MLA Training Summer 2022

Final Project

Image Captioning

Supervised by:

Dr. Marwan Turki

Eng. Adel Atef

Team Norm

Mennatallah Moustafa	6234
Rawan Hindawy	6491
Nouran Hisham	6532

Table of contents:

→Problem statement ……………………………………………………………………………………………………….

→Applications on image captioning ……………………………………………………………………………….

→Datasets ………

→Work using flickr8k dataset

→Loading the data …………………………………………………………………………………………….

→Data description

→Size of data ………………………………………………………………………………………….

→Data visualization …………………………………………………………………………………..

→The most and least common words …………………………………………………………

→Data cleaning ………………………………………………………………………………………………….

→Feature extraction using ResNet50 …………………………………………………………………

→Train/Test split ……………………………………………………………………………………………….

→Data pre-processing ………………………………………………………………………………………..

→LSTM model ………………………………………………………………………………………………………

→Testing process ……………………………………………………………………………………………….

→Work using the flickr30k dataset

→Captions format ………………………………………………………………………………………………

→Train/Test split ……………………………………………………………………………………………….

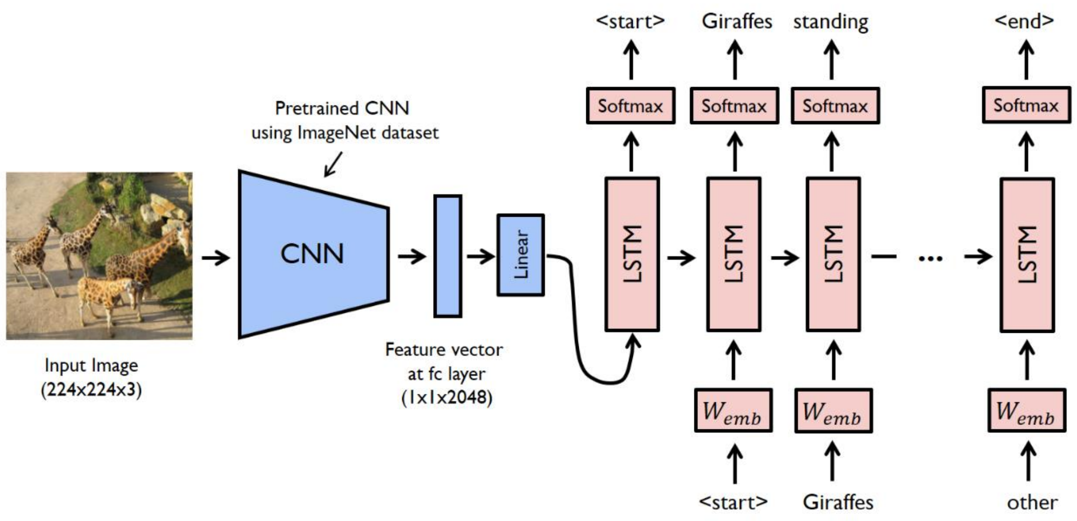
→LSTM model ………………………………………………………………………………………………………

→Links ………

→References ……

Problem Statement:

Image Captioning is the task of describing the content of an image in words. This task lies at the intersection of computer vision and natural language processing. Most image captioning systems use an encoder-decoder framework, where an input image is encoded into an intermediate representation of the information in the image, and then decoded into a descriptive text sequence.



Applications on Image Captioning:

- Probably, will be useful in cases/fields where text is most used and with the use of this, you can infer/generate text from images. As in, use the information directly from any image in a textual format automatically.

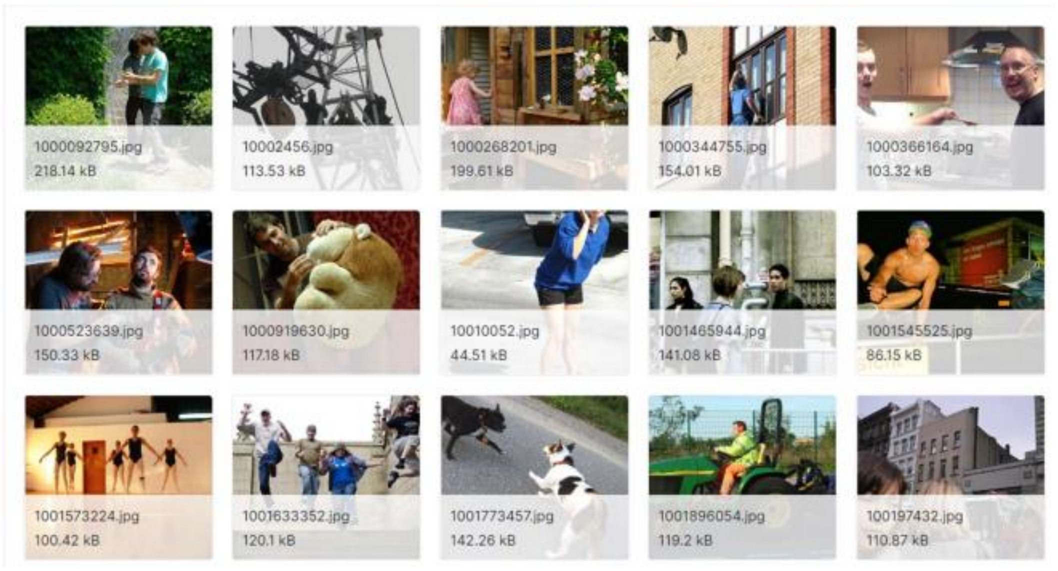
- Would serve as a huge help for visually impaired people. Lots of applications can be developed in that space.

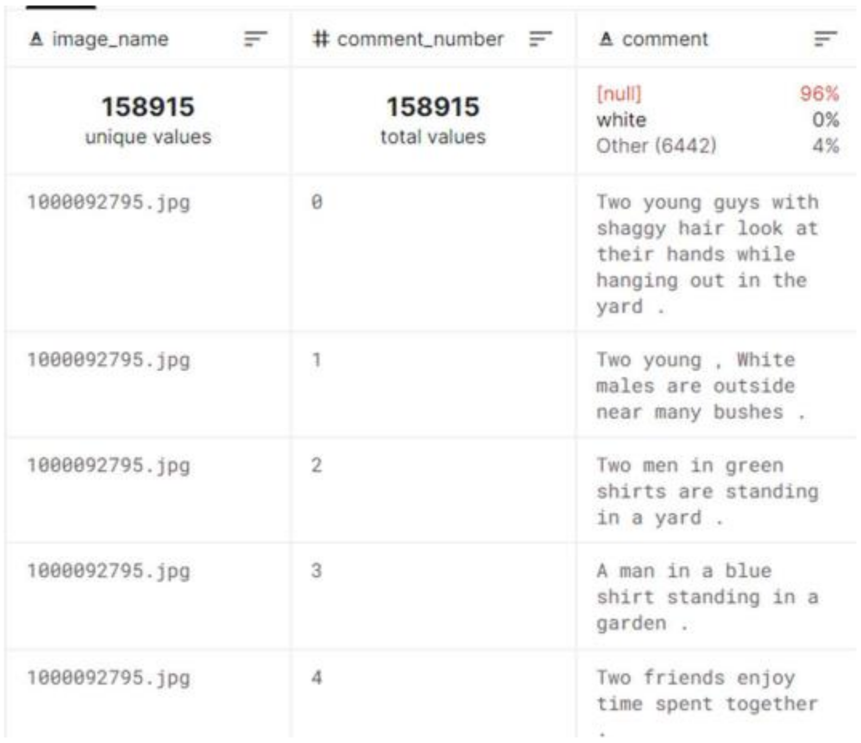
- Social Media. Platforms like Facebook can infer directly from the image, where you are (beach, cafe etc.), what you wear (colour) and more importantly what you're doing also (in a way). See an example to understand it better.

Datasets:

We initially wanted to use the flickr30k dataset. Here's a brief description of it:

- The Flickr30k dataset has become a standard benchmark for sentence-based image description. This dataset presents Flickr30k Entities, which augments the 158k captions from Flickr30k with 244k coreference chains, linking mentions of the same entities across different captions for the same image, and associating them with 276k manually annotated bounding boxes. Such annotations are essential for continued progress in automatic image description and grounded language understanding.





image_name	# comment_number	comment
158915 unique values	158915 total values	[null] 96% white 0% Other (6442) 4%
1000092795.jpg	0	Two young guys with shaggy hair look at their hands while hanging out in the yard .
1000092795.jpg	1	Two young , White males are outside near many bushes .
1000092795.jpg	2	Two men in green shirts are standing in a yard .
1000092795.jpg	3	A man in a blue shirt standing in a garden .
1000092795.jpg	4	Two friends enjoy time spent together .

Due to its huge memory requirements which we, unfortunately, weren't able to fully meet, we had to use the flickr8k dataset, which is exactly the same, but the only difference is that it has about 8,000 pictures instead of 30,000.

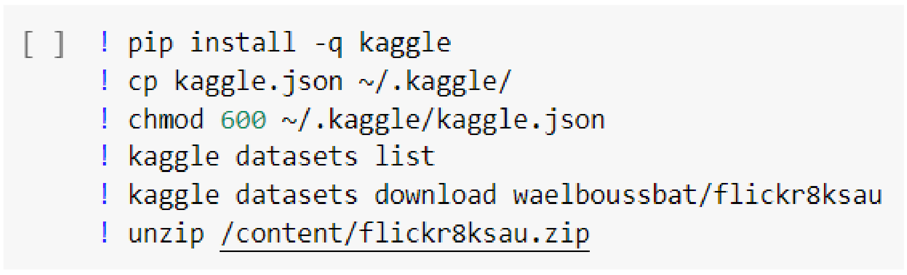
We ended up working mainly on the flickr8k dataset, but we achieved some progress on the flickr30k dataset by trying to at least use half of it, around 15,000 pictures, here are all our trials:

- The free google colab access offers only 12GB of RAM which wasn't enough and has always been all used up during the pre-processing of the data, so we made a colab pro account that offers 25GB of RAM which unfortunately still wasn't enough.
- Afterwards, we tried to divide the data into 4 batches, each of about 7,000 images and captions, perform the pre-processing on each batch separately then upload it to our drive, which was completely empty and had 15GB of storage space. The drive was able to only save 2/4 batches, which took about 12.5/15GB available.
- Finally, we decided to work with only half the dataset, we divided it into 2 batches, performed the pre-processing on one batch only and fed it into our LSTM model.

Work using flickr8k dataset

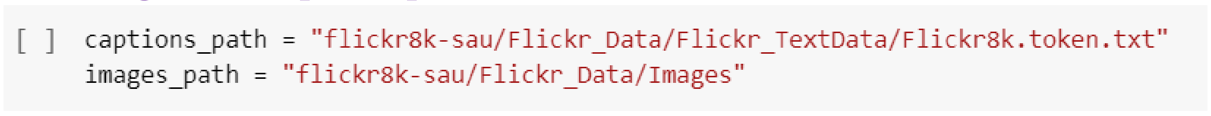
Loading the data:

We used Kaggle's API tokens to connect google colab to Kaggle and be able to use the dataset and work with it.



```
[ ]  ! pip install -q kaggle
     ! cp kaggle.json ~/.kaggle/
     ! chmod 600 ~/.kaggle/kaggle.json
     ! kaggle datasets list
     ! kaggle datasets download waelboussbat/flickr8ksau
     ! unzip /content/flickr8ksau.zip
```

The images and captions paths were as follows:



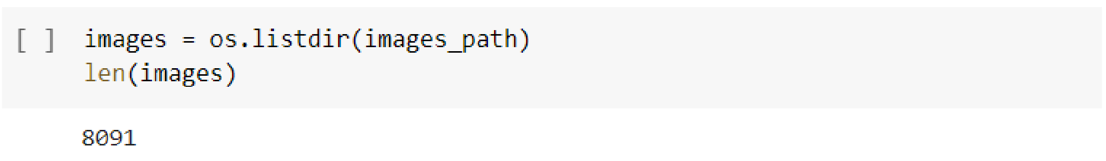
```
[ ]  captions_path = "flickr8k-sau/Flickr_Data/Flickr_TextData/Flickr8k.token.txt"
     images_path = "flickr8k-sau/Flickr_Data/Images"
```

Data description:

Size of data:

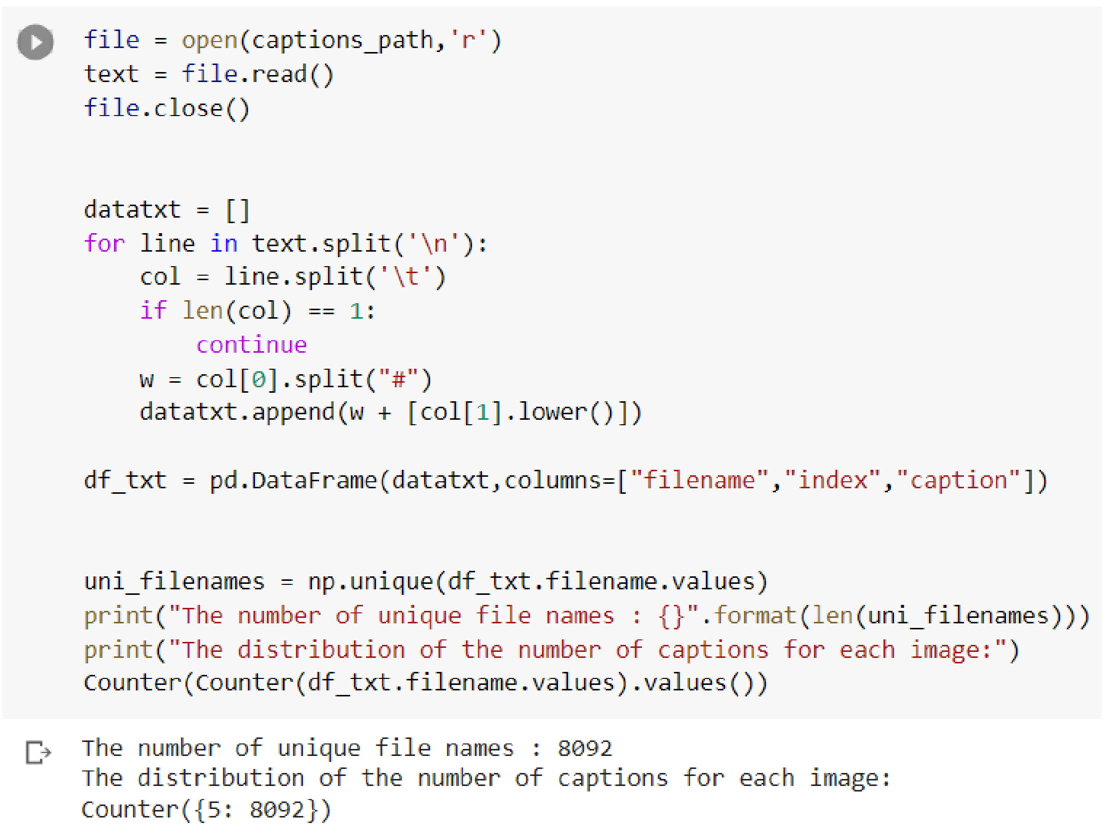
The dataset consists of 8091 images and each image has 5 captions describing it.

The captions text file has the filename, next to it (#), then the index of the caption between 0 and 4 (5 captions) then a space tab and the caption itself, that's how we were able to separate the filename, index and caption.



```
[ ]  images = os.listdir(images_path)
     len(images)
```

```
8091
```

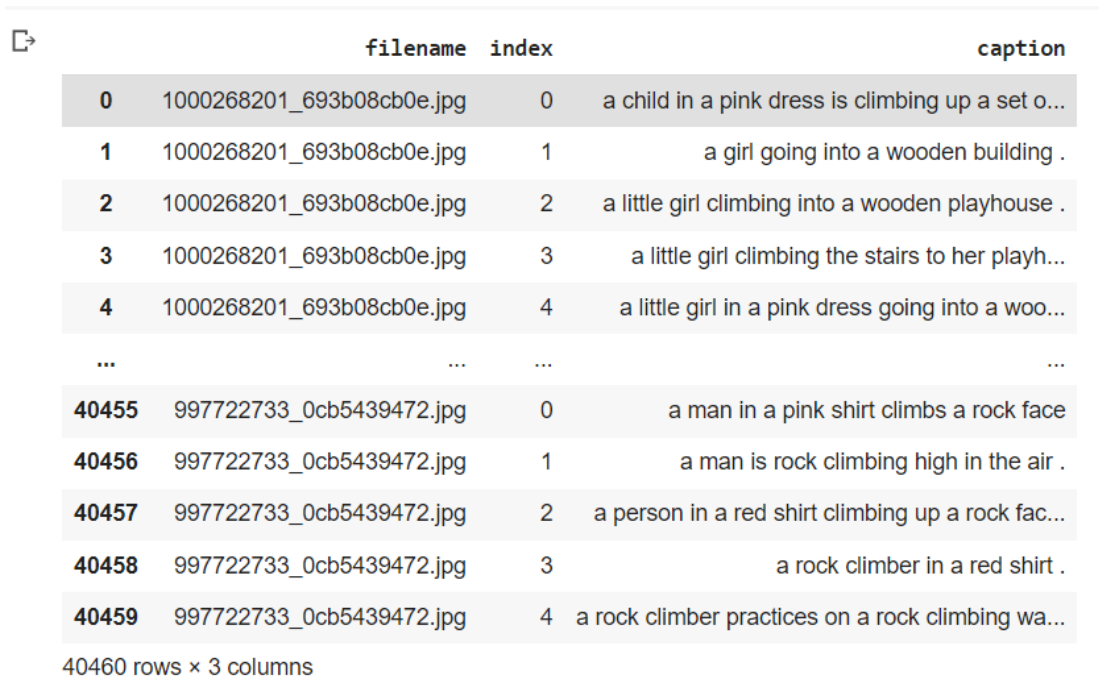


```
file = open(captions_path,'r')
text = file.read()
file.close()


datatxt = []
for line in text.split('\n'):
    col = line.split('\t')
    if len(col) == 1:
        continue
    w = col[0].split("#")
    datatxt.append(w + [col[1].lower()])

df_txt = pd.DataFrame(datatxt,columns=["filename","index","caption"])


uni_filenames = np.unique(df_txt.filename.values)
print("The number of unique file names : {}".format(len(uni_filenames)))
print("The distribution of the number of captions for each image:")
Counter(Counter(df_txt.filename.values).values())
```

```
The number of unique file names : 8092
The distribution of the number of captions for each image:
Counter({5: 8092})
```

Data visualization:

The captions after being separated from the filename and index:



	filename	index	caption
0	1000268201_693b08cb0e.jpg	0	a child in a pink dress is climbing up a set o...
1	1000268201_693b08cb0e.jpg	1	a girl going into a wooden building .
2	1000268201_693b08cb0e.jpg	2	a little girl climbing into a wooden playhouse .
3	1000268201_693b08cb0e.jpg	3	a little girl climbing the stairs to her playh...
4	1000268201_693b08cb0e.jpg	4	a little girl in a pink dress going into a woo...
...	...	...	...
40455	997722733_0cb5439472.jpg	0	a man in a pink shirt climbs a rock face
40456	997722733_0cb5439472.jpg	1	a man is rock climbing high in the air .
40457	997722733_0cb5439472.jpg	2	a person in a red shirt climbing up a rock fac...
40458	997722733_0cb5439472.jpg	3	a rock climber in a red shirt .
40459	997722733_0cb5439472.jpg	4	a rock climber practices on a rock climbing wa...

40460 rows × 3 columns

The images with their 5 corresponding captions:

```python
count = 1
fig = plt.figure(figsize=(10,20))
for jpgfnm in uni_filenames[:10]:
    filename = images_path + '/' + jpgfnm
    captions = list(df_txt["caption"].loc[df_txt["filename"]==jpgfnm].values)

    image_load = load_img(filename, target_size=(224,224,3))

    ax = fig.add_subplot(10,2,count,xticks=[],yticks=[])
    ax.imshow(image_load)
    count += 1

    ax = fig.add_subplot(10,2,count)
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0,1)
    ax.set_ylim(0,len(captions))
    for i, caption in enumerate(captions):
        ax.text(0,i,caption,fontsize=20)
    count += 1
plt.show()
```

```
a little girl in a pink dress going into a wooden cabin .
a little girl climbing the stairs to her playhouse .
a little girl climbing into a wooden playhouse .
a girl going into a wooden building .
a child in a pink dress is climbing up a set of stairs in an entry way .
```

```
two dogs on pavement moving toward each other .
two dogs of different breeds looking at each other on the road .
a black dog and a white dog with brown spots are staring at each other in the street .
a black dog and a tri-colored dog playing with each other on the road .
a black dog and a spotted dog are fighting
```

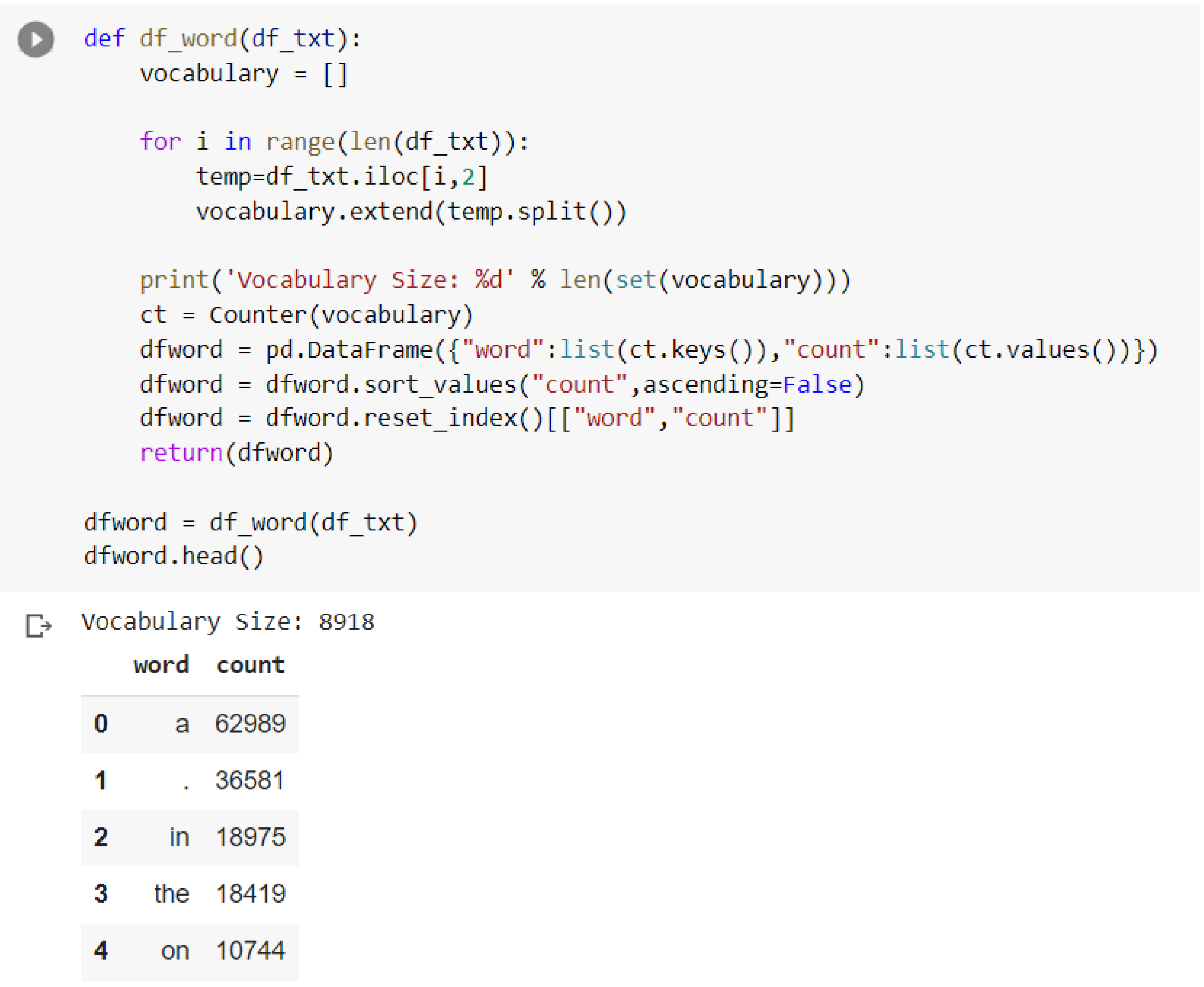
```
young girl with pigtails painting outside in the grass .
there is a girl with pigtails sitting in front of a rainbow painting .
a small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
a little girl is sitting in front of a large painted rainbow .
a little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .
```

```
man laying on bench holding leash of dog sitting on ground
a shirtless man lies on a park bench with his dog .
a man sleeping on a bench outside with a white and black dog sitting next to him .
a man lays on the bench to which a white dog is also tied .
a man lays on a bench while his dog sits by him .
```

```
the man with pierced ears is wearing glasses and an orange hat .
a man with glasses is wearing a beer can crocheted hat .
a man with gauges and glasses is wearing a blitz hat .
a man wears an orange hat and glasses .
a man in an orange hat starring at something .
```

The most and least common words:

We first made a data frame of all the words in all the captions, counted how many times each one of them is used and sorted the values.

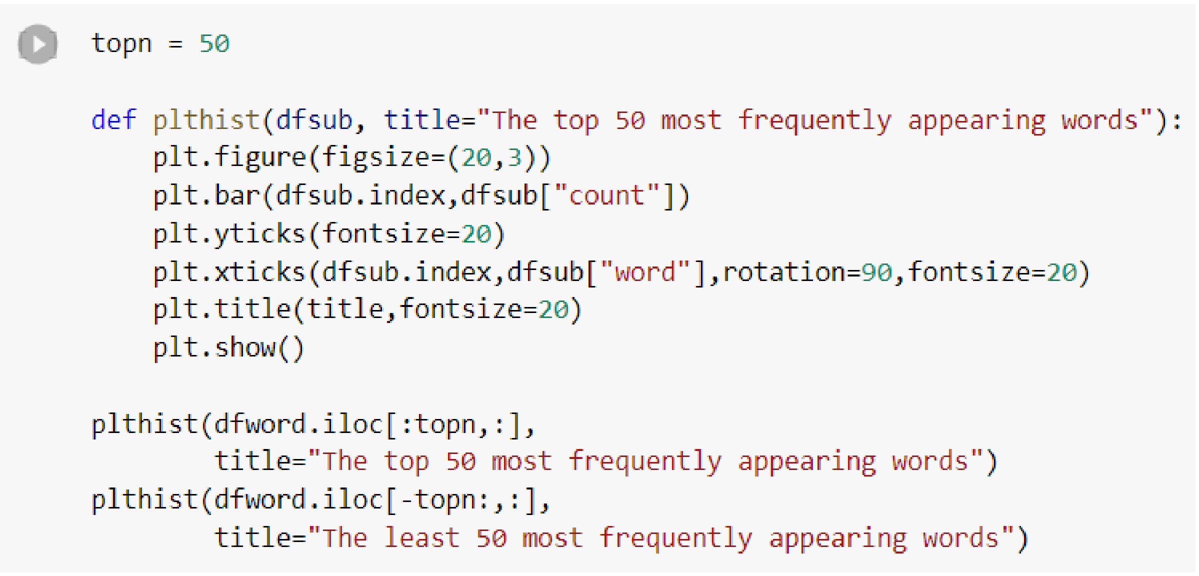


```python
def df_word(df_txt):
    vocabulary = []

    for i in range(len(df_txt)):
        temp=df_txt.iloc[i,2]
        vocabulary.extend(temp.split())

    print('Vocabulary Size: %d' % len(set(vocabulary)))
    ct = Counter(vocabulary)
    dfword = pd.DataFrame({"word":list(ct.keys()),"count":list(ct.values())})
    dfword = dfword.sort_values("count",ascending=False)
    dfword = dfword.reset_index()[["word","count"]]
    return(dfword)

dfword = df_word(df_txt)
dfword.head()
```

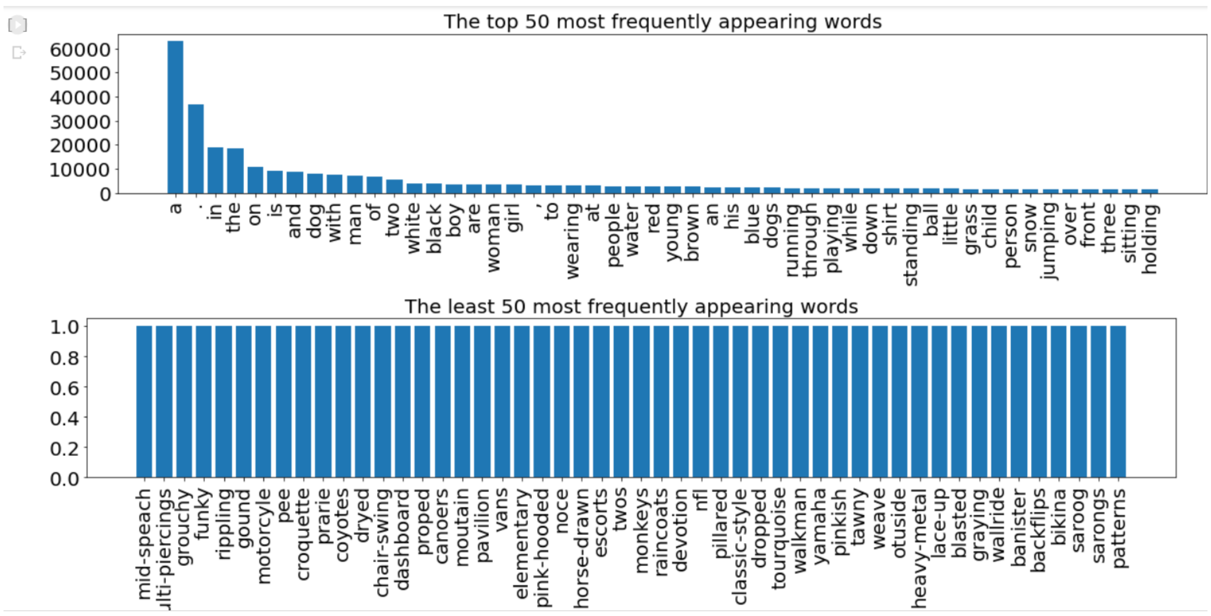
```
Vocabulary Size: 8918
```

	word	count
0	a	62989
1	.	36581
2	in	18975
3	the	18419
4	on	10744

We then plotted the first and last 50 values in the data frame to represent the most and least common words, respectively.

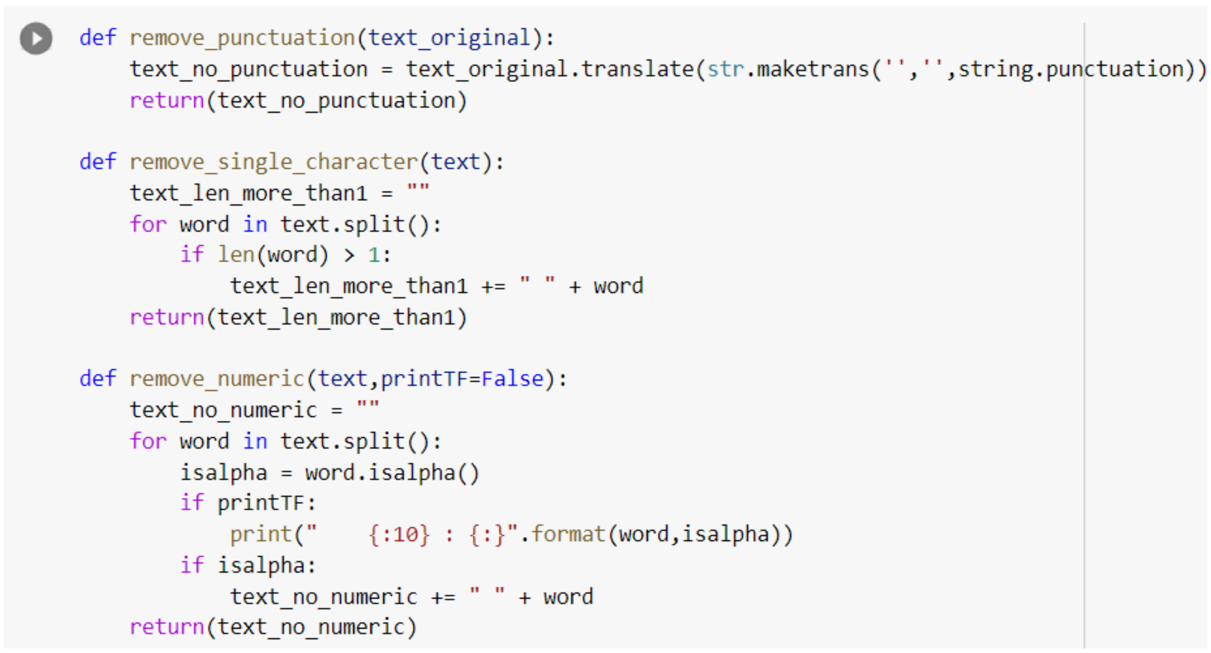


```python
topn = 50

def plthist(dfsub, title="The top 50 most frequently appearing words"):
    plt.figure(figsize=(20,3))
    plt.bar(dfsub.index,dfsub["count"])
    plt.yticks(fontsize=20)
    plt.xticks(dfsub.index,dfsub["word"],rotation=90,fontsize=20)
    plt.title(title,fontsize=20)
    plt.show()

plthist(dfword.iloc[:topn,:],
        title="The top 50 most frequently appearing words")
plthist(dfword.iloc[-topn:,:],
        title="The least 50 most frequently appearing words")
```

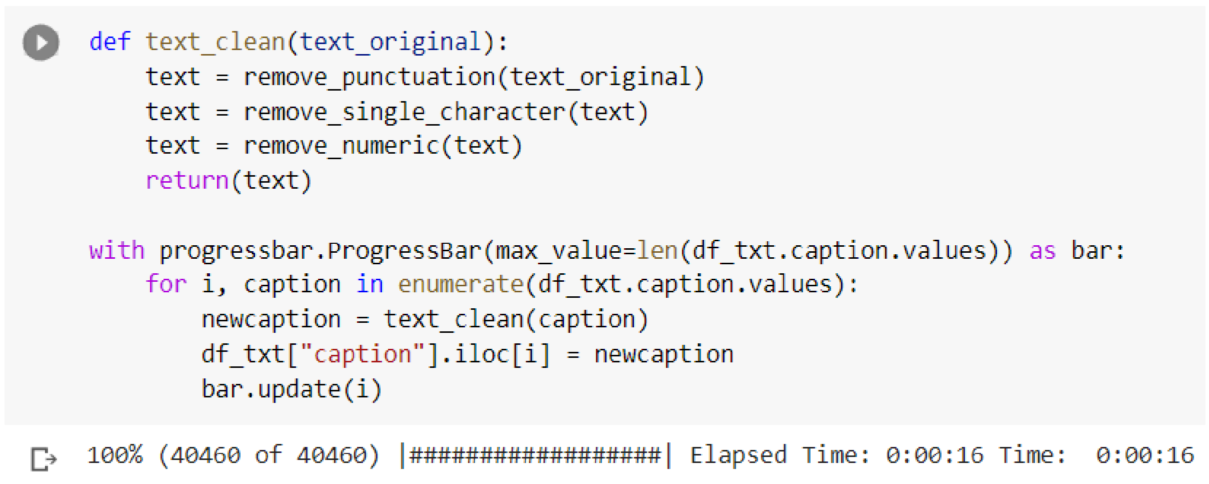


Data cleaning:

Data cleaning has an important role in getting good accuracies in NLP (natural language processing) projects, as things like punctuations or digits affect the results of any text processing approach, especially what depends on the occurrence frequencies of words and phrases.



```
def remove_punctuation(text_original):
    text_no_punctuation = text_original.translate(str.maketrans('','',string.punctuation))
    return(text_no_punctuation)

def remove_single_character(text):
    text_len_more_than1 = ""
    for word in text.split():
        if len(word) > 1:
            text_len_more_than1 += " " + word
    return(text_len_more_than1)

def remove_numeric(text,printTF=False):
    text_no_numeric = ""
    for word in text.split():
        isalpha = word.isalpha()
        if printTF:
            print("    {:10} : {:}".format(word,isalpha))
        if isalpha:
            text_no_numeric += " " + word
    return(text_no_numeric)
```

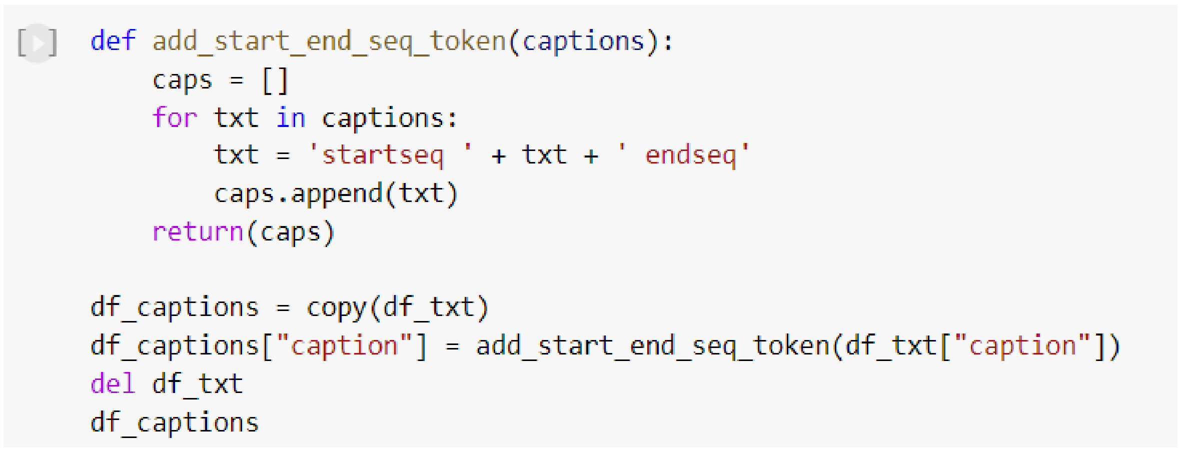


```
def text_clean(text_original):
    text = remove_punctuation(text_original)
    text = remove_single_character(text)
    text = remove_numeric(text)
    return(text)

with progressbar.ProgressBar(max_value=len(df_txt.caption.values)) as bar:
    for i, caption in enumerate(df_txt.caption.values):
        newcaption = text_clean(caption)
        df_txt["caption"].iloc[i] = newcaption
        bar.update(i)
```

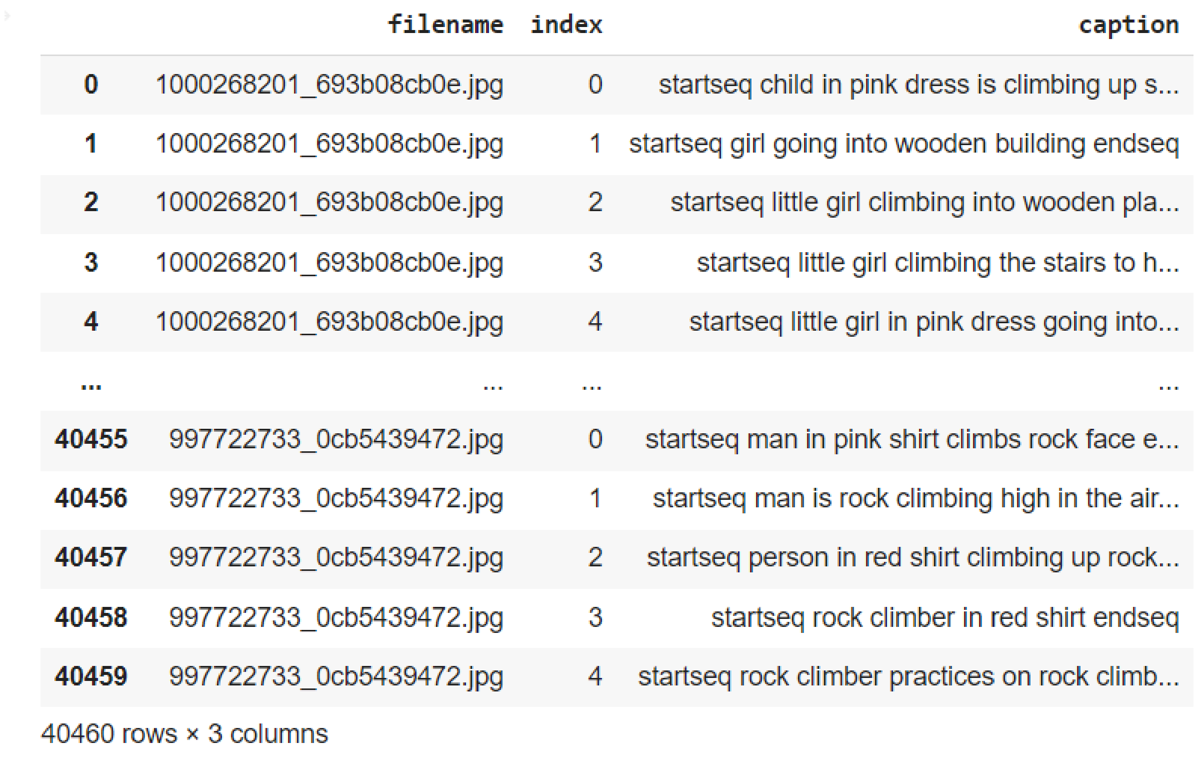
```
100% (40460 of 40460) |##################| Elapsed Time: 0:00:16 Time:  0:00:16
```

The encoder reads the input sequence to construct an embedding representation of the sequence. Terminating the input in an end-of-sequence token signals to the encoder that when it receives that input, the output needs to be the finalized embedding.

The decoder will progress by taking the tokens it emits as inputs, along with the embedding and hidden state, or using the embedding to initialize the hidden state, so before it has emitted anything it needs a token of some kind to start with, which is the start-of-sequence token.



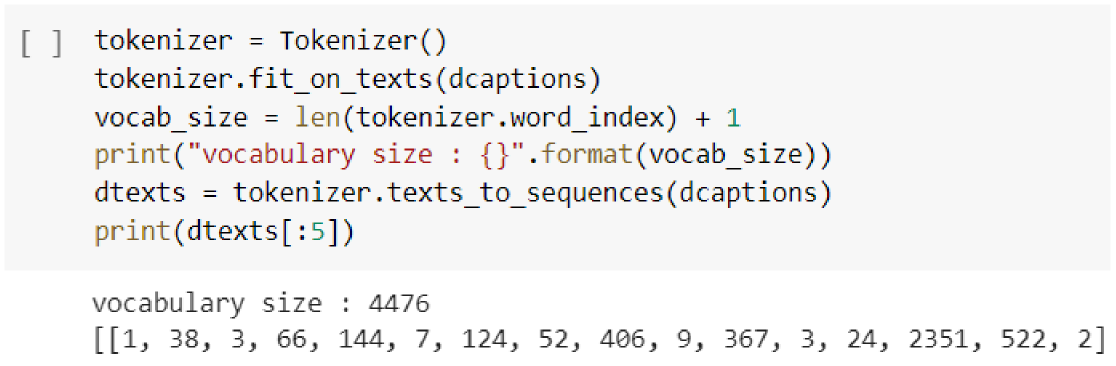
```
def add_start_end_seq_token(captions):
    caps = []
    for txt in captions:
        txt = 'startseq ' + txt + ' endseq'
        caps.append(txt)
    return(caps)

df_captions = copy(df_txt)
df_captions["caption"] = add_start_end_seq_token(df_txt["caption"])
del df_txt
df_captions
```



	filename	index	caption
0	1000268201_693b08cb0e.jpg	0	startseq child in pink dress is climbing up s...
1	1000268201_693b08cb0e.jpg	1	startseq girl going into wooden building endseq
2	1000268201_693b08cb0e.jpg	2	startseq little girl climbing into wooden pla...
3	1000268201_693b08cb0e.jpg	3	startseq little girl climbing the stairs to h...
4	1000268201_693b08cb0e.jpg	4	startseq little girl in pink dress going into...
...	...	...	...
40455	997722733_0cb5439472.jpg	0	startseq man in pink shirt climbs rock face e...
40456	997722733_0cb5439472.jpg	1	startseq man is rock climbing high in the air...
40457	997722733_0cb5439472.jpg	2	startseq person in red shirt climbing up rock...
40458	997722733_0cb5439472.jpg	3	startseq rock climber in red shirt endseq
40459	997722733_0cb5439472.jpg	4	startseq rock climber practices on rock climb...

40460 rows × 3 columns

Tokenization is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning. Our tokenizer is used to convert a set of training data, the captions, into a dictionary where each unique word gets a different ID.



```
[ ]  tokenizer = Tokenizer()
     tokenizer.fit_on_texts(dcaptions)
     vocab_size = len(tokenizer.word_index) + 1
     print("vocabulary size : {}".format(vocab_size))
     dtexts = tokenizer.texts_to_sequences(dcaptions)
     print(dtexts[:5])

     vocabulary size : 4476
     [[1, 38, 3, 66, 144, 7, 124, 52, 406, 9, 367, 3, 24, 2351, 522, 2]
```

Feature Extraction using ResNet:

What is feature extraction?

Feature Extraction aims to reduce the number of features in a dataset by creating new features from the existing ones and then discarding the original features. These new reduced set of features should then be able to summarize most of the information contained in the original set of features.

We need to do this in image captioning because there may be some parts in the image that won't affect the caption generated for it, like watermarks for instance, so it would be better to train the model on the image without such features, containing only the important ones.

It's usually done through pre-trained CNN architecture through transfer learning.

Our feature extractor needs an image 224x224x3 size. The model uses ResNet50 pretrained on ImageNet dataset where the features of the image are extracted just before the last layer of classification. Another dense layer is added and converted to get a vector of length 2048.

ResNet50:

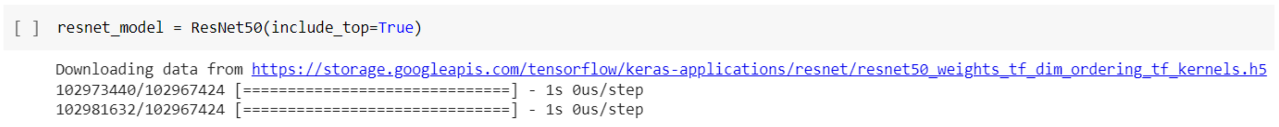
- ResNet-50 is a convolutional neural network that is 50 layers deep.
- A pretrained version of the network trained on more than a million images from the ImageNet database can be loaded.
- The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.
- The network has learned rich feature representations for a wide range of images.
- The network has an image input size of 224-by-224

ResNet50 Architecture:

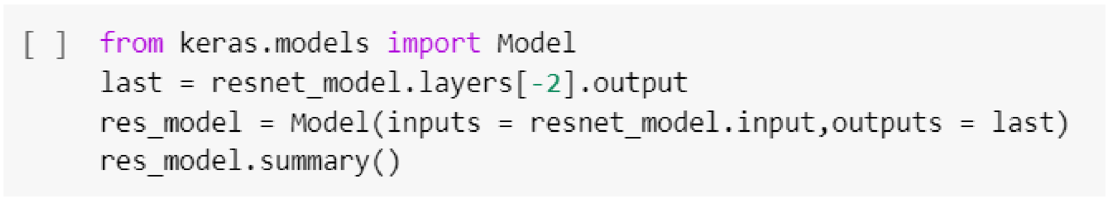
- ResNet50 is a variant of ResNet model which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer.
- It has 3.8 x 10^9 Floating points operations.

Why did we use ResNet50?

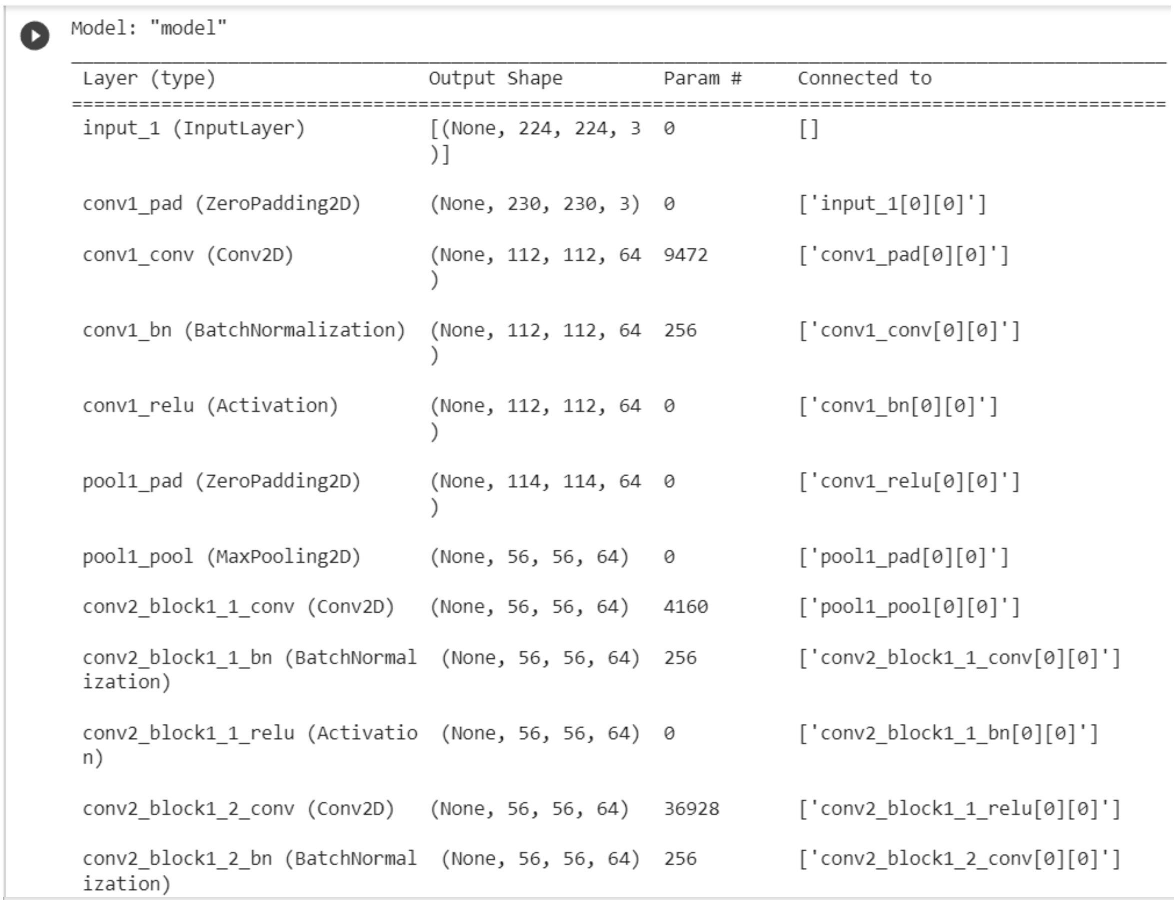
- Networks with large number of layers can be trained easily without increasing the training error percentage.
- ResNets help in tackling the vanishing gradient problem using identity mapping.
- ResNets are relatively faster than other pre-trained CNN transfer models.
- ResNets have relatively a smaller number of parameters than other pre-trained CNN transfer models.



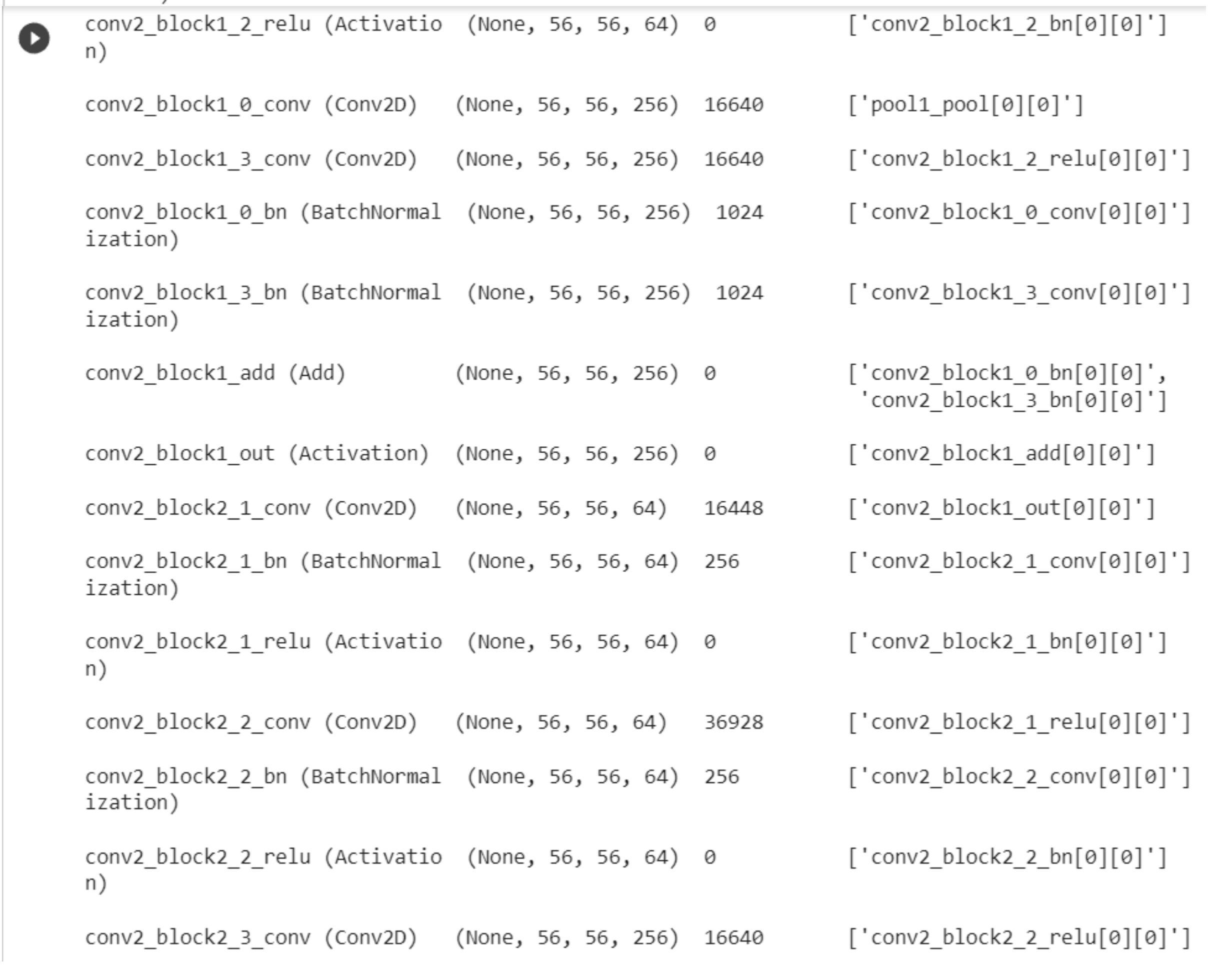
```
[ ] resnet_model = ResNet50(include_top=True)

    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
    102973440/102967424 [==============================] - 1s 0us/step
    102981632/102967424 [==============================] - 1s 0us/step
```



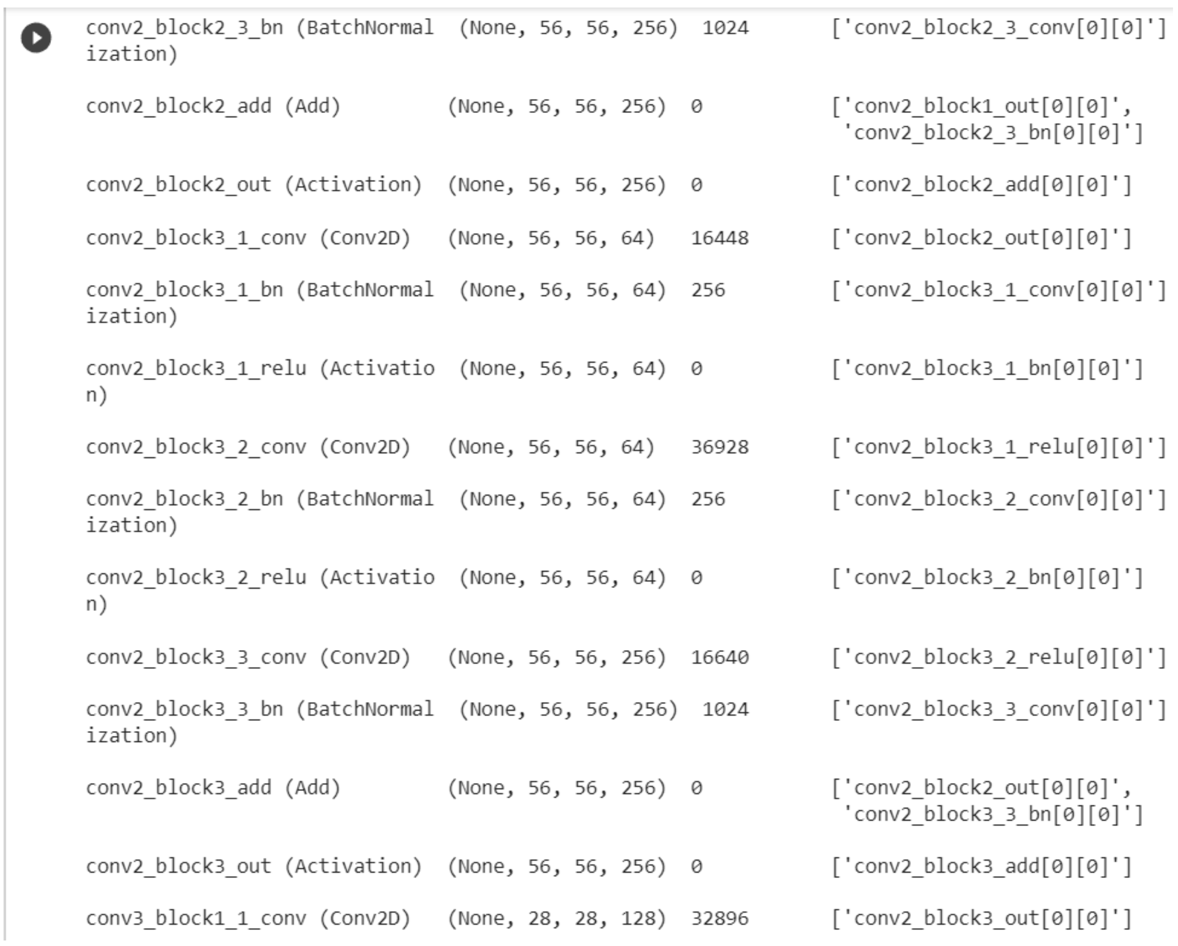
```
[ ] from keras.models import Model
    last = resnet_model.layers[-2].output
    res_model = Model(inputs = resnet_model.input,outputs = last)
    res_model.summary()
```



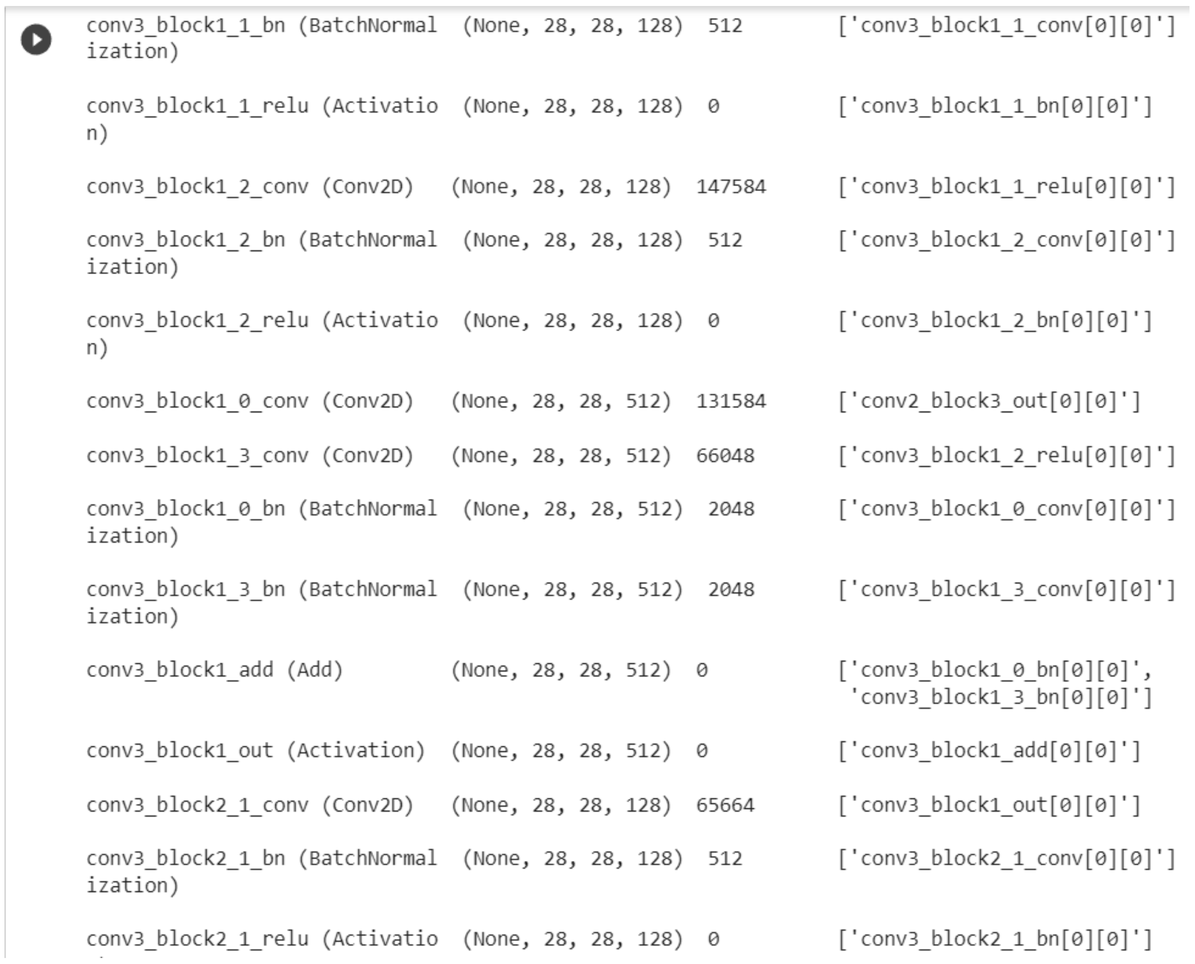
```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 112, 112, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0           ['conv1_relu[0][0]']
                                )

 pool1_pool (MaxPooling2D)      (None, 56, 56, 64)   0           ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2D)   (None, 56, 56, 64)   4160        ['pool1_pool[0][0]']

 conv2_block1_1_bn (BatchNormal  (None, 56, 56, 64)  256         ['conv2_block1_1_conv[0][0]']
 ization)

 conv2_block1_1_relu (Activatio  (None, 56, 56, 64)  0           ['conv2_block1_1_bn[0][0]']
 n)

 conv2_block1_2_conv (Conv2D)   (None, 56, 56, 64)   36928       ['conv2_block1_1_relu[0][0]']

 conv2_block1_2_bn (BatchNormal  (None, 56, 56, 64)  256         ['conv2_block1_2_conv[0][0]']
 ization)
```



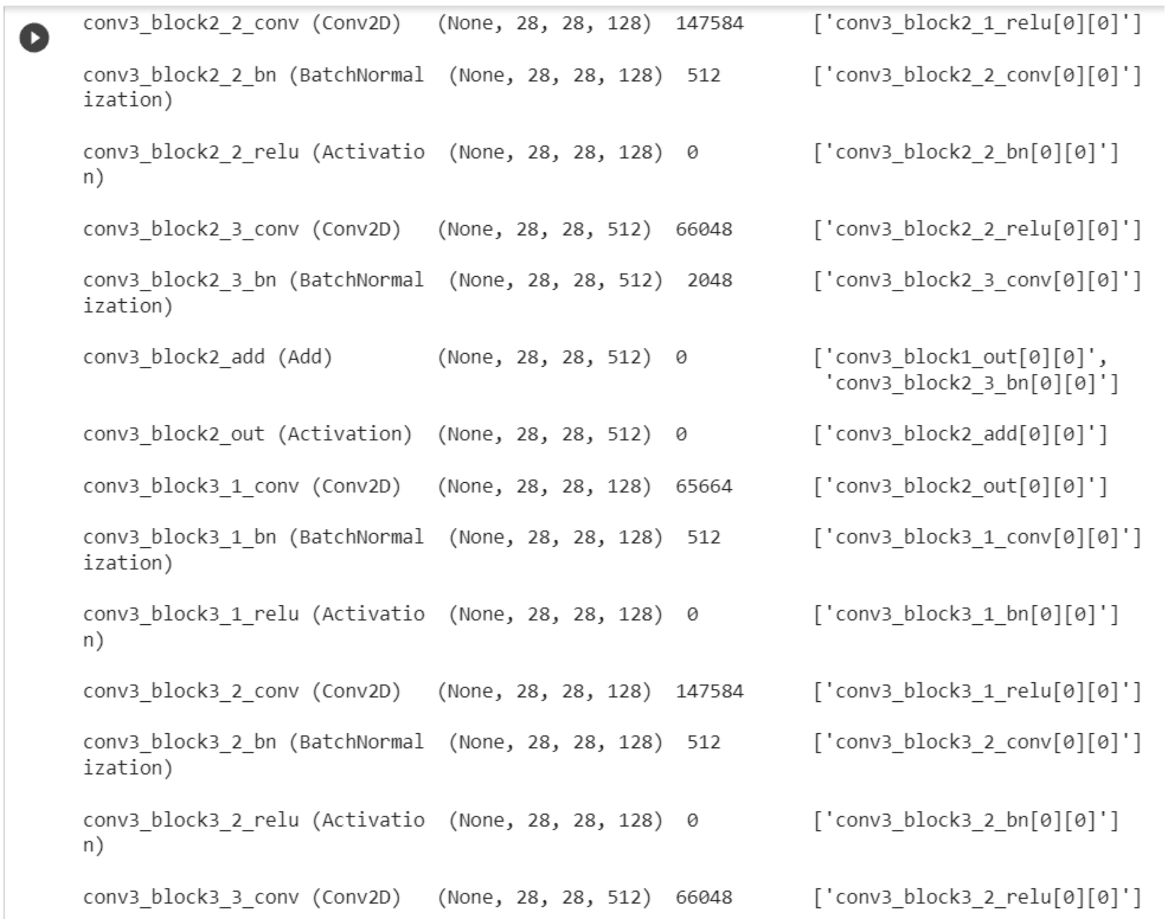
```
 conv2_block1_2_relu (Activatio  (None, 56, 56, 64)  0           ['conv2_block1_2_bn[0][0]']
 n)

 conv2_block1_0_conv (Conv2D)   (None, 56, 56, 256)  16640       ['pool1_pool[0][0]']

 conv2_block1_3_conv (Conv2D)   (None, 56, 56, 256)  16640       ['conv2_block1_2_relu[0][0]']

 conv2_block1_0_bn (BatchNormal  (None, 56, 56, 256)  1024       ['conv2_block1_0_conv[0][0]']
 ization)

 conv2_block1_3_bn (BatchNormal  (None, 56, 56, 256)  1024       ['conv2_block1_3_conv[0][0]']
 ization)

 conv2_block1_add (Add)         (None, 56, 56, 256)  0           ['conv2_block1_0_bn[0][0]',
                                                                  'conv2_block1_3_bn[0][0]']

 conv2_block1_out (Activation)  (None, 56, 56, 256)  0           ['conv2_block1_add[0][0]']

 conv2_block2_1_conv (Conv2D)   (None, 56, 56, 64)   16448       ['conv2_block1_out[0][0]']

 conv2_block2_1_bn (BatchNormal  (None, 56, 56, 64)  256         ['conv2_block2_1_conv[0][0]']
 ization)

 conv2_block2_1_relu (Activatio  (None, 56, 56, 64)  0           ['conv2_block2_1_bn[0][0]']
 n)

 conv2_block2_2_conv (Conv2D)   (None, 56, 56, 64)   36928       ['conv2_block2_1_relu[0][0]']

 conv2_block2_2_bn (BatchNormal  (None, 56, 56, 64)  256         ['conv2_block2_2_conv[0][0]']
 ization)

 conv2_block2_2_relu (Activatio  (None, 56, 56, 64)  0           ['conv2_block2_2_bn[0][0]']
 n)

 conv2_block2_3_conv (Conv2D)   (None, 56, 56, 256)  16640       ['conv2_block2_2_relu[0][0]']
```



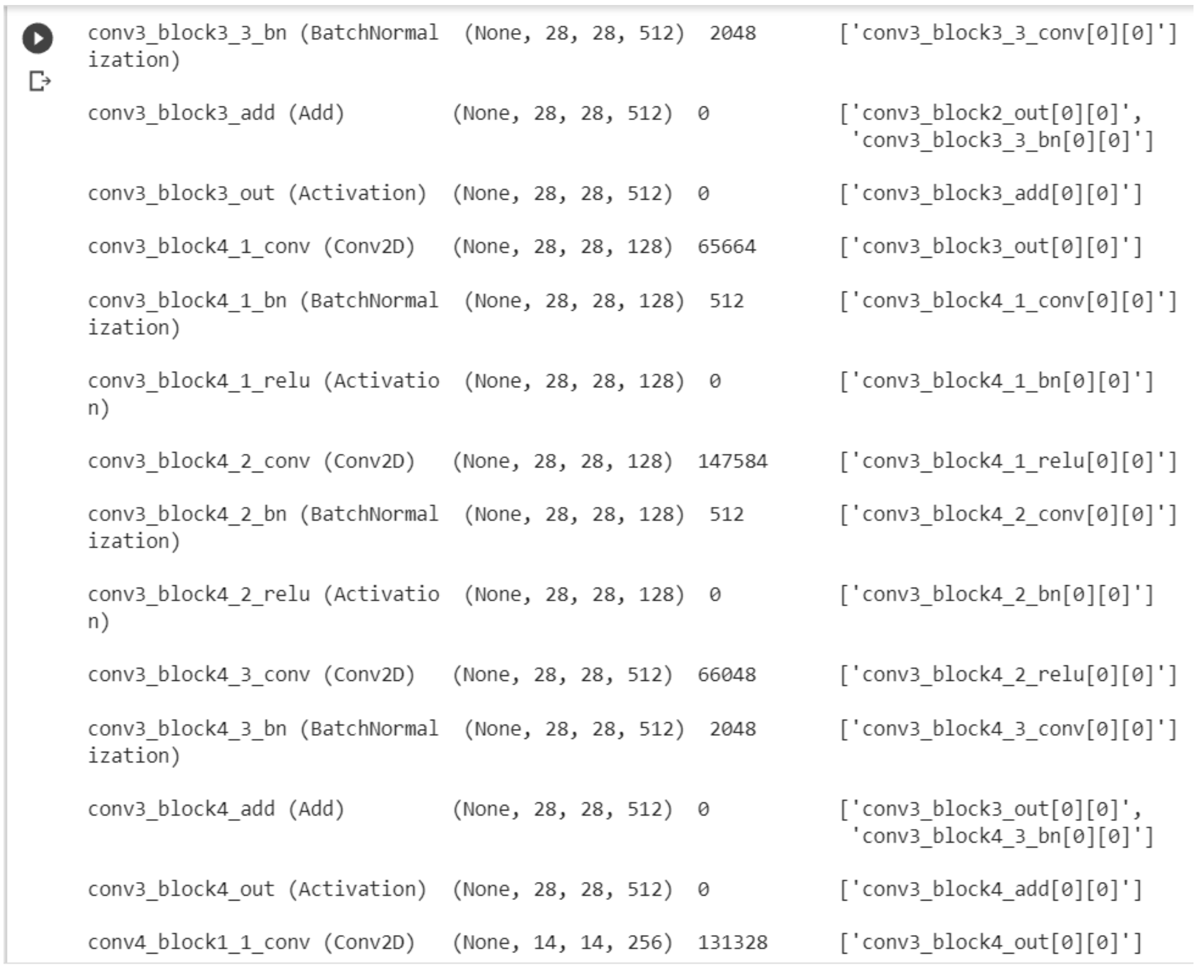
```
conv2_block2_3_bn (BatchNormal  (None, 56, 56, 256)  1024       ['conv2_block2_3_conv[0][0]']
ization)

conv2_block2_add (Add)         (None, 56, 56, 256)  0          ['conv2_block1_out[0][0]',
                                                                 'conv2_block2_3_bn[0][0]']

conv2_block2_out (Activation)  (None, 56, 56, 256)  0          ['conv2_block2_add[0][0]']

conv2_block3_1_conv (Conv2D)   (None, 56, 56, 64)   16448      ['conv2_block2_out[0][0]']

conv2_block3_1_bn (BatchNormal  (None, 56, 56, 64)  256        ['conv2_block3_1_conv[0][0]']
ization)

conv2_block3_1_relu (Activatio  (None, 56, 56, 64)  0          ['conv2_block3_1_bn[0][0]']
n)

conv2_block3_2_conv (Conv2D)   (None, 56, 56, 64)   36928      ['conv2_block3_1_relu[0][0]']

conv2_block3_2_bn (BatchNormal  (None, 56, 56, 64)  256        ['conv2_block3_2_conv[0][0]']
ization)

conv2_block3_2_relu (Activatio  (None, 56, 56, 64)  0          ['conv2_block3_2_bn[0][0]']
n)

conv2_block3_3_conv (Conv2D)   (None, 56, 56, 256)  16640      ['conv2_block3_2_relu[0][0]']

conv2_block3_3_bn (BatchNormal  (None, 56, 56, 256)  1024       ['conv2_block3_3_conv[0][0]']
ization)

conv2_block3_add (Add)         (None, 56, 56, 256)  0          ['conv2_block2_out[0][0]',
                                                                 'conv2_block3_3_bn[0][0]']

conv2_block3_out (Activation)  (None, 56, 56, 256)  0          ['conv2_block3_add[0][0]']

conv3_block1_1_conv (Conv2D)   (None, 28, 28, 128)  32896      ['conv2_block3_out[0][0]']
```



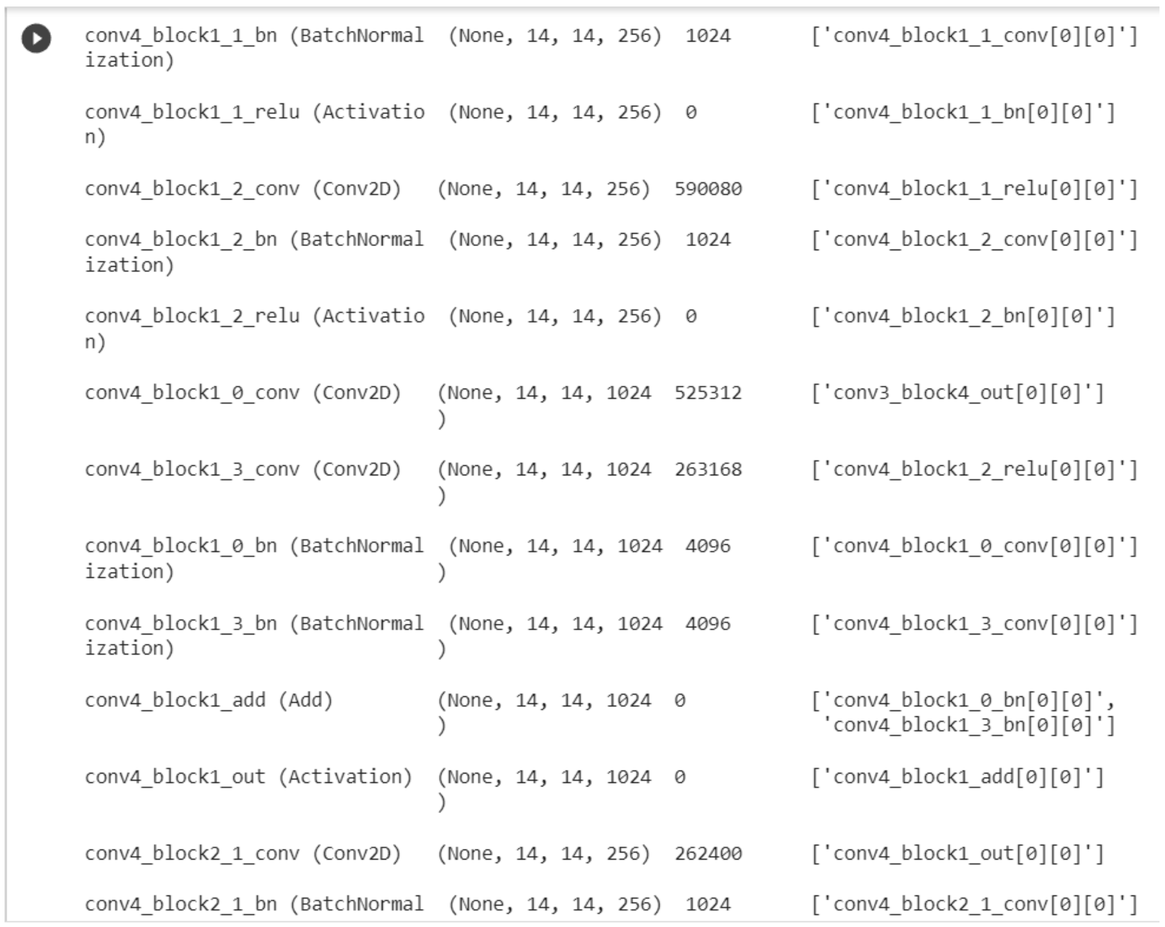
```
conv3_block1_1_bn (BatchNormal  (None, 28, 28, 128)  512        ['conv3_block1_1_conv[0][0]']
ization)

conv3_block1_1_relu (Activatio  (None, 28, 28, 128)  0          ['conv3_block1_1_bn[0][0]']
n)

conv3_block1_2_conv (Conv2D)   (None, 28, 28, 128)  147584     ['conv3_block1_1_relu[0][0]']

conv3_block1_2_bn (BatchNormal  (None, 28, 28, 128)  512        ['conv3_block1_2_conv[0][0]']
ization)

conv3_block1_2_relu (Activatio  (None, 28, 28, 128)  0          ['conv3_block1_2_bn[0][0]']
n)

conv3_block1_0_conv (Conv2D)   (None, 28, 28, 512)  131584     ['conv2_block3_out[0][0]']

conv3_block1_3_conv (Conv2D)   (None, 28, 28, 512)  66048      ['conv3_block1_2_relu[0][0]']

conv3_block1_0_bn (BatchNormal  (None, 28, 28, 512)  2048       ['conv3_block1_0_conv[0][0]']
ization)

conv3_block1_3_bn (BatchNormal  (None, 28, 28, 512)  2048       ['conv3_block1_3_conv[0][0]']
ization)

conv3_block1_add (Add)         (None, 28, 28, 512)  0          ['conv3_block1_0_bn[0][0]',
                                                                 'conv3_block1_3_bn[0][0]']

conv3_block1_out (Activation)  (None, 28, 28, 512)  0          ['conv3_block1_add[0][0]']

conv3_block2_1_conv (Conv2D)   (None, 28, 28, 128)  65664      ['conv3_block1_out[0][0]']

conv3_block2_1_bn (BatchNormal  (None, 28, 28, 128)  512        ['conv3_block2_1_conv[0][0]']
ization)

conv3_block2_1_relu (Activatio  (None, 28, 28, 128)  0          ['conv3_block2_1_bn[0][0]']
```



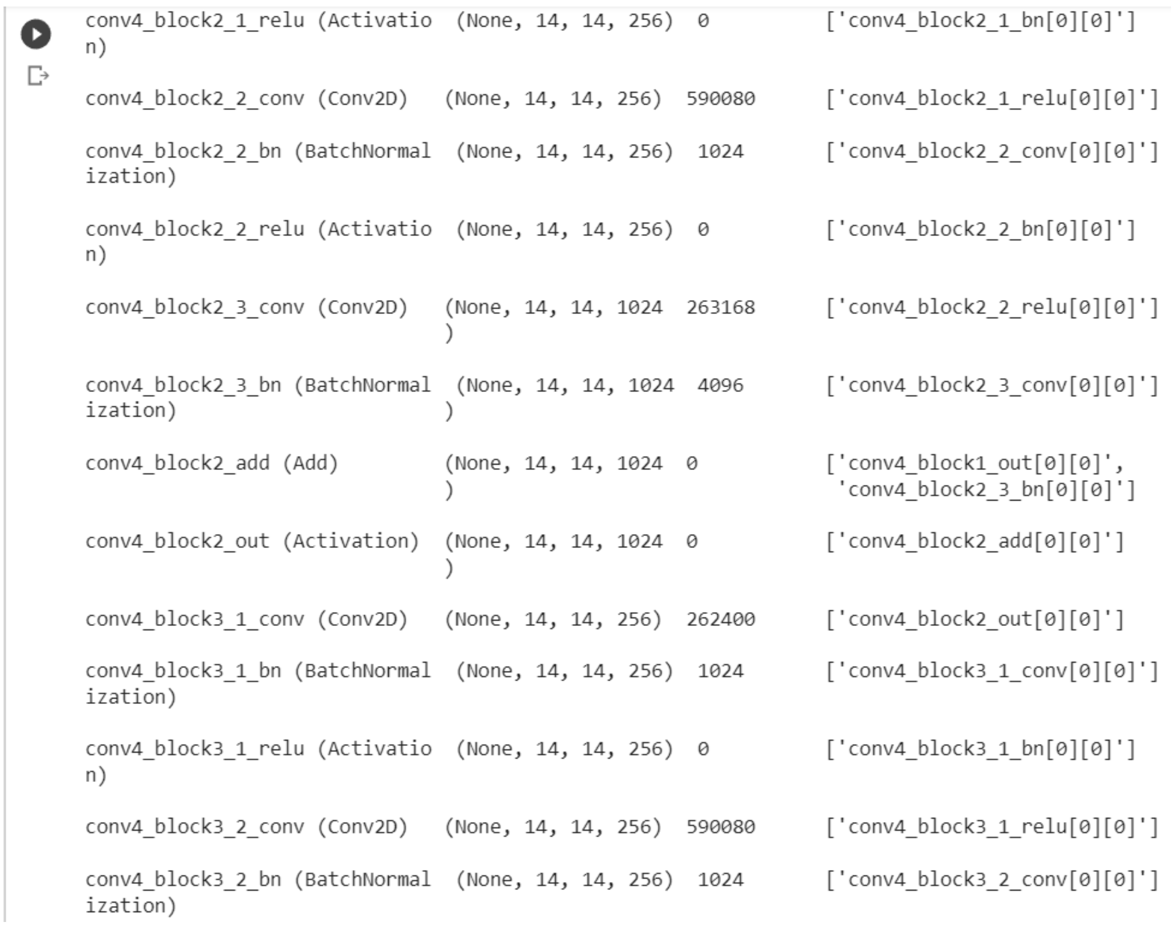
```
conv3_block2_2_conv (Conv2D)   (None, 28, 28, 128)  147584     ['conv3_block2_1_relu[0][0]']

conv3_block2_2_bn (BatchNormal  (None, 28, 28, 128)  512       ['conv3_block2_2_conv[0][0]']
ization)

conv3_block2_2_relu (Activatio  (None, 28, 28, 128)  0         ['conv3_block2_2_bn[0][0]']
n)

conv3_block2_3_conv (Conv2D)   (None, 28, 28, 512)  66048      ['conv3_block2_2_relu[0][0]']

conv3_block2_3_bn (BatchNormal  (None, 28, 28, 512)  2048      ['conv3_block2_3_conv[0][0]']
ization)

conv3_block2_add (Add)         (None, 28, 28, 512)  0          ['conv3_block1_out[0][0]',
                                                                'conv3_block2_3_bn[0][0]']

conv3_block2_out (Activation)  (None, 28, 28, 512)  0          ['conv3_block2_add[0][0]']

conv3_block3_1_conv (Conv2D)   (None, 28, 28, 128)  65664      ['conv3_block2_out[0][0]']

conv3_block3_1_bn (BatchNormal  (None, 28, 28, 128)  512       ['conv3_block3_1_conv[0][0]']
ization)

conv3_block3_1_relu (Activatio  (None, 28, 28, 128)  0         ['conv3_block3_1_bn[0][0]']
n)

conv3_block3_2_conv (Conv2D)   (None, 28, 28, 128)  147584     ['conv3_block3_1_relu[0][0]']

conv3_block3_2_bn (BatchNormal  (None, 28, 28, 128)  512       ['conv3_block3_2_conv[0][0]']
ization)

conv3_block3_2_relu (Activatio  (None, 28, 28, 128)  0         ['conv3_block3_2_bn[0][0]']
n)

conv3_block3_3_conv (Conv2D)   (None, 28, 28, 512)  66048      ['conv3_block3_2_relu[0][0]']
```



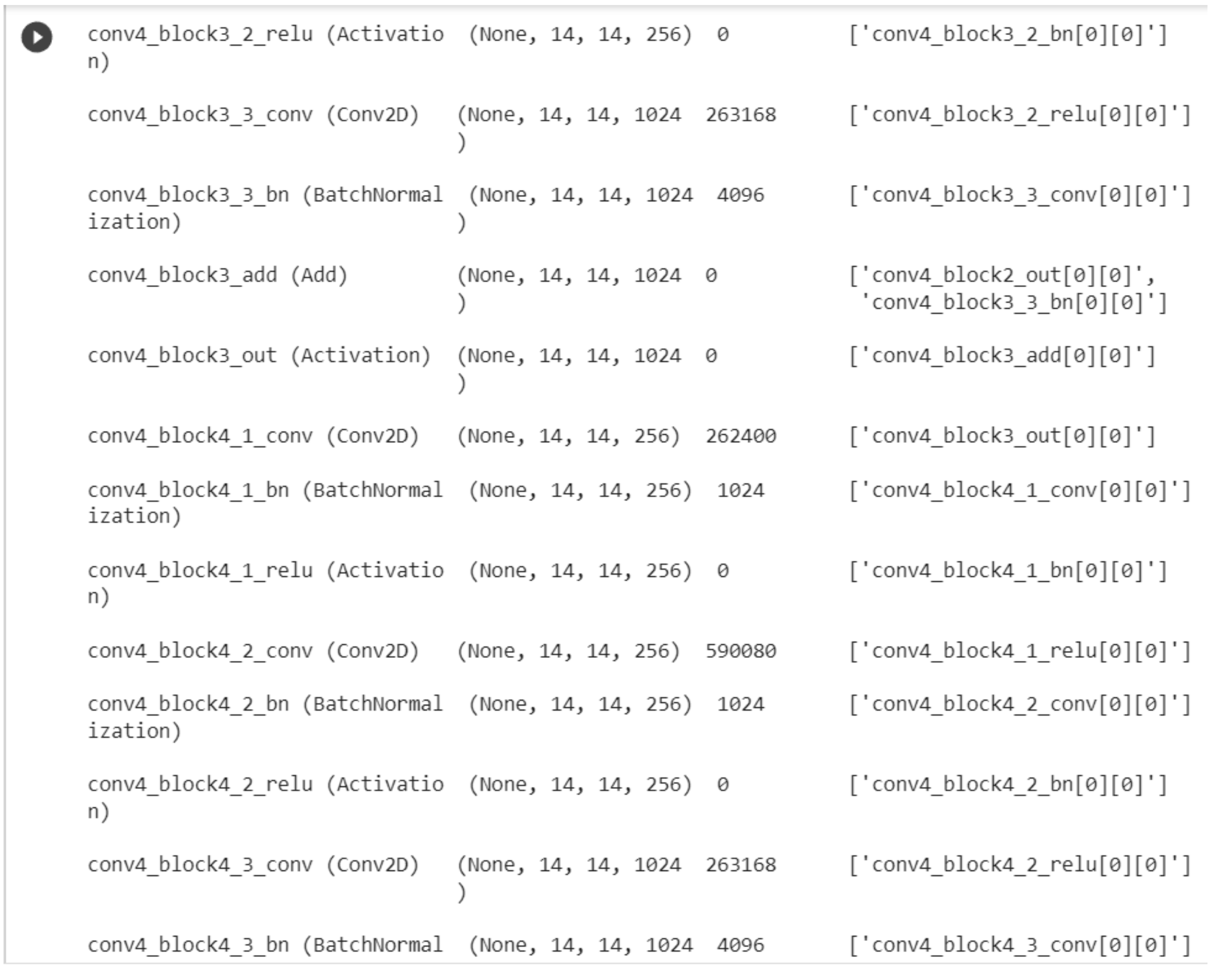
```
conv3_block3_3_bn (BatchNormal  (None, 28, 28, 512)  2048      ['conv3_block3_3_conv[0][0]']
ization)

conv3_block3_add (Add)         (None, 28, 28, 512)  0          ['conv3_block2_out[0][0]',
                                                                'conv3_block3_3_bn[0][0]']

conv3_block3_out (Activation)  (None, 28, 28, 512)  0          ['conv3_block3_add[0][0]']

conv3_block4_1_conv (Conv2D)   (None, 28, 28, 128)  65664      ['conv3_block3_out[0][0]']

conv3_block4_1_bn (BatchNormal  (None, 28, 28, 128)  512       ['conv3_block4_1_conv[0][0]']
ization)

conv3_block4_1_relu (Activatio  (None, 28, 28, 128)  0         ['conv3_block4_1_bn[0][0]']
n)

conv3_block4_2_conv (Conv2D)   (None, 28, 28, 128)  147584     ['conv3_block4_1_relu[0][0]']

conv3_block4_2_bn (BatchNormal  (None, 28, 28, 128)  512       ['conv3_block4_2_conv[0][0]']
ization)

conv3_block4_2_relu (Activatio  (None, 28, 28, 128)  0         ['conv3_block4_2_bn[0][0]']
n)

conv3_block4_3_conv (Conv2D)   (None, 28, 28, 512)  66048      ['conv3_block4_2_relu[0][0]']

conv3_block4_3_bn (BatchNormal  (None, 28, 28, 512)  2048      ['conv3_block4_3_conv[0][0]']
ization)

conv3_block4_add (Add)         (None, 28, 28, 512)  0          ['conv3_block3_out[0][0]',
                                                                'conv3_block4_3_bn[0][0]']

conv3_block4_out (Activation)  (None, 28, 28, 512)  0          ['conv3_block4_add[0][0]']

conv4_block1_1_conv (Conv2D)   (None, 14, 14, 256)  131328     ['conv3_block4_out[0][0]']
```



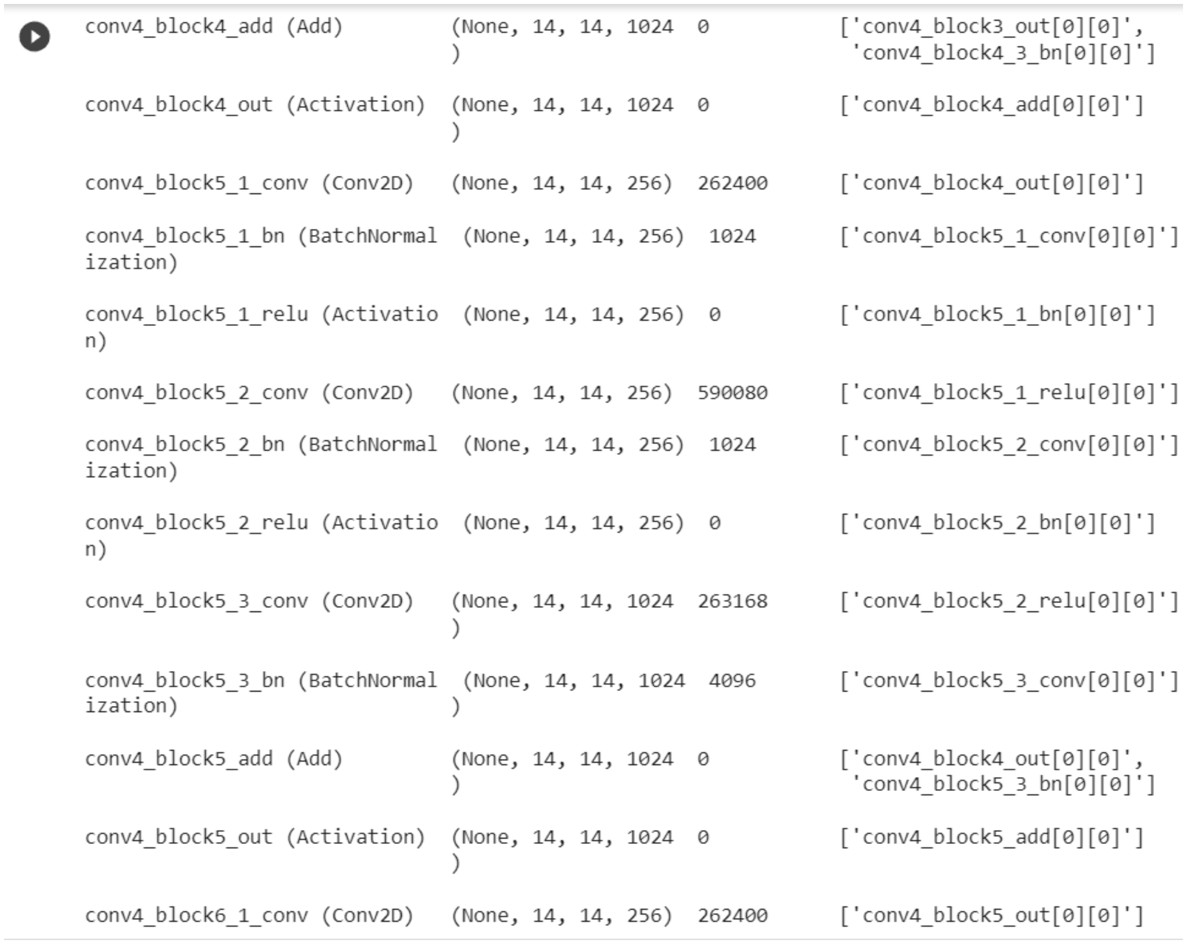
```
conv4_block1_1_bn (BatchNormal  (None, 14, 14, 256)  1024       ['conv4_block1_1_conv[0][0]']
ization)

conv4_block1_1_relu (Activatio  (None, 14, 14, 256)  0          ['conv4_block1_1_bn[0][0]']
n)

conv4_block1_2_conv (Conv2D)   (None, 14, 14, 256)  590080     ['conv4_block1_1_relu[0][0]']

conv4_block1_2_bn (BatchNormal  (None, 14, 14, 256)  1024       ['conv4_block1_2_conv[0][0]']
ization)

conv4_block1_2_relu (Activatio  (None, 14, 14, 256)  0          ['conv4_block1_2_bn[0][0]']
n)

conv4_block1_0_conv (Conv2D)   (None, 14, 14, 1024  525312     ['conv3_block4_out[0][0]']
                               )

conv4_block1_3_conv (Conv2D)   (None, 14, 14, 1024  263168     ['conv4_block1_2_relu[0][0]']
                               )

conv4_block1_0_bn (BatchNormal  (None, 14, 14, 1024  4096       ['conv4_block1_0_conv[0][0]']
ization)                        )

conv4_block1_3_bn (BatchNormal  (None, 14, 14, 1024  4096       ['conv4_block1_3_conv[0][0]']
ization)                        )

conv4_block1_add (Add)         (None, 14, 14, 1024  0          ['conv4_block1_0_bn[0][0]',
                               )                                'conv4_block1_3_bn[0][0]']

conv4_block1_out (Activation)  (None, 14, 14, 1024  0          ['conv4_block1_add[0][0]']
                               )

conv4_block2_1_conv (Conv2D)   (None, 14, 14, 256)  262400     ['conv4_block1_out[0][0]']

conv4_block2_1_bn (BatchNormal  (None, 14, 14, 256)  1024       ['conv4_block2_1_conv[0][0]']
```



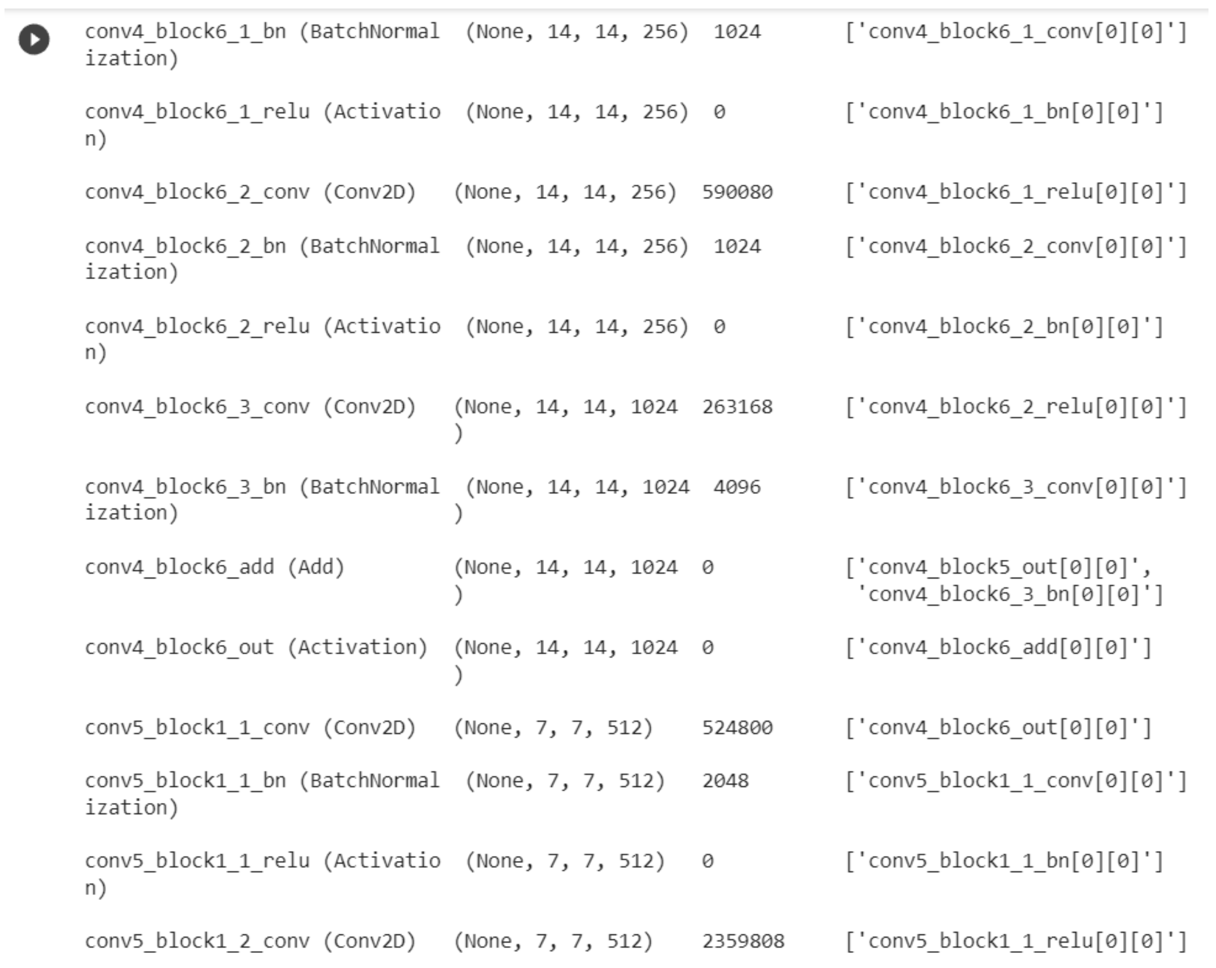
```
conv4_block2_1_relu (Activatio  (None, 14, 14, 256)  0          ['conv4_block2_1_bn[0][0]']
n)

conv4_block2_2_conv (Conv2D)   (None, 14, 14, 256)  590080     ['conv4_block2_1_relu[0][0]']

conv4_block2_2_bn (BatchNormal  (None, 14, 14, 256)  1024       ['conv4_block2_2_conv[0][0]']
ization)

conv4_block2_2_relu (Activatio  (None, 14, 14, 256)  0          ['conv4_block2_2_bn[0][0]']
n)

conv4_block2_3_conv (Conv2D)   (None, 14, 14, 1024  263168     ['conv4_block2_2_relu[0][0]']
                               )

conv4_block2_3_bn (BatchNormal  (None, 14, 14, 1024  4096       ['conv4_block2_3_conv[0][0]']
ization)                        )

conv4_block2_add (Add)         (None, 14, 14, 1024  0          ['conv4_block1_out[0][0]',
                               )                                'conv4_block2_3_bn[0][0]']

conv4_block2_out (Activation)  (None, 14, 14, 1024  0          ['conv4_block2_add[0][0]']
                               )

conv4_block3_1_conv (Conv2D)   (None, 14, 14, 256)  262400     ['conv4_block2_out[0][0]']

conv4_block3_1_bn (BatchNormal  (None, 14, 14, 256)  1024       ['conv4_block3_1_conv[0][0]']
ization)

conv4_block3_1_relu (Activatio  (None, 14, 14, 256)  0          ['conv4_block3_1_bn[0][0]']
n)

conv4_block3_2_conv (Conv2D)   (None, 14, 14, 256)  590080     ['conv4_block3_1_relu[0][0]']

conv4_block3_2_bn (BatchNormal  (None, 14, 14, 256)  1024       ['conv4_block3_2_conv[0][0]']
ization)
```



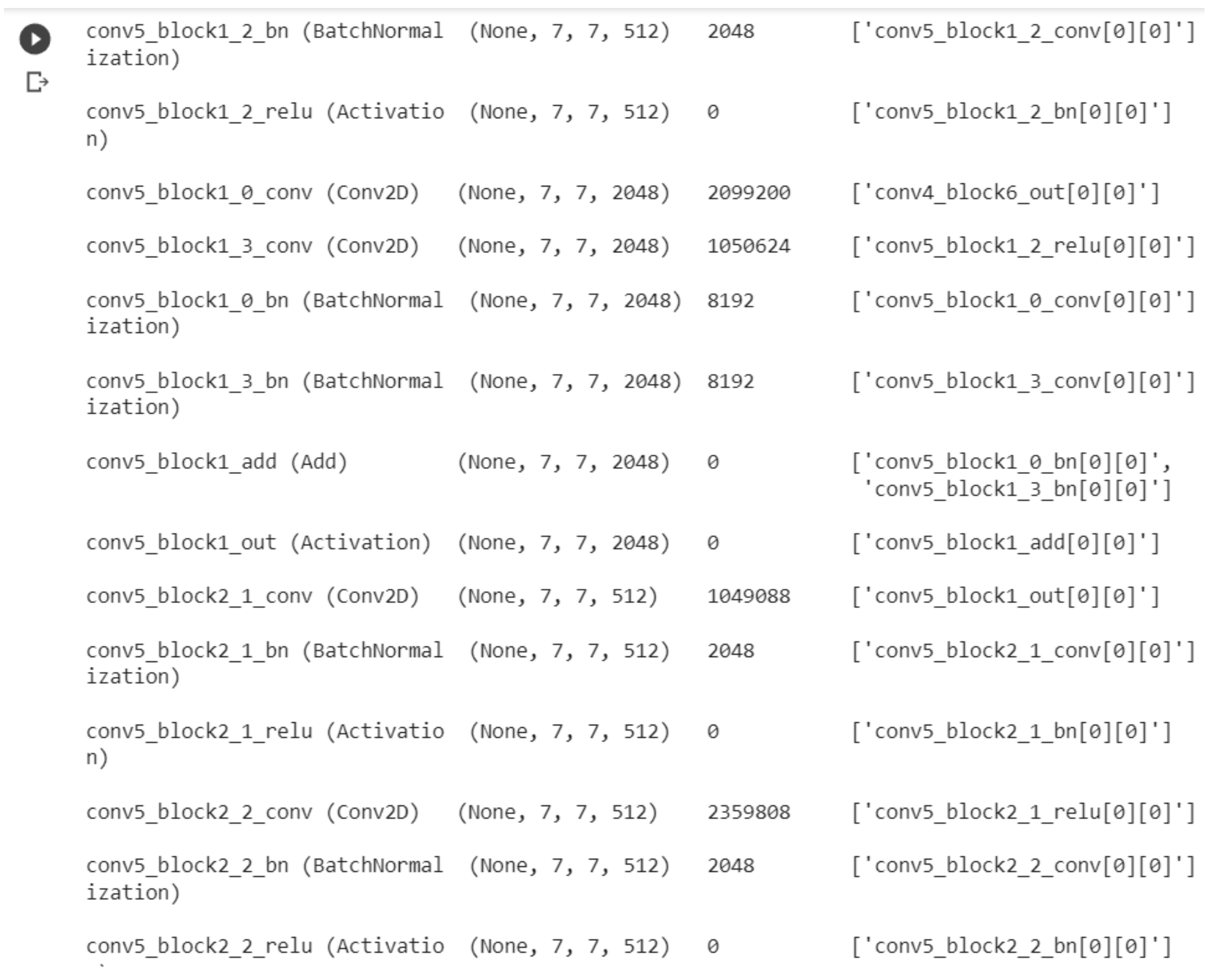
```
conv4_block3_2_relu (Activatio  (None, 14, 14, 256)  0           ['conv4_block3_2_bn[0][0]']
n)

conv4_block3_3_conv (Conv2D)   (None, 14, 14, 1024  263168      ['conv4_block3_2_relu[0][0]']
                               )

conv4_block3_3_bn (BatchNormal  (None, 14, 14, 1024  4096        ['conv4_block3_3_conv[0][0]']
ization)                       )

conv4_block3_add (Add)         (None, 14, 14, 1024  0           ['conv4_block2_out[0][0]',
                               )                                 'conv4_block3_3_bn[0][0]']

conv4_block3_out (Activation)  (None, 14, 14, 1024  0           ['conv4_block3_add[0][0]']
                               )

conv4_block4_1_conv (Conv2D)   (None, 14, 14, 256)  262400      ['conv4_block3_out[0][0]']

conv4_block4_1_bn (BatchNormal  (None, 14, 14, 256)  1024        ['conv4_block4_1_conv[0][0]']
ization)

conv4_block4_1_relu (Activatio  (None, 14, 14, 256)  0           ['conv4_block4_1_bn[0][0]']
n)

conv4_block4_2_conv (Conv2D)   (None, 14, 14, 256)  590080      ['conv4_block4_1_relu[0][0]']

conv4_block4_2_bn (BatchNormal  (None, 14, 14, 256)  1024        ['conv4_block4_2_conv[0][0]']
ization)

conv4_block4_2_relu (Activatio  (None, 14, 14, 256)  0           ['conv4_block4_2_bn[0][0]']
n)

conv4_block4_3_conv (Conv2D)   (None, 14, 14, 1024  263168      ['conv4_block4_2_relu[0][0]']
                               )

conv4_block4_3_bn (BatchNormal  (None, 14, 14, 1024  4096        ['conv4_block4_3_conv[0][0]']
```



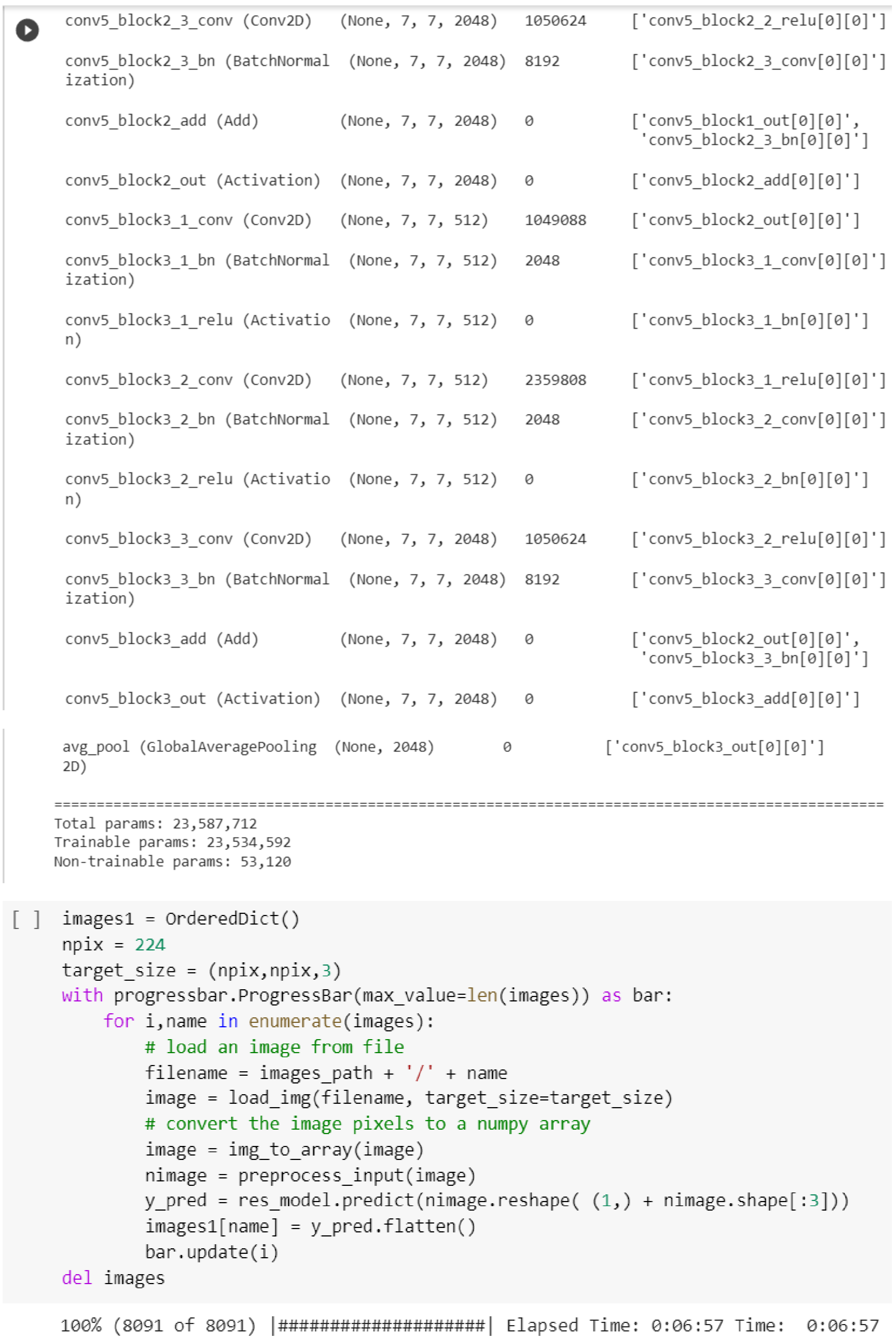
```
conv4_block4_add (Add)         (None, 14, 14, 1024  0           ['conv4_block3_out[0][0]',
                               )                                 'conv4_block4_3_bn[0][0]']

conv4_block4_out (Activation)  (None, 14, 14, 1024  0           ['conv4_block4_add[0][0]']
                               )

conv4_block5_1_conv (Conv2D)   (None, 14, 14, 256)  262400      ['conv4_block4_out[0][0]']

conv4_block5_1_bn (BatchNormal  (None, 14, 14, 256)  1024        ['conv4_block5_1_conv[0][0]']
ization)

conv4_block5_1_relu (Activatio  (None, 14, 14, 256)  0           ['conv4_block5_1_bn[0][0]']
n)

conv4_block5_2_conv (Conv2D)   (None, 14, 14, 256)  590080      ['conv4_block5_1_relu[0][0]']

conv4_block5_2_bn (BatchNormal  (None, 14, 14, 256)  1024        ['conv4_block5_2_conv[0][0]']
ization)

conv4_block5_2_relu (Activatio  (None, 14, 14, 256)  0           ['conv4_block5_2_bn[0][0]']
n)

conv4_block5_3_conv (Conv2D)   (None, 14, 14, 1024  263168      ['conv4_block5_2_relu[0][0]']
                               )

conv4_block5_3_bn (BatchNormal  (None, 14, 14, 1024  4096        ['conv4_block5_3_conv[0][0]']
ization)                       )

conv4_block5_add (Add)         (None, 14, 14, 1024  0           ['conv4_block4_out[0][0]',
                               )                                 'conv4_block5_3_bn[0][0]']

conv4_block5_out (Activation)  (None, 14, 14, 1024  0           ['conv4_block5_add[0][0]']
                               )

conv4_block6_1_conv (Conv2D)   (None, 14, 14, 256)  262400      ['conv4_block5_out[0][0]']
```



```
conv4_block6_1_bn (BatchNormal  (None, 14, 14, 256)  1024       ['conv4_block6_1_conv[0][0]']
ization)

conv4_block6_1_relu (Activatio  (None, 14, 14, 256)  0          ['conv4_block6_1_bn[0][0]']
n)

conv4_block6_2_conv (Conv2D)   (None, 14, 14, 256)  590080     ['conv4_block6_1_relu[0][0]']

conv4_block6_2_bn (BatchNormal  (None, 14, 14, 256)  1024       ['conv4_block6_2_conv[0][0]']
ization)

conv4_block6_2_relu (Activatio  (None, 14, 14, 256)  0          ['conv4_block6_2_bn[0][0]']
n)

conv4_block6_3_conv (Conv2D)   (None, 14, 14, 1024  263168     ['conv4_block6_2_relu[0][0]']
                               )

conv4_block6_3_bn (BatchNormal  (None, 14, 14, 1024  4096       ['conv4_block6_3_conv[0][0]']
ization)                       )

conv4_block6_add (Add)         (None, 14, 14, 1024  0          ['conv4_block5_out[0][0]',
                               )                                 'conv4_block6_3_bn[0][0]']

conv4_block6_out (Activation)  (None, 14, 14, 1024  0          ['conv4_block6_add[0][0]']
                               )

conv5_block1_1_conv (Conv2D)   (None, 7, 7, 512)    524800     ['conv4_block6_out[0][0]']

conv5_block1_1_bn (BatchNormal  (None, 7, 7, 512)   2048       ['conv5_block1_1_conv[0][0]']
ization)

conv5_block1_1_relu (Activatio  (None, 7, 7, 512)   0          ['conv5_block1_1_bn[0][0]']
n)

conv5_block1_2_conv (Conv2D)   (None, 7, 7, 512)    2359808    ['conv5_block1_1_relu[0][0]']
```



```
conv5_block1_2_bn (BatchNormal  (None, 7, 7, 512)   2048       ['conv5_block1_2_conv[0][0]']
ization)

conv5_block1_2_relu (Activatio  (None, 7, 7, 512)   0          ['conv5_block1_2_bn[0][0]']
n)

conv5_block1_0_conv (Conv2D)   (None, 7, 7, 2048)   2099200    ['conv4_block6_out[0][0]']

conv5_block1_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624    ['conv5_block1_2_relu[0][0]']

conv5_block1_0_bn (BatchNormal  (None, 7, 7, 2048)  8192       ['conv5_block1_0_conv[0][0]']
ization)

conv5_block1_3_bn (BatchNormal  (None, 7, 7, 2048)  8192       ['conv5_block1_3_conv[0][0]']
ization)

conv5_block1_add (Add)         (None, 7, 7, 2048)   0          ['conv5_block1_0_bn[0][0]',
                                                                 'conv5_block1_3_bn[0][0]']

conv5_block1_out (Activation)  (None, 7, 7, 2048)   0          ['conv5_block1_add[0][0]']

conv5_block2_1_conv (Conv2D)   (None, 7, 7, 512)    1049088    ['conv5_block1_out[0][0]']

conv5_block2_1_bn (BatchNormal  (None, 7, 7, 512)   2048       ['conv5_block2_1_conv[0][0]']
ization)

conv5_block2_1_relu (Activatio  (None, 7, 7, 512)   0          ['conv5_block2_1_bn[0][0]']
n)

conv5_block2_2_conv (Conv2D)   (None, 7, 7, 512)    2359808    ['conv5_block2_1_relu[0][0]']

conv5_block2_2_bn (BatchNormal  (None, 7, 7, 512)   2048       ['conv5_block2_2_conv[0][0]']
ization)

conv5_block2_2_relu (Activatio  (None, 7, 7, 512)   0          ['conv5_block2_2_bn[0][0]']
```



```
 conv5_block2_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624     ['conv5_block2_2_relu[0][0]']

 conv5_block2_3_bn (BatchNormal  (None, 7, 7, 2048)  8192        ['conv5_block2_3_conv[0][0]']
 ization)

 conv5_block2_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block1_out[0][0]',
                                                                  'conv5_block2_3_bn[0][0]']

 conv5_block2_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block2_add[0][0]']

 conv5_block3_1_conv (Conv2D)   (None, 7, 7, 512)    1049088     ['conv5_block2_out[0][0]']

 conv5_block3_1_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block3_1_conv[0][0]']
 ization)

 conv5_block3_1_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block3_1_bn[0][0]']
 n)

 conv5_block3_2_conv (Conv2D)   (None, 7, 7, 512)    2359808     ['conv5_block3_1_relu[0][0]']

 conv5_block3_2_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block3_2_conv[0][0]']
 ization)

 conv5_block3_2_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block3_2_bn[0][0]']
 n)

 conv5_block3_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624     ['conv5_block3_2_relu[0][0]']

 conv5_block3_3_bn (BatchNormal  (None, 7, 7, 2048)  8192        ['conv5_block3_3_conv[0][0]']
 ization)

 conv5_block3_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block2_out[0][0]',
                                                                  'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block3_add[0][0]']

 avg_pool (GlobalAveragePooling  (None, 2048)        0           ['conv5_block3_out[0][0]']
 2D)

==================================================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
```

```python
images1 = OrderedDict()
npix = 224
target_size = (npix,npix,3)
with progressbar.ProgressBar(max_value=len(images)) as bar:
    for i,name in enumerate(images):
        # load an image from file
        filename = images_path + '/' + name
        image = load_img(filename, target_size=target_size)
        # convert the image pixels to a numpy array
        image = img_to_array(image)
        nimage = preprocess_input(image)
        y_pred = res_model.predict(nimage.reshape( (1,) + nimage.shape[:3]))
        images1[name] = y_pred.flatten()
        bar.update(i)
del images
```

```
100% (8091 of 8091) |####################| Elapsed Time: 0:06:57 Time:  0:06:57
```

Train/Test split:

We used 80% of the data for training and 20% for testing.

Then, we used 10% of the training data for validation while building the models.

```
Ntest = int(len(dtexts)*0.2)

def split_test_train(dtexts,Ntest):
    return(dtexts[:Ntest],
           dtexts[Ntest:])

dt_test, dt_train   = split_test_train(dtexts,Ntest)
di_test, di_train   = split_test_train(dimages,Ntest)
fnm_test, fnm_train  = split_test_train(fnames,Ntest)
```

Data pre-processing:

We need padding as sentences can be different in length, we need to have the inputs with the same size, this is where the padding is necessary. So, identifier sequences are padded at the end with null tokens to ensure that they are all the of same length.

```
[ ]  def preprocessing(dtexts,dimages):
         N = len(dtexts)
         print("# captions/images = {}".format(N))

         assert(N==len(dimages))
         Xtext, Ximage, ytext = [],[],[]
         for text,image in zip(dtexts,dimages):

             for i in range(1,len(text)):
                 in_text, out_text = text[:i], text[i]
                 in_text = pad_sequences([in_text],maxlen=maxlen).flatten()
                 out_text = to_categorical(out_text,num_classes = vocab_size)

                 Xtext.append(in_text)
                 Ximage.append(image)
                 ytext.append(out_text)

         Xtext  = np.array(Xtext)
         Ximage = np.array(Ximage)
         ytext  = np.array(ytext)
         print(" {} {} {}".format(Xtext.shape,Ximage.shape,ytext.shape))
         return(Xtext,Ximage,ytext)


     Xtext_train, Ximage_train, ytext_train = preprocessing(dt_train,di_train)
```

```
# captions/images = 6473
 (65984, 30) (65984, 2048) (65984, 4476)
```

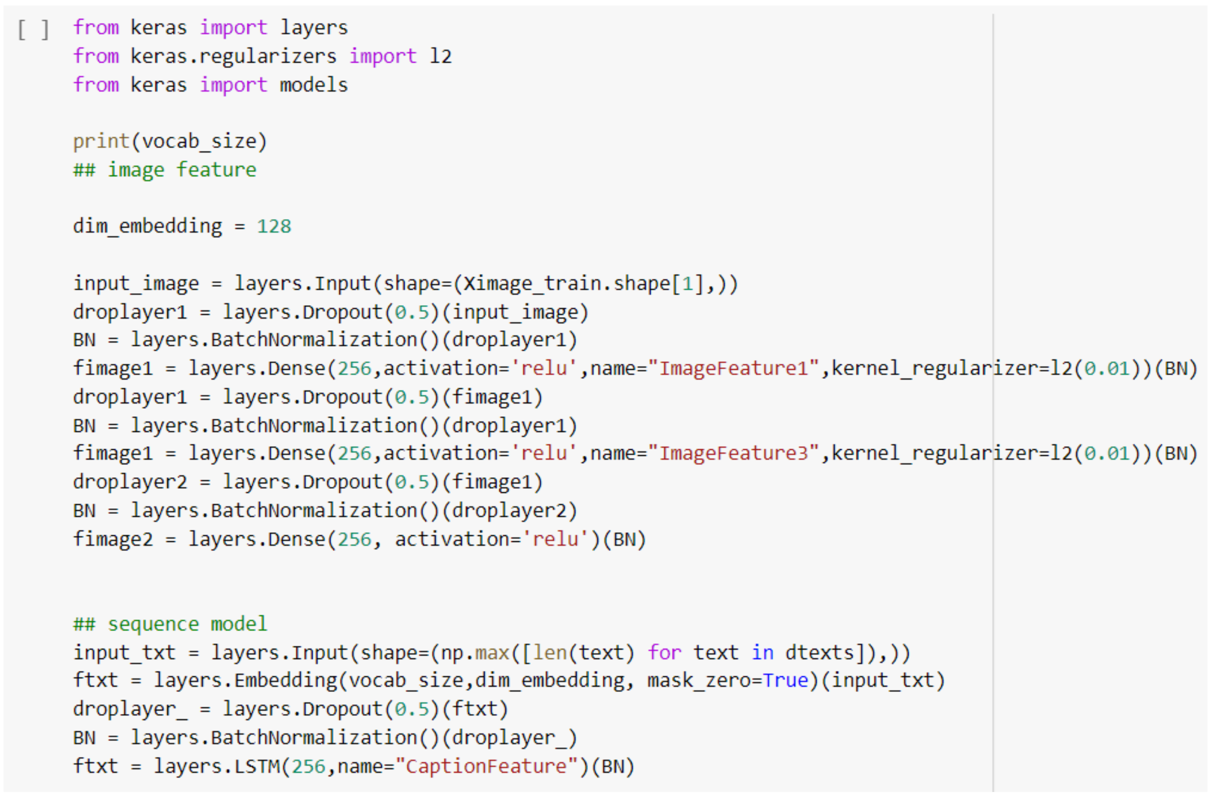
LSTM model:

The Encoder-Decoder architecture is utilized for settings where a varied-length input sequence is mapped to the varied-length output sequence. This network can be used for image captioning. In image captioning, the core idea is to use CNN as encoder and a normal RNN (LSTM) as decoder.

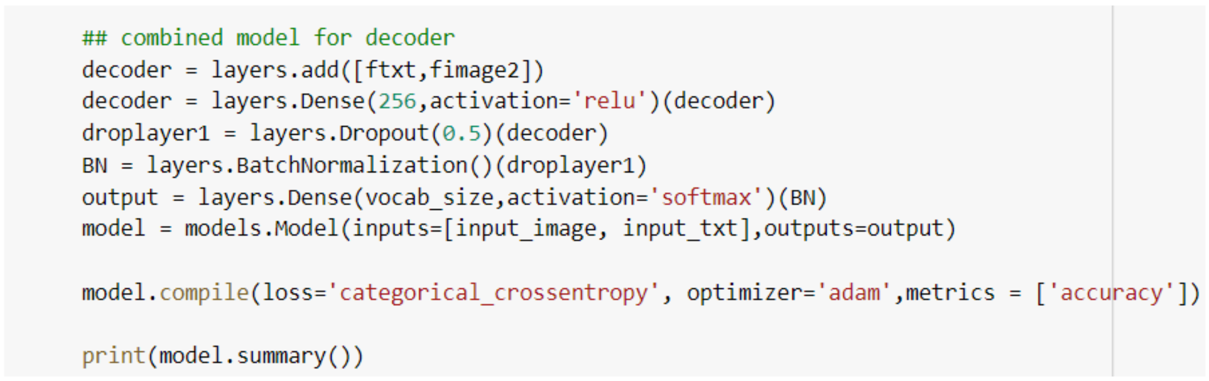
We tried 4 different LSTM models, 3 models where there was a great deal of overfitting, so we tried to overcome it using batch normalization layers, dropout layers and l2 regularization.

However, due to the lack of enough data due to our limited resources, we couldn't pass the accuracy over 50% for train accuracy and around 30% for validation accuracy. We believe that this accuracy can be improved by using the same model on larger datasets.

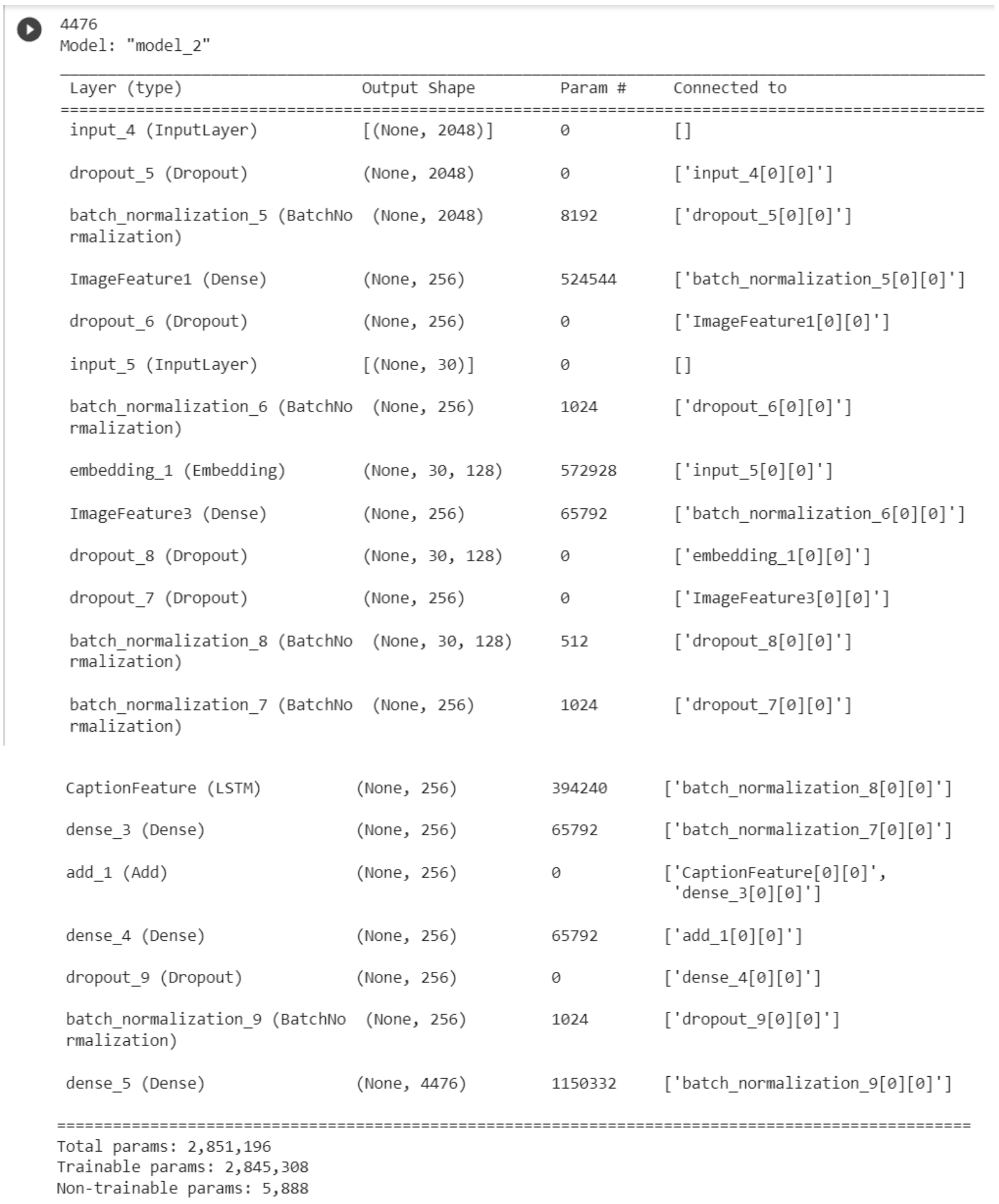
These are screenshots of our best model:



```
[ ] from keras import layers
    from keras.regularizers import l2
    from keras import models

    print(vocab_size)
    ## image feature

    dim_embedding = 128

    input_image = layers.Input(shape=(Ximage_train.shape[1],))
    droplayer1 = layers.Dropout(0.5)(input_image)
    BN = layers.BatchNormalization()(droplayer1)
    fimage1 = layers.Dense(256,activation='relu',name="ImageFeature1",kernel_regularizer=l2(0.01))(BN)
    droplayer1 = layers.Dropout(0.5)(fimage1)
    BN = layers.BatchNormalization()(droplayer1)
    fimage1 = layers.Dense(256,activation='relu',name="ImageFeature3",kernel_regularizer=l2(0.01))(BN)
    droplayer2 = layers.Dropout(0.5)(fimage1)
    BN = layers.BatchNormalization()(droplayer2)
    fimage2 = layers.Dense(256, activation='relu')(BN)


    ## sequence model
    input_txt = layers.Input(shape=(np.max([len(text) for text in dtexts]),))
    ftxt = layers.Embedding(vocab_size,dim_embedding, mask_zero=True)(input_txt)
    droplayer_ = layers.Dropout(0.5)(ftxt)
    BN = layers.BatchNormalization()(droplayer_)
    ftxt = layers.LSTM(256,name="CaptionFeature")(BN)
```

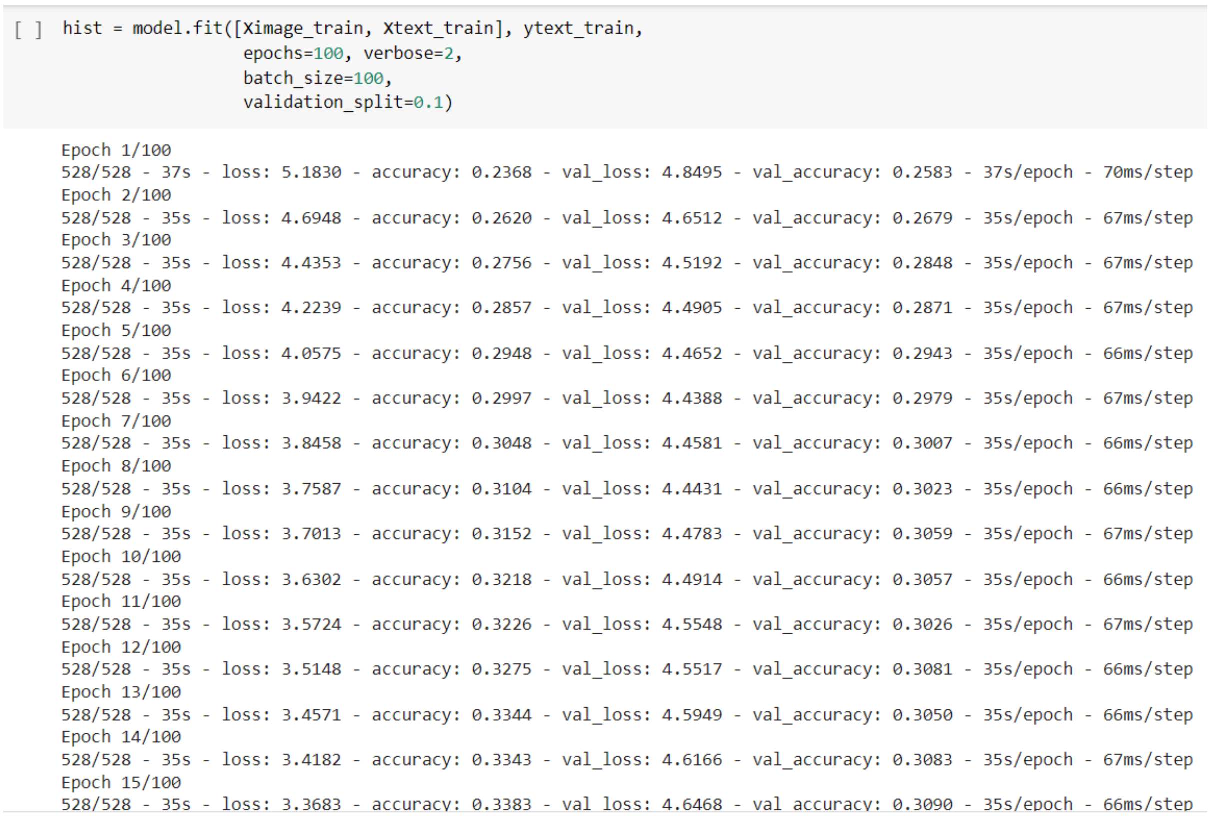


```
## combined model for decoder
decoder = layers.add([ftxt,fimage2])
decoder = layers.Dense(256,activation='relu')(decoder)
droplayer1 = layers.Dropout(0.5)(decoder)
BN = layers.BatchNormalization()(droplayer1)
output = layers.Dense(vocab_size,activation='softmax')(BN)
model = models.Model(inputs=[input_image, input_txt],outputs=output)

model.compile(loss='categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])

print(model.summary())
```



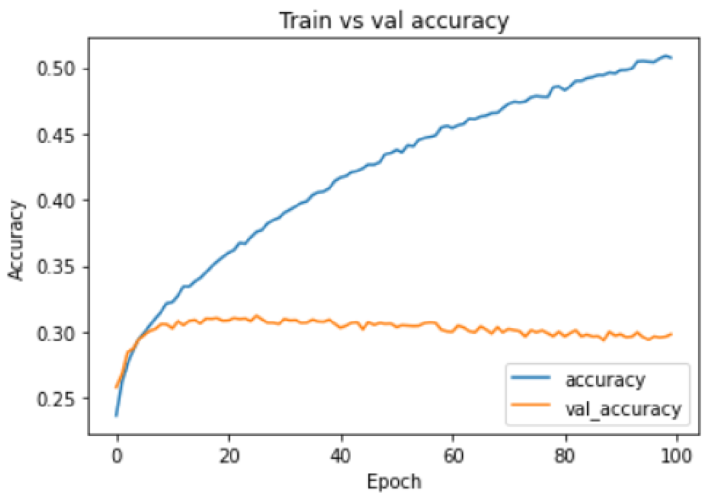
```
4476
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_4 (InputLayer)           [(None, 2048)]       0           []

 dropout_5 (Dropout)            (None, 2048)         0           ['input_4[0][0]']

 batch_normalization_5 (BatchNo  (None, 2048)        8192        ['dropout_5[0][0]']
 rmalization)

 ImageFeature1 (Dense)          (None, 256)          524544      ['batch_normalization_5[0][0]']

 dropout_6 (Dropout)            (None, 256)          0           ['ImageFeature1[0][0]']

 input_5 (InputLayer)           [(None, 30)]         0           []

 batch_normalization_6 (BatchNo  (None, 256)         1024        ['dropout_6[0][0]']
 rmalization)

 embedding_1 (Embedding)        (None, 30, 128)      572928      ['input_5[0][0]']

 ImageFeature3 (Dense)          (None, 256)          65792       ['batch_normalization_6[0][0]']

 dropout_8 (Dropout)            (None, 30, 128)      0           ['embedding_1[0][0]']

 dropout_7 (Dropout)            (None, 256)          0           ['ImageFeature3[0][0]']

 batch_normalization_8 (BatchNo  (None, 30, 128)     512         ['dropout_8[0][0]']
 rmalization)

 batch_normalization_7 (BatchNo  (None, 256)         1024        ['dropout_7[0][0]']
 rmalization)

 CaptionFeature (LSTM)          (None, 256)          394240      ['batch_normalization_8[0][0]']

 dense_3 (Dense)                (None, 256)          65792       ['batch_normalization_7[0][0]']

 add_1 (Add)                    (None, 256)          0           ['CaptionFeature[0][0]',
                                                                  'dense_3[0][0]']

 dense_4 (Dense)                (None, 256)          65792       ['add_1[0][0]']

 dropout_9 (Dropout)            (None, 256)          0           ['dense_4[0][0]']

 batch_normalization_9 (BatchNo  (None, 256)         1024        ['dropout_9[0][0]']
 rmalization)

 dense_5 (Dense)                (None, 4476)         1150332     ['batch_normalization_9[0][0]']

==================================================================================================
Total params: 2,851,196
Trainable params: 2,845,308
Non-trainable params: 5,888
```

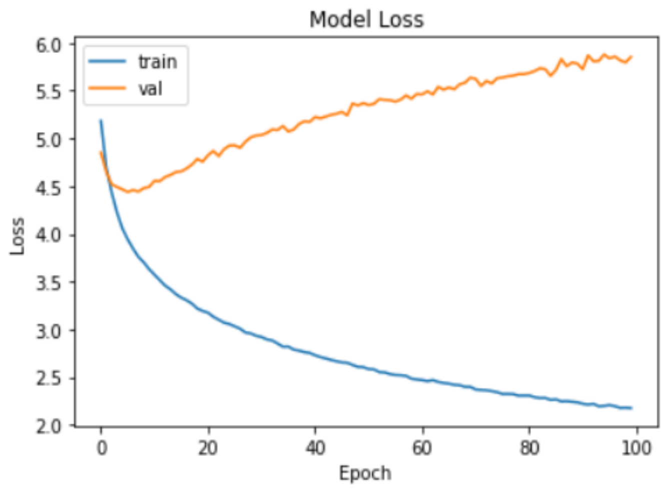
First 15 epochs:



```
[ ] hist = model.fit([Ximage_train, Xtext_train], ytext_train,
                     epochs=100, verbose=2,
                     batch_size=100,
                     validation_split=0.1)
```

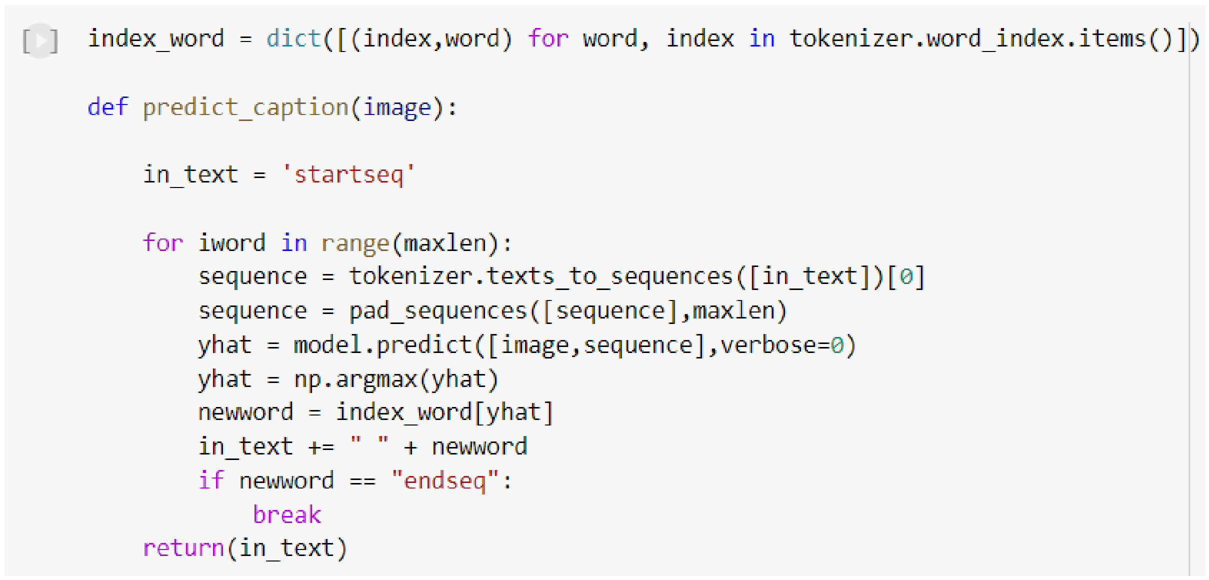
```
Epoch 1/100
528/528 - 37s - loss: 5.1830 - accuracy: 0.2368 - val_loss: 4.8495 - val_accuracy: 0.2583 - 37s/epoch - 70ms/step
Epoch 2/100
528/528 - 35s - loss: 4.6948 - accuracy: 0.2620 - val_loss: 4.6512 - val_accuracy: 0.2679 - 35s/epoch - 67ms/step
Epoch 3/100
528/528 - 35s - loss: 4.4353 - accuracy: 0.2756 - val_loss: 4.5192 - val_accuracy: 0.2848 - 35s/epoch - 67ms/step
Epoch 4/100
528/528 - 35s - loss: 4.2239 - accuracy: 0.2857 - val_loss: 4.4905 - val_accuracy: 0.2871 - 35s/epoch - 67ms/step
Epoch 5/100
528/528 - 35s - loss: 4.0575 - accuracy: 0.2948 - val_loss: 4.4652 - val_accuracy: 0.2943 - 35s/epoch - 66ms/step
Epoch 6/100
528/528 - 35s - loss: 3.9422 - accuracy: 0.2997 - val_loss: 4.4388 - val_accuracy: 0.2979 - 35s/epoch - 67ms/step
Epoch 7/100
528/528 - 35s - loss: 3.8458 - accuracy: 0.3048 - val_loss: 4.4581 - val_accuracy: 0.3007 - 35s/epoch - 66ms/step
Epoch 8/100
528/528 - 35s - loss: 3.7587 - accuracy: 0.3104 - val_loss: 4.4431 - val_accuracy: 0.3023 - 35s/epoch - 66ms/step
Epoch 9/100
528/528 - 35s - loss: 3.7013 - accuracy: 0.3152 - val_loss: 4.4783 - val_accuracy: 0.3059 - 35s/epoch - 67ms/step
Epoch 10/100
528/528 - 35s - loss: 3.6302 - accuracy: 0.3218 - val_loss: 4.4914 - val_accuracy: 0.3057 - 35s/epoch - 66ms/step
Epoch 11/100
528/528 - 35s - loss: 3.5724 - accuracy: 0.3226 - val_loss: 4.5548 - val_accuracy: 0.3026 - 35s/epoch - 67ms/step
Epoch 12/100
528/528 - 35s - loss: 3.5148 - accuracy: 0.3275 - val_loss: 4.5517 - val_accuracy: 0.3081 - 35s/epoch - 66ms/step
Epoch 13/100
528/528 - 35s - loss: 3.4571 - accuracy: 0.3344 - val_loss: 4.5949 - val_accuracy: 0.3050 - 35s/epoch - 66ms/step
Epoch 14/100
528/528 - 35s - loss: 3.4182 - accuracy: 0.3343 - val_loss: 4.6166 - val_accuracy: 0.3083 - 35s/epoch - 67ms/step
Epoch 15/100
528/528 - 35s - loss: 3.3683 - accuracy: 0.3383 - val_loss: 4.6468 - val_accuracy: 0.3090 - 35s/epoch - 66ms/step
```





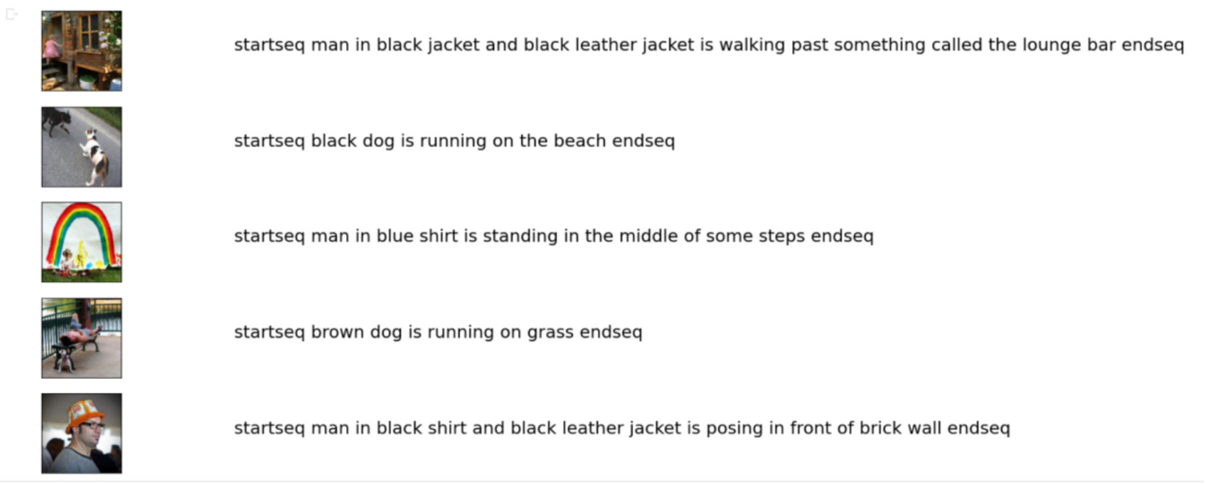
Testing process:

We first made a method that predicts the caption of the images in the test set through our trained model.



```python
index_word = dict([(index,word) for word, index in tokenizer.word_index.items()])

def predict_caption(image):

    in_text = 'startseq'

    for iword in range(maxlen):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence],maxlen)
        yhat = model.predict([image,sequence],verbose=0)
        yhat = np.argmax(yhat)
        newword = index_word[yhat]
        in_text += " " + newword
        if newword == "endseq":
            break
    return(in_text)
```

Then we plotted some of the test set images with their predicted captions.



```
startseq man in black jacket and black leather jacket is walking past something called the lounge bar endseq

startseq black dog is running on the beach endseq

startseq man in blue shirt is standing in the middle of some steps endseq

startseq brown dog is running on grass endseq

startseq man in black shirt and black leather jacket is posing in front of brick wall endseq
```

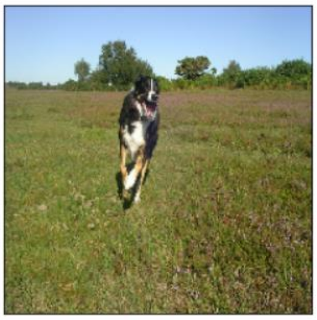
To be able to test the accuracy in a professional way, we used BLEU accuracy. BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated, we provide it with the true caption and the predicted caption of the test image to be able to calculate it.

Then, we divide the predicted captions into two sets, good captions having bleu probability of 0.5 or more and bad captions having bleu probability of 0.2 or less, finally we plot both sets.

```
[ ]  from nltk.translate.bleu_score import sentence_bleu

     index_word = dict([(index,word) for word, index in tokenizer.word_index.items()])

     nkeep = 20
     pred_good, pred_bad, bleus = [], [], []
     count = 0
     for jpgfnm, image_feature, tokenized_text in zip(fnm_test,di_test,dt_test):
         count += 1
         if count % 200 == 0:
             print("  {:4.2f}% is done..".format(100*count/float(len(fnm_test))))

         caption_true = [ index_word[i] for i in tokenized_text ]
         caption_true = caption_true[1:-1] ## remove startreg, and endreg
         ## captions
         caption = predict_caption(image_feature.reshape(1,len(image_feature)))
         caption = caption.split()
         caption = caption[1:-1]## remove startreg, and endreg

         bleu = sentence_bleu([caption_true],caption)
         bleus.append(bleu)

         if bleu > 0.5 and len(pred_good) < nkeep:
             pred_good.append((bleu,jpgfnm,caption_true,caption))
         elif bleu < 0.2 and len(pred_bad) < nkeep:
             pred_bad.append((bleu,jpgfnm,caption_true,caption))
```

Good captions:

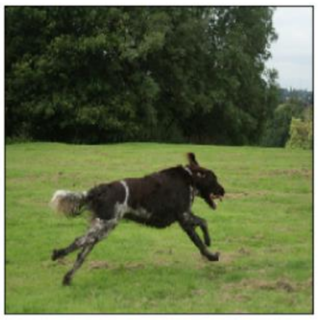
```
Good Captions
```



true: black and white dog is running through the field

pred: black and white dog is running through the grass

BLEU: 0.8633400213704505

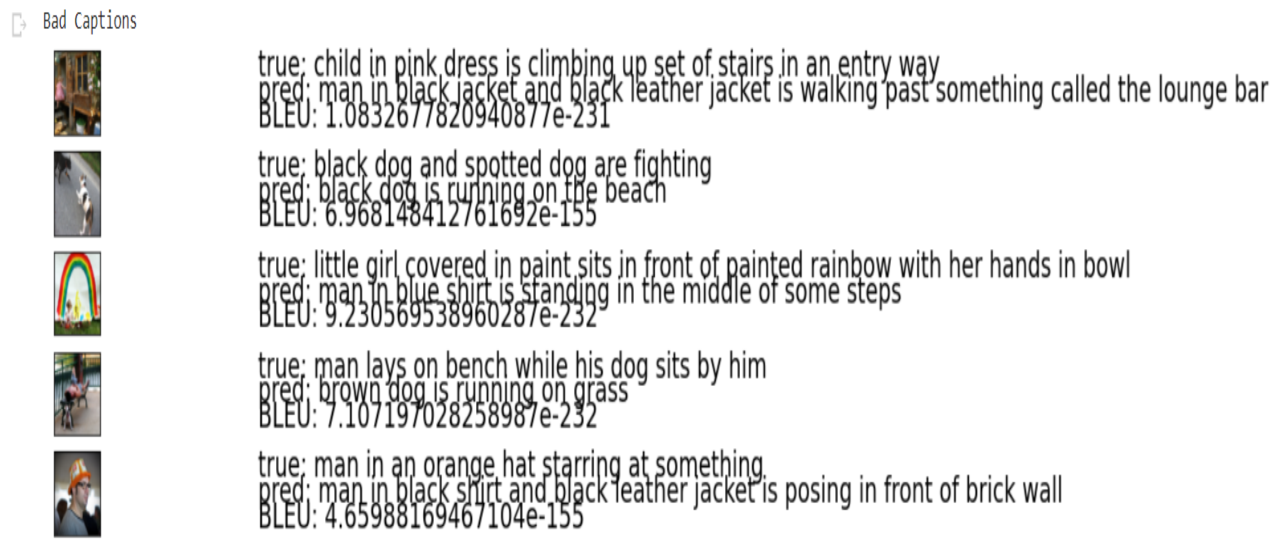


true: black and white dog is running in the grass

pred: black and white dog is running through the grass

BLEU: 0.6606328636027614

Bad captions:



Bad Captions

```
true: child in pink dress is climbing up set of stairs in an entry way
pred: man in black jacket and black leather jacket is walking past something called the lounge bar
BLEU: 1.0832677820940877e-231

true: black dog and spotted dog are fighting
pred: black dog is running on the beach
BLEU: 6.968148412761692e-155

true: little girl covered in paint sits in front of painted rainbow with her hands in bowl
pred: man in blue shirt is standing in the middle of some steps
BLEU: 9.230569538960287e-232

true: man lays on bench while his dog sits by him
pred: brown dog is running on grass
BLEU: 7.107197028258987e-232

true: man in an orange hat starring at something
pred: man in black shirt and black leather jacket is posing in front of brick wall
BLEU: 4.659881694671104e-155
```

Work using the flick30k dataset

As mentioned before, we tried working with the flickr30k dataset, but we faced a lot of obstacles with the runtime and memory, so we ended up using only half the dataset.

The only differences between the flickr8k and flickr30k datasets are the format in which the captions are written, and the train/test split we applied.

Captions format:

Here, the captions are recorded as filename (space) comment number (tab space) the caption. So, in order to break this down we wrote the following method.

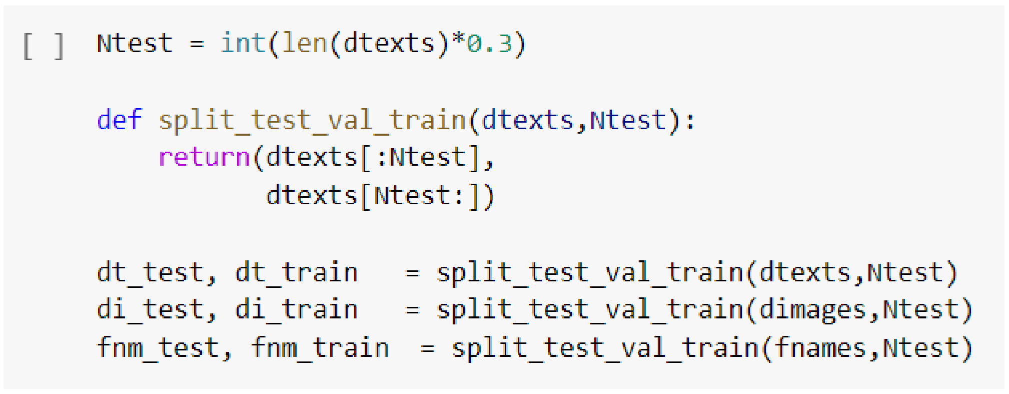
```
def load_csv(directory):
    desc=dict()
    text = pd.read_csv(directory, delimiter='|',header=None,
                       names=["image_name","comment_number",
                              "comment"])
    text = text.iloc[1:,:]
    df_new = text[text.iloc[:,2].notnull()]
    return df_new
```

```
[ ] data = load_csv(captions_path)
    data
```

Train/Test split:

In this dataset, we used 70% of the data for training and 30% for testing.

We used 5% of the training set for validation during our training process.



```
[ ]  Ntest = int(len(dtexts)*0.3)

     def split_test_val_train(dtexts,Ntest):
         return(dtexts[:Ntest],
                dtexts[Ntest:])

     dt_test, dt_train   = split_test_val_train(dtexts,Ntest)
     di_test, di_train   = split_test_val_train(dimages,Ntest)
     fnm_test, fnm_train  = split_test_val_train(fnames,Ntest)
```

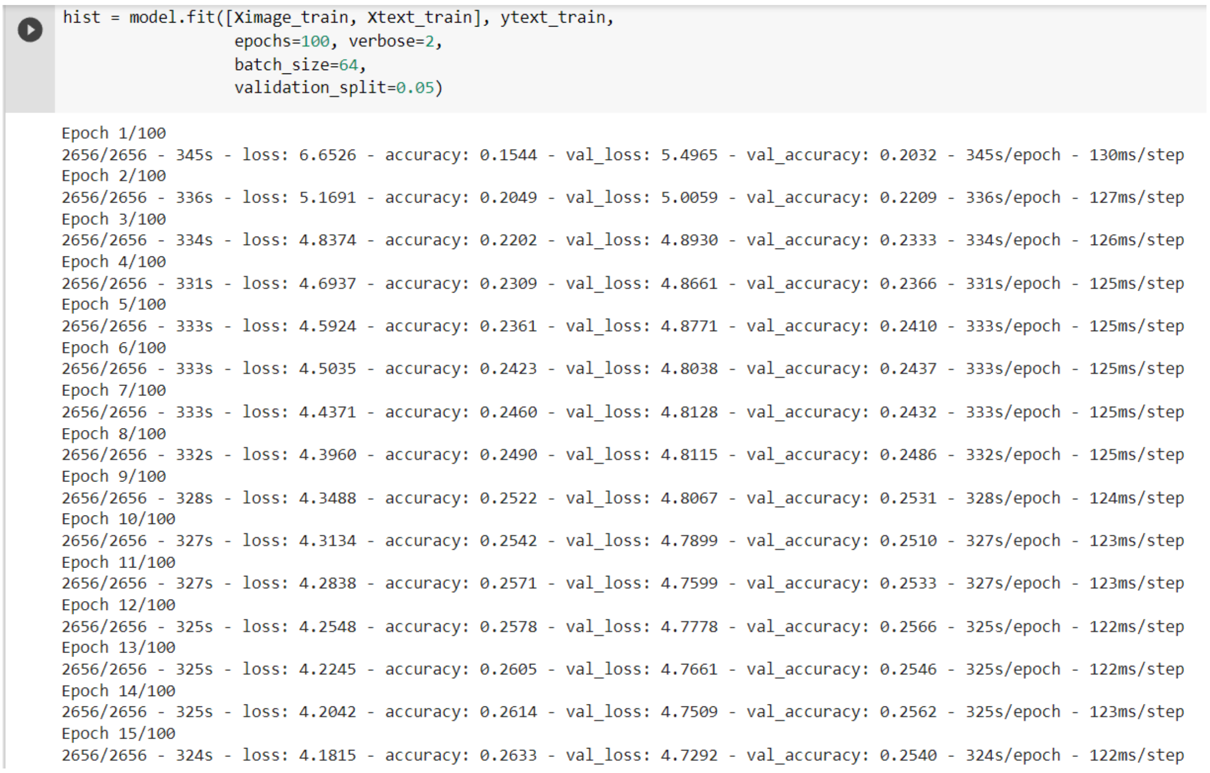
LSTM model:

We used the same model we used on the flickr8k dataset, but here we noticed that the overfitting was less than the flickr8k dataset.

Each epoch took around 330 seconds to run which made the runtime to stop at 91/100 epochs, reaching a train accuracy of around 30% and validation accuracy of around 25%.

```
4h 54m 10s    completed at 4:43 AM
```

First 15 epochs:



```
hist = model.fit([Ximage_train, Xtext_train], ytext_train,
                  epochs=100, verbose=2,
                  batch_size=64,
                  validation_split=0.05)
```

```
Epoch 1/100
2656/2656 - 345s - loss: 6.6526 - accuracy: 0.1544 - val_loss: 5.4965 - val_accuracy: 0.2032 - 345s/epoch - 130ms/step
Epoch 2/100
2656/2656 - 336s - loss: 5.1691 - accuracy: 0.2049 - val_loss: 5.0059 - val_accuracy: 0.2209 - 336s/epoch - 127ms/step
Epoch 3/100
2656/2656 - 334s - loss: 4.8374 - accuracy: 0.2202 - val_loss: 4.8930 - val_accuracy: 0.2333 - 334s/epoch - 126ms/step
Epoch 4/100
2656/2656 - 331s - loss: 4.6937 - accuracy: 0.2309 - val_loss: 4.8661 - val_accuracy: 0.2366 - 331s/epoch - 125ms/step
Epoch 5/100
2656/2656 - 333s - loss: 4.5924 - accuracy: 0.2361 - val_loss: 4.8771 - val_accuracy: 0.2410 - 333s/epoch - 125ms/step
Epoch 6/100
2656/2656 - 333s - loss: 4.5035 - accuracy: 0.2423 - val_loss: 4.8038 - val_accuracy: 0.2437 - 333s/epoch - 125ms/step
Epoch 7/100
2656/2656 - 333s - loss: 4.4371 - accuracy: 0.2460 - val_loss: 4.8128 - val_accuracy: 0.2432 - 333s/epoch - 125ms/step
Epoch 8/100
2656/2656 - 332s - loss: 4.3960 - accuracy: 0.2490 - val_loss: 4.8115 - val_accuracy: 0.2486 - 332s/epoch - 125ms/step
Epoch 9/100
2656/2656 - 328s - loss: 4.3488 - accuracy: 0.2522 - val_loss: 4.8067 - val_accuracy: 0.2531 - 328s/epoch - 124ms/step
Epoch 10/100
2656/2656 - 327s - loss: 4.3134 - accuracy: 0.2542 - val_loss: 4.7899 - val_accuracy: 0.2510 - 327s/epoch - 123ms/step
Epoch 11/100
2656/2656 - 327s - loss: 4.2838 - accuracy: 0.2571 - val_loss: 4.7599 - val_accuracy: 0.2533 - 327s/epoch - 123ms/step
Epoch 12/100
2656/2656 - 325s - loss: 4.2548 - accuracy: 0.2578 - val_loss: 4.7778 - val_accuracy: 0.2566 - 325s/epoch - 122ms/step
Epoch 13/100
2656/2656 - 325s - loss: 4.2245 - accuracy: 0.2605 - val_loss: 4.7661 - val_accuracy: 0.2546 - 325s/epoch - 122ms/step
Epoch 14/100
2656/2656 - 325s - loss: 4.2042 - accuracy: 0.2614 - val_loss: 4.7509 - val_accuracy: 0.2562 - 325s/epoch - 123ms/step
Epoch 15/100
2656/2656 - 324s - loss: 4.1815 - accuracy: 0.2633 - val_loss: 4.7292 - val_accuracy: 0.2540 - 324s/epoch - 122ms/step
```

Links:

- Link for the flickr8k dataset: https://www.kaggle.com/datasets/waelboussbat/flickr8ksau
- Link for the flickr30k dataset: https://www.kaggle.com/datasets/hsankesara/flickr-image-dataset
- Link for image captioning on flickr8k notebook: https://colab.research.google.com/drive/1W8KdNUa77cyEkD0u6l9O8RhjEnFbQd_Z?usp=sharing
- Link for image captioning on flickr30k notebook: https://colab.research.google.com/drive/1XxNaKIe_cqIWg0e_1wl0owAc5W2Xnbf6?usp=sharing

References:

- https://www.geeksforgeeks.org/image-caption-generator-using-deep-learning-on-flickr8k-dataset/
- https://www.researchgate.net/publication/329037107_Image_Captioning_Based_on_Deep_Neural_Networks
- https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2
- https://machinelearningprojects.net/image-captioning/