METU MMI727
SHOW, ATTEND, AND TELL

PROJECT REPORT

Enes Muvahhid ŞAHİN
Department of Electrical and Electronics Engineering
Middle East Technical University
enes.sahin@metu.edu.tr

July 2021

ABSTRACT

In this project a well-known image captioning model called *Show, Attend, and Tell* [1] is investigated. Different improvement strategies are applied on the baseline model. 5 modified architectures are implemented. All 6 models (baseline + 5 modified models) are trained on COCO [2] dataset from scratch. Trained models are compared both quantitatively and qualitatively. Comparison results are provided. Experimental results are discussed and some conclusions are made.

1 Problem Definition

Computer Vision (CV) and Natural Language Processing (NLP) are two of the pioneer branches of deep learning. These two topics have gain significant popularity in the recent years. Moreover, there have been research topics that aim to combine CV and NLP. Image Captioning and Visual Question Answering are two of such topics. In this project, I have worked on Image Captioning topic. Image Captioning models aim to generate natural-sounding textual descriptions of images. They take an image as an input and output a text describing the given image. An example image and possible caption for this image is given in Figure 1.

Figure 1: A dog and cat sitting on the ground.

There are many practical applications of image captioning. One of the most beneficial usage is guiding visually impaired people by describing the scene around them. Another use case is making robots navigate by giving them textual commands. Robots can interpret their surroundings using a camera, describe the scene the camera sees, and apply commands given to them by matching description and textual command. Image captioning can also be used in image searching. If each image on the internet have captions, then searching an image would be as simple as searching the description for that image.

Since the aim in image captioning is generating natural sounding descriptions for images, an image captioning model needs to relate image domain to text domain. It needs to fully understand contents of an image and be able to generate a grammatically correct and natural sounding description which summarizes what is in the image.

Convolutional Neural Networks (CNN) have been widely used in image understanding/recognition tasks. They help encoding an image. Encoding can be seen as extracting discriminative and representative features of an image so that it becomes easier for machines to understand, recognize, classify, match that image. Hence, most of the image captioning models utilize different CNN architectures as the first stage in order to encode the given image.

Recurrent Neural Networks (RNN) are very powerful networks that help representing sequential, time-series data. Their inherent recurrent structure helps understanding and generating sequential data by representing relationships between elements of that data at different time instants. Hence, RNNs are heavily used in Machine Translation, Language Modelling and Language Generation tasks.

Show, Attend, and Tell [1] is a well-known image captioning model that utilizes a CNN as an image encoder and an RNN as a sentence-generating decoder. Furthermore, it makes use of attention mechanism which will be discussed later in more detail.

While training image captioning models, either one-hot representation of GT captions can be used or an embedding layer can be used to transform one-hot vectors with dimension equal to vocabulary size into embedded input vectors. The predictions of decoder blocks generally have dimension equal to vocabulary size. By applying softmax to the predicted output vector, one can determine the predicted word of the model from the vocabulary.

Organization of this report is as follows:

In Section 2, literature review for image captioning is provided. In Section 3, the idea behind Show, Attend, and Tell is discussed thoroughly. In Section 4, details of the dataset used in experiments are provided. In Section 5, baseline model and the models in which I implemented my improvement strategies are discussed. In Section 6 implementation details are provided. In Section 7, both quantitative and qualitative results are provided. 6 models are compared in terms of accuracy, speed, and model size. In Section 8, work conducted in this project is summarized. Limitations of the models are stated, some future works are discussed.

2 Literature Review

One of the first deep learning based image captioning model was Show and Tell [3]. In Show and Tell, encoder-decoder approach from Machine Translation [4] is brought into image captioning. In [4], an encoder RNN encodes sentence to be translated and creates a context vector. Decoder RNN decodes this encoded vector and generates translated sentence. In Show and Tell, instead of an RNN, a CNN is used as encoder to encode input image and create the context vector. Decoder part is an LSTM [5] which takes encoded context vector and <*start*> token as inputs and starts generating caption for the input image. Generation stops when RNN predicts <*end*> token or maximum sentence length is reached. Overview of Show and Tell is given in Figure 2.

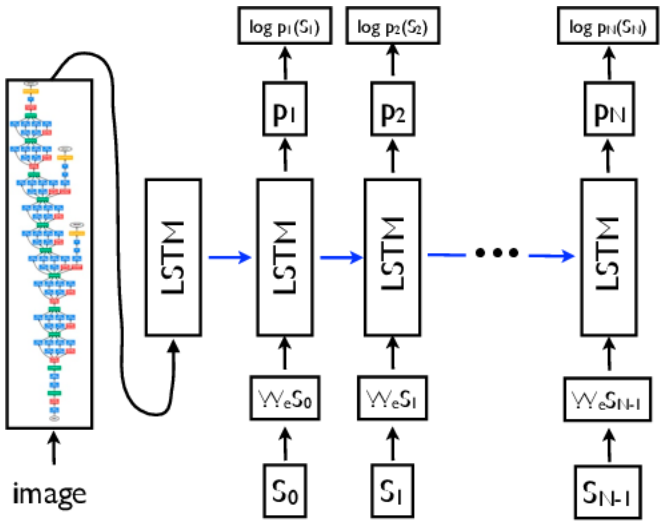


Figure 2: Overview of Show and Tell [3]

Another popular image captioning model was proposed in [6]. In this method, they approached image captioning as an alignment problem between visual and textual data. To learn this alignment, a model is trained to map an image and the corresponding caption to a common h-dimensional multi-modal space. They measure the similarity between mapped image and caption vectors. They try to maximize this similarity during training. Example caption generation is illustrated in Figure 3.

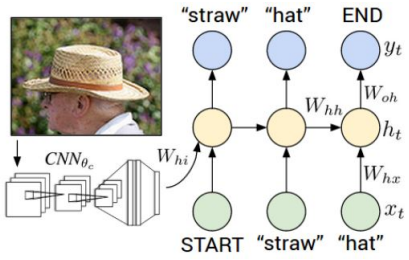


Figure 3: Overview of Deep Visual Semantic Alignments [6]

Show, Attend, and Tell [1] achieved significant improvement after these two models by introducing attention mechanism into image captioning. With attention mechanism, decoder RNN can focus on different parts of the input image while generating different words. Detailed explanation of Show, Attend, and Tell is discussed in the next section.

There have been many improvements since Show, Attend, and Tell. For example in [7], they utilized bottom-up attention as well. In bottom-up attention, a deep learning based object detector is utilized to detect objects in the image. Then, instead of the whole image, decoder RNN attends to only detected objects while generating captions.

Currently, state-of-the art image captioning models are based on transformer networks [8]. Out of these works, [9] and [10] are the most popular ones which aim to align image and text features using self-attention and self-supervised learning mechanisms.

In the next section, Show, Attend, and Tell method is explained in detail.

3 Show, Attend, and Tell

Inspired from [11], Show, Attend, and Tell [1] introduces attention mechanism into image captioning. In [11], attention mechanism improved machine translation performance for long sentences. Instead of using fixed context vector for a given sentence as in [4], they utilized a varying context vector for each different output word. While generating an output translated word, context vector contains weighted sum of states of the encoder RNN where weights represent the relation of the current decoder state to the corresponding input word.

Show, Attend, and Tell [1] uses an architecture similar to [3] where a CNN-based block encodes the image and an RNN-based block, LSTM [5], decodes the encoded image and generates the caption. Different from [3] and similar to [11], Show, Attend, and Tell [1] utilizes an attention mechanism to allow decoder block to selectively focus on different parts of the image while generating output words. Illustration of the attended regions while generating different words are given in Figure 4.

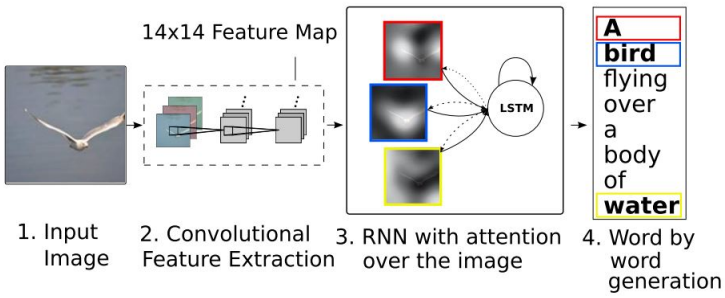


Figure 4: Show, Attend, and Tell [1] attends to different regions while generating different words.

In the encoder block, L different D-dimensional feature vectors corresponding to L different regions of the image are extracted by the CNN:

$$a = \{a_1, ..., a_L\}, a_i \in \mathbb{R}^D \tag{1}$$

At each time instant t of the output decoder RNN, a different context vector, c_t, is calculated using L D-dimensional feature vectors, $\{a_1, ..., a_L\}$:

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L} \exp(e_{tk})} \tag{2}$$

$$e_{ti} = f_{att}(a_i, h_{t-1}) \tag{3}$$

$$c_t = \phi(\{a_i\}, \{\alpha_{ti}\}) \tag{4}$$

where h_{t-1} is the previous hidden state of the decoder block at time instant t, α_{ti} is the weight for i-th feature vector that relates it to the t-th time instant of the decoder RNN, f_{att} is the attention function which calculates the relation of an image feature vector and hidden state of the decoder RNN, ϕ is a function that calculates context vector, c_t, given set of feature vectors, $\{a_i\}$, and corresponding weights, $\{\alpha_{ti}\}$. It is defined as follows:

$$\phi(\{a_i\}, \{\alpha_{ti}\}) = \beta \sum_{i}^{L} \alpha_i a_i \tag{5}$$

where β is called gating scalar and it is defined as follows:

$$\beta_t = \sigma(f_\beta(h_{t-1})) \tag{6}$$

They claim that using gating scalar puts more emphasis on the objects in the images.

In [1], they proposed two different attention mechanisms called "hard attention" and "soft attention". In "hard attention", context vector c_t is selected as one of the feature vectors, a_i, by sampling the set of feature vectors $\{a_i\}$ from a Multinoulli distribution specified by the attention weights, $\{\alpha_{ti}\}$. In "soft attention", context vector c_t is calculated as weighted sum of the feature vectors:

$$c_t = \sum_{i=1}^{L} \alpha_{ti} a_i \tag{7}$$

Visualizations for "hard attention" and "soft attention" are given in Figure 5 and Figure 6, respectively. White regions indicate the attended image regions while generating the word at the top left corner of corresponding patch.

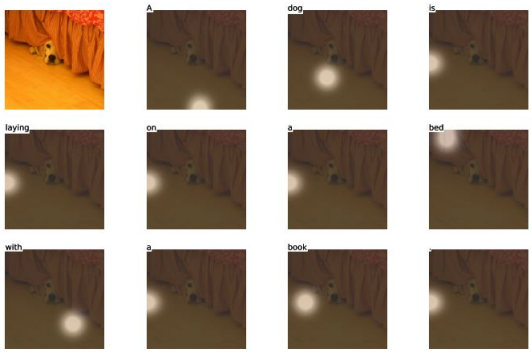


Figure 5: Attention map for "hard attention"

Figure 6: Attention map for “soft attention”

In this project, I have examined the model with "soft attention". The loss function used in this model is straight-forward. Cross entropy loss between predicted output vector of RNN and ground truth word is calculated and averaged over all time instants.
For the "soft attention" model, authors have also introduced another term to the loss function called *doubly stochastic attention*. This term forces $\sum_t \alpha_{ti} \approx 1$. This means that, over the course of decoding (different time instants, t), model is forced to attend all parts (different positions, i) of the encoded feature map. They claim that *doubly stochastic attention* helps generating rich and descriptive captions.

Overall loss function is defined as follows:

$$L_d = -\log(P(y \mid x)) + \lambda \sum_{i}^{L} (1 - \sum_{t}^{C} \alpha_{ti})^2 \tag{8}$$

where the first term is modelled with cross entropy loss, second term stands for *doubly stochastic attention*, and λ is regularization parameter.

In the original paper, they used VGGNet [12] as the encoder CNN. They utilized the output of forth convolutional layer which results in feature map of size $14 \times 14 \times 512$ for input image of size 224×224. This means that $L = 196$ and $D = 512$. Official implementation of the paper is available at [13]. It is based on Theano library.

I used the implementation available in [14] as baseline and implemented my improvement strategies. In [14], they used ResNet101 [15] backbone. The architecture of ResNet is summarized in Figure 7.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 7: ResNet architecture

They used the *conv5_x* output of ResNet101 which results in feature map of size $8 \times 8 \times 2048$ for input image of size 256×256. After that, adaptive average pooling [16] is performed to make feature map resolution $14 \times 14 \times 2048$.

Encoder and decoder blocks for the default implementation in [14] is illustrated in Figure 8 and Figure 9, respectively.

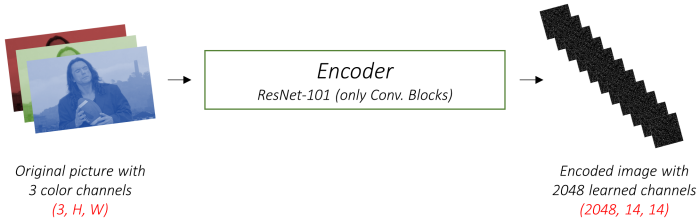


Figure 8: Illustration of encoder, Source [14]

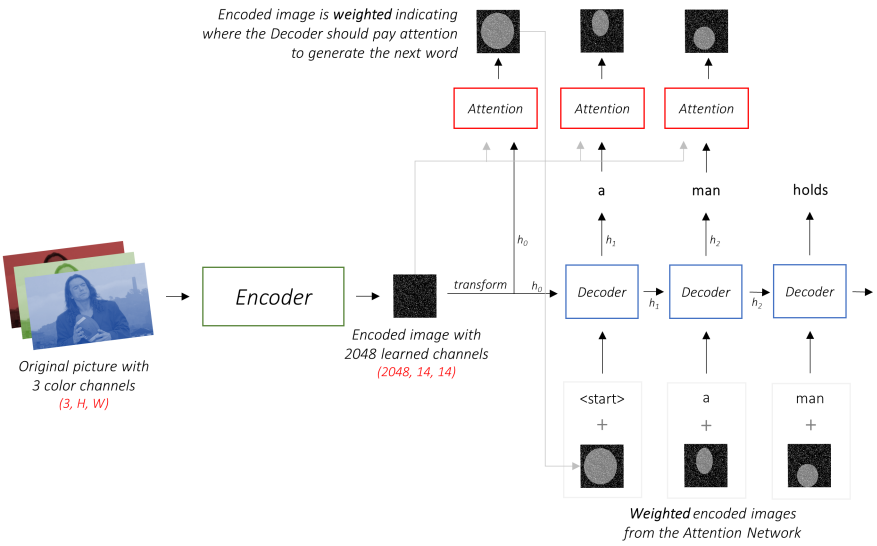


Figure 9: Illustration of decoder, Source [14]

4 Dataset

During my experiments, I used Microsoft COCO (MSCOCO) dataset [2]. This dataset contains more than 120000 images each having 5 ground truth captions. Karpathy's splits [17] are used to determine train, test, and validation splits. Training set contains around 80000 images. Validation and test sets are split equally. All papers on image captioning utilize COCO dataset since it is one of the largest and most popular image captioning datasets. Papers mentioned in Section 2 are also trained and evaluated on COCO dataset.

At first, my aim was experimenting on Flickr8k dataset [18]. However, after a few training attempts, I observed that models were not able to achieve good BLEU [19] scores since dataset is too small. Hence, I decided to go with COCO.

5 Method

In this project, I have tried to improve default model provided in [14] by modifying either encoder, attention, or decoder submodules. In total, one default model and 5 modified models are experimented.

In all models, ResNet101 [15] model pretrained on ImageNet [20] is used. In some models, final layers of ResNet101 is modified. All images are resized to 256×256 and they are normalized by the mean and standard deviation of ImageNet dataset.

Maximum caption length of 50 is used. Captions longer than this length is ignored. Shorter captions are padded with special <*pad*> token.

In the next subsections, I will explain details of each model.

5.1 Model 1 - Default (Baseline)

This is the default model provided in [14].

In the encoder block, it uses *conv5_x* output of ResNet101 which results in feature map of size $8 \times 8 \times 2048$ for input image of size 256×256. After that, adaptive average pooling [16] is performed to obtain feature map, a, with resolution equal to $14 \times 14 \times 2048$. Adaptive average pooling allows passing the network variable-sized input images.

In the decoder block, it uses LSTM with hidden dimension size equal to 512. Input captions are passed through an embedding layer which maps one-hot encoded words, o_t, to embedded vectors, e_t, of size 512. This embedding layer is trained together with the whole model, in and end-to-end fashion.

$$e_t = Embedding(o_t), \quad o_t \in \mathbb{R}^V, e_t \in \mathbb{R}^{512} \tag{9}$$

where V is the size of vocabulary.

To predict words, LSTM's state, h_t, is fed into fully-connected layer to obtain output vector of size V:

$$y_t = FC(h_t) \quad h_t \in \mathbb{R}^{512}, y_t \in \mathbb{R}^V \tag{10}$$

After applying $Softmax$ to y_t, distribution of model's prediction for the current word is obtained.

In the attention block, both encoder output and previous decoder hidden state are mapped to a common attention space with dimension equal to 512. Encoder output of size $14 \times 14 \times 2048$ is flattened and a tensor, a, of size 196×2048 is obtained. This tensor is mapped to the common attention space with a fully-connected layer. A tensor, m, with size 196×512 is obtained. Similarly, previous hidden state, h_{t-1}, of the decoder which is of size 512 is passed through a fully-connected layer and mapped to common attention space of size 512. A tensor with size 512 is obtained. This tensor is expanded to the same size of m. A tensor, n, with size 196×512 is obtained. Two tensors coming from encoder and decoder blocks are summed. ReLU activation function is applied to summation result and it is passed through a fully-connected layer to obtain attention tensor, k, of size 196. After passing k through softmax, attention weights for all 196 positions in 14×14 feature map is obtained.

$$m = FC(a), \quad m \in \mathbb{R}^{196 \times 512} \tag{11}$$

$$n_t = Expand(FC(h_{t-1})), \quad n_t \in \mathbb{R}^{196 \times 512} \tag{12}$$

$$k_t = FC(ReLU(m + n_t)), \quad k_t \in \mathbb{R}^{196} \tag{13}$$

$$\alpha_t = Softmax(k_t), \quad \alpha_t \in \mathbb{R}^{196} \tag{14}$$

Obtained weights, α_{ti}, are applied to the feature map $a = \{a_1, ..., a_{196}\}, a_i \in \mathbb{R}^{2048}$ and context vector $c_t \in \mathbb{R}^{512}$ is obtained:

$$c_t = \beta_t \sum_{i=1}^{196} \alpha_{ti} a_i \tag{15}$$

where β_t is gating scalar calculated according to (6).

Context vector c_t and embedded input word e_t is concatenated and fed into the LSTM as the input.

5.2 Model 2 - Concatenated Attention

This model differs from Model 1 in its attention block. In Model 1, feature map and previous hidden state of LSTM is mapped to a common space and tensors m and n_t are obtained. Then these two tensors are summed up and passed through ReLU and fully-connected layer according to (13). The summation operation seemed counterintuitive to me. I thought it would be more representative if we concatenate m and n_t and then pass through ReLU and fully-connected layer. So, in this model, I modified attention block and used ConcatenatedAttention. Rest of the network is same as Model 1. Only difference is in (13). I modified it as follows:

$$k_t = FC(ReLU(Concat(m, n_t))), \quad k_t \in \mathbb{R}^{196} \tag{16}$$

where $Concat$ is concatenation operation along attention dimension.

5.3 Model 3 - GRU

This model differs from Model 1 in its decoder block. In Model 1, LSTM is used as the decoder. In this model, I used Gated Recurrent Unit (GRU) [21] as the decoder RNN. Hidden dimension size is same as Model 1, 512. In this model, my aim was comparing representation power and speed of LSTM and GRU. So, GRU is substituted with LSTM. The rest of the network is the same as Model 1.

5.4 Model 4 - Wider Encoder

This model differs from Model 1 in its encoder block. As described in Model 1, network is passed through ResNet101, and feature map of size $8 \times 8 \times 2048$ is obtained for input image of size 256×256. Since original paper used feature map of size 14×14, in default implementation [14], they used adaptive average pooling to increase size of feature map from $8 \times 8 \times 2048$ to $14 \times 14 \times 2048$. I thought increasing size of feature map from 8×8 to 14×14 would result in degraded performance since we introduce non-existing information to the final feature map by performing interpolation. So, instead of using *conv5_x* output of ResNet101 [15], I used *conv4_x* output. For an image of size 256×256, *conv4_x* produces output feature map of size $16 \times 16 \times 1024$. Number of channels in Model 1 was 2048. In order to achieve same channel size in this model, I implemented a bottleneck layer after *conv4_x* output. This bottleneck layer is inspired from ResNet.

This bottleneck layer is illustrated below:

```
nn.Sequential(
    nn.Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False),
    nn.BatchNorm2d(512),
    nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), \
        padding=(1, 1), bias=False),
    nn.BatchNorm2d(512),
    nn.Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False),
    nn.BatchNorm2d(2048),
    nn.ReLU(inplace=True)
)
```

I first reduce number of channels from 1024 to 512 using 1×1 convolutions. Then, a 3×3 convolution is applied. After that, number of channels is increased from 512 to 2048 using 1×1 convolutions. Output of this layer is of size $16 \times 16 \times 2048$ for an input image of size 256×256. Finally, adaptive average pooling is applied to obtain feature map of size $14 \times 14 \times 2048$ which is the same as Model 1.

One thing to note here is that, the ResNet layers are pretrained on ImageNet but the weights of bottleneck layer I implemented are initialized randomly. Hence, this might cause degraded performance.

5.5 Model 5 - Feature Pyramid Network

This model differs from Model 1 in its encoder block. In this model, I introduced the idea in [22], namely Feature Pyramid Networks (FPN). In CNNs, shallow layers contain high resolution feature maps with strong localization information. Deeper layers have reduced resolution but they contain much more semantic information. Hence, FPN tries to combine *high resolution - low semantic information* feature maps with *low resolution - high semantic information* feature maps. This idea is illustrated in Figure 10.

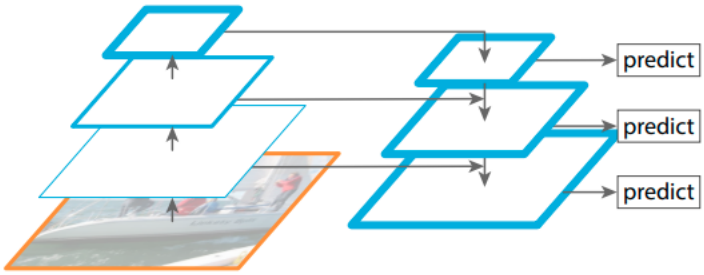


Figure 10: Feature Pyramid Networks

In Figure 10, bottom-up path is general architecture of CNN. Feature maps in this path are named as C_1, C_2, C_3 from bottom to top. Feature maps in top-down path are named as P_3, P_2, P_1 from top to bottom. P_3 is upsampled and added with C_2 to obtain P_2 which is a high-resolution feature map with strong semantic information.

I implemented a 2-layer FPN. In my implementation, *conv4_x* output of ResNet101 is called C_1 and *conv5_x* output is called C_2. Top feature map P_2 is equal to C_2. Then, I upsampled P_2 and added it to C_1 to obtain P_1. P_1 is of size $16 \times 16 \times 2048$ for an input image of size 256×256. Similar to Model 1, adaptive average pooling is applied to obtain encoder output of size $14 \times 14 \times 2048$.

With this model, my aim was using a wider feature map just like Model 4. I also wanted to increase semantic information in this wider feature map which does not exists in Model 4. So, FPN was suitable for my aim. Furthermore, in FPN implementation, I used original ResNet layers with pretrained weights on ImageNet.

5.6 Model 6 - 2-Layer Decoder

This model differs from Model 2 in its decoder block. In this model, I used 2-Layer LSTM instead of original 1-Layer LSTM. Both LSTMs have hidden state of size 512.

In this model, similar to Model 1, 1st LSTM layer is fed with concatenated embedded input word and attention-weighted context vector. Output of 1st LSTM, h_t^1, is fed into the 2nd LSTM as input. 2nd LSTM's output, h_t^2, is used to predict words just like in Model 1.

The calculations in attention block are similar to Model 1 except the use of h_{t-1}. In this model, attention is calculated using previous hidden state of 2nd LSTM, h_{t-1}^2.

My aim in implementing this model was to see if the representation power of 2-layer RNN affects quality of generated words or not.

6 Implementation Details

Even though I used implementation available at [14], I was not able to use provided code and pretrained model given in that repositroy. Because the code was written for PyTorch 0.4 but PyTorch 0.4 was not available for CUDA 10.2. Hence, first I modified the necessary parts in code to make it working with PyTorch 1.8.1. Furthermore, due to updates in the PyTorch's built-in functions, I could not load provided pretrained model. My aim was loading pretrained model and finetuning on a different dataset such as Flickr8k [18]. But due to version mismatches, I could not use pretrained model.

Due to these reasons, I trained all 6 models from scratch on COCO dataset with PyTorch 1.8.1 on NVIDIA Tesla P100 GPU with CUDA 10.2 driver. I performed my experiments on the test set of COCO.

In [14], they used 2-stage training where in the first stage, encoder block is not trained. It is trained for few epochs in the second stage. However, in all my trainings, I used end-to-end training approach. I finetuned encoder layers starting from *conv3_x*. I used learning rate of 0.0001 for both encoder and decoder. I used batch size of 64. I trained models for at most 30 epochs. If BLEU-4 score calculated on validation set does not improve for 8 consecutive epochs, learning rate is decayed by multiplying with 0.8. After training, I selected checkpoints with highest BLEU-4 score on validation set and performed comparisons using those checkpoints for all models.

While generating captions, I used beam search. With beam search, instead of selecting the words with highest score at each time step t, we take top-k words at each time instant. These k words are used as previous words at time step $t+1$ and top-k sequences are chosen as candidate partial captions. This goes on like this until <*end*> token is generated. In

the end, sentence with highest score out of top-k sequences is chosen as the predicted caption. k is known as the beam size. This procedure is illustrated in Figure 11.

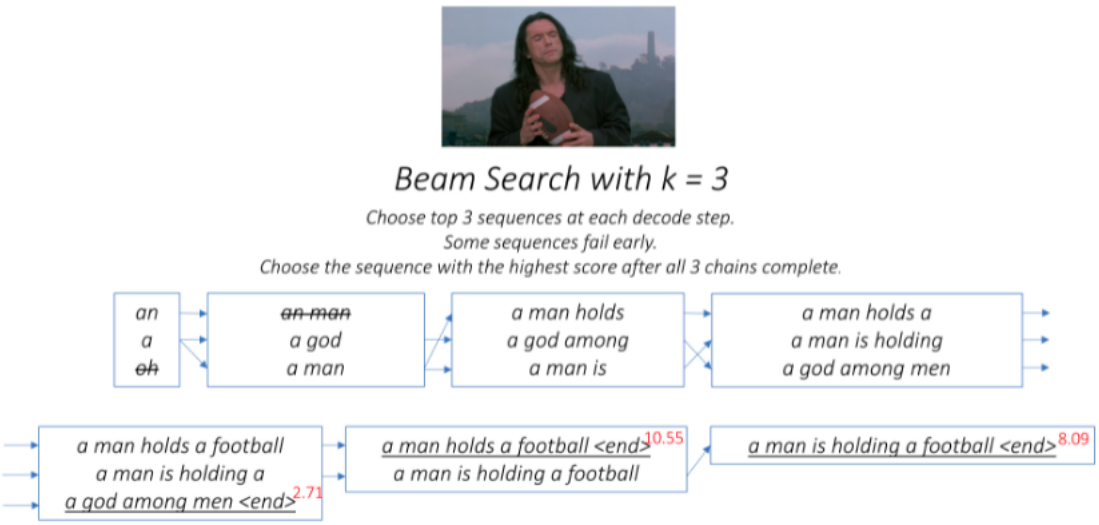


Figure 11: Beam Search, Source [14]

The training logs and graphics for all 6 models are provided in the uploaded zip file and Jupyter Notebook.

7 Results

7.1 Quantitative Results and Discussions

In this section, 6 models are compared quantitatively in terms of accuracy and speed on the test set of COCO.

In Table 1, 6 models which are trained by me and the results provided in the paper [1] and in the reference repository [14] are compared in terms of accuracy using BLEU-4 score with two different beam sizes, 1 and 5. The results are obtained on the test set of COCO dataset.

First of all, all 6 models trained by me and the baseline result is better than paper result. I think this is due to representative power of ResNet compared to VGGNet which was the encoder used in the paper. Furthermore, the default model, Model 1, trained by me surpasses the results reported in the reference repository [14]. I think this is due to training both encoder and decoder in an end-to-end fashion.

When we compare models trained by me, we can see that Model 5 - FPN slightly improves the results compared to the baseline model (Model 1). I think the reason why Model 1 performs worse is due to upsampling of 8×8 feature map to 14×14 using adaptive average pooling. In Model 5 - FPN, high semantic feature map of size 8×8 is added to the high resolution feature map. Hence, this model can utilize high semantic information in a wider feature map. I think the reason why Model 4 performed worse is due to newly introduced bottleneck component in the encoder block. Weights of layers in this component are initialized randomly as opposed to other models whose all layers are initialized with pretrained ImageNet weights.

Performance of Model 3 - GRU is medicore but still not far away from other models. I thought it would be the worst among all models but it seems that GRU can handle this task well.

Performance of Model 6 - 2-Layer Decoder is the worst among all models. This is a little suprising for me. I thought adding a 2-Layer LSTM would help generating more novel and representative captions. But the results say the opposite. I think in this model, the bridges between 2 LSTM layers, encoder output, and attention block are not constructed well. I think there is room for improvement for this model. Altering dimensions of hidden states of 2 LSTMs, modifying attention mechanism so that both layers' hidden states play role in attention weights could help boosting the performance of this model.

Table 1: Accuracy Comparison of Models on COCO Test Set. Red is the best, Blue is the 2nd best, Green is the 3rd best among the models trained by me. *: Results are taken from [1] and [14].

Model Name	BLEU-4 (Beam Size = 1)	BLEU-4 (Beam Size = 5)
Model 1 - Default	0.3066	0.3371
Model 2 - Concatenated Attention	0.3040	0.3337
Model 3 - GRU	0.2976	0.3302
Model 4 - Wider Encoder	0.3001	0.3301
Model 5 - Feature Pyramid Network	**0.3068**	**0.3392**
Model 6 - 2-Layer Decoder	0.2975	0.3175
*Baseline Result [14]	0.3028	0.3329
*Paper Result [1]	0.243 (Beam Size not mentioned)	

Table 2: Execution Time Comparison of Models

Model Name	Execution Time in ms (Beam Size = 1)	Execution Time in ms (Beam Size = 5)
Model 1 - Default	26.873	44.708
Model 2 - Concatenated Attention	27.274	45.351
Model 3 - GRU	27.242	44.721
Model 4 - Wider Encoder	26.074	43.329
Model 5 - Feature Pyramid Network	28.765	46.365
Model 6 - 2-Layer Decoder	28.094	46.642

Model 2 - Concatenated Attention did not improve the performance compared to Model 1. This was again a little surprising for me. I had thought concatenating two vectors in the common space would help calculating attention weights compared to just summing two vectors up. It did not turn out that way but the accuracy gap is not too big either.

In Table 2, 6 models are compared in terms of execution speed. Tests are performed on NVIDIA GTX 1060 GPU with CUDA 11.2 and PyTorch 1.8.0. Tests are performed by captioning 40 images in a loop of 10 iterations and median of 400 execution times (from the beginning of encoder until the end of generating caption) are reported. The execution time differences between models are very small. I think all perform similarly in terms of speed. One notable difference is that, Model 5 - FPN and Model 6 - 2-Layer Decoder are slower compared to the other models since their architectures are more complex. Furthermore, Model 3 - GRU is not as fast as I expected. In this case, it seems using LSTM is more suitable since it gives higher accuracy for similar execution times.

In Table 3, number of parameters for 6 models are provided. We can see that Model 4 - Wider Encoder has much fewer parameters compared to the rest of the models. This might also cause this model to have worse results compared to the others. Model 3 - GRU has 3 million less parameters compared to LSTM-based versions. Model 5 - FPN introduces 2 million parameters to the encoder and Model 6 - 2-Layer Decoder introduces 2 million parameters to the decoder.

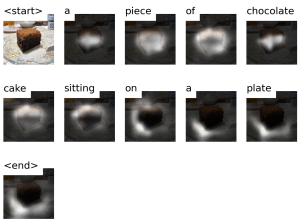
7.2 Qualitative Results and Discussions

In this section, example outputs of 6 models for custom images captured by me are provided.

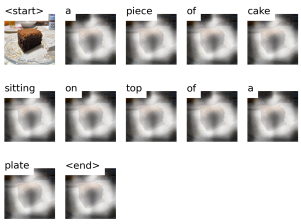
Table 3: Comparison of Models in Terms of Number of Parameters

Model Name	# of Params-Encoder	# of Params-Decoder	# of Params-Total
Model 1 - Default	42,500,160	20,483,859	62,984,019
Model 2 - Concatenated Attention	42,500,160	20,484,371	62,984,531
Model 3 - GRU	42,500,160	17,860,883	60,361,043
Model 4 - Wider Encoder	31,473,728	20,483,859	51,957,587
Model 5 - Feature Pyramid Network	44,599,360	20,483,859	65,083,219
Model 6 - 2-Layer Decoder	42,500,160	22,585,107	65,085,267

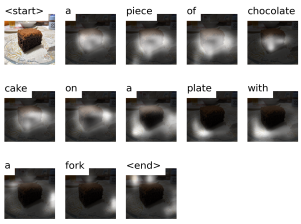
Figure 12: Image - 1



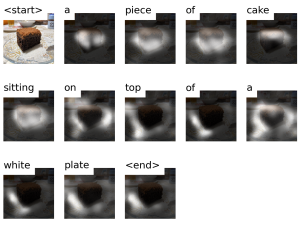
(a) Model 1 - Default



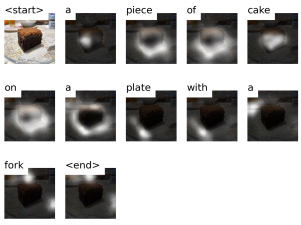
(b) Model 2 - Concatenated Attention



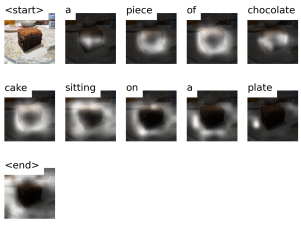
(c) Model 3 - GRU



(d) Model 4 - Wide Encoder



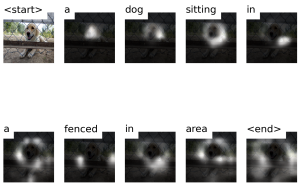
(e) Model 5 - FPN



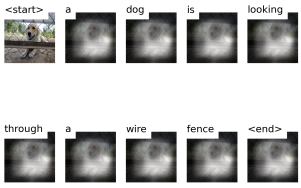
(f) Model 6 - 2-Layer Decoder

Figure 13: Outputs of 6 models for Image - 1

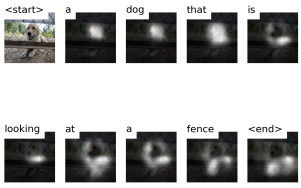
Figure 14: Image - 2



(a) Model 1 - Default

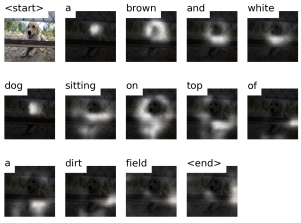


(b) Model 2 - Concatenated Attention

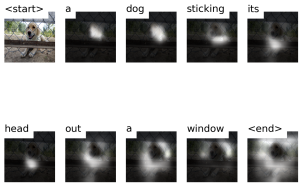


(c) Model 3 - GRU

(d) Model 4 - Wide Encoder



(e) Model 5 - FPN



(f) Model 6 - 2-Layer Decoder

Figure 15: Outputs of 6 models for Image - 2

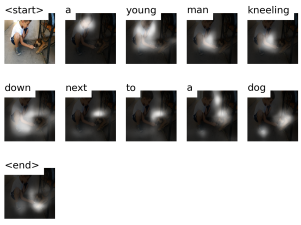
Figure 16: Image - 3

(a) Model 1 - Default

(b) Model 2 - Concatenated Attention

(c) Model 3 - GRU

(d) Model 4 - Wide Encoder



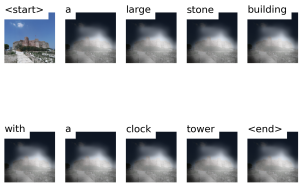
(e) Model 5 - FPN

(f) Model 6 - 2-Layer Decoder

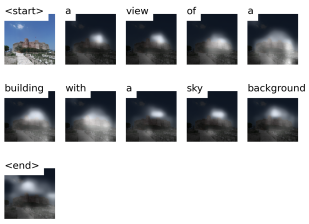
Figure 17: Outputs of 6 models for Image - 3

Figure 18: Image - 4

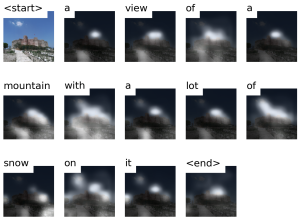
(a) Model 1 - Default



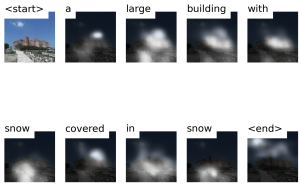
(b) Model 2 - Concatenated Attention



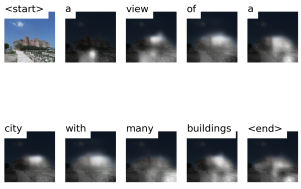
(c) Model 3 - GRU



(d) Model 4 - Wide Encoder



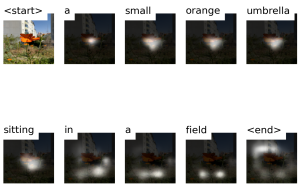
(e) Model 5 - FPN



(f) Model 6 - 2-Layer Decoder

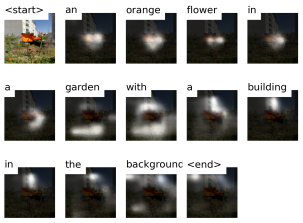
Figure 19: Outputs of 6 models for Image - 4

Figure 20: Image - 5

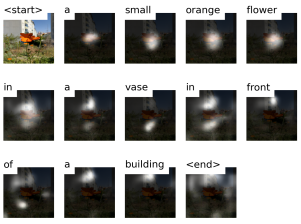


(a) Model 1 - Default

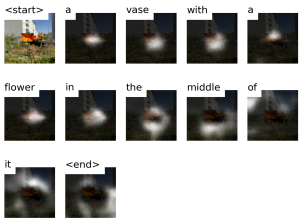
(b) Model 2 - Concatenated Attention



(c) Model 3 - GRU



(d) Model 4 - Wide Encoder



(e) Model 5 - FPN

(f) Model 6 - 2-Layer Decoder

Figure 21: Outputs of 6 models for Image - 5

The results of 6 models differ from image to image. First of all, there is something wrong in the visualization of Model 2 - Concatenated Attention. It shows same attention map for all words. However, when I debugged, I could not detect the problem, output of the network was correct. There might be a bug in visualization code.

In Figure 13, we can see that all models generate fairly accurate captions. Only two of them were able to mention about the fork at the bottom right of the image, Model 3 and Model 5. Three of the models were able to recognize the cake as *chocolate cake*, Model 1, Model 3, Model 6.

In Figure 15, Model 1's output is grammatically wrong. Model 6 confused the fence with a window. Model 5 is the only model that mentions about the properties of dog, though it is not fully correct. However, Model 5 did not mention about the fence.

In Figure 17, Model 2 and Model 3 mentioned about skateboard which does not exist in the image. Model 6 mentioned about the 'kid' as 'man'. Model 5 used 'kneel down' phrase correctly. Its caption is also good.

In Figure 19, most of the models failed. Model 4 and Model 5 hallucinated about 'snow' which does not exist in the image. Model 1 hallucinates about two buildings. Model 2 hallucinates about clock tower. Model 6 hallucinates many buildings. Only Model 3's caption seems accurate. Also, it is the only one that mentions about the sky.

In Figure 21, Model 1 is completely wrong since it confuses flower with umbrella. Model 2's output is fairly good. Model 3's output is very good since it mentions about orange flower, garden, and the building in the background. Model 4 and Model 5 hallucinated about the vase. Model 6's output is also fairly good since it tried to mention more than one flower.

These results show that different models perform differently depending on the image. There is not a clear winner among these models qualitatively. However, I am impressed by the captions generated by Model 3 - GRU since I expected it to be less representative than LSTM.

8 Conclusion

In this project I have examined and experimented with a well-known image captioning model, Show, Attend, and Tell [1]. I tried to improve baseline model by implementing 5 different architectures in which either encoder, decoder, or attention block is modified. I have trained all 6 models on COCO dataset from scratch. I have benchmarked and compared these models in terms of accuracy, speed, and size. Finally, I have provided some qualitative results and comments about the outputs of models on custom images captured by me.

Models' performances change from image to image. I liked the results of Model 3 - GRU and Model 5 - FPN. However, even these models may fail on some basic scenes. I think the limitation of these models come from the fact that alignment between visual and textual data is not constructed very well. About this issue, I really liked the idea proposed in [9] which utilizes names of the objects in the image as an alignment tool. This kind of approach may also be implemented in basic models (which do not utilize transformers) like Show, Attend, and Tell. This might increase the accuracy.

References

[1] Kelvin Xu et al. "Show, attend and tell: Neural image caption generation with visual attention". In: *32nd International Conference on Machine Learning, ICML 2015* 3 (2015), pp. 2048–2057.

[2] Xinlei Chen et al. "Microsoft COCO Captions: Data Collection and Evaluation Server". In: (2015), pp. 1–7. URL: http://arxiv.org/abs/1504.00325.

[3] Oriol Vinyals et al. "Show and tell: A neural image caption generator". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 07-12-June (2015), pp. 3156–3164. ISSN: 10636919. DOI: 10.1109/CVPR.2015.7298935.

[4] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference* (2014), pp. 1724–1734. DOI: 10.3115/v1/d14-1179.

[5] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735.

[6] Andrej Karpathy and Li Fei-Fei. "Deep Visual-Semantic Alignments for Generating Image Descriptions". In: *Proceedings of Conference on Computer Vision and Pattern Recognition* (2015). URL: http://arxiv.org/abs/1412.2306.

[7] Peter Anderson et al. "Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2018), pp. 6077–6086. ISSN: 10636919. DOI: 10.1109/CVPR.2018.00636.

[8] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. Vol. 2017-Decem. Neural information processing systems foundation, June 2017, pp. 5999–6009. URL: https://arxiv.org/abs/1706.03762v5.

[9] Xiujun Li et al. "Oscar: Object-Semantics Aligned Pre-training for Vision-Language Tasks". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12375 LNCS (2020), pp. 121–137. ISSN: 16113349. DOI: 10.1007/978-3-030-58577-8{_}8.

[10] Pengchuan Zhang et al. "VinVL: Making Visual Representations Matter in Vision-Language Models". In: *CoRR* abs/2101.00529 (2021). arXiv: 2101.00529. URL: https://arxiv.org/abs/2101.00529.

[11] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), pp. 1–15.

[12] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations*. 2015.

[13] https://github.com/kelvinxu/arctic-captions.

[14] https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning.

[15] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[16] https://pytorch.org/docs/stable/generated/torch.nn.AdaptiveAvgPool2d.html.

[17] https://cs.stanford.edu/people/karpathy/deepimagesent/caption_datasets.zip.

[18] https://www.kaggle.com/adityajn105/flickr8k.

[19] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040.

[20] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[21] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[22] Tsung-Yi Lin et al. "Feature Pyramid Networks for Object Detection". In: *CoRR* abs/1612.03144 (2016). arXiv: 1612.03144. URL: http://arxiv.org/abs/1612.03144.