**AI825 Term 2 2021-22**

**Visual Recognition Part 2**

**Mini Project**

**Image Captioning CNN-LSTM**

**Group Number 6**

***Adrij Sharma – IMT2019004  
Shrey Tripathi – IMT2019084***

***Problem 1:***

***CNN-LSTM system for image captioning on Flicker8K dataset***

**About Flick8K dataset**

We used the Flickr dataset for training, validation, and testing. The collection contains 8,000 photos, each of which is accompanied by five different captions that provide detailed descriptions of the important entities and events. The photos were hand-picked from six separate Flickr groups and don't feature any well-known persons or places, but they do portray a range of scenarios and circumstances.

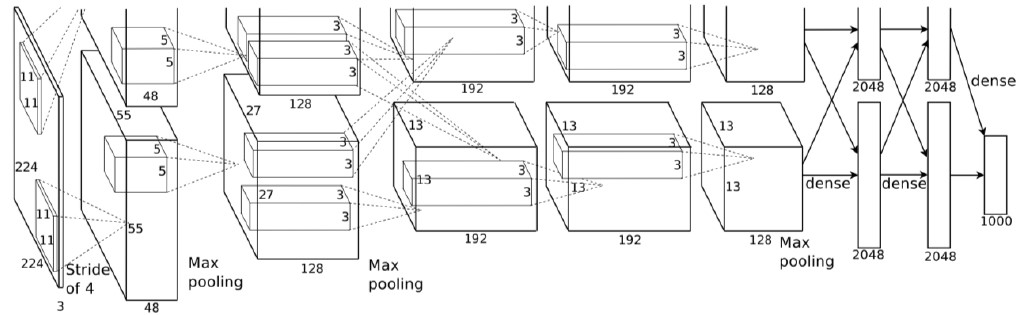
**Definition of CNN**

CNN is a deep learning technique that is widely applied to image analysis. Unlike older methods, which require hard engineering of image features, CNN allows us to provide input as is and let the picture learn the features/characteristics on its own. Multiple layers of artificial neurons make up CNNs. The output of one CNN layer is sent to the next. The first layer of a CNN usually recognises basic features like horizontal and vertical edges. It starts to recognise higher-level elements like objects and faces as you advance further into the layer. Convolutional, pooling, and fully-connected CNN layers are available.

**Image Features Extraction**

We employ pre-trained deep CNNs to extract critical characteristics from our photos because they are (typically) trained on very big datasets like ImageNet and can create great features. We experiment with 2 different CNNs.

**Alexnet –** A CNN with five convolutional layers, three max-pooling layers, two normalisation layers, two fully connected layers, and one softmax layer. Convolutional filters and a nonlinear activation function ReLU make up each convolutional layer. Max pooling is done using the pooling layers. Due to the presence of fully connected layers, the input size is fixed. AlexNet is broken down into three sections: features, avgpool, and classifier.



Diagram

Description automatically generated**VGGNet –** This method arose from a desire to lower the amount of parameters in the CONV layers while also reducing training time. VGGNet has several variations (VGG16, VGG19, and so on) that change solely in the total number of layers in the network. There are 138 million parameters in VGG16. The crucial thing to remember here is that all conv kernels are 3x3 in size, while maxpool kernels are 2x2 with a stride of two. The first thing we do is create rich features from images, which should be representative of its contents. It is best to use pre-trained deep CNNs for this task, since they are (usually) trained on very large datasets like ImageNet and can generate excellent features. We use VGG-16 by PyTorch, which is VGGNet with 16 layers. The model's general architecture looks like:

The final layer in this model is a fully connected layer mapping to 1000 units. This is kept for classification purposes (the ImageNet dataset has 1000 classes) and is not necessary for us. We'll retain only up to the penultimate layer of the model, which will give us a feature vector of 4096 dimensions for every image.

Diagram

Description automatically generated**Model Architecture**

The inputs to the model are the preprocessed image *(224, 244, 3)* and the integer tokens of the captions. The image's features are extracted and reduced to *256* dimensions using a Linear layer with ReLU activation. *256* is an arbitrary choice. We use **Categorical Crossentropy** **loss** **(Log softmax** + **Nonlinear logloss** in PyTorch) for updating the parameters. Also, we use an Adam optimizer with constant learning rate.

Providing the caption tokens, however, is not that straightforward. Imagine how YOU, as a human, would go about this.

Providing the caption tokens work in the following steps:

1. Look at the image, extract some information and keep that in mind.
2. Generate the first word.
3. Look at the first word, remember what you saw in the image, and generate a second word so that the sentence so far is grammatically correct.
4. Look at the first and second words and generate a third word. Refer to the image again if needed.
5. Continue this process until all of the words have been generated.

We follow this pattern for the network. So, the caption inputs look like this.

A picture containing text, clipart, screenshot

Description automatically generated

**Preprocessing**

**Image feature generation**

All images that go through the network must have the same shape. To do this, we define a transformation pipeline using transforms. This pipeline performs the following operations.

1. Resizes the image to (256, 256, 3).
2. Crops the image to only retain its center, which is of size (224, 224, 3). This assumes that the edges of the image do not contain any valuable information, which is reasonable most of the times.
3. Converts this PIL image into a torch tensor.
4. Normalizes each of the three channels of the image (RGB) with means (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225) respectively. These numbers were found to give optimal performance in ILSVRC (a very large computer vision competition), so we use it here as well.

**Caption processing**

Every image is associated with 5 captions. We organize this information in an easier-to-work-with manner. So, we create a lookup table where the keys are the filename and a list of 5 captions associated with that file as the corresponding values. To separate the keys from the sentences, we split each sentence on the tab delimiter *"\t"*. Plus, we also remove the number tags (#0, #1, etc.) from the filenames.

**Tokenization**

We generate a vocabulary of all the words we have (bag of words) and assign an integer to each word. This process is called **tokenization**.

**Preprocessing**

Some elements of caption sentences do not carry any meaning. Examples include uppercase alphabets, punctuation and numbers (at least for our purposes). We will clean each sentence of these entities. Then, we add two special tokens at the start and end of each sentence, to indicate the beginning and ending of the sentence (*startseq* and *endseq* respectively). These help the model know when to start and stop predicting. Once that is done, we tokenize all the lines.

**Padding**

Neural networks expect inputs to be of fixed size when provided in a batch. Since captions might be of different lengths, we pad each sentence at the end with special tokens (usually zeros) so that all inputs have the same shape. This process is known as **padding**.

**Recurrent Neural Networks**

Generating image captions involves a “sequence” of predicted words. Due to this, the assumption that fully connected networks involve (inputs provided are independent of each other) falls apart. Hence, instead of dense layers, RNNs are used.

An RNN is capable of retaining some memory of the examples it has seen earlier by feeding back the activations of its hidden layer to itself. We can unroll the RNN, where each input layer + hidden layer + output is the **same network at different times**. Also, the outputs taken from the RNN layer are usually its hidden layer activations i.e. *y0 = h0, y1 = h1*

Diagram

Description automatically generated

The network itself behaves as a fully connected network, with the output given by the expression

It incorporates the effect of the current input as well as the previous hidden layer activations, as a weighted sum.

**LSTMs**

Diagram

Description automatically generatedRNNs suffer from two issues, namely **Vanishing and Exploding Gradients**. This greatly affects their performance, making them rather unpopular in the practical space. A novel solution to this is **Long Short Term Memory (LSTM)** cells, as a replacement for the simple RNN cell. Its structure is can be shown as.

These cells have the capability to selectively read, forget and output information from inputs and past hidden states. The input-output specification fort LSTMs can be shown as follows:

Diagram

Description automatically generated

**Word embeddings (GLoVe)**

Integers are not the best way to capture information about words and their meanings. So, we give the network 8730 vectors which are randomly initialized (with smaller dimension, say a few hundred features) and let it learn the required spatial relations between words by modifying these vectors as it learns. Vectors generated through this process are called Word Embeddings. GloVe stands for **Global Vectors**. These vectors are considered as a lookup table for our words. We go to this "learned" table, ask for the vector corresponsding to a word, and replace the word's integer token with this vector. **GloVe** is an excellent group of pre-trained vectors trained on very large corpora such as Common Crawl and Wikipedia. These vectors capture co-occurences of words really well, and we use the same here.

**Activation Functions**

ReLU - In deep learning models, the Rectified Linear Unit is the most widely employed activation function. If the function receives any negative input, it returns 0, but if it receives any positive input, it returns that value. As a result, f(x) = max can be written (0, x). The slope of the ReLU function is always either 0 (for negative values) or 1 (for positive values) (for positive values). Compared to the sigmoid function or similar activation functions, Rectified linear units enable for faster and more successful training of deep neural architectures on vast and complicated datasets.

Tanh - The tanh function is similar to the logistic sigmoid, however it is more efficient. The tanh function has a range of -1 to 1. Tanh is sigmoidal as well (s - shaped). The advantage is that in the tanh graph, negative inputs will be strongly negative and zero inputs will be mapped near zero, allowing for faster convergence to the optimal solution.

**Training strategy for the model**

1. For every epoch, we initialize the data generator and loss counter.
2. We Perform *len(captions\_map)* number of iterations on the generator. This ensures that we go over each image and all of its captions in the training dataset.
3. We perform a learning step for the batch generated during an iteration and update relevant objects.
4. Perform some console outputs to track its progress.
5. Save the model.

**Evaluation**

**BLEU Score** - To compare model performance for seq2seq tasks (ones where it generates a sequence and we have to match it with reference sequences to see how well it has done), we use BLEU scores.

The *evaluate\_model* function checks the correctness of the generated sentence at multiple levels, as specified by the user. That is, it checks uni-gram matches (single words, BLEU-1), bi-gram matches (two word pairs, BLEU-2), tri-grams, and so on. We check the values of BLEU-1 to BLEU-4. A decent model should have its BLEU scores in the following ranges (taken from Where to put the Image in an Image Caption Generator, a 2017 paper).

* BLEU-1: 0.415 to 0.591
* BLEU-2: 0.167 to 0.384
* BLEU-3: 0.104 to 0.253
* BLEU-4: 0.042 to 0.183

To compute BLEU scores, we use the *corpus\_bleu* function from *nltk.translate.bleu\_score*. It needs 3 parameters:

1. **List of references**: List of documents (lists as well), each document being the set of possible correct translations.
2. **List of hypotheses**: List of predictions.
3. **Weights**: These determine the value of K in BLEU-K scores.

The model performs quite well on the training dataset, and decent on the test dataset. There could be multiple reasons to this.

1. **Overfitting**. There might have been too many parameters to train for the amount of data we have. Given the gap in training and test metric values (and the brilliant performance on training data), this is the most likely case.
2. There were elements in the images of test dataset that weren't encountered by the model very often in the train dataset. It didn't learn how to use the words describing them very well, and it couldn't reproduce them during testing.
3. The model needs to train more. This would depend on heuristics, however.

**Translation** - All that is necessary for the translate function, which is one of the VGG-16 capabilities, is an image. The "begin" character initiates the prediction. We anticipate a word each time, insert it into our sentence, tokenize (and pad) the new sentence, and repeat the process. When either "end" or "maximum length" is reached, the loop ends.

**Results**

CNN model = Alexnet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Activation function | Epoch | Loss | BLEU Training | BLEU Testing |
| tanh | 20 | 0.9688 | BLEU1: 0.4240  BLEU2: 0.2557  BLEU3: 0.1664  BLEU4: 0.1112 | BLEU1: 0.3955  BLEU2: 0.2685  BLEU3: 0.1553  BLEU4: 0.1252 |
| ReLU | 20 | 0.8440 | BLEU1: 0.3239  BLEU2: 0.1556  BLEU3: 0.0765  BLEU4: 0.0333 | BLEU1: 0.3352  BLEU2: 0.1522  BLEU3: 0.0698  BLEU4: 0.0401 |
| tanh | 30 | 0.7484 | BLEU1: 0.3434  BLEU2: 0.1955  BLEU3: 0.1100  BLEU4: 0.0565 | BLEU1: 0.3526  BLEU2: 0.1833  BLEU3: 0.1213  BLEU4: 0.0657 |
| ReLU | 30 | 0.7166 | BLEU1: 0.3569  BLEU2: 0.1934  BLEU3: 0.1255  BLEU4: 0.1122 | BLEU1: 0.3657  BLEU2: 0.1832  BLEU3: 0.1323  BLEU4: 0.1037 |

CNN model = VGGNet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Activation function | Epoch | Loss | BLEU Training | BLEU Testing |
| tanh | 20 | 0.9724 | BLEU1: 0.4156  BLEU2: 0.2636  BLEU3: 0.1523  BLEU4: 0.1039 | BLEU1: 0.3975  BLEU2: 0.2723  BLEU3: 0.1571  BLEU4: 0.1129 |
| ReLU | 20 | 0.8699 | BLEU1: 0.3379  BLEU2: 0.1996  BLEU3: 0.0823  BLEU4: 0.0636 | BLEU1: 0.3397  BLEU2: 0.1833  BLEU3: 0.0796  BLEU4: 0.0602 |
| tanh | 30 | 0.7136 | BLEU1: 0.3453  BLEU2: 0.2424  BLEU3: 0.1199  BLEU4: 0.0577 | BLEU1: 0.3466  BLEU2: 0.2243  BLEU3: 0.1266  BLEU4: 0.0581 |
| ReLU | 30 | 0.6994 | BLEU1: 0.3129  BLEU2: 0.1855  BLEU3: 0.1029  BLEU4: 0.0722 | BLEU1: 0.3208  BLEU2: 0.1900  BLEU3: 0.0991  BLEU4: 0.0715 |

**Image Predictions**

Prediction: group of people ice street with building surrounding them

1. 

Prediction: the person in the marching band

Prediction: man jumps on bike with another man on it and black and red helmet

Prediction: two men are playing in front of small grass

1. 

Prediction: two dogs run through the grass

1. 

Prediction: group of people playing soccer in the snow

***Problem 2: Language Bias of the system***

**Definition of language bias**

Even without much information into the content of the image, LSTM-based image captioning may 'blindly' learn the structure of the language and anticipate meaningful words. This is referred to as the system's "language bias."

**Metric for system comparison**

The method we use for judging the language bias comparison between system 1 and system 1 modified is that of noun bias. For comparing the language bias, we extract all nouns from both the actual and anticipated captions for the test image. If a noun appears in the predicted caption but not in the actual phrase, then it is obvious that the model has learned that term from the corpus.

**Outputs:**

1. 

*Real captions:*

* A woman wearing a white helmeted looking up as she is calming a huge rock wall.
* A woman is climbing up a steep mountain.
* People mountain climbing.
* People rock crawl up a steep obstacle.
* These helmeted people are rock climbing outdoors.s

*Predicted caption:* A woman wearing a helmet is climbing up the rock water.

*Nouns in real captions:* woman, helmet, helmeted people, huge rock wall, steep obstacle, climbing outdoor, mountain.

*Nouns in predicted caption:* woman, helmet, rock, water.

*Observations:* 2 out of the 4 nouns in the predicted caption are also present in the actual caption. The LSTM model was trained over a lot of images containing water so it got biased. Although there is no presence of water in the image, the predicted caption contains the noun water.

1. 

*Real captions:*

* A black dog on a rocky beach.
* A black dog walks along an ocean front.
* A black dog walks on the beach near the rocks.
* A black dog walks on the rocks.
* Black dog walking on the beach after swimming in the ocean.

*Predicted caption:* A black dog walking on the sand on the beach.

*Nouns in real caption:* black dog, rocks, beach, swimming in the ocean, ocean front, rocky beach.

*Nouns in the predicted caption:* black dog, beach, sand.

*Observation:* out of the 3 nouns in the predicted caption, 2 were present in the real captions too. The LSTM model was trained over a lot of images containing sand so it got biased. Although there is no presence of sand in the captions of the image, the predicted caption contains the noun water.

1. 

*Real captions:*

* a man on a bicycle jumping off a dirt ramp with one foot on the ground.
* a man on a bmx.
* a man riding a bicycle performs a trick above a dirt mound.
* a young man in tan shorts and green shirt is falling off his bicycle.
* man doing a jumping bike trick on dirt mound at night.

*Predicted caption:* a man riding a brown bike is riding a bike through brown grass

*Nouns in real caption:* dirt ramp, ground, bmx, bicycle performs, dirt mound, young man, tan shorts, green shirt, bicycle, bike trick, dirt mound, night.

*Nouns in predicted caption:* brown bike, bike, brown grass.

*Observations:* 1 out of 3 predicted caption noun is same in actual captions. The LSTM model attempts to autocomplete a sentence with grass-related information.

1. 

*Real captions:*

* A boy eats with a spoon.
* A little boy holds a spoon up to his mouth.
* A little boy is eating his food off of a spoon while sitting on a patio.
* A small child dressed in green is eating with a spoon.
* A young child holds a spoon to its mouth while sitting in a chair.

*Predicted caption:* A boy holds the spoon on green grass.

*Nouns in real caption:* boy, little boy, small child, young child, spoon, mouth, patio, food, chair.

*Nouns in predicted caption:* boy, spoon, green grass.

*Observations:* 2 out of 3 nouns in the predicted caption are also present in the actual captions. The image has a green chair and there is no presence of grass in the image. The LSTM model, because of the language bias predicts it as green grass because it has seen a lot of images containing grass.

1. 

*Real captions:*

* A beige and dark brown dog plays in the swimming pool with his mouth open.
* A black and white dog trying to bite a stream of water.
* A dog is making a splash in a blue paddling pool.
* A dog is splashing through water trying to catch ice in its mouth.
* A dog playing in some water.

*Predicted caption:* A brown dog eats water with open mouth on the blue beach.

*Nouns in real caption:* beige, dark-brown, dog, black, white, mouth open, swimming pool, water, catch ice, mouth, blue padding pool, stream of water.

*Nouns in predicted caption:* brown, dog, water, open mouth, blue, beach.

*Observations:* 3 out of 5 nouns in the predicted caption are also present in the actual captions. The image has a swimming pool having blue tiles. There is no beach in the image. The LSTM model, because of the language bias predicts the dog to be playing on a blue beach because it has seen a lot of images containing a beach and these images will certainly have a dominance of the colour blue in them.

**References**

Documents & Articles:

* https://blog.birost.com/a?ID=01550-2890dc9c-be68-4a19-81e6-c71c49d709ae
* <https://towardsdatascience.com/batch-normalization-explainedalgorithm-breakdown-23d2794511c>
* https://towardsdatascience.com/adam-latest-trends-in-deep learningoptimization-6be9a291375c
* <https://arxiv.org/abs/1502.03044>
* <https://towardsdatascience.com/batch-normalization-explainedalgorithm>breakdown-23d2794511c
* <https://towardsdatascience.com/adam-latest-trends-in-deep-learningoptimization-6be9a291375c>
* https://medium.com/@raman.shinde15/image-captioning-with-flickr8k-dataset-bleu-4bcba0b52926

Pytorch Tutorial:

* <https://pytorch.org/vision/0.8/datasets.html>
* https://pytorch.org/tutorials/beginner/nlp/sequence\_models\_tutorial.html
* <https://pytorch.org/hub/pytorch_vision_alexnet/>
* https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

Videos:

* <https://www.youtube.com/watch?v=f2ccs2xziLk>
* <https://www.youtube.com/watch?v=fUSTbGrL1tc&t=2669s>
* <https://www.youtube.com/watch?v=y2BaTt1fxJU>
* <https://www.youtube.com/watch?v=fUSTbGrL1tc>
* <https://www.youtube.com/watch?v=yWAhC95n5RM&t=545s>
* https://www.youtube.com/watch?v=X\_MaVHIBJHc

Repositories

* https://zhenye-na.github.io/2018/10/07/pytorch-resnet-cifar100.html
* https://www.kaggle.com/datasets/ming666/flicker8k-dataset?select=Flickr8k\_text
* <https://github.com/NishantPrabhu/Image-Captioning-with-PyTorch-and-Keras>
* <https://github.com/aladdinpersson/Machine-Learning-Collection>
* <https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/image_captioning>
* <https://github.com/sankalp1999/Image_Captioning_With_Attention>
* <https://github.com/aswintechguy/Deep-Learning-Projects/tree/main/Image%20Caption%20Generator%20-%20Flickr%20Dataset>
* <https://github.com/kelvinxu/arctic-captions>
* https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning