# A
# REPORT
# ON

"Image Description Generation using Computer Vision and Deep Learning"

## By


| | |
|---|---|
| Durba Satpathi | 2019A7PS0972H |
| R Adarsh | 2019A7PS0230H |
| Shubham Priyank | 2019AAPS0467H |


## At

Happiest Minds Technologies

**A Practice School –I Station of**


## BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI (Rajasthan)

**22nd July 2021**

# A
# REPORT
# ON

"Image Description Generation using Computer Vision and Deep Learning"

**By**


| | | |
|---|---|---|
| Durba Satpathi | 2019A7PS0972H | CSE |
| R Adarsh | 2019A7PS0230H | CSE |
| Shubham Priyank | 2019AAPS0467H | ECE |


Prepared in partial fulfilment of the
Practice School-1 Course Nos.
BITS C221/BITS C231/BITS C241

**At**

Happiest Minds Technologies

**A Practice School –I Station of**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**
**(Rajasthan)**

**22nd July 2021**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI**
**(RAJASTHAN)**
**Practice School Division**

**Station:** Happiest Minds Technologies **Centre:** Bangalore
**Duration**: 2 months **Date of Start:** 8th June 2021
**Date of Submission:** 22nd July 2021
**Title of Project: Image:** Description Generation using Computer Vision and Deep Learning

| ID Nos | Name | Discipline |
|---|---|---|
| 2019A7PS0972H | Durba Satpathi | CSE |
| 2019A7PS0230H | R Adarsh | CSE |
| 2019AAPS0467H | Shubham Priyank | ECE |

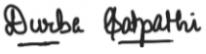
**Name and designation of expert:** Mr Vivek Ananthan, Senior Data Scientist

**Name of PS faculty:** Prof. Ramakrishna Dantu

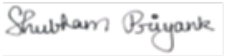
**Keywords:** Deep learning, ResNet, Transformers, Convolutional neural network, Transformers, Encoding, Embeddings

**Project Areas:** Deep learning, Natural Language Processing, Computer vision

**Abstract:** Image description generation or image captioning is the process of generating sentences related to a given image. Image description generation has a wide range of uses, like image indexing, accessibility for visually impaired people, social networking, search engines and recommendation systems. Image description generation can be considered an end to end seq2seq problem because it deals with the conversion of images (sequence of pixels) to sentences (sequence of words). We have used an encoder-decoder network to generate the captions for the images. In the encoder part, we have used convolutional neural networks (CNNs) to extract the features from the images and in the decoder part, we have used transformers to generate the captions.

Durba Satpathi 

R Adarsh 

Shubham Priyank 

**Signatures of students**
Date: 21/07/2021

Prof. Ramakrishna Dantu
**Signature of PS faculty**
**Date:** 22/07/2021

## Acknowledgements

We would like to extend our sincere gratitude to our PS Faculty Dr Ramakrishna Dantu for his consistent support and motivation throughout the course of PS-1 and for his persistent efforts to get us onboarded to the company.

We are also thankful to our industry mentors Mr Andrew Anand, Mr Umesh and Mr Vivek Ananthan for giving us this project which has enhanced our knowledge of machine learning and computer vision.

We would like to thank our project mentor Mr Vivek Ananthan in particular for his exemplary guidance, monitoring and constant encouragement throughout the course of this project and for providing the necessary information and resources for this project. He has helped us a lot in improving our technical skills and got us accustomed to the corporate culture, helped us follow and stick to deadlines and give constructive suggestions to enhance our work.

Lastly, we would like to thank our PS division and BITS Pilani for providing us with this opportunity to develop our industrial knowledge and give us this enriching experience in the form of PS-1 that has not only helped us witness what we have been studying but also implement them in the real world.

# Table of contents

---

# 1.Introduction

---

**About the project:**

We created a sentence-based image description generator; this project comes under two flourishing domains of deep learning image recognition and natural language processing. First, we generated index-based encoding of the captions present in the dataset our model uses these text encodings to train on. We then used the pre-trained ResNet models to extract image embeddings which we used as input to transformers' decoder layer, which uses these image embeddings to generate sentences that describe the image.

**Workflow followed:**

1. Collected data
2. Performed Data Augmentation
3. Word/Sentence Embedding
4. Combined Model
    - 4.1 CNN for encoding-ResNet
    - 4.2 Transformers for decoder network
    - 4.3 Combined Loss and Fine Tuning
5. Validation of Description
6. Generalization/Decoupling of Code

**Platform:**

We decided to use Google Colab for generating and training our model, as it provided us with necessary hardware such as GPU for accelerating our training rate and a platform where we can share our work easily and it even allowed multiple people to work on it at once. We worked with python3 and used many of its libraries, primarily PyTorch, cv2, PIL, pandas and pickle.

# 2. Dataset Preparation

We have used the Flickr8K dataset for developing this model. It consists of 8000 images and each image is paired with 5 descriptions.
For example, the following is the first image and the 5 captions provided of the dataset.


```
0    A child in a pink dress is climbing up a set of stairs in an entry way .
1                                    A girl going into a wooden building .
2                           A little girl climbing into a wooden playhouse .
3                         A little girl climbing the stairs to her playhouse .
4                  A little girl in a pink dress going into a wooden cabin .
Name: caption, dtype: object
```

## 2.1 Caption Preprocessing

We first preprocessed the given text captions. This was done by removing all punctuation marks from the captions and converting all the captions to lowercase. We added <START> and <END> tokens to the beginning and the end of the sentence respectively. We then added <PAD> tokens to the sentences to make all the sentences of equal length, the number of <PAD> tokens added were thus equal to the length of the largest caption minus the length of the given sentence. We then split the sentence into its constituent words. Then we created a vocabulary and enumerated the words and replaced the words of the sequence with their corresponding index numbers.

| | image | caption | clean_data | seq |
|---|---|---|---|---|
| 0 | 1000268201_693b08cb0e.jpg | A child in a pink dress is climbing up a set o... | [<START>, a, child, in, a, pink, dress, is, cl... | [2, 3, 1331, 3653, 3, 5277, 2205, 3743, 1435, ... |
| 1 | 1000268201_693b08cb0e.jpg | A girl going into a wooden building . | [<START>, a, girl, going, into, a, wooden, bui... | [2, 3, 3062, 3117, 3725, 3, 8255, 970, 1, 0, 0... |
| 2 | 1000268201_693b08cb0e.jpg | A little girl climbing into a wooden playhouse . | [<START>, a, little, girl, climbing, into, a, ... | [2, 3, 4152, 3062, 1435, 3725, 3, 8255, 5340, ... |
| 3 | 1000268201_693b08cb0e.jpg | A little girl climbing the stairs to her playh... | [<START>, a, little, girl, climbing, the, stai... | [2, 3, 4152, 3062, 1435, 7437, 6941, 7535, 343... |
| 4 | 1000268201_693b08cb0e.jpg | A little girl in a pink dress going into a woo... | [<START>, a, little, girl, in, a, pink, dress,... | [2, 3, 4152, 3062, 3653, 3, 5277, 2205, 3117, ... |

## 2.2 Image Augmentation

Image augmentation is the process of generating new images by adding variations to the existing image dataset to expand the existing image dataset. This can be done by various techniques cropping, rotating, flipping, blurring to name a few. Image augmentation is required to train deep learning convolutional neural networks as they require a large number of images to be trained effectively, it also improves the performance of the model by a better generalization and reducing overfitting.

We applied augmentation at the preprocessing step rather than in real-time and saved them to the disk as we had a smaller dataset which we wanted to expand to reduce overfitting and were not constrained by disk space but rather by the primary memory (RAM). The augmentation we performed are as followed:-
For reference, this is the original image without performing any augmentation.

```
[ ]  plt.imshow(cv2.cvtColor(cv2.imread('Images/997722733_0cb5439472.jpg'),cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f0d5141e2d0>
```


### 2.2.1 Centre Cropping:-

We applied the centre crop by taking a 299×299 pixel frame at the centre of each image and cropping it.

```
[ ]
    # creates a center cropped image for all the images present in the dataset and saves it to the disk
    def center_crop(desc):
      for key in desc:
        img = cv2.imread('/content/Images/'+key+'.jpg')
        width=img.shape[1]
        height = img.shape[0]
        dim = (299,299)
        mid_x = int(width/2)
        mid_y = int(height/2)
        w = 299
        h = 299
        x = mid_x - w/2
        y = mid_y - h/2
        crop_img = img[int(y):int(y+h), int(x):int(x+w)]
        cv2.imwrite('/content/Images/'+key+'_crop'+'.jpg',crop_img)
    center_crop(desc)
```

**Function for adding augmented centre crop image to the image dataset.*

```
plt.imshow(cv2.cvtColor(cv2.imread('/content/Images/997722733_0cb5439472_crop.jpg'),cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f0d445f4850>
```


**Augmented image after performing centre cropping,*

### 2.2.2 Changing brightness:-

We increased the brightness of the image by converting the image into HSV colour space and adding a random integer between 10 and 100 to the colour value channel of the image and if the value increases 255 it is set to 255.

```
#generates a image with brightness random increase in their brightness and saves them to the disk.
def image_brightness_augment(desc):
  for key in desc:
    img = cv2.imread('/content/Images/'+key+'.jpg')
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)
    value = random.randint(10,100)
    lim = 255 - value
    v[v > lim] = 255
    v[v <= lim] += value

    final_hsv = cv2.merge((h, s, v))
    img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
    cv2.imwrite('/content/Images/'+key+'_bright'+'.jpg',img)
image_brightness_augment(desc)
```

**Function for adding an augmented brightened image to the image dataset.*

```
plt.imshow(cv2.cvtColor(cv2.imread('/content/Images/997722733_0cb5439472_bright.jpg'),cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f0d4469dcd0>
```


**Augmented image after changing the image brightness.*

### 2.2.3 Adding blur to the image:-

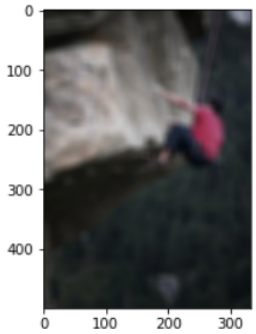
We added a random amount of blur to the image (between 5 and 15) by using the OpenCV library blur function.

```
[ ] #generates blurred images of the images present in the dataset by adding random amount of blur to it and saves it to the disk.
    def image_blur_augment(desc):
      for key in desc:
        img = cv2.imread('/content/Images/'+key+'.jpg')
        blur_value = random.randint(5,15)
        blur_image = cv2.blur(img,(blur_value,blur_value))
        cv2.imwrite('/content/Images/'+key+'_blur'+'.jpg',blur_image)

    image_blur_augment(desc)
```

**Function for adding an augmented blurred image to the image dataset.*

```
[ ] plt.imshow(cv2.cvtColor(cv2.imread('/content/Images/997722733_0cb5439472_blur.jpg'),cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f0d44619390>
```




**Augmented image after adding blur to the image*

## 2.3 Image Preprocessing

After augmenting the necessary images we started with getting the images ready to be fed into the Resnet18 model for finally generating their vector embedding. As the Resnet18 model accepts images of the size 224*224 dimensions, we first started with resizing the image to the appropriate size. 8-bit images have a value ranging from 0 to 255 for each pixel, but neural network models (a regression model) prefer a floating-point representation within a smaller range to get better results, so after that, we normalized the image by using the Normalize function which subtracts each channel (Red, blue and green) with their respective means and divides each channel by its standard deviation. Next, we convert the normalized image to tensor by using the ToTensor() function which takes in an image( or NumPy array) of a specific dimension and returns a tensor of the same dimension but with values ranging between 0 and 1.Thus by processing the images through this “transform_image()” function we make the images ready to be embedded by the Resnet18 model.

```
[ ] class transform_image():
      def __init__(self, data):
            self.data = data
            self.scaler = transforms.Resize([224, 224])
            self.normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225])
            self.to_tensor = transforms.ToTensor()
      def __len__(self):
            return self.data.shape[0]
      def __getitem__(self,idx):
            image_name = self.data.iloc[idx]['image']
            transformed_img= self.normalize(self.to_tensor(self.scaler(Image.open('/content/Images/'+str(image_name)))))
            return image_name, transformed_img
```

** Function for performing preprocessing to the image*

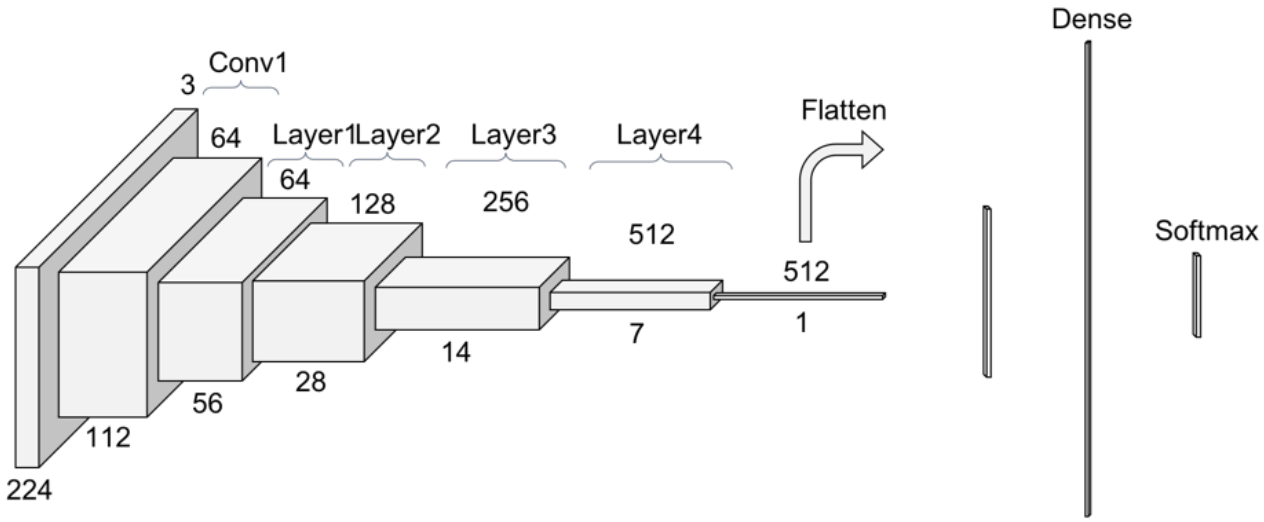
# 3. Model Description

## 3.1 Encoder Networks

We experimented with a variety of encoder networks to generate captions and we have summarized the final losses obtained using each encoder in the results section.

### 3.1.1 Using ResNet18:-

The ResNet18 architecture was used as encoder CNN to extract the features of images and get the vector embeddings. The images are resized as 224*224 which is the input shape for the ResNet18 model. The images are then normalized and converted to tensors. We then created a data loader of transformed images and each and every transformed image is passed forward through the ResNet18 architecture. The output embedding is obtained from the fourth layer of the Resnet. The image now shows 512 feature maps of dimension 7 X 7.




** Architecture of ResNet18*

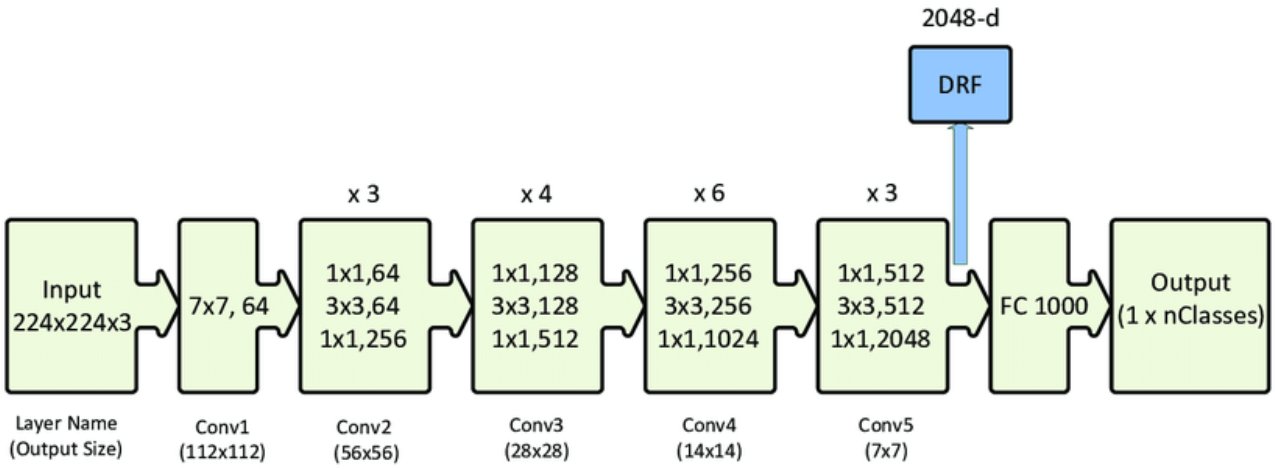
```
#get the layer4 of resNet18
resNet18Layer4 = resnet18._modules.get('layer4').to(device)
resNet18Layer4
```

```
Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

** Modules of layer4 of ResNet18*

### 3.1.2 Using ResNet50:-

Similar to ResNet18, ResNet50 was used as a CNN encoder to extract the features of images and get the vector embeddings. Again similar to ResNet18 the images are resized to 224*224 dimension which is the input shape of the ResNet model. Then the images are normalized and converted to tensors. Then, similar to the ResNet18 model the transformed images are passed through the ResNet50 architecture. The output is then obtained from the fourth layer of the model which is a 2048 feature map of dimension 7×7.




**(ResNet-50 Architecture [26] Shown with the Residual Units, the Size of the Filters and the Outputs of Each Convolutional Layer., 2020)*

```
#get the layer4 of resNet18
resNet50Layer4 = resnet50._modules.get('layer4').to(device)
resNet50Layer4
```

```
Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
)
```

** Modules of layer4 of ResNet50*

## 3.2 Decoder Network

### 3.2.1 Decoder Network using Transformers

We used a transformer decoder to generate the outputs of our model.

**Architecture:** Given a continuous sequence x=($x_1, x_2 \ldots\ldots x_n$), the decoder will generate the output sequence y=($y_1, y_2 \ldots\ldots y_n$). This is done in an autoregressive fashion wherein the decoder uses the earlier generated outputs to generate the next output. The decoder part of the transformers consists of a stack of several identical layers each composed of 3 sublayers.

*Sublayer-1*: Performs multi-headed attention with masking so that the current position does not attend to the succeeding positions.

*Sublayer-2*: Performs multi-head attention with a padding mask wherein the query receives the output of the sub-layer 1

*Sublayer-3*:A position-wise fully connected simple feed-forward network

There are residual connections around each of the sublayers. The next step is to normalize the layers. The self-attention in this decoder layer is modified in such a manner that the following positions from a given position are not attended(masking). Furthermore, the input to the decoder stacks is also 1 timestep right shifted. As a result, the predictions at a given position would depend only on the output at the previous positions and not the succeeding ones.

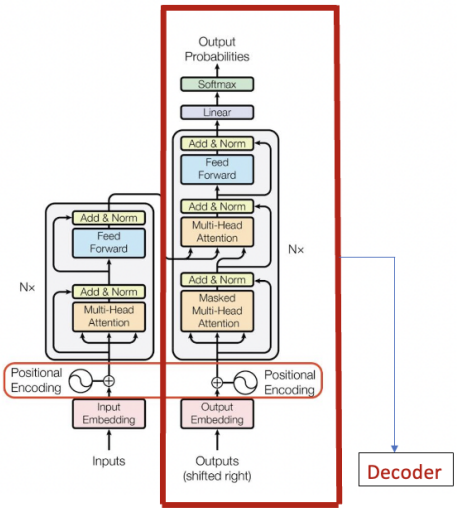



*Fig-3.2.1 Architecture of Transformer*

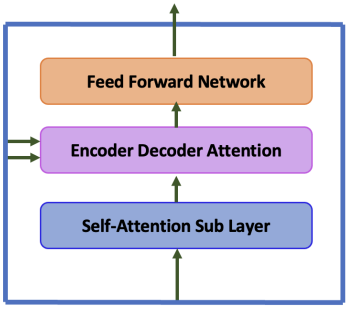
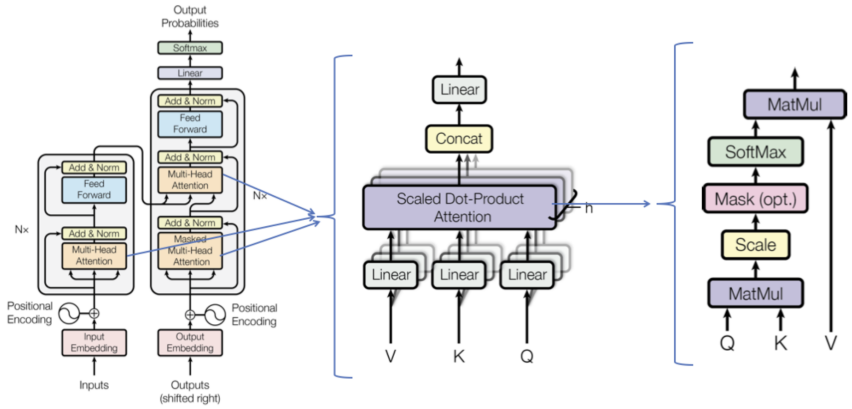



*Fig-3.2.2 Sublayers in each Stack of a decoder*

**Attention-** The attention mechanism helps the decoder focus on relevant parts of the sequence given in the input. In simple words, the self-attention mechanism helps the inputs of the model to allow the interaction inputs amongst themselves and determine the relevant parts to focus on. Each input has 3 representations namely key, value and query. Key, value, query and outputs are vectors.




The scaled dot product attention is calculated as

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where

- Q, K, V are the matrices where the set of queries, keys and values are packed respectively,
- $d_k$ is the dimension of the queries and keys
- $d_v$ is the dimension of the values
- $\text{Softmax}(z_i) = \frac{exp(z_i)}{\sum_{j=1}^{k} exp(z_j)}$ where $z=(z_1, z_2, \ldots. z_k)$

To prevent attending the subsequent positions, all the values of attending to illegal functions are set at -infinity at the input of the softmax to make the scaled dot product attention as zero i.e the subsequent positions are effectively masked and are not attended to while generating the output.

With multi-headed attention, we will have 8 (in case of transformers) sets of query, key and value weight matrices which are initialized randomly from a uniform distribution. Then each set of these matrices project the input embeddings or the embeddings obtained from previous stacks into a representational subspace. Multi headed attention concatenates the attention from all these subspaces and helps the model attend to all of them. Multi headed attention is thus calculated as

$$MultiHeadAttention(Q,K,V) = Concat(head_1, head_2, \ldots. head_H)W^O$$

where

- H= number of parallel attention layers
- $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$ *where the projection matrices* $W_i^Q \in R^{d_{model} * d_K}, W_i^K \in R^{d_{model} * d_K}, W_i^V \in R^{d_{model} * d_V}$ *and* $W_i^Q \in R^{Hd_V * d_{model}}$

  Here we use $d_v = d_k = \frac{d_{model}}{H}$=64

In the decoder layer the queries are obtained from the layer below and the key-value pairs are obtained from the encoder.

**Feed Forward Network and Softmax-**

Above the attention layers, there is a position-wise fully connected simple feed-forward network that consists of two linear transformations and a ReLu activation.

$$FeedForwardNetwork(z) = max(z W_1 + b_1, 0) W_2 + b_2$$

↑

ReLu Activation

Then a softmax function is used on the obtained decoder output to get the next output token probability. The highest probability output token is chosen and the word corresponding to this output is predicted as the word of this time-step. The same weight matrix is used in the 2 embedding layers and the linear transformation before softmax except that in the embedding layers the weights are also multiplied by $\sqrt{d_{model}}$

**Positional Encodings-**
Because every word flows into the decoder stack of the transformer simultaneously, we used positional encodings to incorporate the position of the word in the sentence to the model.

We generated positional encoding using the equation

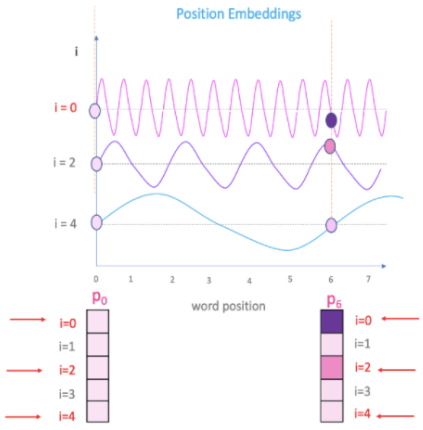
$$PositionalEncoding(p,2i) = sin(\frac{p}{10000^{(\frac{2i}{d})}})$$

and

$$PositionalEncoding(p,2i+1) = cos(\frac{p}{10000^{(\frac{2i}{d})}})$$

where

- p is the position of the word in the sentence
- d is the size of the encoding
- i is the current dimension and ranges between 0 to d-1 and each dimension corresponds to a sinusoid




Also, a dropout is applied to the sum of embeddings and positional encodings.

## 3.3 Training the model

We used a cross-entropy loss for training the model. Since the sequences are padded we also applied a padding mask while calculating the losses.

The cross-entropy is calculated as

$$L(\theta) = -\sum_{i=1}^{output\ size} y_i * log(\hat{y}_i)$$

$$where\ \hat{y}_i = predicted\ value(using\ softmax\ function)\ and\ y_i = target\ value$$

Adam optimizer was used to accelerate the gradient descent to find the parameters to minimize the loss. Also, a learning rate reducer was used to reduce the learning rate on the flow when the learning stagnates i.e when the loss stops reducing.

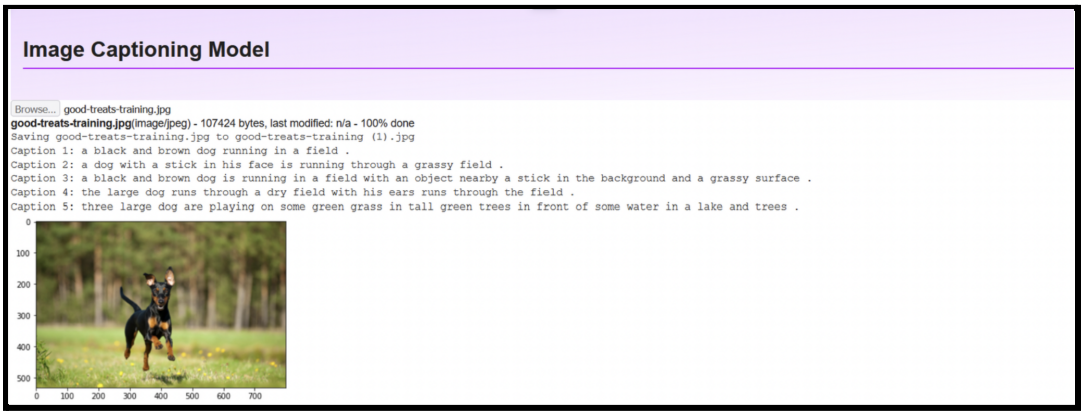
```
    image_embed = image_embed.squeeze(1).to(device)
    caption_seq = caption_seq.to(device)
    target_seq = target_seq.to(device)

    output, padding_mask = ictModel.forward(image_embed, caption_seq)
    output = output.permute(1, 2, 0)

    loss = criterion(output,target_seq)

    loss_masked = torch.mul(loss, padding_mask)

    final_batch_loss = torch.sum(loss_masked)/torch.sum(padding_mask)

    final_batch_loss.backward()
    optimizer.step()
    total_epoch_train_loss += torch.sum(loss_masked).detach().item()
    total_train_words += torch.sum(padding_mask)


total_epoch_train_loss = total_epoch_train_loss/total_train_words
tb.add_scalar('Train Loss',total_epoch_train_loss,epoch)
```

We further calculated the train and test loss for every epoch and the current model was saved only if the loss in the current epoch was lower than the previous epoch. And this saved model was finally used to generate the captions.

```
if min_val_loss > total_epoch_valid_loss:
    print("Writing Model at epoch ", epoch)
    torch.save(ictModel, './BestModel')
    min_val_loss = total_epoch_valid_loss
```

## 3.4 Generating the captions

Top k (here 5) captions were generated for each image. This was done by passing the image through the encoder network to obtain the embeddings and then passing these embeddings through the decoder network and obtaining the outputs.




**Top 5 captions generated by our model for this particular image*

# 4. Results

| SNo | Encoder CNN | Decoder Network | Image Augmentation Done | Number of Epochs (Till which testing loss decreased) | Learning Rate | Final training Loss (Cross Entropy) | Final testing Loss (Cross Entropy) |
|---|---|---|---|---|---|---|---|
| 1 | ResNet18 | Transformers | No | 15 | 0.00001 | 2.583574 | 2.907768 |
| 2 | ResNet18 | Transformers | Yes | 13 | 0.00001 | 1.988753 | 2.877248 |
| 3 | ResNet50 | Transformers | No | 6 | 0.00001 | 2.341792 | 2.984258 |

## 4.1 ResNet18 + Transformers(Without Data Augmentation):-

First to test out our model as a whole from encode to decode we decided to train the ResNet 18 model without any data augmentation. Training took 47 minutes, with 7282 images in the train set and 810 images in the test set. The train and test loss at the end of the first epoch were 4.9430 and 3.9946, respectively. After training for 15 epochs with a learning rate of$1 \times 10^{-5}$ and a scheduler which will reduce the learning rate by 20% if the loss plateaus or increases for 2 epochs, the train and test losses are reduced to 2.5835 and 2.9077 respectively.


**Train and test loss of ResNet18+Transformers model over the epochs.*

The final output model came out to be 100 megabytes large, which was the smallest compared to other models we got—testing the model on test set images gave satisfactory results considering the size of the trained model and as no image augmentations were used.

### 4.1.1 Results:-

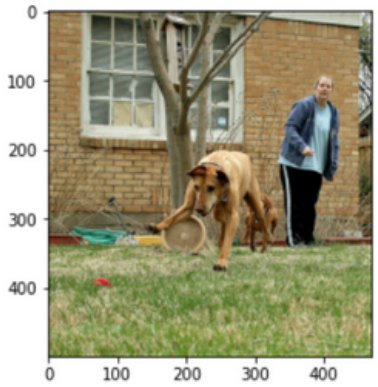
**Example 1:**

Actual captions:

- A woman watches a brown dog run away from a house across the grass.
- A woman throws a Frisbee for her two brown dogs to chase.
- brown dogs and a woman in a yard.
- A brown dog persues a Frisbee across the grass as the thrower watches.
- A woman in a blue jacket watches as her two brown dogs play with a red ball in a grassy yard.

Generated Caption:

```
Predicted caption :
a brown dog is jumping up to catch a toy in the air .
```




**Caption generated for an image in the test set.*

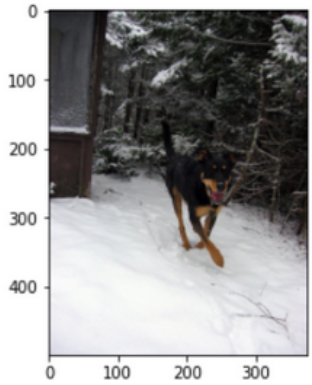
**Example 2:**

Actual captions:

- A black and brown dog walks through the snow near a building .
- A black dog , running in the snow.
- A black dog running in the snow by some trees.
- A large black and tan dog is running across the snow in a wooded area.
- The black and brown dog is running through the snow.

Generated caption:

```
Predicted caption :
a black dog is running in the snow .
```




**Caption generated for an image in the test set.*

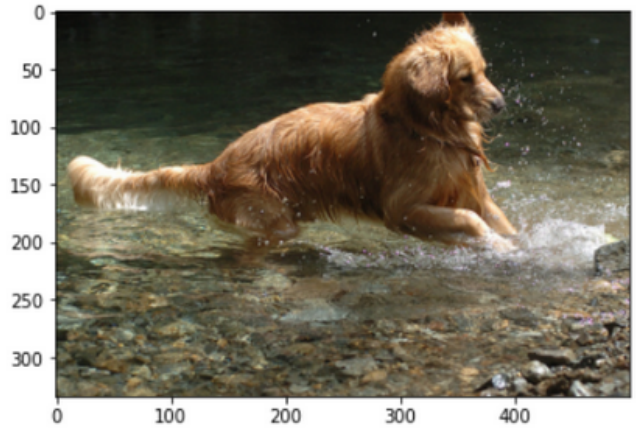
**Example 3:**

Actual captions:

- A brown dog is running though a river.
- a brown dog jumping in a stoney brook.
- A brown dog runs through shallow water.
- A dog wading through shallow water.
- a golden retriever splashes in the water.

Generated caption:-

```
Predicted caption :
a brown dog is running through the water .
```




**Caption generated for an image in the test set.*

Performance on random uploaded images was not as up to the mark as in the test set but was acceptable. Overall, the model was unpredictable; sometimes, sometimes generated captions which did not make any sense and sometimes described things which were not present. Here is an example of a caption generated from a random image we uploaded to the model.

```
Predicted Caption: the white puppy with brown hair has its teeth .
```


**Caption generated for a random image uploaded to the model.*

# 4.2 ResNet18 + Transformers (With Data Augmentation):-

We decided to train the ResNet18 model with a dataset that includes only three augmentations and original images that were centre cropping, brightening and blurring as we were limited with both the storage and primary memory(RAM) on Google Colab. The training took almost an hour, with 19419

images in the train set and 4855 images in the test set. The train and test loss at the end of the first epoch were 4.2011 and 3.5387, respectively. After training for 15 epochs with a learning rate of $1 \times 10^{-5}$ and a scheduler which will reduce the learning rate by 20% if the loss plateaus or increases for 2 epochs, the train and test losses are reduced to 1.9280 and 2.8861 respectively. Training for more epochs will not have helped as the loss started to plateau, so we decided to stop at 15 epochs which gave an outstanding balance between reducing loss and overfitting.


**Train and test loss of ResNet18+Transformers model with image augmentation over the epochs.*

The final output model came out to be 122 megabytes large, which was quite small compared to other models we got—testing the model on test set images gave good results considering the size of the trained model.

### 4.2.1 Results:

**Example 1:**

Actual captions:

- A biker makes a high jump.
- A man doing an aerial stunt on a bike.
- A man on a bicycle performing a jump.
- A person on a bicycle wears a helmet and jumps , soaring over the ground in front of a skyline and lake.
- The cyclist is making a high jump with his bike .

Generated Caption

```
Predicted caption :
a man in a blue uniform is airborne on a bike .
```


**Caption generated for an image in the test set.*

**Example 2:**

Actual captions:

- A dirt biker competes while onlookers watch.
- A man is performing a midair stunt on a dirt bike while people in the background watch.
- A man in a white jacket and helmet does tricks on his bike.
- A mountain bike racer jumps his bike as spectators watch.
- A bicyclist jumps their bike while a crowd spread across a hillside watches.

Generated Caption:

```
Predicted caption :
a man is riding a bike on a dirt path .
```


**Caption generated for an image in the test set.*

**Example 3:**

Actual captions:

- Two children are in a pool with a beach ball floating nearby.
- The boy laughed as he swam in the pool.
- Children play in an inflatable pool.
- A young boy plays around in a jacuzzi.
- A boy with a beach ball behind him playing in a pool.

Generated Caption:

```
Predicted caption :
a boy and a girl are playing in a pool .
```


**Caption generated for an image in the test set.*

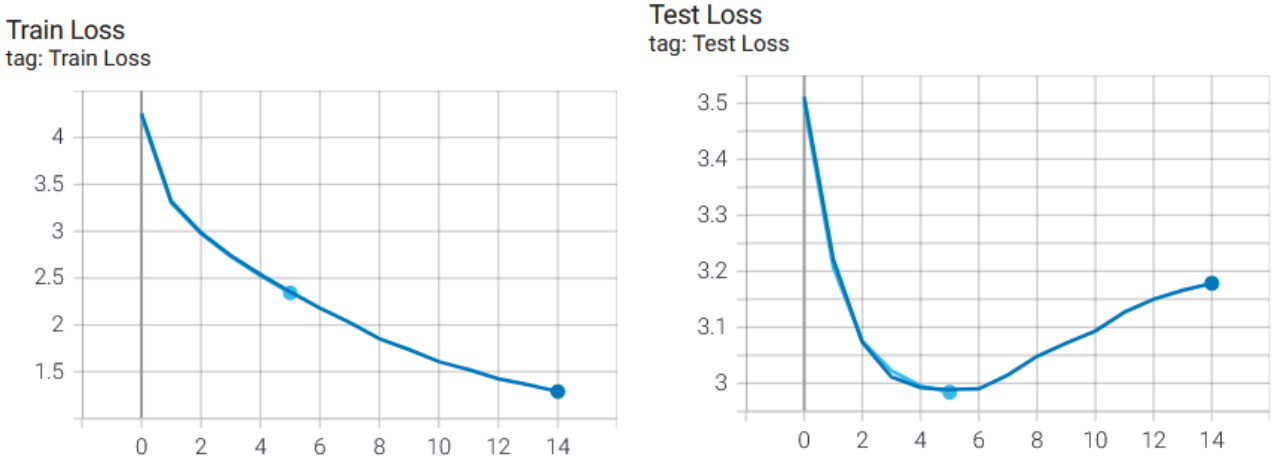
Performance on random uploaded images was not as up to the mark as in the test set but was acceptable. Overall, the model was a little unpredictable; sometimes, the model generated random words, which did not make a cohesive sentence when put together but sometimes generated exceptional captions. Here is an example of a caption generated from a random image we uploaded to the model.

```
Caption 1: a black and brown dog running in a field .
Caption 2: a dog with a stick in his face is running through a grassy field .
Caption 3: a black and brown dog is running in a field with an object nearby a stick in the background and a grassy surface .
Caption 4: the large dog runs through a dry field with his ears runs through the field .
Caption 5: three large dog are playing on some green grass in tall green trees in front of some water in a lake and trees .
```


**Caption generated for a random image uploaded to the model.*

## 4.3 ResNet50 + Transformers (Without Data Augmentation):-

We decided to train the ResNet50 model without any augmentation. The hardware limited us as we were working on Google Colab. As the image embeddings we got from the ResNet50 model were of the dimension of $2048 \times 7 \times 7$ which is four times larger than the embedding obtained from ResNet18, it saturated the computational power we had by itself, so we could not add any image augmentation. The training took 51 minutes and 48 seconds with 6473 images in the train set and 1619 images in the test set. The train and test losses at the start of training were 4.2255 and 3.4986, respectively. After training for six epochs, the train and test losses were reduced to 2.3417 and 2.9842, respectively. We decided to stop the training after six epochs as the test loss started to increase starting from the 7th epoch, which implied the model began to overfit on the train set.




**Train and test loss of ResNet50+Transformers model over the epochs*

The final model came out to be 1.02 gigabytes large which is significantly larger than the ResNet18 model. The model performed well with images from the test set, as was the case with the ResNet18+Transformers model.

### 4.3.1 Results:

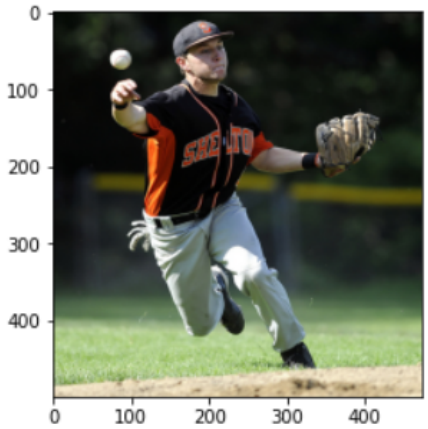
**Example 1:**

Actual captions:

- The baseball player is running after the ball.
- A male dressed in a sports outfit tries to catch a baseball.
- A man playing baseball.
- A baseball pitcher with Shelton on his shirt throws the ball.
- A baseball player in a black and orange shirt , and with a glove on , pitches the ball.

Generated Caption:

```
Predicted caption :
a baseball player in a red uniform is throwing the ball .
```




**Caption generated for an image in the test set.*

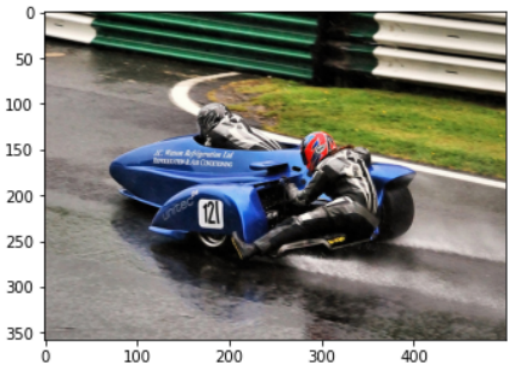
**Example 2:**

Actual captions:

- A blue motorcycle and sidecar is being raced on a wet track.
- A man races in a small vehicle while another man hangs off the back.
- a racer in a blue car towing a man in a black suit on the street.
- Racing on wet road while man hangs on back.
- Two people on a blue motorcycle.

Generated Caption:

```
Predicted caption :
a man in a blue helmet is driving a motorcycle .
```




**Caption generated for an image in the test set.*

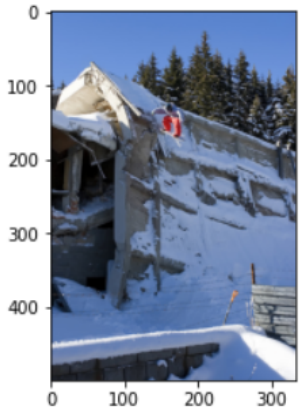
**Example 3:**

Actual captions:

- A lone snowboarder jumping in midair with a snow covered mine in the background.
- A man in orange pants is climbing on something.
- A snowboarder jumps off an old , snow-covered building.
- Someone is ski-ing off the top of some type of building.
- The snowboarder is jumping from a snow covered height.

Generated Caption:

```
Predicted caption :
a snowboarder is jumping over a snow covered rail .
```




**Caption generated for an image in the test set.*

# 5.Conclusion

Overall the model performed similarly to the ResNet18+Transformers model with slightly better results in recognising people. The model had the same problem with generating images for random images uploaded to the model. That is, some generated captions were incoherent and did not form a cohesive sentence. Here is an example of a caption generated from a random image we uploaded to the model.

```
Caption 1: a black and brown dog running through a field .
Caption 2: the black and black dog is running through the field .
Caption 3: black dog runs on a grassy green grass .
Caption 4: the dog with the toy on a lawn runs through grass and grass in his hand in front .
Caption 5: a brown black dog and black lab playing on a dirt .
```


**Caption generated for a random image uploaded to the model.*

### Scope for improvement:-

- As we were limited by the hardware of Google Colab, given that we have sufficient resources the Resnet50+Transformers model can be improved by adding image augmentations to the dataset that can create variations thus reducing the chance of overfitting during training and allowing us to train the model for a longer duration and thereby reducing the loss.
- The vocabulary obtained from the flickr 8K dataset was limited and restricted the predictive power of the model..

# 6. References


**[1]**Peng, J., Kang, S., Ning, Z., & Deng, H. (2019). *Architecture of ResNet50 model* [Graph]. Residual Convolutional Neural Network for Predicting Response of Transarterial Chemoembolization in Hepatocellular Carcinoma from CT Imaging July 2019European Radiology 30(5):1–12. https://www.researchgate.net/figure/The-architecture-of-ResNet50-and-deep-learning-model-flowchart-a-b-Architecture-of_fig1_334767096

**[2]**Phi, M. (2020, June 28). *Illustrated Guide to Transformers- Step by Step Explanation*. Medium. https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0

**[3]***ResNet-50 architecture [26] shown with the residual units, the size of the filters and the outputs of each convolutional layer.* (2020, January). [Graph]. Automatic Hierarchical Classification of Kelps Using Deep Residual Features. https://www.researchgate.net/figure/ResNet-50-architecture-26-shown-with-the-residual-units-the-size-of-the-filters-and_fig1_338603223

**[4]**Rosebrock, A. (2021, June 17). *ImageNet: VGGNet, ResNet, Inception, and Xception with Keras*. PyImageSearch. https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

**[5]**Roy, A. (2020, December 11). *A Guide to Image Captioning - Towards Data Science*. Medium. https://towardsdatascience.com/a-guide-to-image-captioning-e9fd5517f350

**[6]**Srinivasan, L., Sreekanthan, D., & A.L, A. (2018). Image captioning - a deep learning approach. *International Journal of Applied Engineering Research*, *13*. https://www.ripublication.com/ijaer18/ijaerv13n9_102.pdf

**[7]**Tian, Y., & Li, T. (2016). *Project final report*. Stanford University. http://cs231n.stanford.edu/reports/2016/pdfs/364_Report.pdf

**[8]***Transformer model for language understanding | Text |*. (2021). TensorFlow. https://www.tensorflow.org/text/tutorials/transformer

**[9]**Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is all you need*. Neural Information Processing Systems, Long Beach, CA, USA. https://dl.acm.org/doi/epdf/10.5555/3295222.3295349

**[10]***What is the positional encoding in the transformer model?* (2019, April 28). Data Science Stack Exchange. https://datascience.stackexchange.com/questions/51065/what-is-the-positional-encoding-in-the-transformer-model

# 7. Glossary

- CNN - Convolutional Neural Network
- HSV - Hue, saturation, value which is an alternative representation of the RGB color model
- LSTM - Long Short Term Memory networks
- ReLu Function - Rectified Linear Activation function
- ResNet - Residual Networks
- RGB- It is an additive color model where the various colors are obtained by adding the colors Red, Green and Blue
- SoftMax Function - Softargmax or Normalized exponential function
- Tensor - Multidimensional matrix consisting of elements of  a single data-type