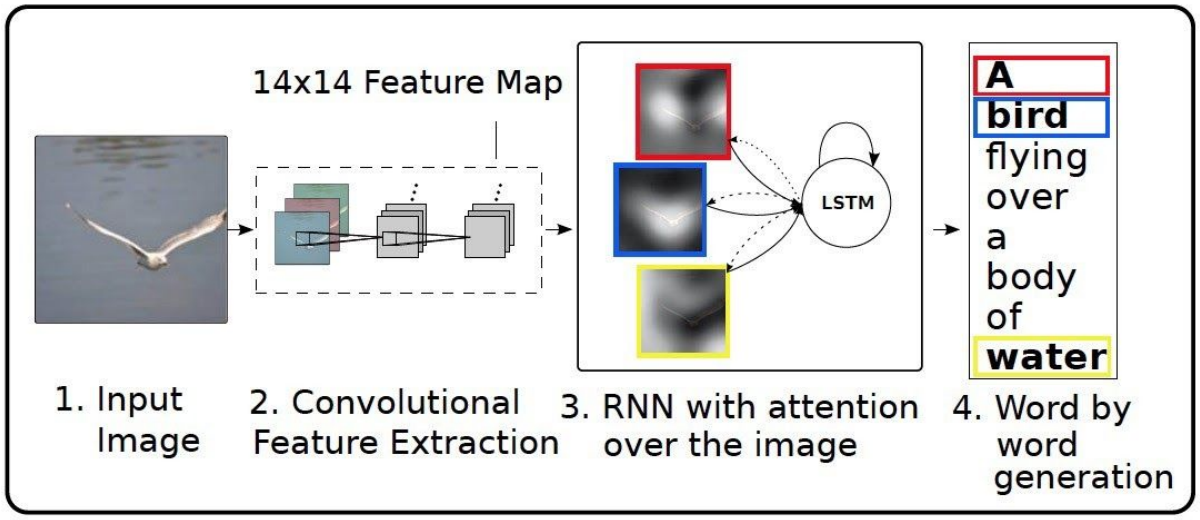
Pattern Recognition

Implementation of Show, Attend and Tell: Neural Image Caption Generation with Visual Attention paper

COCO Image Captioning Dataset



14x14 Feature Map
LSTM
A
bird
flying
over
a
body
of
water
1. Input Image
2. Convolutional Feature Extraction
3. RNN with attention over the image
4. Word by word generation

Noha Nomier	4638
Fatma Sherif	4701
Hagar Barakat	4703
Omar Abouzeid	4813
Amr Fahmy	5119

Spring ‘20

Contents

Introduction

The main objective of implementing Show, Attend and Tell: Neural Image Caption Generation with Visual Attention paper to generate image captions.

Captions are generated by the proposed CNN-LSTM network, in order to capture multiple objects inside an image, features are extracted from the lower convolutional layers.

We used four transfer learning networks for feature extraction:

1. Visual Geometry Group (VGG)
2. Residual neural network (ResNet)
3. MobileNet
4. Inception V3

Deliverables

Code: Colab Link

Summary: Summary Link

Presentation: Presentation Link

Individual Reports:

- Noha Nomier_4638
- Fatma Sherif_4701
- Hagar Barakat 4703
- Omar Abouzeid 4813
- Amr Fahmy 5119

Dataset

COCO dataset is a well known dataset used for benchmarking multiple tasks. such as classification, detection, segmentation and captioning, and for this task specifically it's used for captioning.

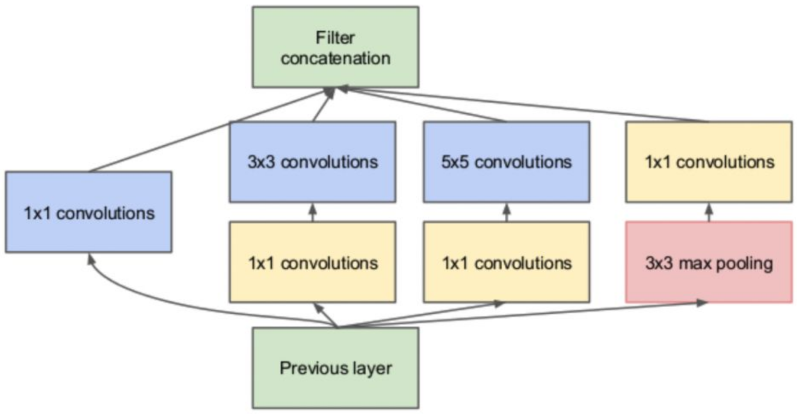
Preprocessing of data

1. Using Inception V3

Generally, Inception V3 is a convolutional neural network architecture.

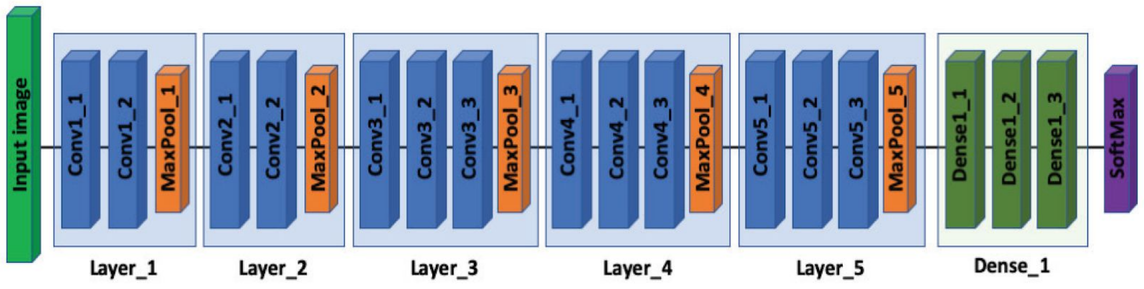
InceptionV3 (which is pre-trained on Imagenet) is used to classify each image. The model expects a format:

- Image of size 299px by 299px
- Normalized image that contains pixels in the range of -1 to 1, which matches the format of the images used to train InceptionV3.



2. Using VGG

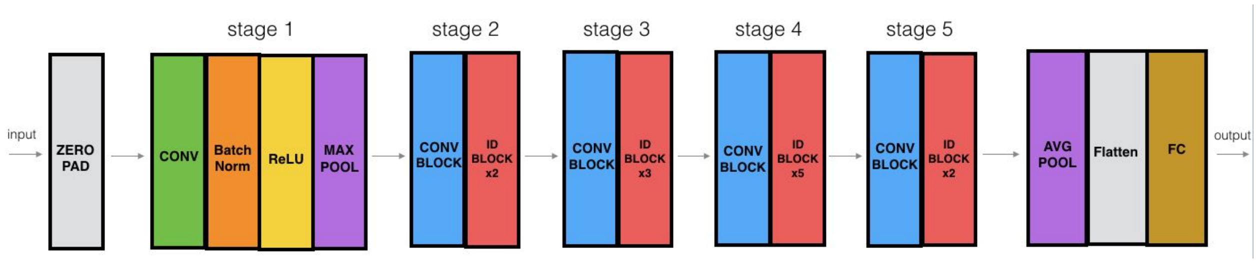
The input to the network is an image of dimensions (224, 224, 3). The first two layers have 64 channels of 3*3 filter size and same padding. Then after a max pool layer of stride (2, 2), two layers which have convolution layers of 256 filter size and filter size (3, 3). This followed by a max pooling layer of stride (2, 2) which is the same as the previous layer. Then there are 2 convolution layers of filter size (3, 3) and 256 filters. After that there are 2 sets of 3 convolution layers and a max pool layer. Each has 512 filters of (3, 3) size with the same padding.This image is then passed to the stack of two convolution layers. In these convolution and max pooling layers, the filters we use is of the size 3*3 instead of 11*11 in AlexNet and 7*7 in ZF-Net. In some of the layers, it also uses 1*1 pixel which is used to manipulate the number of input channels. There is a padding of 1-pixel (same padding) done after each convolution layer to prevent the spatial feature of the image.



After the stack of convolution and max-pooling layer, we got a (7, 7, 512) feature map. We flatten this output to make it a (1, 25088) feature vector.After this there are 3 fully connected layer, the first layer takes input from the last feature vector and outputs a (1, 4096) vector, second layer also outputs a vector of size (1, 4096) but the third layer output a 1000 channels for 1000 classes of ILSVRC challenge, then after the output of 3rd fully connected layer is passed to softmax layer in order to normalize the classification vector. After the output of classification vector top-5 categories for evaluation. All the hidden layers use ReLU as its activation function. ReLU is more computationally efficient because it results in faster learning and it also decreases the likelihood of vanishing gradient problems.

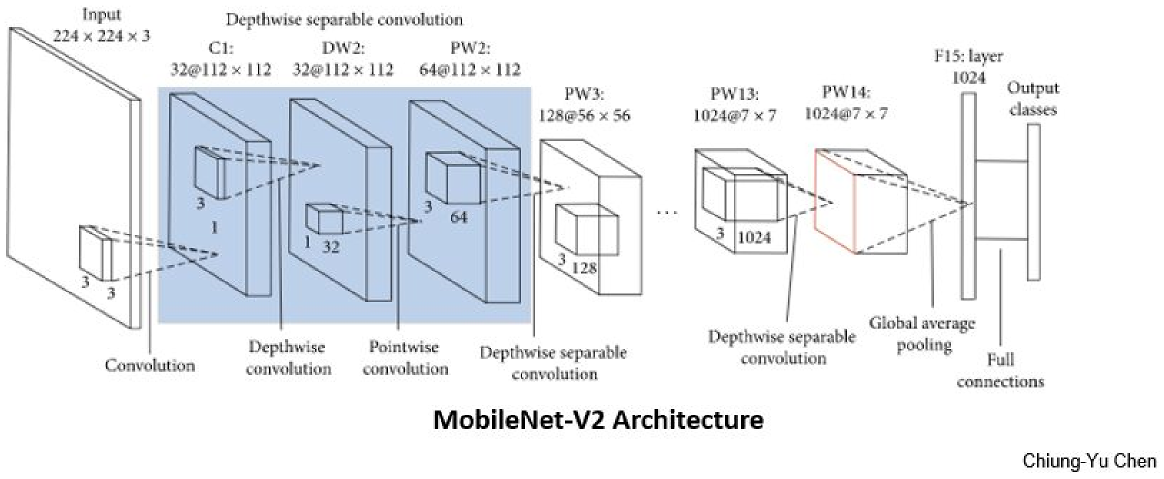
3. Using ResNet

The ResNet-50 model consists of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. The ResNet-50 has over 23 million trainable parameters.



- ResNet uses skip connection to add the output from an earlier layer to a later layer. This helps it mitigate the vanishing gradient problem
- ResNet's input shape has to be (224, 224, 3)

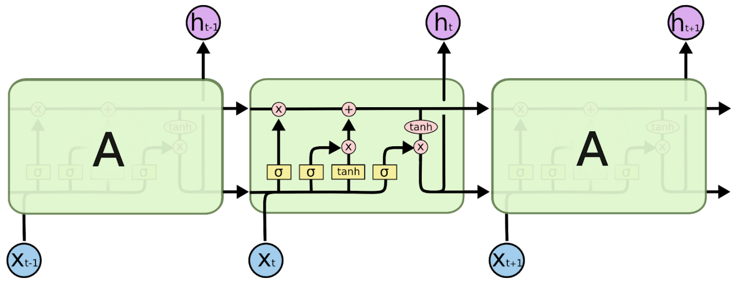
4. Using MobileNet



- In MobileNetV2, there are two types of blocks. One is a residual block with stride of 1. Another one is a block with stride of 2 for downsizing.
- There are 3 layers for both types of blocks.
- This time, the first layer is 1×1 convolution with ReLU6.
- The second layer is the depthwise convolution.
- The third layer is another 1×1 convolution but without any non-linearity. It is claimed that if ReLU is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain.

RNN & LSTM

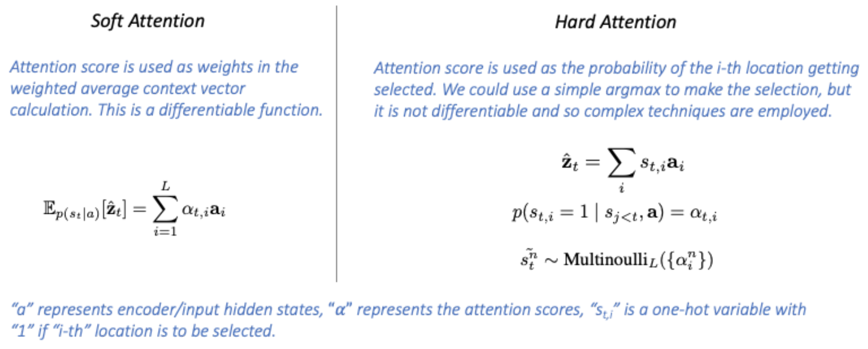
- LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior.
- LSTMs have chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



- At each time-step, the LSTM considers the previous cell state (initial state is the image embedding) and outputs a prediction for the most probable next value in the sequence. This process is repeated until the end token is sampled, signaling the end of the caption
- The top N most probable candidates are kept and utilized in the next inference step. This process continues until either the max sentence length is reached or all sentences have generated the end-of-sentence token.

Soft and Hard attention

Soft attention is when we calculate the context vector as a weighted sum of the encoder hidden states as we had seen in the figures above. Hard attention is when, instead of a weighted average of all hidden states, we use attention scores to select a single hidden state. The selection is an issue, because we could use a function like argmax to make the selection, but it is not differentiable (we are selecting an index corresponding to max score when we use argmax and nudging the weights to move the scores a little as part of backprop will not change this index selection) and therefore more complex techniques are employed.



Soft Attention

Attention score is used as weights in the weighted average context vector calculation. This is a differentiable function.

$$\mathbb{E}_{p(s_t|a)}[\hat{\mathbf{z}}_t] = \sum_{i=1}^{L} \alpha_{t,i}\mathbf{a}_i$$

Hard Attention

Attention score is used as the probability of the i-th location getting selected. We could use a simple argmax to make the selection, but it is not differentiable and so complex techniques are employed.

$$\hat{\mathbf{z}}_t = \sum_i s_{t,i}\mathbf{a}_i$$

$$p(s_{t,i} = 1 \mid s_{j<t}, \mathbf{a}) = \alpha_{t,i}$$

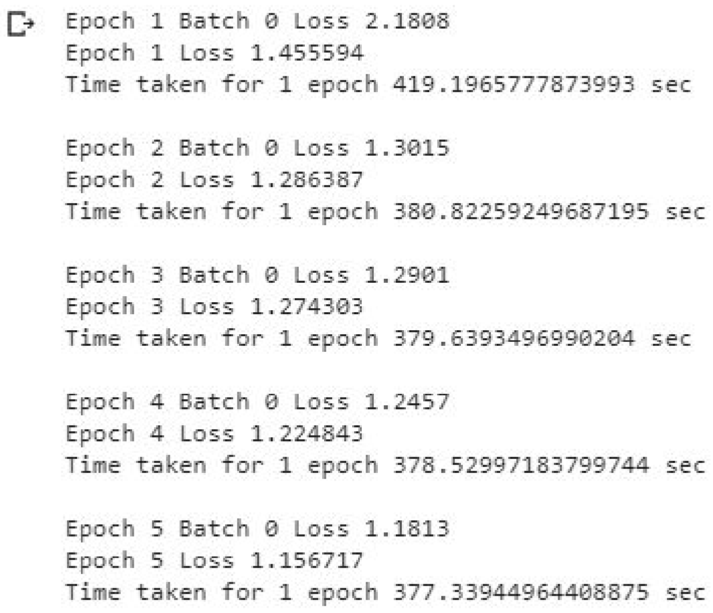
$$\tilde{s_t^n} \sim \text{Multinoulli}_L(\{\alpha_i^n\})$$

"a" represents encoder/input hidden states, "α" represents the attention scores, "$s_{t,i}$" is a one-hot variable with "1" if "i-th" location is to be selected.

Runtime and accuracies

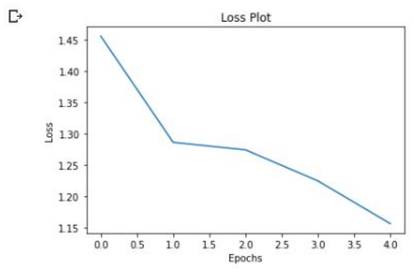
1. Inception V3

- Input shape: has to be `(299, 299, 3)`
- Output shape: `(64, 2048) ”8*8*2048”`
- **Epochs:**

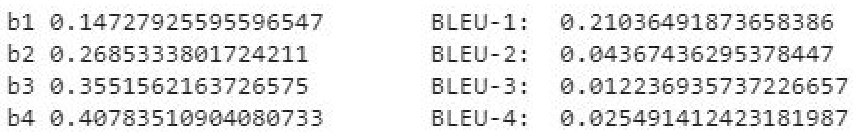


```
Epoch 1 Batch 0 Loss 2.1808
Epoch 1 Loss 1.455594
Time taken for 1 epoch 419.1965777873993 sec

Epoch 2 Batch 0 Loss 1.3015
Epoch 2 Loss 1.286387
Time taken for 1 epoch 380.82259249687195 sec

Epoch 3 Batch 0 Loss 1.2901
Epoch 3 Loss 1.274303
Time taken for 1 epoch 379.6393496990204 sec

Epoch 4 Batch 0 Loss 1.2457
Epoch 4 Loss 1.224843
Time taken for 1 epoch 378.52997183799744 sec

Epoch 5 Batch 0 Loss 1.1813
Epoch 5 Loss 1.156717
Time taken for 1 epoch 377.33944964408875 sec
```

- **Loss Plot:**



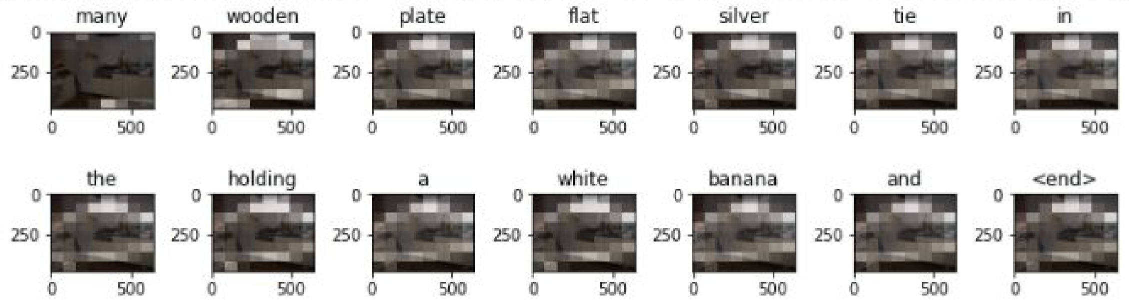
- **Bleu Score:**



```
b1 0.14727925595596547        BLEU-1:  0.21036491873658386
b2 0.2685333801724211         BLEU-2:  0.04367436295378447
b3 0.3551562163726575         BLEU-3:  0.012236935737226657
b4 0.40783510904080733        BLEU-4:  0.025491412423181987
```

- **Image:**

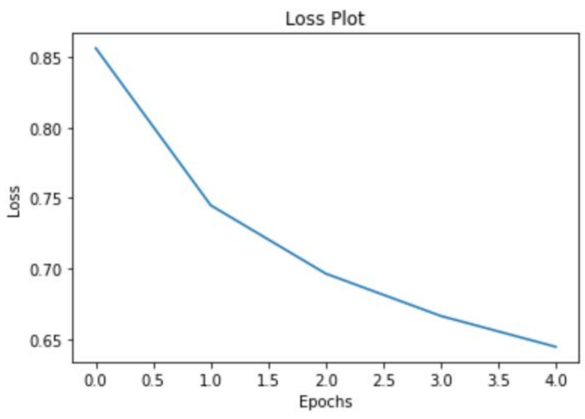
```
Real Caption: <start> a small kitchen with all of the appliances <end>
Prediction Caption: many wooden plate flat silver tie in the holding a white banana and <end>
```



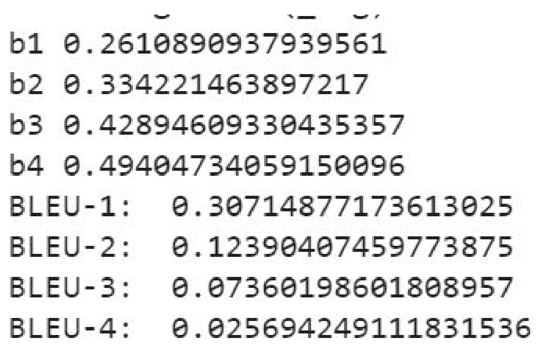
2. MobileNet
   - Input shape: has to be `(224, 224, 3)`
   - Output shape: `(49,1280) "7*7*1280"`
   - **Epochs:**

```
Epoch 1 Batch 0 Loss 0.9439
Epoch 1 Batch 100 Loss 0.8578
Epoch 1 Loss 0.856095
Time taken for 1 epoch 1655.788557767868 sec

Epoch 2 Batch 0 Loss 0.7938
Epoch 2 Batch 100 Loss 0.7438
Epoch 2 Loss 0.744858
Time taken for 1 epoch 1647.800597190857 sec

Epoch 3 Batch 0 Loss 0.7133
Epoch 3 Batch 100 Loss 0.7163
Epoch 3 Loss 0.696496
Time taken for 1 epoch 1648.0806696414948 sec

Epoch 4 Batch 0 Loss 0.6903
Epoch 4 Batch 100 Loss 0.6756
Epoch 4 Loss 0.666541
Time taken for 1 epoch 1646.9259233474731 sec

Epoch 5 Batch 0 Loss 0.6560
Epoch 5 Batch 100 Loss 0.6661
Epoch 5 Loss 0.644687
Time taken for 1 epoch 1646.244445323944 sec
```

- **Loss Plot:**

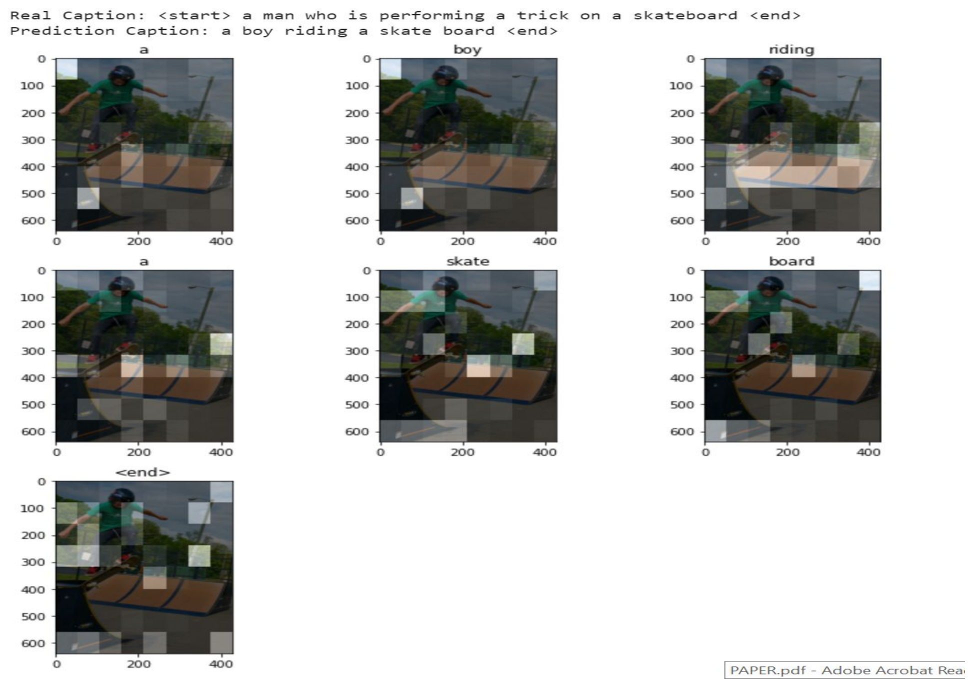


- **Bleu Score:**



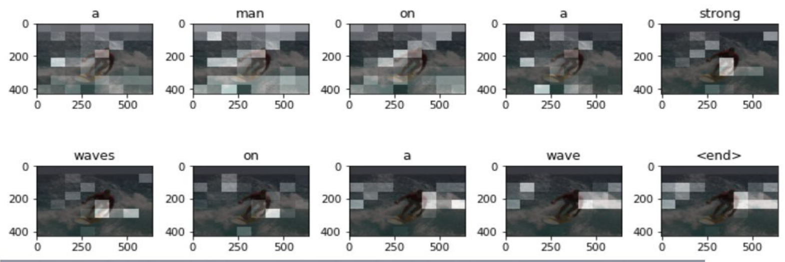
```
b1 0.2610890937939561
b2 0.334221463897217
b3 0.42894609330435357
b4 0.49404734059150096
BLEU-1:  0.30714877173613025
BLEU-2:  0.12390407459773875
BLEU-3:  0.07360198601808957
BLEU-4:  0.025694249111831536
```

- **Validation Image:**



```
Real Caption: <start> a man who is performing a trick on a skateboard <end>
Prediction Caption: a boy riding a skate board <end>
```

- **Test Image:**

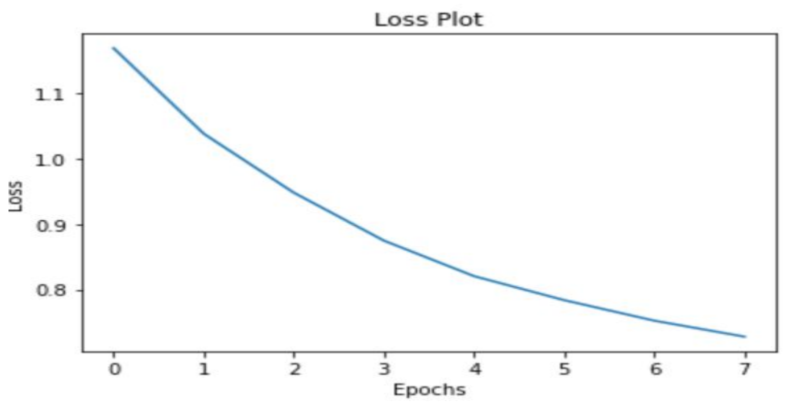


3. ResNet

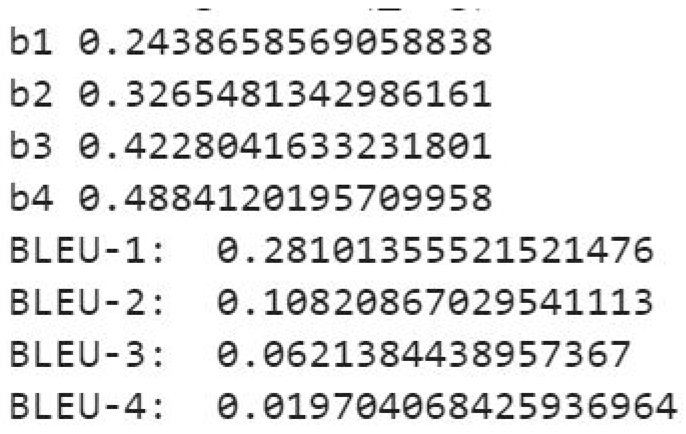
- Input shape: has to be `(224, 224, 3)`
- Output shape: `(49,2048) "7*7*2048"`
- **Epochs:**

```
Epoch 3 Batch 0 Loss 1.2462
Epoch 3 Loss 1.168575
Time taken for 1 epoch 821.5742135047913 sec

Epoch 4 Batch 0 Loss 1.0688
Epoch 4 Loss 1.037944
Time taken for 1 epoch 767.6885824203491 sec

Epoch 5 Batch 0 Loss 0.9749
Epoch 5 Loss 0.947917
Time taken for 1 epoch 764.2853863239288 sec

Epoch 6 Batch 0 Loss 0.8823
Epoch 6 Loss 0.874695
Time taken for 1 epoch 765.4291055202484 sec

Epoch 7 Batch 0 Loss 0.8291
Epoch 7 Loss 0.820458
Time taken for 1 epoch 765.7617399692535 sec

Epoch 8 Batch 0 Loss 0.7721
Epoch 8 Loss 0.783826
Time taken for 1 epoch 765.9377481937408 sec

Epoch 9 Batch 0 Loss 0.7427
Epoch 9 Loss 0.752494
Time taken for 1 epoch 766.6396014690399 sec

Epoch 10 Batch 0 Loss 0.7283
Epoch 10 Loss 0.728077
Time taken for 1 epoch 766.1348278522491 sec
```

- **Loss Plot:**

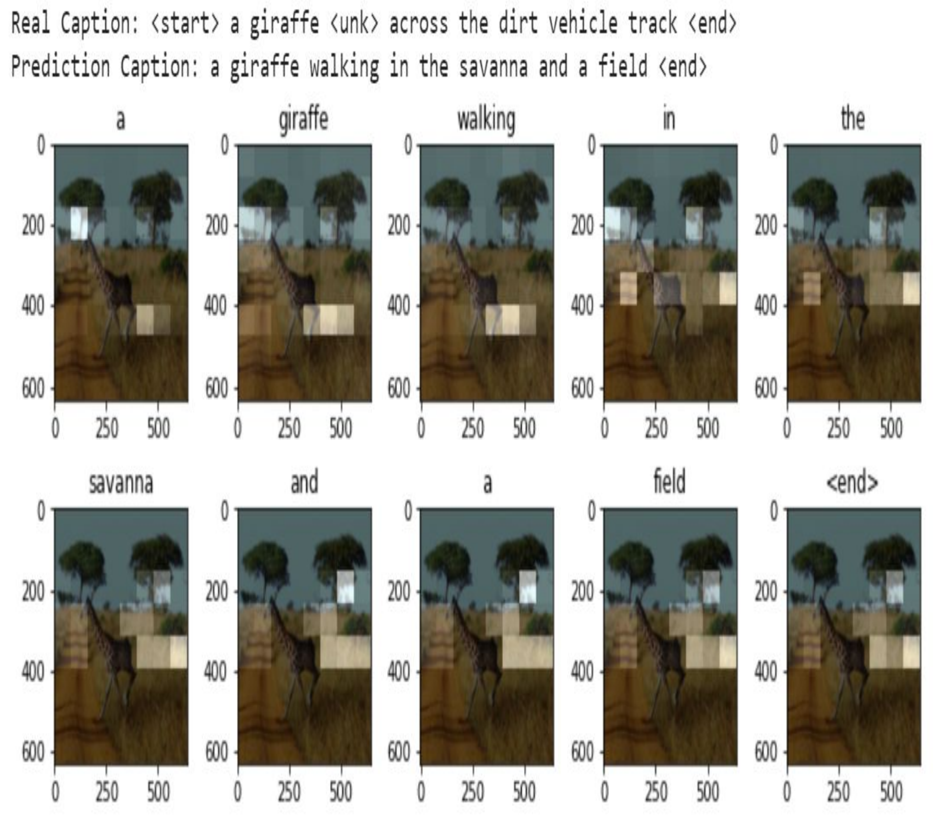


- **Bleu Score:**



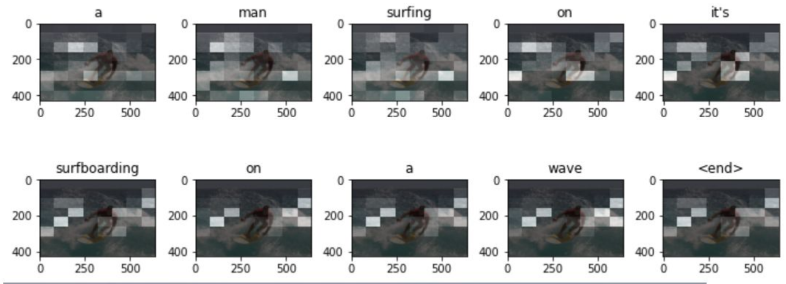
```
b1 0.2438658569058838
b2 0.3265481342986161
b3 0.4228041633231801
b4 0.4884120195709958
BLEU-1:  0.28101355521521476
BLEU-2:  0.10820867029541113
BLEU-3:  0.0621384438957367
BLEU-4:  0.019704068425936964
```

- **Images:**



Real Caption: <start> a giraffe <unk> across the dirt vehicle track <end>
Prediction Caption: a giraffe walking in the savanna and a field <end>

Prediction Caption: a man surfing on it's surfboarding on a wave <end>

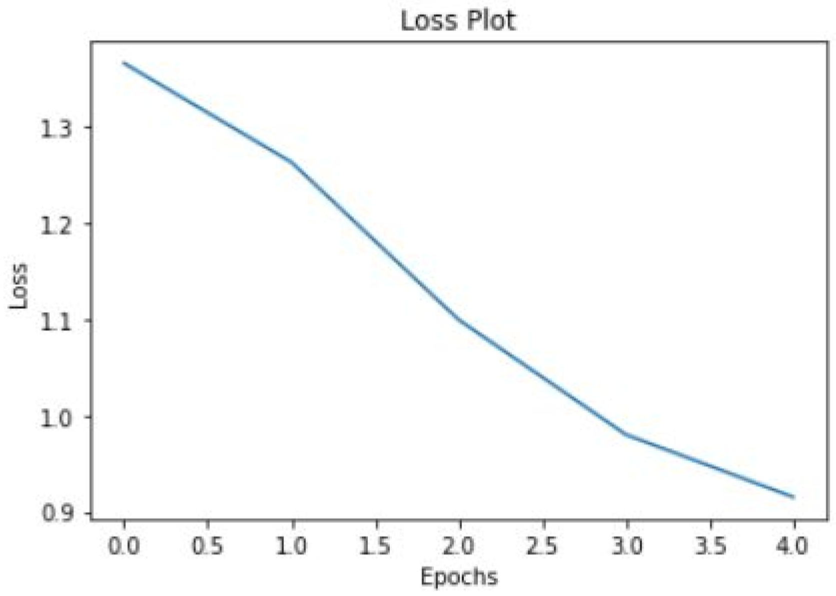


4. VGG

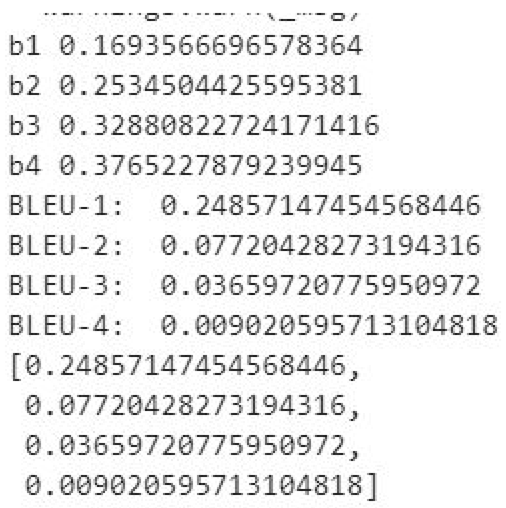
- Input shape: has to be `(224, 224, 3)`
- Output shape: `(49,512)"7*7*512"`
- **Epochs: 5**

```
Epoch 1 Batch 0 Loss 2.1356
Epoch 1 Loss 1.364952
Time taken for 1 epoch 217.50674033164978 sec

Epoch 2 Batch 0 Loss 1.2657
Epoch 2 Loss 1.262421
Time taken for 1 epoch 99.00246119499207 sec

Epoch 3 Batch 0 Loss 1.1418
Epoch 3 Loss 1.099102
Time taken for 1 epoch 110.71923637390137 sec

Epoch 4 Batch 0 Loss 1.0062
Epoch 4 Loss 0.980034
Time taken for 1 epoch 86.16047382354736 sec

Epoch 5 Batch 0 Loss 0.9196
Epoch 5 Loss 0.915326
Time taken for 1 epoch 77.23488068580627 sec
```

- **Loss Plot:**



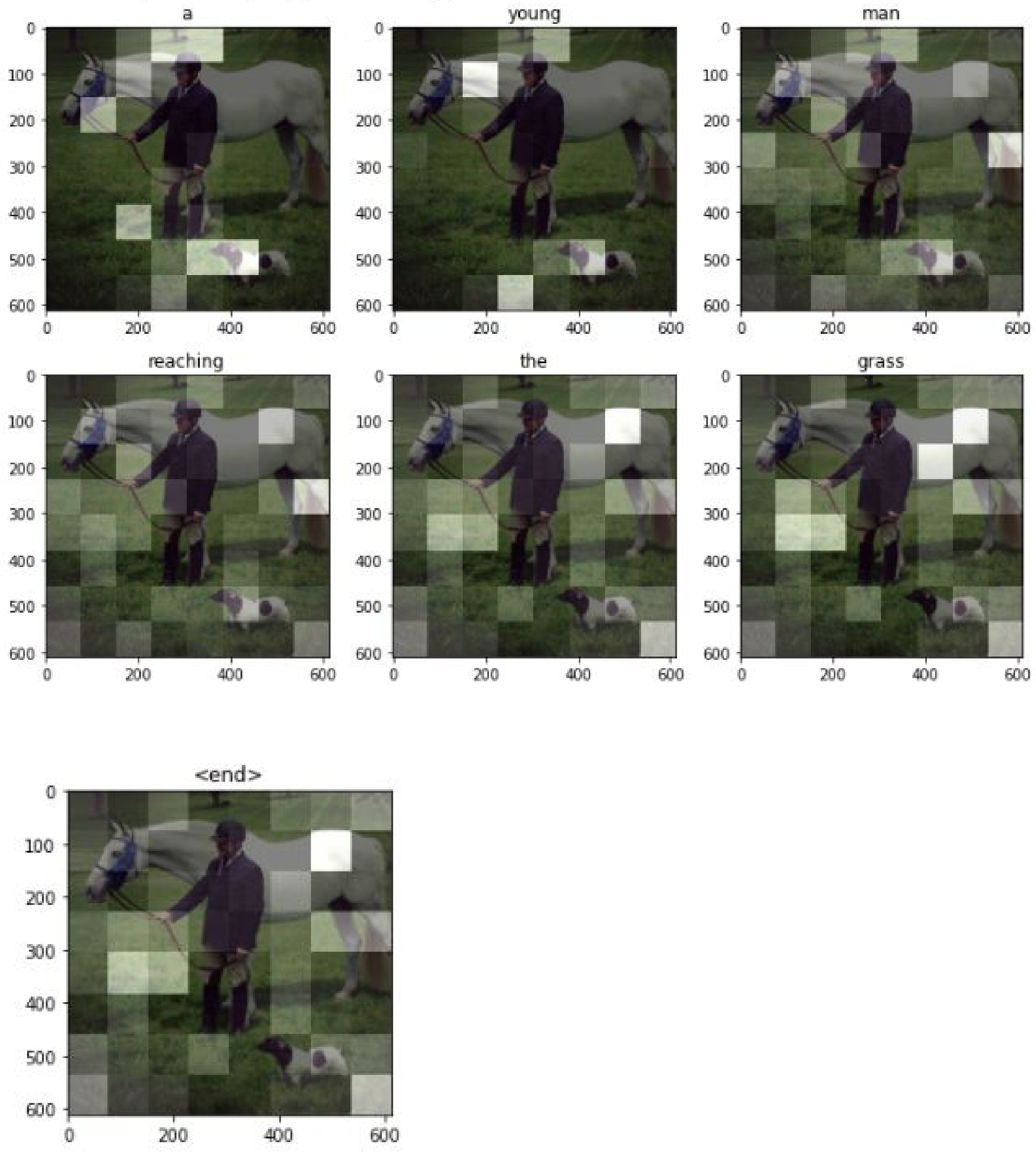
- **Bleu Score:**



```
b1 0.1693566696578364
b2 0.2534504425595381
b3 0.32880822724171416
b4 0.3765227879239945
BLEU-1:  0.24857147454568446
BLEU-2:  0.07720428273194316
BLEU-3:  0.03659720775950972
BLEU-4:  0.009020595713104818
[0.24857147454568446,
 0.07720428273194316,
 0.03659720775950972,
 0.009020595713104818]
```

- **Image:**

```
Real Caption: <start> a jockey with his horse and dog standing in a field <end>
Prediction Caption: a young man reaching the grass <end>
```

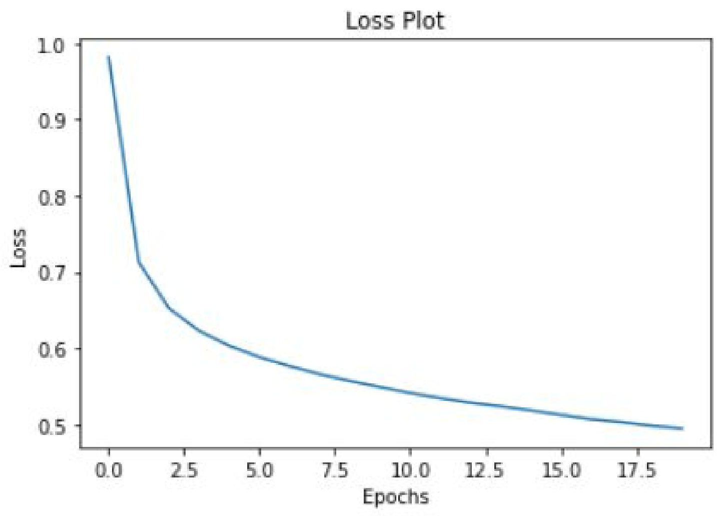


VGG

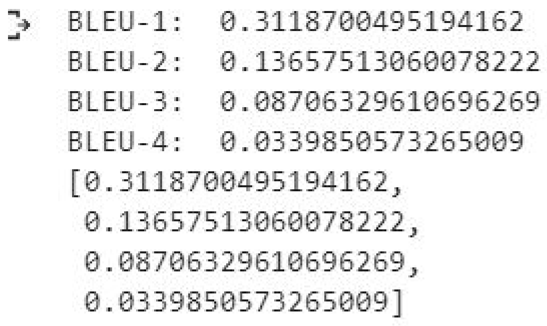
Epochs: 20

```
Epoch 17 Loss 0.506829
Time taken for 1 epoch 917.6034195423126 sec

Epoch 18 Batch 0 Loss 0.4983
Epoch 18 Batch 100 Loss 0.5088
Epoch 18 Batch 200 Loss 0.5067
Epoch 18 Batch 300 Loss 0.4960
Epoch 18 Loss 0.502945
Time taken for 1 epoch 910.9847905635834 sec

Epoch 19 Batch 0 Loss 0.4841
Epoch 19 Batch 100 Loss 0.5092
Epoch 19 Batch 200 Loss 0.4946
Epoch 19 Batch 300 Loss 0.4967
Epoch 19 Loss 0.498118
Time taken for 1 epoch 912.1112067699432 sec

Epoch 20 Batch 0 Loss 0.4934
Epoch 20 Batch 100 Loss 0.4932
Epoch 20 Batch 200 Loss 0.4826
Epoch 20 Batch 300 Loss 0.4900
Epoch 20 Loss 0.494667
Time taken for 1 epoch 910.8196933269501 sec
```

Loss plot:



Bleu Score:



```
BLEU-1:  0.3118700495194162
BLEU-2:  0.13657513060078222
BLEU-3:  0.08706329610696269
BLEU-4:  0.0339850573265009
[0.3118700495194162,
 0.13657513060078222,
 0.08706329610696269,
 0.0339850573265009]
```