# Deep Learning: Final Prroject

Due on January 27, 2020 at 11:55pm

*Professor Emad Fatemizadeh*

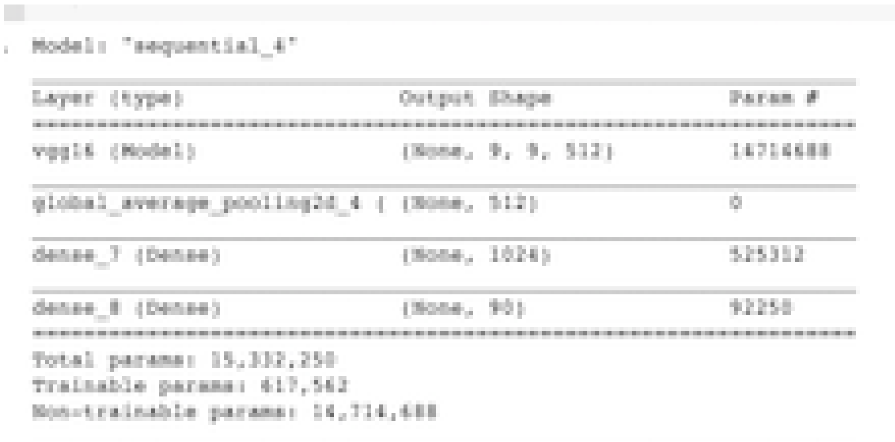
**Amirhossein Yousefi**

**97206984**

# Image Captioning Part1

**Multi Label Classification :**

In this section *VGG* net as a pretrained model which was trained on imagenet dataset, is used to classify 80 different categories and *Adam* is used as an optimizer and *binary cross entropy* is used for loss function.The structure of network and results is shown bellow.



Model: "sequential_4"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| vgg16 (Model) | (None, 9, 9, 512) | 14714688 |
| global_average_pooling2d_4 ( | (None, 512) | 0 |
| dense_7 (Dense) | (None, 1024) | 525312 |
| dense_8 (Dense) | (None, 90) | 92250 |

Total params: 15,332,250
Trainable params: 617,562
Non-trainable params: 14,714,688

Figure 1: Structure of network

:

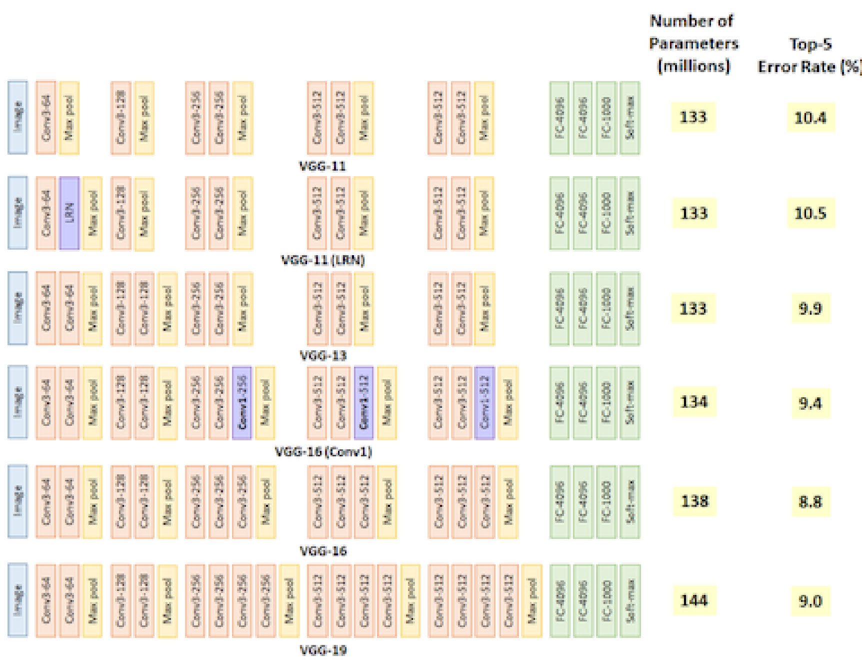



Figure 2: Structure of VGG net

:

Note that as we do multi class classification, *sigmoid* for last layer is used. For better usage of Ram, we load images partially not entirely and also label smoothing is used in order to prevent model from overfitting. Here is results:

```
Epoch 1/15
500/500 [==============================] - 115s 230ms/step - loss: 0.0884 - top_k_categorical_accuracy: 0.7891 - acc: 0.9646 - val_loss: 0.0717 - val_top_k_categorical_accuracy: 0.8420 - val_acc: 0.9655
Epoch 2/15
500/500 [==============================] - 115s 230ms/step - loss: 0.0685 - top_k_categorical_accuracy: 0.8638 - acc: 0.9649 - val_loss: 0.0686 - val_top_k_categorical_accuracy: 0.8613 - val_acc: 0.9655
Epoch 3/15
500/500 [==============================] - 114s 228ms/step - loss: 0.0631 - top_k_categorical_accuracy: 0.8818 - acc: 0.9646 - val_loss: 0.0667 - val_top_k_categorical_accuracy: 0.8707 - val_acc: 0.9648
Epoch 4/15
500/500 [==============================] - 114s 228ms/step - loss: 0.0596 - top_k_categorical_accuracy: 0.8925 - acc: 0.9646 - val_loss: 0.0656 - val_top_k_categorical_accuracy: 0.8740 - val_acc: 0.9643
Epoch 5/15
500/500 [==============================] - 113s 227ms/step - loss: 0.0565 - top_k_categorical_accuracy: 0.9031 - acc: 0.9646 - val_loss: 0.0661 - val_top_k_categorical_accuracy: 0.8683 - val_acc: 0.9642
Epoch 6/15
500/500 [==============================] - 114s 227ms/step - loss: 0.0536 - top_k_categorical_accuracy: 0.9091 - acc: 0.9647 - val_loss: 0.0645 - val_top_k_categorical_accuracy: 0.8790 - val_acc: 0.9638
Epoch 7/15
500/500 [==============================] - 114s 228ms/step - loss: 0.0507 - top_k_categorical_accuracy: 0.9186 - acc: 0.9648 - val_loss: 0.0648 - val_top_k_categorical_accuracy: 0.8770 - val_acc: 0.9639
Epoch 8/15
500/500 [==============================] - 115s 230ms/step - loss: 0.0478 - top_k_categorical_accuracy: 0.9280 - acc: 0.9649 - val_loss: 0.0659 - val_top_k_categorical_accuracy: 0.8737 - val_acc: 0.9633
Epoch 9/15
500/500 [==============================] - 115s 230ms/step - loss: 0.0452 - top_k_categorical_accuracy: 0.9318 - acc: 0.9651 - val_loss: 0.0668 - val_top_k_categorical_accuracy: 0.8663 - val_acc: 0.9618
Epoch 10/15
500/500 [==============================] - 114s 229ms/step - loss: 0.0424 - top_k_categorical_accuracy: 0.9392 - acc: 0.9653 - val_loss: 0.0662 - val_top_k_categorical_accuracy: 0.8727 - val_acc: 0.9631
Epoch 11/15
500/500 [==============================] - 113s 226ms/step - loss: 0.0394 - top_k_categorical_accuracy: 0.9472 - acc: 0.9655 - val_loss: 0.0677 - val_top_k_categorical_accuracy: 0.8743 - val_acc: 0.9640
Epoch 12/15
500/500 [==============================] - 114s 227ms/step - loss: 0.0367 - top_k_categorical_accuracy: 0.9519 - acc: 0.9657 - val_loss: 0.0683 - val_top_k_categorical_accuracy: 0.8800 - val_acc: 0.9624
Epoch 13/15
500/500 [==============================] - 113s 226ms/step - loss: 0.0338 - top_k_categorical_accuracy: 0.9590 - acc: 0.9659 - val_loss: 0.0693 - val_top_k_categorical_accuracy: 0.8737 - val_acc: 0.9633
Epoch 14/15
500/500 [==============================] - 113s 226ms/step - loss: 0.0313 - top_k_categorical_accuracy: 0.9606 - acc: 0.9660 - val_loss: 0.0710 - val_top_k_categorical_accuracy: 0.8700 - val_acc: 0.9623
Epoch 15/15
500/500 [==============================] - 114s 228ms/step - loss: 0.0285 - top_k_categorical_accuracy: 0.9665 - acc: 0.9663 - val_loss: 0.0712 - val_top_k_categorical_accuracy: 0.8660 - val_acc: 0.9620
<keras.callbacks.History at 0x7ff76f09efd0>
```

Figure 3: Results

:

## Image Captioning Part2

**Captioning:**

As mentioned some preprocessing needed for captioning such as padding and tokenizing it and using *word2vec* models. Here *GRU* is used as *RNN* network and *inceptionV2* as a pretrained model and in *20 Epochs* and also 6000 top word is chosen for vocabulary and other hyper parameters is shown bellow.

**Model hyperparameters**

```
BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
vocab_size = top_k + 1
num_steps = len(img_name_train) // BATCH_SIZE
# Shape of the vector extracted from InceptionV3 is (64, 2048)
# These two variables represent that vector shape
features_shape = 2048
attention_features_shape = 64
```

Figure 4: HyperParameters

:

**Attention** structure and is shown bellow and it is adapted from *tensorflow.org* And here is a brief *sudo code*

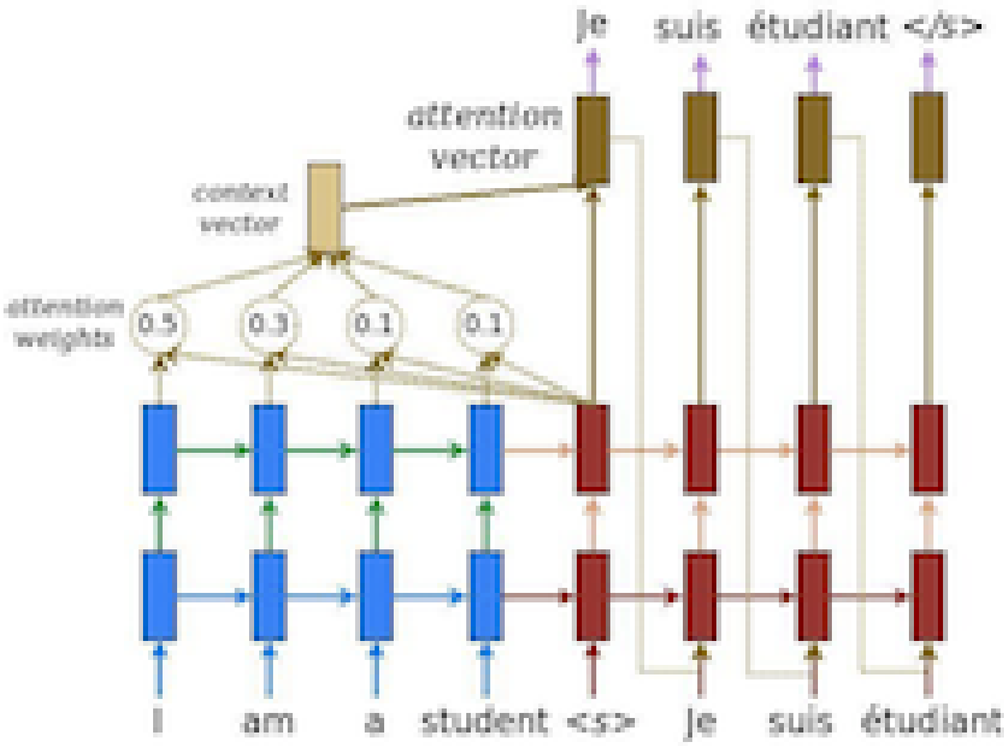



Figure 5: Bahdanau attention model

:

of that.

$$\alpha_{ts} = \frac{\exp\left(\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \quad \text{[Attention weights]} \quad (1)$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \quad \text{[Context vector]} \quad (2)$$

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t]) \quad \text{[Attention vector]} \quad (3)$$

$$\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \boldsymbol{W} \bar{\boldsymbol{h}}_s & \text{[Luong's multiplicative style]} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_1 \boldsymbol{h}_t + \boldsymbol{W}_2 \bar{\boldsymbol{h}}_s\right) & \text{[Bahdanau's additive style]} \end{cases} \quad (4)$$

This tutorial uses Bahdanau attention for the encoder. Let's decide on notation before writing the simplified form:

- FC = Fully connected (dense) layer
- EO = Encoder output
- H = hidden state
- X = input to the decoder

And the pseudo-code:

- `score = FC(tanh(FC(EO) + FC(H)))`

Figure 6: Bahdanau attention model adapted from tensorflow.org

:

Note that here we used *Adam* as an optimizer and *SparseCategoricalCrossentropy* as a loss function and distributed method of *gradient* is used by *GradientTape()* and *@tf.function* is used to compile sub functions or model into a graph for faster execution and running on *GPU* and *teacher forcing* is used for training *GRU*. At the end *Attention* is visualized on photos for better illustration. Here is some captions generated by network on validation set and it's real caption which show amazing results in some cases.

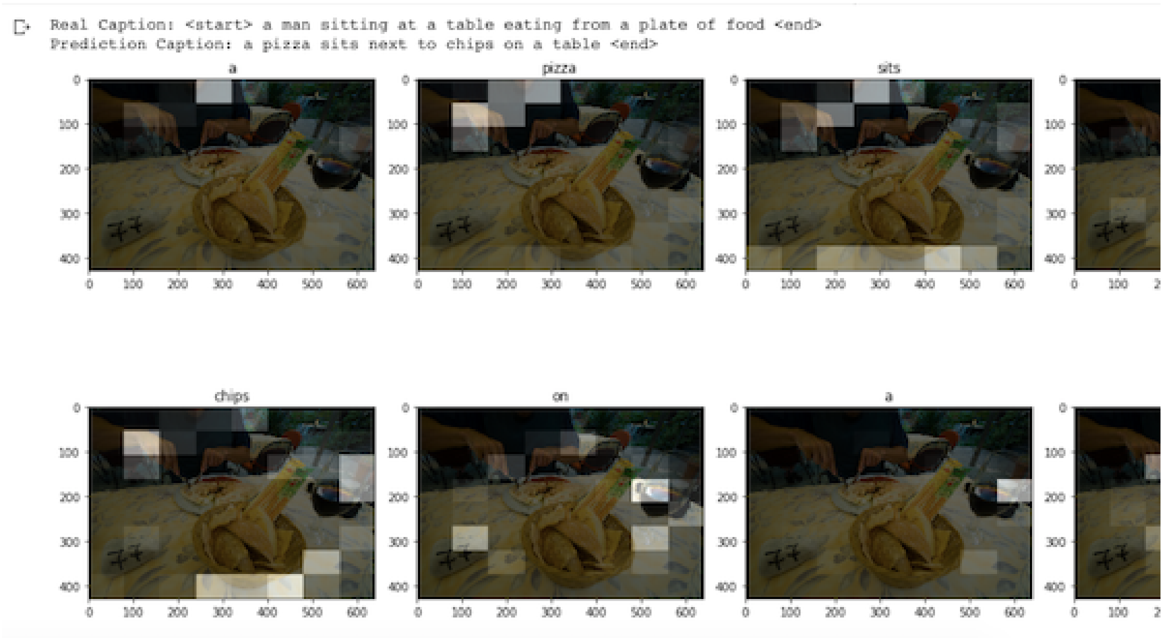



Figure 7: Caption

:

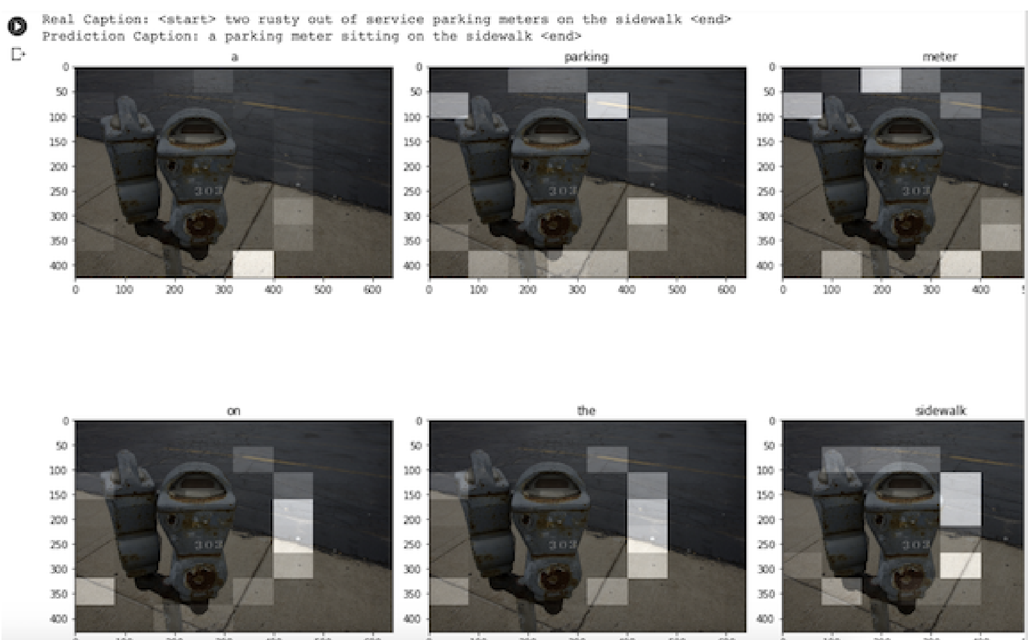



Figure 8: Caption

:

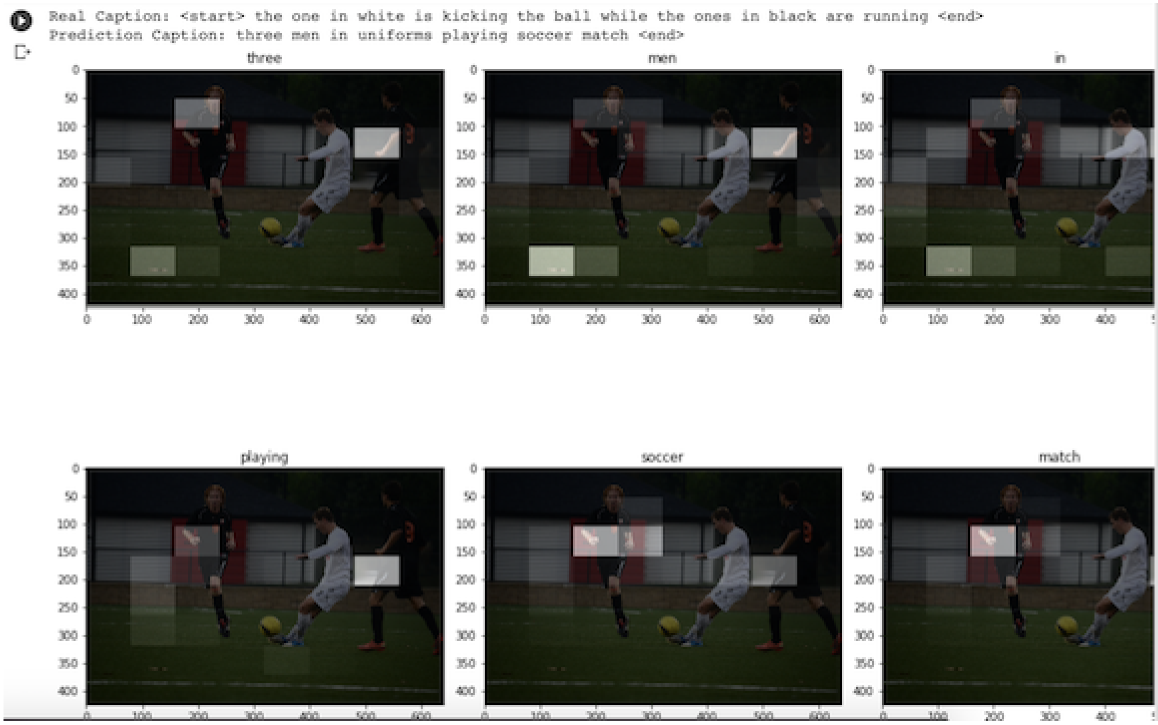



Figure 9: Caption

:

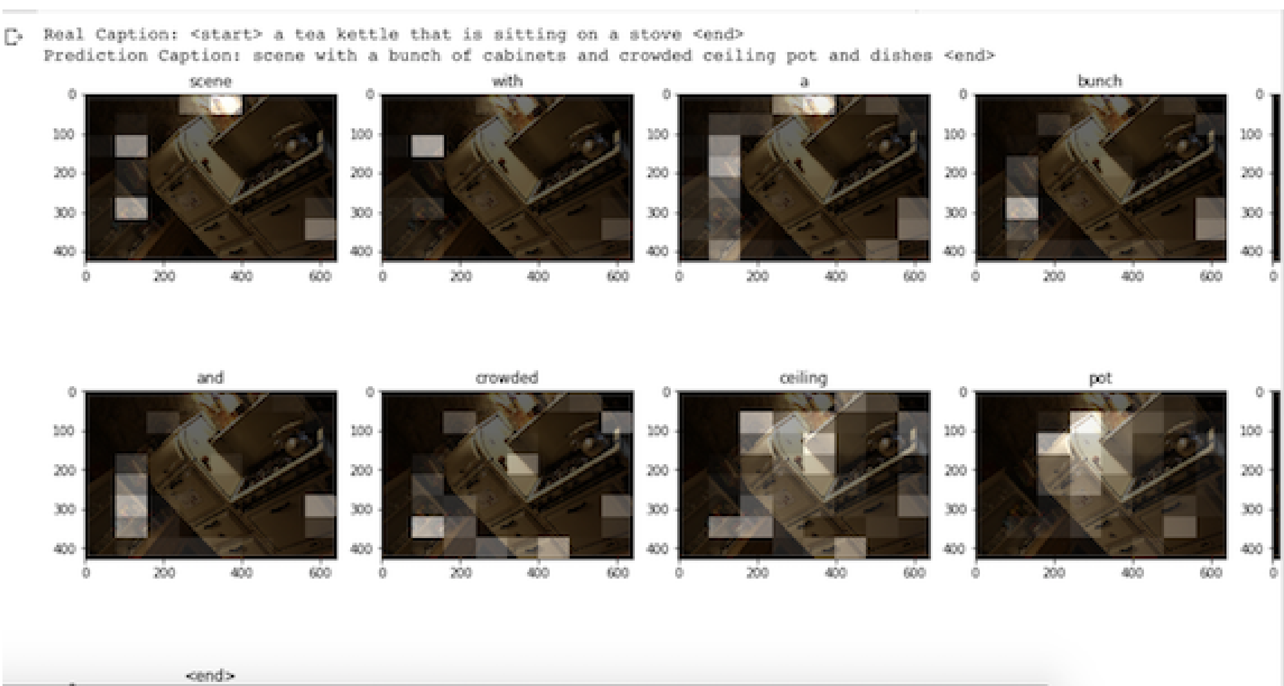



Figure 10: Caption

:

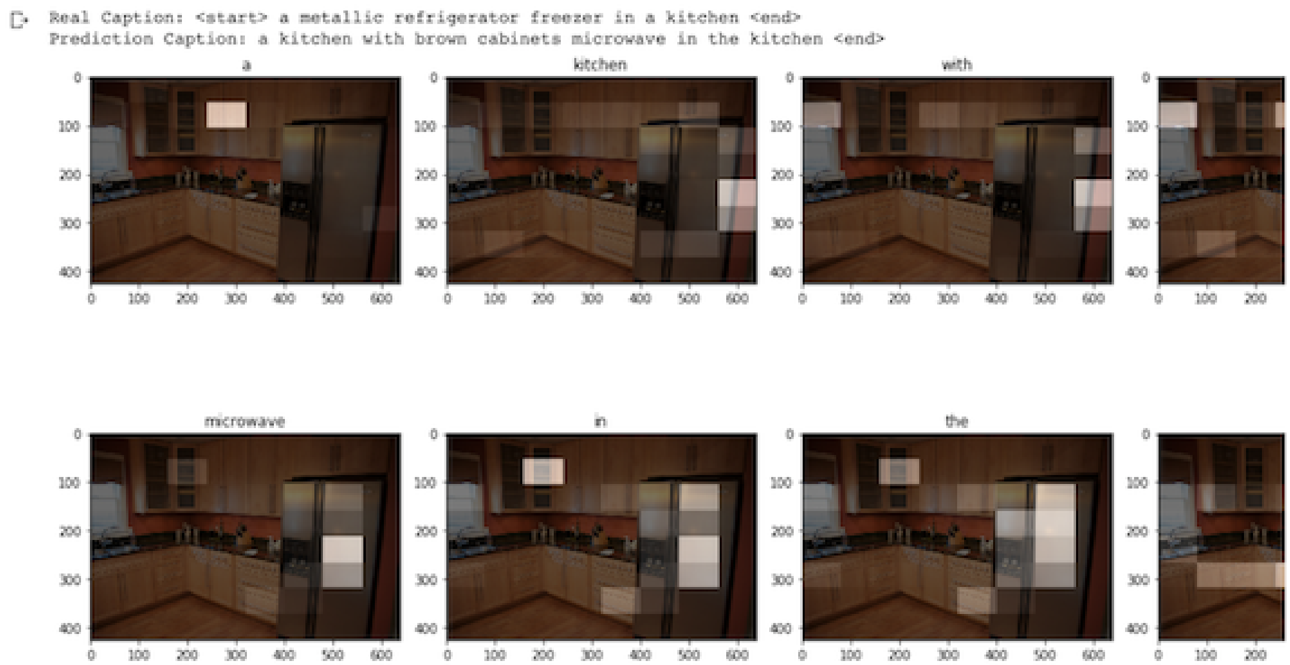



Figure 11: Caption

:


Figure 12: Caption

:

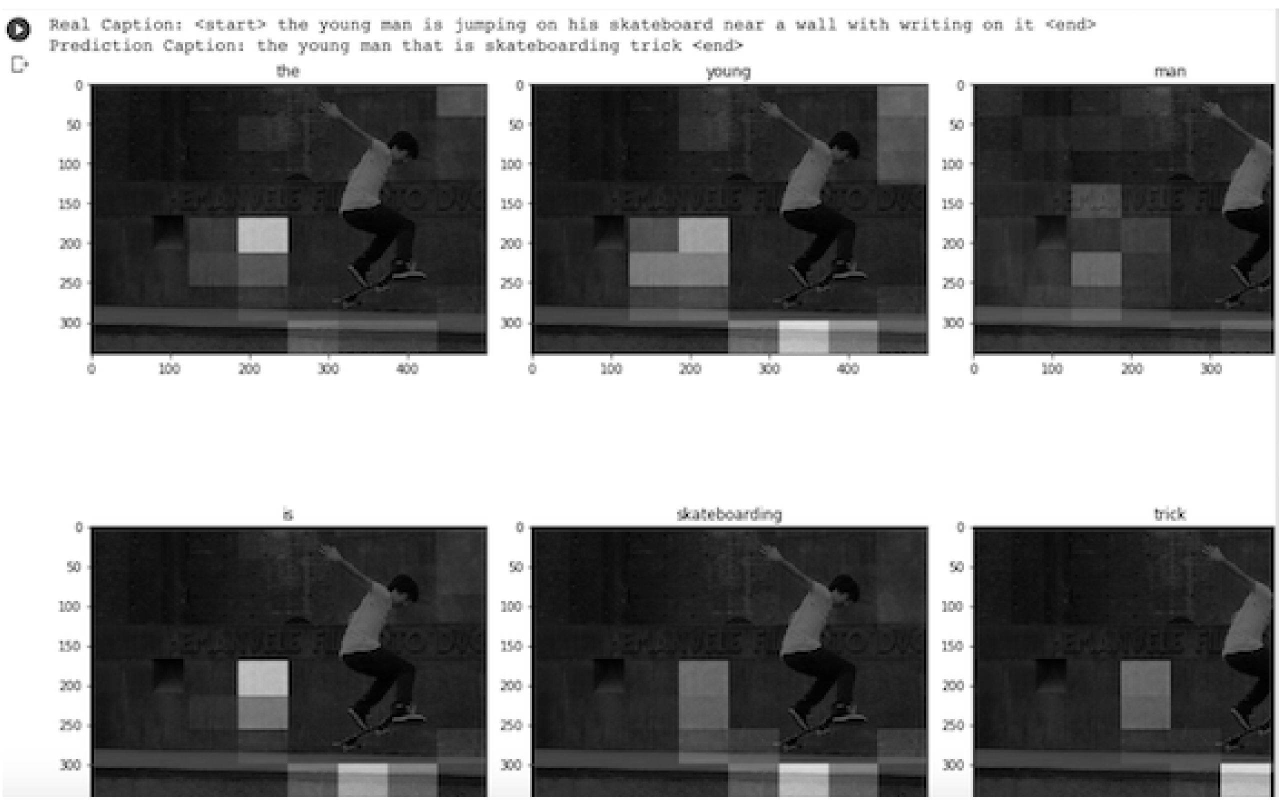



Figure 13: Caption

: