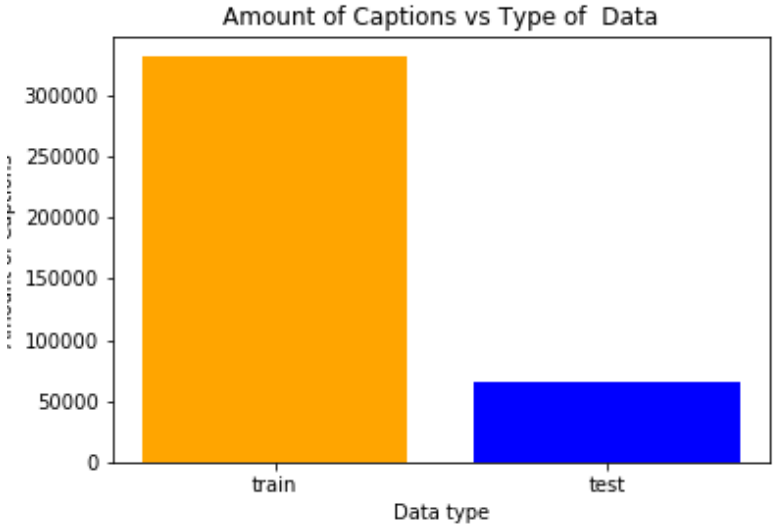
3.5 Description of the Dataset

The dataset is a part of Common Object in Context dataset. It is an opensource dataset containing image each with at most 5 captions.

There are total 82783 images each with 5 captions. In total, there are total 414113 unique pairs of image and captions.

Below images best describe the dataset and the split of the dataset into training and testing





3.6 Project Implementation: Phase 1- Pre-processing Captions

Data Pre Processing refers to the various and unique techniques used to produce transformations to a particular dataset, that looks to rid it of noise and unwanted data entry elements, so that we are left with data that can be easily modeled to produce the most accurate results.

As one of the inputs of our data is of a textual nature, our preprocessing techniques consisted of the following things:

- Remove Punctuations
- Remove Stop Words
- Add <start> tag to denote the starting of the captions
- Add <end> tag at the end to denote the ending of the captions
- Create a vocabulary of words from captions
- Remove words which are not in vocabulary and replace than with <unk> tag.
- Tokenize the captions and encode the word into word index

```
train_unprocessed_df=pd.read_csv(os.path.join("MSCOCO_14","training_data.csv"))
validation_unprocessed_df=pd.read_csv(os.path.join("MSCOCO_14","testing_data.csv"))
```

```
# adding the <start> tag at the begining of each caption and the <end> tag at the end of each caption
train_unprocessed_df.captions=train_unprocessed_df.captions.map(lambda data: "<start> "+data+" <end>")
validation_unprocessed_df.captions=validation_unprocessed_df.captions.map(lambda data: "<start> "+data+" <end>")
```

```
if os.path.exists("tokenizer.pickle")==True:
    print("Tokenizer found")
    with open('tokenizer.pickle', 'rb') as handle:
        tokenizer = pickle.load(handle)

else:
    print("Creating tokenizer. Tokenizer not found")
    df=pd.concat([train_unprocessed_df,validation_unprocessed_df],ignore_index=True)
    df.captions=df.captions.map(lambda data: "<start> "+data+" <end>")
    tokenizer = keras.preprocessing.text.Tokenizer(num_words=VOCAB_COUNT,
                                                   oov_token="<unk>",
                                                   filters='!"#$%&()*+.,-/:;=?@[\]^_`{|}~ ')
    tokenizer.fit_on_texts(df.captions)
    df['tokenize_captions']=tokenizer.texts_to_sequences(df.captions)
    MAX_LENGTH=df['tokenize_captions'].map(lambda data: len(data)).max()
    tokenizer.word_index['<pad>']=0
    tokenizer.index_word[0]="<pad>"
    with open('tokenizer.pickle', 'wb') as handle:
        pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
    del df
```

```
Tokenizer found
```

```
train_unprocessed_df['tokenize_captions']=tokenizer.texts_to_sequences(train_unprocessed_df['captions'])
validation_unprocessed_df['tokenize_captions']=tokenizer.texts_to_sequences(validation_unprocessed_df['captions'])
```

3.7 Project Implementation: Phase 2 Image Pre-Processing and Image Feature Extraction

Pre-Processing image is a way to convert the image into a acceptable format to input the image data to a system.

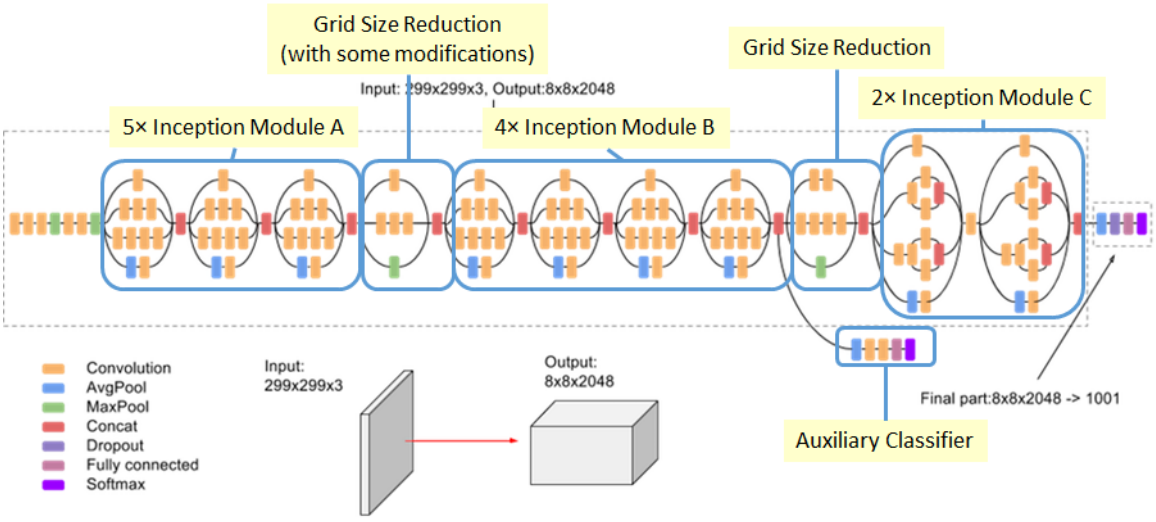
Feature extraction is a way to decrease the dimensionality of the image for higher computation speed. Feature Extraction is the name for methods that select and/or combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original dataset.

3.7.1 Using Transfer Learning to Extract Image Features

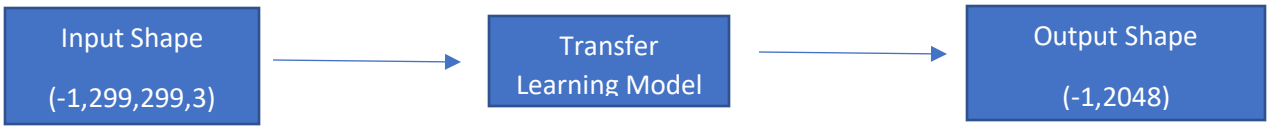
Transfer Learning is a machine learning method where a model developed for a certain task is reused as a starting point for a model on a second task. It saves us the time it would take to for us to create a model and saves a huge computation task it would require for us to train the new model in isolation.

Transfer Learning in this case, can help us to extract important and differential features from the image while also reducing the dimension of the original image. The model we used in the image is trained on the word renowned image net dataset which is a part of Large-Scale Visual Recognition Challenge. The image net dataset consisted of 3.2 million labelled images, separated into 5,247 categories, sorted into 12 subtrees like mammal, vehicle, and furniture.

The model we are using is called InceptionV3 which is a 1st Runner Up in Image Classification in ILSVRC 2015. The model contains 42 layers and is the third edition of Google's Inception Convolutional Neural Network.



In reference to this project, the input shape of the Inceptionv3 transfer learning model is (Batch Size, 299,299,3) and the output dimension of the model is (Batch Size , 2048)



Loading the image model using the Keras library

```
def getFeatureExtractorModel():
    if os.path.exists("image_feature_extractor"):
        print("Loading saved image model")
        image_model=keras.models.load_model("image_feature_extractor")
    else:
        print("no image feature extractor found. Downloading the file")
        image_model=keras.applications.InceptionV3(weights="imagenet")
        image_model=keras.Model(image_model.input,image_model.layers[-2].output)
        image_model.save("image_feature_extractor")
        return image_model
```

3.7.2 Preprocessing Image and Generating Image Feature

In the previous section, we stated we are using transfer learning and the google InceptionV3 model to generate features from the images. To use transfer learning, we need to change the image which is acceptable for the InceptionV3 model.

Our preprocessing techniques and feature extraction from the InceptionV3 model consists of the following items

- Load the image from the file
- Change the height and width of the image to 299 and 299
- Convert the image into an image array
- Divide the image array by 255 to convert the image data between 0 and 1
- Generate the features using the inceptionV3 image model
- Store the features in a file for later use

```
def getFeatureExtractorModel():
    if os.path.exists("image_feature_extractor"):
        print("Loading saved image model")
        image_model=keras.models.load_model("image_feature_extractor")
    else:
        print("no image feature extractor found. Downloading the file")
        image_model=keras.applications.InceptionV3(weights="imagenet")
        image_model=keras.Model(image_model.input,image_model.layers[-2].output)
        image_model.save("image_feature_extractor")
        return image_model
```

```
def generate_and_save_features(filename):
    img_data=keras.preprocessing.image.load_img(os.path.join(TRAING_IMG_FOLDER,filename),target_size=(299, 299))
    img_data = keras.preprocessing.image.img_to_array(img_data)
    img_data = img_data.reshape((1, img_data.shape[0], img_data.shape[1], img_data.shape[2]))
    img_data=keras.applications.inception_v3.preprocess_input(img_data)
    features=image_model.predict(img_data)
#     return features
    np.save(os.path.join(FEATURE_FOLDER,filename+"_new.npy"),features)
    return os.path.join(FEATURE_FOLDER,filename+"_new.npy")
```

```
image_caption_df=pd.read_csv(os.path.join("image_caption_df.csv"))
unique_images=np.unique(image_caption_df.image_location)
```

```
feature_file_array=os.listdir(FEATURE_FOLDER)
if len(feature_file_array)==len(unique_images):
    print("Feature Files found")
    del unique_images
else:
    print("no feature files found. Creating feature files")
    print("This may take upto 2 hours to finish depending on your computer.")
    image_model=getFeatureExtractorModel()
    vfunc=np.vectorize(generate_and_save_features)
    unique_images_new=vfunc(unique_images)
    del image_model
```

3.8 Project Implementation: Phase 3 Building the Model

Once we have pre-processed the captions and generated features from the images, we can now build the model using TensorFlow and Keras. Before we build the model, we need to know about different models

3.8.1 Different Model Layers

3.8.1.1 Dense Layer

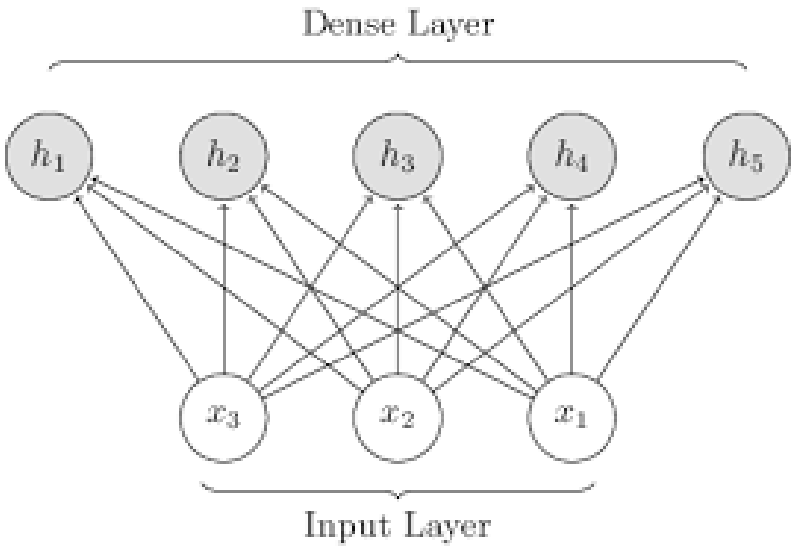
Dense layers add non-linearity property to the model. They are non-stochastic in nature wherein the same input will give the same output. It implements the following operations:

$$output = activation(dot(input + kernal) + bias)$$

The activation is an element-wise activation function which can be RELU , uniform etc. The kernel is a weights matrix created and built during the training of the model. The bias vector is a bias vector created by the layer.

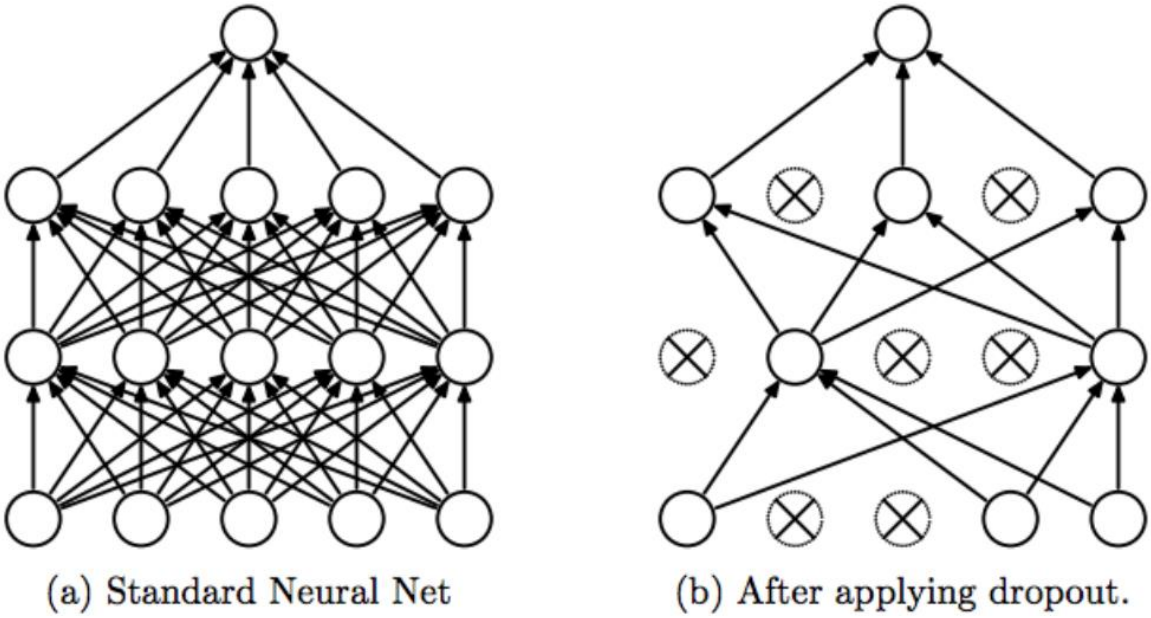
```
keras.layers.Dense(Units,activation="relu",kernel_initializer='normal')
```

The dense layer looks as below:



3.8.1.2 Dropout Layer

Dropout layer helps us to prevent overfitting in the model. During the training phase based on the dropout parameter certain units are ignored at random and are not considered during a forward or backward pass.



(a) Standard Neural Net (b) After applying dropout.

The code to create the Dropout layer using keras is as follows:

```
keras.layers.Dropout(rate)
```

3.8.1.3 Embedding Layer

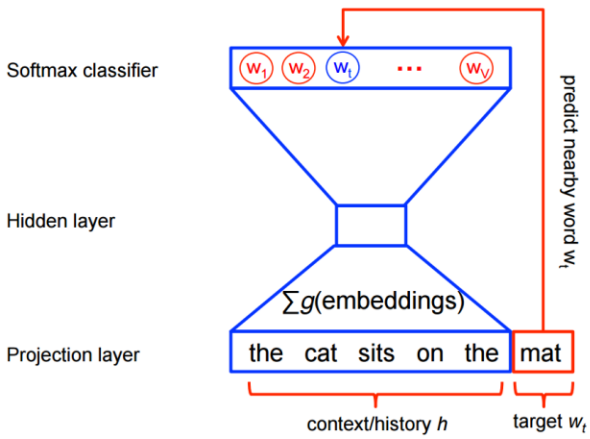
Embedding is a feature engineering algorithm used in text mining which helps us in getting textual features with semantics, syntactic relation and relation with other words. It gets similar words near in the vector space which generates semantic features

To create word embedding we need the following things:

- Vocabulary Size
- Embedding Size
- Sentences converted into list of indices from the Vocabulary

The code for creating word Embedding is as follows:

```
keras.layers.Embedding(VOCABULARY_COUNT,EMBEDDING_SIZE)
```



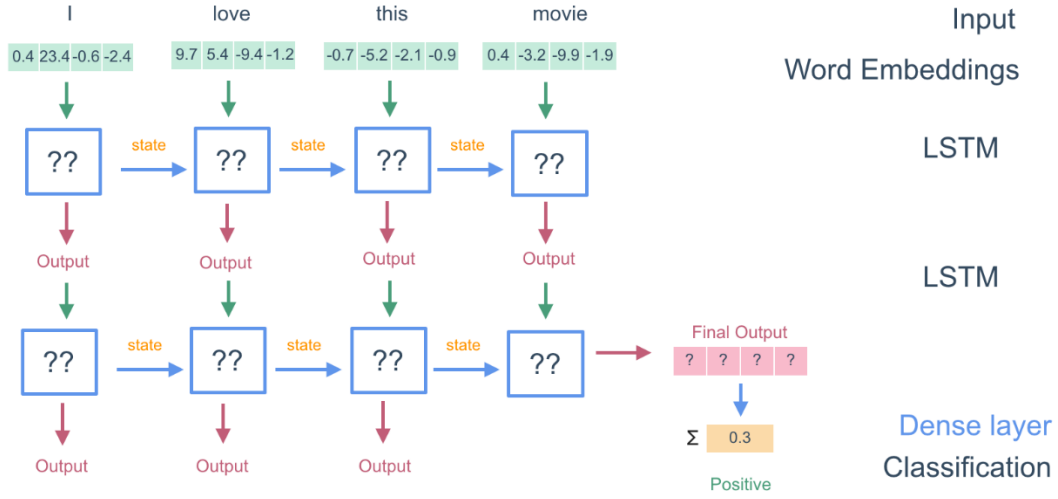
3.8.1.3 LSTM Layer

Long Short Term Memory: This is a version of recurrent neural networks (RNNs) that has found very wide acceptance and popularity with solving problems related to recurrent neural networks, due to their special ability to retain long term contextual dependencies as they carry out recurrent analysis. This is incredibly important to our intended outcome, as we will look to retain dependencies between key features that tend to signify a certain kind of sentiment, which in turn can help us identify churn, within tweets.

The code for the creating LSTM model

```
keras.layers.LSTM(NUM_UNITS)
```

The LSTM layer is mostly used after the Embedding Layer so that it can predict the probability of the next word given the current word and the previous word context



3.8.2 Defining the model

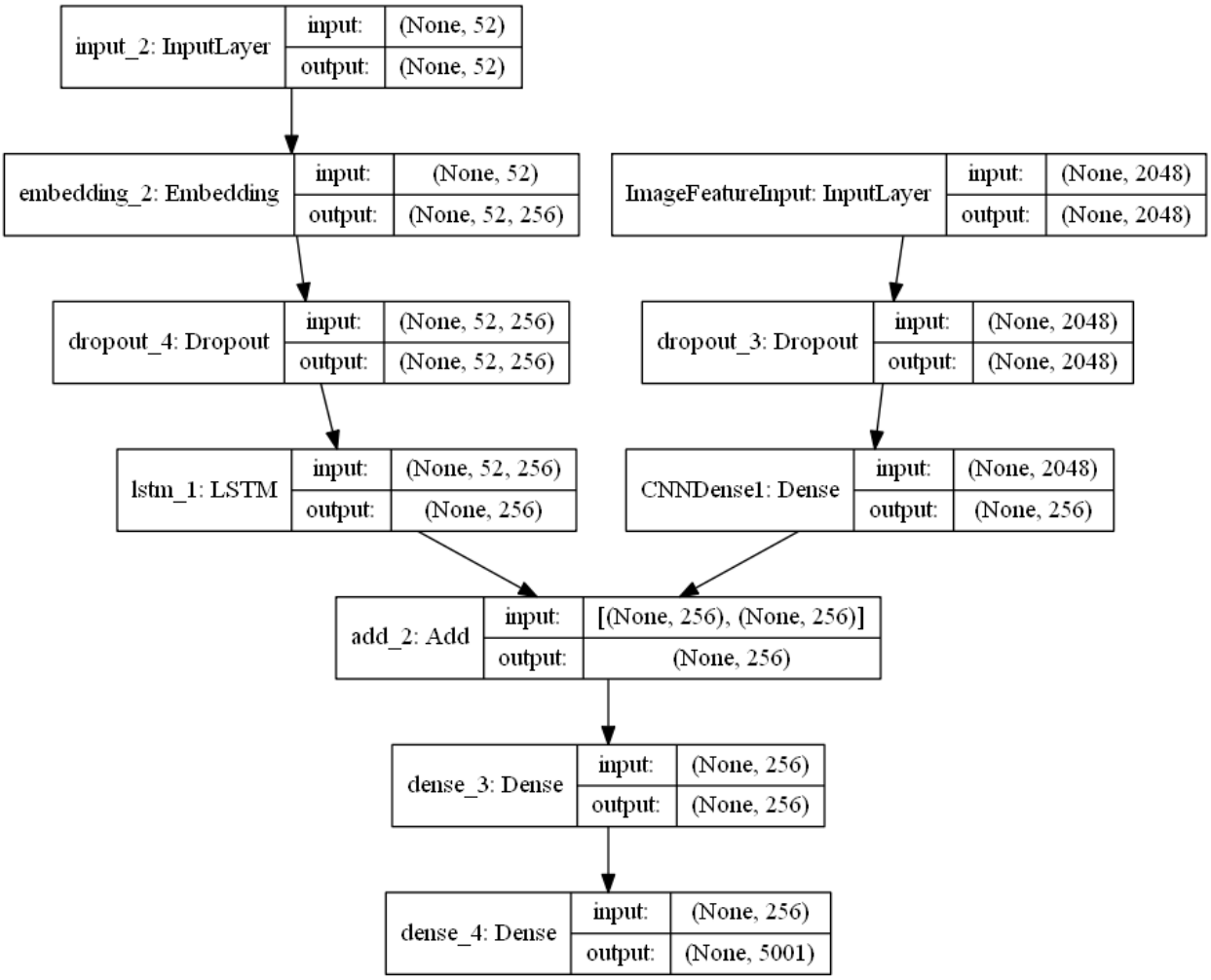
Below is the code to create the model using keras and by using all the layers described in section 3.8.1

```
# Feature Extraction Layer
input1=keras.layers.Input(shape=(2048,),name="ImageFeatureInput")
fe_dropout=keras.layers.Dropout(0.2)(input1)
fe1=keras.layers.Dense(256,activation='relu',name="CNNDense1",kernel_initializer="normal")(fe_dropout)
# Sequence model
input2=keras.layers.Input(shape=(MAX_LENGTH,))
se1=keras.layers.Embedding(VOCAB_COUNT+1,256,mask_zero=True)(input2)
se_dropout=keras.layers.Dropout(0.2)(se1)
se2=keras.layers.LSTM(256)(se_dropout)
# Decoder Model
decoder=keras.layers.add([fe1,se2])
decoder2=keras.layers.Dense(256, activation='relu')(decoder)
output=keras.layers.Dense(VOCAB_COUNT+1, activation='softmax')(decoder2)
model = keras.Model(inputs=[input1, input2], outputs=output)
model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy','mse'])
```

The description for the model is as below

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            (None, 52)           0
__________________________________________________________________________________________________
ImageFeatureInput (InputLayer)  (None, 2048)         0
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 52, 256)      1280256     input_2[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 2048)         0           ImageFeatureInput[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 52, 256)      0           embedding_2[0][0]
__________________________________________________________________________________________________
CNNDense1 (Dense)               (None, 256)          524544      dropout_3[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 256)          525312      dropout_4[0][0]
__________________________________________________________________________________________________
add_2 (Add)                     (None, 256)          0           CNNDense1[0][0]
                                                                 lstm_1[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 256)          65792       add_2[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 5001)         1285257     dense_3[0][0]
==================================================================================================
Total params: 3,681,161
Trainable params: 3,681,161
Non-trainable params: 0
__________________________________________________________________________________________________
```

The flow of input and output for the model is as below:



4.0 Training the model

Once we create the model, we can start training the model. The inputs for training the model is the image data and the captions.

4.0.1 Teacher Training

Teacher Training is a method for training the model. It is a very robust and a fast way to training the model. Teacher Training is mainly used with LSTM model wherein the expected output from the prior step is used as an input in the current step.

Example let's say we are trying to train the below sentence

```
A man hits a ball rapidly with his bat as two other men are squatting behind him
```

We are going to start and end tag around the caption

```
<start> A man hits a ball rapidly with his bat as two other men are squatting behind him <end>
```

Now imagine the model generates the word 'an' instead of word 'a'

Predicted Input	Predicted Outcome
[Start]	An

Now if we are using regular training method, we would put the word 'an' back into model to train it further

Predicted Input	Predicted Outcome
[Start]	An
[Start],An	Apple

As you can see from the above example, the model has gone off balance and will further add more loss into the training which in result will take a lot of time for the model to train.

Instead we can use the teacher training wherein irrespective of the predicted outcome, the input for the next sequence will be the actual output from the previous sequence

Predicted Input	Predicted Outcome
[Start]	An
[Start],A	Apple
[Start],A man	Orange

With teacher training, the model will learn the correct sequence or correct statistical properties for the sequence quickly.

4.0.2 Generating Sequence and Progressive Loading

With pre-processing we have converted the captions into word index token. Once the captions are converted into word index, we need to convert every single caption to multiple sequence. The model we are trying to predict is not going to predict the whole sentence as one but rather it would predict one word at a time until the model predicts the end tag.

For example, when we are trying to train the image with the following caption with the start and the end tag

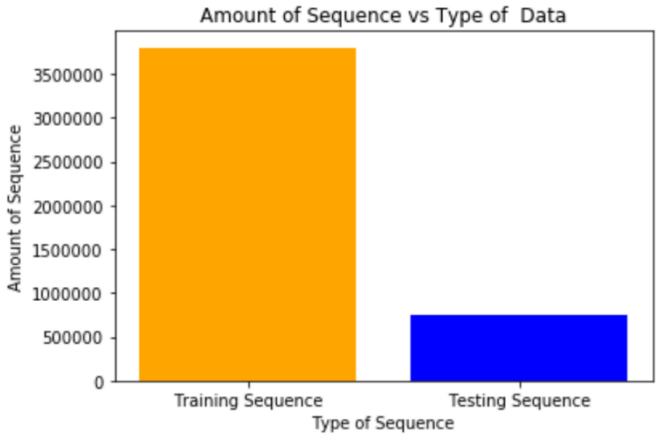
```
<start> A man hits a ball rapidly with his bat as two other men are squatting behind him <end>
```

From the caption, the input and output to the model are as follows:

X1	X2	Predicted Outcome
image data	[Start]	?
image data	[Start],A	?
image data	[Start],A,man	?
image data	[Start],A,man,hits	?
image data	[Start],A,man,hits,a	?

Since, we also have an image data associated with captions the dataset can get huge inside the ram and so we are going to use a python generator to generate a smaller portion of the bigger dataset. It saves the ram from getting full.

The image below shows the total size of sequence vs type of data.



The code below shows the function to generate the sequence from the dataset

```
def generate_sequences(df,batch_size=64):
    while True:
        x1=list()
        x2=list()
        y=list()
        for index in range(len(df)):
            single_row=df.loc[index]
            img_data=np.load(os.path.join(FEATURE_FOLDER,single_row[0]+"_new.npy"))
            img_data=np.squeeze(img_data)
#             img_data=np.reshape(img_data,(-1))
            cap=single_row[2]
            for i in range(1,len(cap)):
                in_seq, out_seq = cap[:i], cap[i]
                # pad input sequence
                in_seq = keras.preprocessing.sequence.pad_sequences([in_seq], maxlen=MAX_LENGTH,padding="post")[0]
                # encode output sequence
                out_seq = keras.utils.to_categorical([out_seq], num_classes=VOCAB_COUNT+1)[0]
                # store
                x1.append(img_data)
                x2.append(in_seq)
                y.append(out_seq)
                if len(y)==batch_size:

                    yield [[np.array(x1),np.array(x2)],np.array(y)]
                    x1=list()
                    x2=list()
                    y=list()
            if len(y)==batch_size:
                yield [[np.array(x1),np.array(x2)],np.array(y)]
                x1=list()
                x2=list()
                y=list()
        if len(y)>0:
            yield [[np.array(x1),np.array(x2)],np.array(y)]
```

```
train_data_generator=generate_sequences(train_unprocessed_df,BATCH_SIZE)
```

```
validation_data_generator=generate_sequences(validation_unprocessed_df,BATCH_SIZE)
```

The above code creates the generator for both training and validation data. For training the model the *training_data_generator* will be used and the *validation_data_generator* will be used to validate the model.

4.0.3 Checkpointing and Training the model

We used the following parameters for training the model. For our training model, the vocabulary count is 5000 and for embedding, the embedding size is 256. The number of epoch for our training model is 5, which means we will go through our training data 5 times and will try to improve the model

```
MAX_LENGTH=52
# folder which contain the training size
TRAING_IMG_FOLDER=os.path.join("MSCOCO_14","train2014")
TRAINING_CSV_FOLDER=os.path.join("MSCOCO_14","training_csv_folder")
VALIDATION_CSV_FOLDER=os.path.join("MSCOCO_14","validation_csv_folder")
VOCAB_COUNT=5000
FEATURE_FOLDER="Img_Features"
NUM_FEATURE_GENERATION_THREADS=100
BATCH_SIZE=64
EMBEDDING_SIZE=256
CHECKPOINT_FOLDER="MSCOCO_Checkpoint"
RESULT_FOLDER="ModelWithoutDropout"
NUM_EPOCH=5
```

We will add checkpointing to the model and will be store the checkpoint by monitoring validation loss. If the validation loss decreases each epoch, we will store the model. If the validation loss doesn't increase, we are not going to store the model.

```
filepath=os.path.join(CHECKPOINT_FOLDER,"everything-new-weights-dropout-improvement-{epoch:02d}-{loss:.4f}.hdf5")
checkpoint = keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
callbacklist=[checkpoint]
```

Apart from saving the model, we are going to predict some images after each epoch so that we can see how the model is improving with each epoch

```
def evaluate_model(image_data):

    text="<start> "
    previous=-1
    for i in range(MAX_LENGTH):
        sequence=tokenizer.texts_to_sequences([text])[0]
        sequence=keras.preprocessing.sequence.pad_sequences([sequence],maxlen=MAX_LENGTH,padding="post")
        yhat = model.predict([image_data,sequence], verbose=0)
        yhat = np.argmax(yhat)
        text=text+" "+tokenizer.index_word[yhat]

        if tokenizer.index_word[yhat].strip()=="<end>":
            break

    print(text)
```

```
class PredictionCallback(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        df_to_use=validation_unprocessed_df
        for i in range(10):
            a_index=random.randint(0,len(df_to_use))
            display(Image(filename=os.path.join(TRAING_IMG_FOLDER,df_to_use.image_location[a_index])))
            print("Data for index =",a_index)
            print("Original Captions")
            print(df_to_use.captions[a_index])
            print("Generated captions")
            image_data=np.load(os.path.join(FEATURE_FOLDER,df_to_use.image_location[a_index]+"_new.npy"))
            evaluate_model(image_data)
            print("-"*100)
callbacklist=[checkpoint,PredictionCallback()]
```

Both Prediction Callback and checkpointing will be running after each epoch. The Prediction Call back will predict the captions for 10 random images after each epoch.

```
TRAIN_STEPS=math.ceil(total_train_length/BATCH_SIZE)
VALIDATION_STEPS=math.ceil(total_validation_length/BATCH_SIZE)
print("Total Train steps is ",TRAIN_STEPS)
print("Total Validation steps is ",VALIDATION_STEPS)
```

```
Total Train steps is  59418
Total Validation steps is  11784
```

Before we start the training, we need to define the number of steps required for training and validation. The number of steps is the size of training data divided by the batch size

```
history=model.fit_generator(train_data_generator,epochs=NUM_EPOCH,
                            steps_per_epoch=TRAIN_STEPS,verbose=1,callbacks=callbacklist,initial_epoch=0,
                            validation_data=validation_data_generator,validation_steps=VALIDATION_STEPS)
```

5.0 Results

5.0.1 Evaluation Parameters

The key feature of our project is that, given the luxury of time, we were able to try more than one model in predicting churn. However, it was not enough to merely predict churn, we had to have an idea of how good these predictions are. To this end, we researched the following metrics for evaluating the output from both our models

- **Categorical Cross Entropy Loss**
  Categorical Cross entropy is a loss function that is used for single label categorization. This is when only one category is applicable for each data point. In other words, an example can belong to one class only.

  The formula for categorical cross entropy loss is

$$L(y,\hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * log(\hat{y}_{ij}))$$

  where ŷ is the predicted value

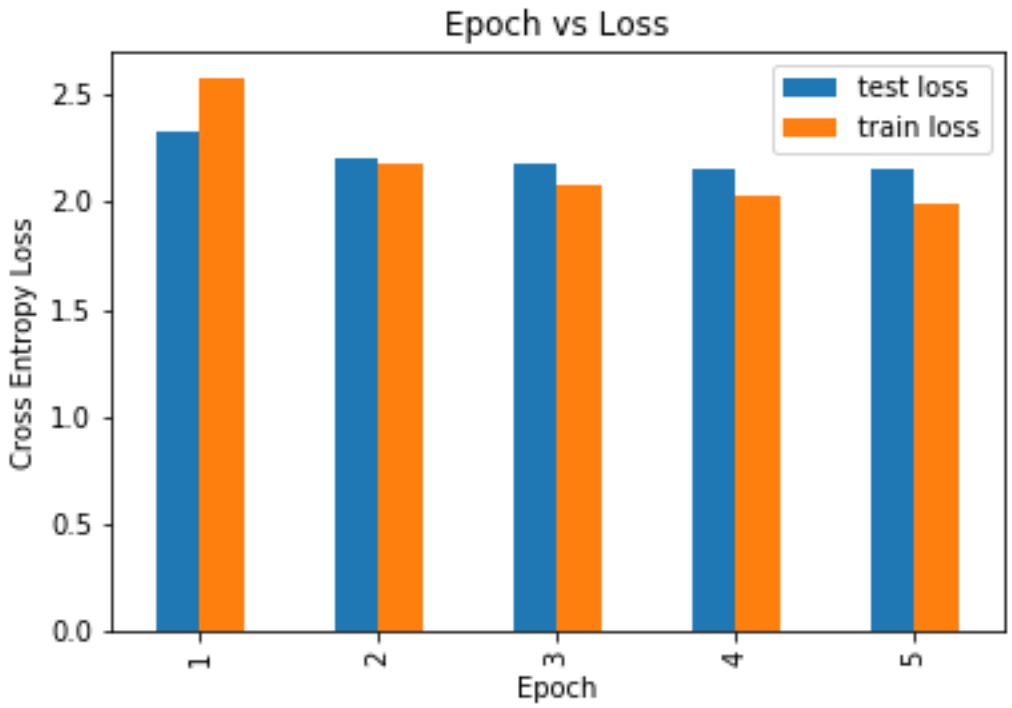
  Categorical cross entropy will compare the distribution of the predictions (the activations in the output layer, one for each class) with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes. To put it in a different way, the true class is represented as a one-hot encoded vector, and the closer the model's outputs are to that vector, the lower the loss.

- **Bleu Score**
  Bleu (Bilingual Evaluation Understudy) is a score for comparing a candidate translation of text to one or more reference translations. A perfect match results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0.

  The approach works by counting matching n-grams in the candidate translation to n-grams in the reference text, where 1-gram or unigram would be each token and a bigram comparison would be each word pair. The comparison is made regardless of word order.

5.0.2 Evaluation Results

- **Categorical Cross Entropy Loss per epoch**



The above chart shows the training loss and testing loss over the number of epoch. In our case, we trained the model for 5 epochs which means that we went over the training dataset 5 times.

In the above case, we can see that with each epoch, the training loss and validation loss are both decreases. With each epoch the amount of drop in the loss is becoming less and less and so after a few more epoch, the loss wouldn't improve.

Also, for the first epoch, the training loss is more than the testing loss which makes sense as the model is being trained for the first time and the weights haven't been trained yet. So for the first epoch, the training loss is higher. Also the training loss is the average of all the batch and for the first batch , the model will have a higher loss which when averaged over all the batches in the epoch makes the training loss to be higher than testing loss.

For the rest of the epoch, the testing loss is higher than training loss as the model is being trained on the training dataset while the testing data is unseen which results in the model having more loss in the testing data.

The code below shows the code to plot the loss chart

```
result_df=pd.DataFrame(history.history)
```

```
result_df.index=result_df.index+1
```

```
result_df.to_csv(os.path.join(RESULT_FOLDER,"result.csv"))
```

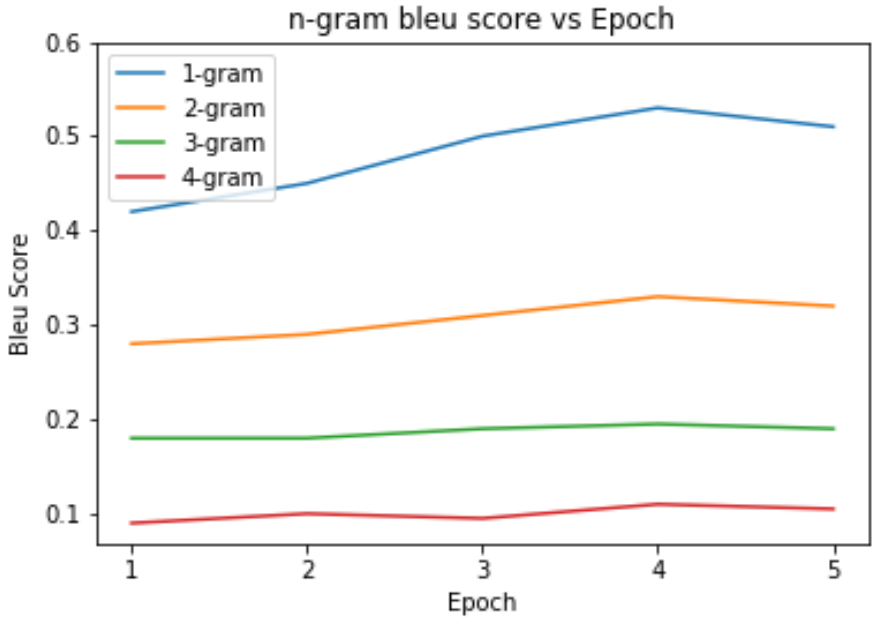
```
result_df[['val_loss','loss']].plot.bar()
plt.legend(['test loss','train loss'])
plt.xlabel("Epoch")
plt.ylabel("Cross Entropy Loss")
plt.title("Epoch vs Loss")
plt.savefig(os.path.join(RESULT_FOLDER,"LossVsEpoch.png"))
```

In the code above, Keras already calculates the training and the testing loss in the history object which we can use to draw the above chart.

- **Bleu Score over epoch**

The chart below shows the calculated n-gram bleu score vs epoch. The bleu score was calculated over the testing dataset after each epoch. The total time to calculate the bleu score for the testing dataset took more than 6 hours.

To generate the bleu score, we need to generate the whole caption for the image, but the model only predicts the next possible word given the previous words and the image data.



From the above chart, one can see that for the first 4 epoch, we can see considerable improvement in the n-gram bleu score.

The 1-gram bleu score is the highest, which makes sense as 1-gram bleu score is the calculation of the similarity of the words between the actual and predicted sentences irrespective of the ordering of the words.

The 2-gram bleu score means the calculation of the similarity of word pairs between the actual and predicted sentences irrespective of where in the predicted sentence, the word pair appears. So, it makes sense that the 2-gram bleu score is lower than 1-gram bleu score and higher than 3-gram and 4-gram bleu score.

Sincerely, 3-gram and 4-gram is calculation of the similarity of trigram and quad gram between the actual caption and the generated captions. Since, the captions are short descriptions of the image, the trigram and quad gram similarity between the actual captions and the generated captions will be small.

The below code calculates the bleu score in the validation dataset.

```
def generate_prediction_array(image_data,model_ref):
    text="<start> "
    previous=-1
    text_as_array=[]
    for i in range(MAX_LENGTH):
        sequence=tokenizer.texts_to_sequences([text])[0]
        sequence=keras.preprocessing.sequence.pad_sequences([sequence],maxlen=MAX_LENGTH,padding="post")
        y = model_ref.predict([image_data,sequence], verbose=0)
        y = np.argmax(y)
        y_word=tokenizer.index_word[y]
        text=text+" "+y_word

        if y_word.strip()=="<end>":
            break

        text_as_array.append(y_word)


    return text_as_array
```

```
modelList=os.listdir(os.path.join(CHECKPOINT_FOLDER))
epoch_bleu_score=[]
one_gram_bleu_score=[]
two_gram_bleu_score=[]
three_gram_bleu_score=[]
four_gram_bleu_score=[]
for a_model in modelList:
    count=0
    startTime=time.time()
    model=keras.models.load_model(os.path.join(CHECKPOINT_FOLDER,a_model))
    for i in range(len(validation_unprocessed_df)):
        validation_unprocessed_df.predicted_arr[i]=generate_prediction_array(
            np.load(os.path.join(FEATURE_FOLDER,validation_unprocessed_df.image_location[i]+"_new.npy")),
            model)
        count=count+1
        if count%1024==0:
            print("finished ",i)
            endTime=time.time()
            print("time taken is ",(endTime-startTime))
            startTime=time.time()
#     bleu_result=[]
    one_gram_bleu_score.append(nltk.translate.bleu_score.corpus_bleu(list(t.actual_arr),list(t.predicted_arr),weights=(1,0,0,0)))
    two_gram_bleu_score.append(nltk.translate.bleu_score.corpus_bleu(list(t.actual_arr),list(t.predicted_arr),weights=(0.5,0.5,0,0)))
    three_gram_bleu_score.append(nltk.translate.bleu_score.corpus_bleu(list(t.actual_arr),list(t.predicted_arr),weights=(0.3,0.3,0.3,0)))
    four_gram_bleu_score.append(nltk.translate.bleu_score.corpus_bleu(list(t.actual_arr),list(t.predicted_arr),weights=(0.25,0.25,0.25,0.25)))
```

```
bleu_score_dict={
    '1-gram':one_gram_bleu_score,
    '2-gram':two_gram_bleu_score,
    '3-gram':three_gram_bleu_score,
    '4-gram':four_gram_bleu_score
}
bleu_score_df=pd.DataFrame(bleu_score_dict)
bleu_score.to_csv(os.path.join(RESULT_FOLDER,"bleu_score_df.csv"))
bleu_score.index+=1
bleu_score.plot()
plt.xlabel("Epoch")
plt.ylabel("Bleu Score")
plt.title("n-gram bleu score vs Epoch")
plt.xticks([1,2,3,4,5])
plt.yticks([0.1,0.2,0.3,0.4,0.5,0.6])
plt.savefig(os.path.join(RESULT_FOLDER,"bleuScore.png"))
```

5.0.3 Generating and Viewing the Captions

a woman walking down a street holding an umbrella

a traffic light on a city street with a sky <unk>

a cat is sitting on a refrigerator in a kitchen

a man sitting on a bench talking on a cell phone

a man is playing tennis on a tennis court

a large airplane flying through the sky

The below code is used to generate the caption

```
def predictCaption(image_url,model_ref):
    startTime=time.time()
    # loading the image using keras and changing the size of image to 299,299
    img_data=keras.preprocessing.image.load_img(image_url,target_size=(299, 299))
    img_data = keras.preprocessing.image.img_to_array(img_data)
    # reshape the image to the shape (1,299,299,3)
    img_data = img_data.reshape((1, img_data.shape[0], img_data.shape[1], img_data.shape[2]))
    # preprocess input for the feature extraction model
    img_data=keras.applications.inception_v3.preprocess_input(img_data)
    # get features from the inceptionv3 model
    features=image_model.predict(img_data)
    text="<start>"
    captionText=""
    for i in range(MAX_LENGTH):
        # convert text to tokenizer
        sequence=tokenizer.texts_to_sequences([text])[0]
        sequence=tokenizer.texts_to_sequences([text])[0]
        # pad the tokenizer output to maxlength
        sequence=keras.preprocessing.sequence.pad_sequences([sequence],maxlen=MAX_LENGTH,padding="post")
        # predict the next word
        yhat = model_ref.predict([features,sequence], verbose=0)
        yhat = np.argmax(yhat)
        # add the next word into the sentence
        text=text+" "+tokenizer.index_word[yhat]

        #ignore the word if the word is the start tag or the end tag
        if tokenizer.index_word[yhat].strip()=="<start>":
           continue

        if tokenizer.index_word[yhat].strip()=="<end>":
            break


        captionText=captionText+" "+tokenizer.index_word[yhat]
    endTime=time.time()
    # display the image inside jupyter notebook
    display(Image(filename=image_url,height=300,width=300))
    # clean caption by removing extra spaces
    print(captionText.strip())
    print("-"*100)
```

6.0 Conclusion and Future Works

6.0.1 Concluding Observations from comparison of Outputs Models

- Transfer Learning is an important step in our process, as the it saves much computation time and help us to get important feature from the image
- The training the model is time consuming as there are many sequences for each image and caption pair
- The bleu score for unigram words is higher than bigram, trigram and quad gram words
- There are certain images, where the caption generated is bad because while training the context taken from the caption was bad
- The features and object segmentation part comes from the image but the actual context to connect different items of the image comes from the captions while training.

6.0.2 Unique Challenges Overcome

- **Resource Exhaustion Error**

  Sometimes while training, we were running out of resources and we were getting resource exhaustion error. To overcome this error, we did the following things:

  - Decreased the batch size
  - Used progressive loading
  - Used memory growth with TensorFlow to make it to progressively use GPU memory

- **Vanishing Gradient**

  Sometimes while training, the neurons would set to zero which would make the features received from the image useless and the model would just predict gibberish. To avoid this, we used a better initializer and kernel in our model dense layer.

6.0.3 Future Work

- Try to use different and better model for transfer learning instead of inceptionv3.
- Instead of word level sequence generator, we would like to generate a character level generator which would generate the caption character by character.
- Try to use glove to create embedding vector in our model
- We would like to further our work from image captioning to video captioning by using the principal, we learned while working on this project.

7.0 Glossary

https://www.tensorflow.org/

https://keras.io/

https://heartbeat.fritz.ai/using-a-keras-embedding-layer-to-handle-text-data-2c88dc019600?gi=72cad32a0bc5

https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/

https://gombru.github.io/2018/05/23/cross_entropy_loss/

https://stats.stackexchange.com/questions/284072/is-it-acceptable-to-have-a-slightly-lower-validation-loss-than-training-loss

https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

https://www.tensorflow.org/tutorials/text/image_captioning

https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

https://machinelearningmastery.com/transfer-learning-for-deep-learning/

https://cs.stanford.edu/people/karpathy/main.pdf

http://www.image-net.org/

http://cocodataset.org/#home