# It was a day

Eupharion, Zaero, Quis

## 1 Dataset and Models

Dataset: MSCOCO
Two models were trained. The first model is the typical one that is introduced by Yunjey on github[1].

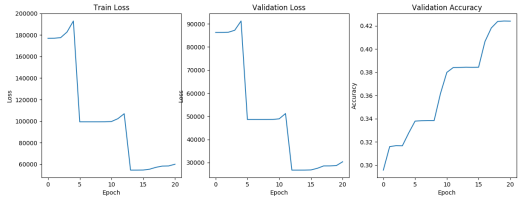
The captioning model utilises a CNN, an encoder, to encode the image, reducing it to its features. The features are then fed into the decoder network consisting of an LSTM, which then outputs the words accordingly.

The models we utilised are built as follows:

1. Vgg 19 - batch norm, pretrained, frozen except last layer.
2. Linear layer, taking the place of the Vgg network's last layer. This linear layer is not frozen.
3. a Batch Normalisation layer after the linear layer.
4. an word embedding layer.
5. LSTM layer.
6. One final linear layer that maps the output to words.

### 1.1 Yunjey's model

Yunjey's model involves concatenating the image and the captions as input during training and validation, attempting guide the outputs in a similar direction to purely training based off images and expecting an output which points to the correct caption. Validation is done in the same manner with the features concatenated with the captions to search for the right outputs. Sampling however, does not take in the caption as an additional parameter, with the decoder only taking in the original input.



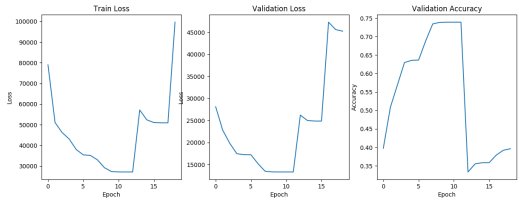

### 1.2 Alternative model

#### 1.2.1 Version 1: Image as Hidden State

This alternative model we propose is different in where the input from the encoder is fed into the network via the hidden state, and the input is simply a start token at the start. During training, the correct words from the $t_{-1}$ time step are fed into the input so as to guide the network into learning the relationship between the words appearances and the input hidden state. This was however a mistake on our part, as it leads to inefficient memory space usage. It is possible that feeding it into the hidden state at the start and expecting that the hidden state outputs the relevant output means that parameters have to be wasted, focused on translating the relevant information into the cell state instead of the approximating the correct word into the output hidden state for the next cycle as opposed to purely basing itself off the existing cell states. In other words, we placed the data into the output area (hidden state)instead of the vector that was directly reserved for memory (cell state). While the neural network was able to adapt, it might have impacted performances negatively.

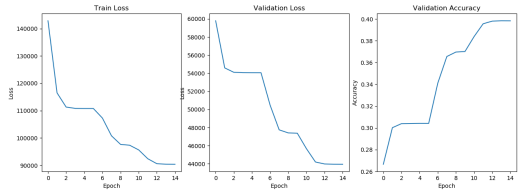
[1] `https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/image_captioning`

Of note is that we did not manage to implement the training phase correctly for the model. The mistake was only noticed during the twelfth epoch, where we noticed that loss was at an all time low and accuracy was peaking at 75% that samples were still turning up with the same top values of the same word. Utilising categorical sampling as an additional measure to see if the model was somehow predicting the right words, we realised to our abject horror we had been feeding in the training and validation inputs incorrectly, with the target word always being the input word for any time step. We also realised we had been feeding the input from the encoder into the hidden state instead of the cell state, whereupon we promptly decided to try version 2 of the model instead, since it seemed to make more sense.




### 1.2.2 Version 2: Image as Cell State

This alternative model we propose is similar to the first version, but instead of using the input of the encoder as the hidden state, we used it as the cell state instead. This corrects the issue that we stated above where now we have the input image stored as a part of memory. During training, the correct words from the $t_{-1}$ time step are fed into the input so as to guide the network into learning the relationship between the words appearances and the input cell state.




## 2 Hyper parameter choices

The following hyper parameters were used for both models initially.

- total epochs = 30
- embedding size = 50
- learning rate = 0.001
- momentum = 0.9
- number of layered LSTMs = 2
- number of hidden dimensions in LSTM = 300
- Loss function = CrossEntropyLoss
- decay rate = 0.9 (Multiplier applied with every epoch end)
- training transforms:

```
from torchvision import transforms
transforms.Compose([
transforms.Resize(300),
transforms.RandomCrop(224),
transforms.RandomHorizontalFlip(),
transforms.ToTensor(),
transforms.Normalize(
(0.485, 0.456, 0.406),
(0.229, 0.224, 0.225))])
```

- testing transforms:

```
transforms.Compose([
transforms.Resize(224),
transforms.CenterCrop(224),
transforms.ToTensor(),
transforms.Normalize(
(0.485, 0.456, 0.406),
(0.229, 0.224, 0.225))])
```

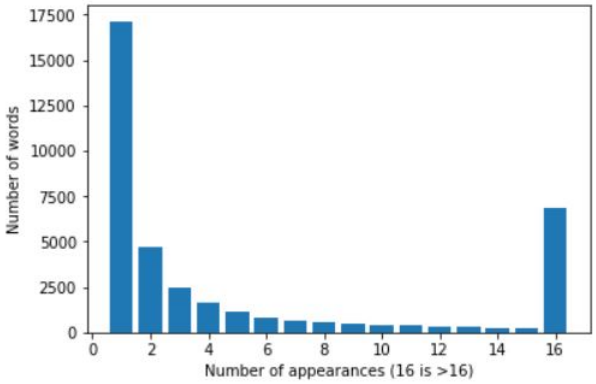
- batchsize = 32/16/8

Batch sizes were altered as we began to experiences plateaus in training, validation and sampling performance. Each time the batch size is altered to become smaller, we force the gradient to update along smaller numbers of features. This slows training somewhat due to the increased number of batch propagations, but due to the larger number of updates, the gradient is forced to learn even more specific features from each batch with every iteration. Because the model is forced to learn even more features with each iteration, the space in which the local minima was originally constructed changes, with the ability to capture even more features to further decrease loss rate is enhanced.

Batch sizes were altered for the model based-off Yunjey's example as follows: 32 till epoch number 8, halved to 16 till epoch 15 before being halved one more time to 8.

Batch sizes for our alternative model stayed at 32 except for the last 2 epochs where it was switched to 16. However, there are several complications in our training process.

# 3 Word frequency analysis




As seen in the bar chart above, where the 16th category is the number of words that have more than 16 appearances in the entire dataset, a majority of words do not appear more than once (17150 words). This is a significant majority of words since there are only a total of 38406 unique words in the dataset. As a result, we opt not to edit the dataset in our training, using those words as part of the training or validation data accordingly. This is because we believed it would result in a large amount of the unknown word token showing up in the training set, heavily impacting performance.

# 4 Evaluation

The evaluation metrics we used is Bilingual Evaluation Understudy Score (BLEU) score which is the state of the art metrics to evaluate a generated sentence with a reference sentence with 1.0 being a perfect match between sentences and 0.0 being no matches.
For us, we evaluated both of our alternative models as we felt that they seem to predict captions better than Yunjey's version. We are using 1-4 ngram corpus BLEU score

## 4.1 Image Used as Hidden State

Using top k to predict the word:
1: 0.08152598353205782
2: 4.171115022461521e-155
3: 3.8125002348036435e-204
4: 9.558226113561296e-232
Using sampling to predict the word:
1: 0.06552827314881415
2: 4.888507480249821e-06
3: 3.749523101562328e-107
4: 2.925910477864098e-159

## 4.2 Image Used as Cell State

Using top k to predict the word:
1: 0.08251634148204581
2: 4.205684304364918e-155
3: 3.83884023597533e-204
4: 9.617969610692441e-232 Using sampling to predict the word:
1: 0.0712434329779113
2: 5.188493402467781e-06
3: 4.051501842982537e-107
4: 3.1904772186924304e-159

# 5 Packages used

1. NLTK - version 3.4
   You will need to also run

   ```
   nltk.download('punkt')
   ```

   In your python file (NOT IN SHELL)
2. Torch - version 1.0.0
3. Python 3.6.8