**1. 5Create a function to read a file and return its content**

def load\_doc(filename):

file = open(filename, 'r')

text = file.read()

file.close()

return text

def load\_data(data\_dir):

text = []

images = []

files\_in\_folder = os.listdir(data\_dir)

files\_in\_folder.sort()

for filename in tqdm(files\_in\_folder):

#Add an image

if filename[-3:] == "npz":

image = np.load(data\_dir+filename)

images.append(image['features'])

else:

**1.a Add text and wrap it in a start and end tag**

syntax = '<START> ' + load\_doc(data\_dir+filename) + ' <END>'

**1.b Seperate each word with a space**

syntax = ' '.join(syntax.split())

**1.c Add a space between each comma**

syntax = syntax.replace(',', ' ,')

text.append(syntax)

images = np.array(images, dtype=float)

return images, text

**2 . Intialize the function to create the vocabulary**

tokenizer = Tokenizer(filters='', split=" ", lower=False)

**2.a Create the vocabulary in a specific order**

tokenizer.fit\_on\_texts([load\_doc('bootstrap.vocab')])

dir\_name = '../../../../eval/'

train\_features, texts = load\_data(dir\_name)

**3. load model and weights**

json\_file = open('../../../../model.json', 'r')

loaded\_model\_json = json\_file.read()

json\_file.close()

loaded\_model = model\_from\_json(loaded\_model\_json)

# load weights into new model

loaded\_model.load\_weights("../../../../weights.hdf5")

print("Loaded model from disk")

**4. map an integer to a word**

def word\_for\_id(integer, tokenizer):

for word, index in tokenizer.word\_index.items():

if index == integer:

return word

return None

print(word\_for\_id(17, tokenizer))

**5. generate a description for an image**

def generate\_desc(model, tokenizer, photo, max\_length):

photo = np.array([photo])

# seed the generation process

in\_text = '<START> '

# iterate over the whole length of the sequence

print('\nPrediction---->\n\n<START> ', end='')

for i in range(150):

# integer encode input sequence

sequence = tokenizer.texts\_to\_sequences([in\_text])[0]

# pad input

sequence = pad\_sequences([sequence], maxlen=max\_length)

# predict next word

yhat = loaded\_model.predict([photo, sequence], verbose=0)

# convert probability to integer

yhat = argmax(yhat)

# map integer to word

word = word\_for\_id(yhat, tokenizer)

# stop if we cannot map the word

if word is None:

break

# append as input for generating the next word

in\_text += word + ' '

# stop if we predict the end of the sequence

print(word + ' ', end='')

if word == '<END>':

break

return in\_text

max\_length = 48

**6. evaluate the skill of the model**

def evaluate\_model(model, descriptions, photos, tokenizer, max\_length):

actual, predicted = list(), list()

# step over the whole set

for i in range(len(texts)):

yhat = generate\_desc(model, tokenizer, photos[i], max\_length)

# store actual and predicted

print('\n\nReal---->\n\n' + texts[i])

actual.append([texts[i].split()])

predicted.append(yhat.split())

# calculate BLEU score

bleu = corpus\_bleu(actual, predicted)

return bleu, actual, predicted

bleu, actual, predicted = evaluate\_model(loaded\_model, texts, train\_features, tokenizer, max\_length)

**7. Compile the tokens into HTML and css**

dsl\_path = "compiler/assets/web-dsl-mapping.json"

compiler = Compiler(dsl\_path)

compiled\_website = compiler.compile(predicted[0], 'index.html')

print(compiled\_website )

print(bleu)