Is Fine Tuning CNN always the choice? Examining the Necessity of Fine-Tuning Pre-trained CNN Features Extractor for Image Caption Generation

Mingrui Sun
Georgia Tech
msun333@gatech.edu

Zhiqin He
Georgia Tech
zhe362@gatech.edu

Ruoshui Zhang
Georgia Tech
rzhang669@gatech.edu

Hansong Zeng
Georgia Tech
hzeng64@gatech.edu

Abstract

The Encoder-Decoder model is a fundamental component of machine learning that allows arbitrary input forms to be encoded into arbitrary output forms. However, the flexibility of the Encoder-Decoder model comes with extra complexity compared to traditional machine learning models, making it difficult to train. To address this difficulty in training, researchers commonly use a pre-trained encoder such as the CNN model. This paper builds an Encoder-Decoder network for Microsoft COCO image captioning using three distinct CNN networks - VGG16, Resnet101, and InceptionV3 - as encoders. The ImageNet pre-trained weights of these CNNs were fine-tuned on the Microsoft COCO dataset for a multi-label classification task. The paper compares the feature extraction quality of the pre-trained CNNs and the fine-tuned CNNs for image captioning tasks by training an LSTM decoder network on top of each CNN to generate captions. The paper also tests the initialization quality of the GloVe 6B 200 dimension embedding for all the Encoder-Decoder networks in this work. The results provide a cost and benefit analysis of utilizing pre-trained CNN or embedding for image captioning tasks. By comparing the performance of pre-trained and fine-tuned CNNs, the paper aims to determine whether the additional training time required for fine-tuning is worth the improved performance for the target task. We found that fine tuning only improve performance of VGG16 encoder configuration's image caption ability.

1. Introduction

The Encoder-Decoder model is a fundamental component of machine learning today, allowing for arbitrary input forms to be encoded into arbitrary output forms without limitations. First introduced by Sutskever et al. as the Seq2Seq model,[8] the Encoder-Decoder model was further improved by Bahdanau et al. with the addition of an attention mechanism.[1] However, the flexibility of the Encoder-Decoder model comes with extra complexity compared to traditional ML models, making it difficult to train. To address the difficulty in training the Encoder-Decoder network, researchers commonly use a pre-trained encoder. The encoder learns a useful transformation for a surrogate or previous similar task, allowing training to focus on teaching the decoder how to interpret that transformation. A popular pre-trained encoder is the CNN model, which has learned a large amount of ImageNet examples for classification and has shown to be useful for a range of applications as a feature extractor.[6]
However, the question of whether pre-trained CNN extracted features are already good enough for a target data domain, or whether further fine-tuning is required for the target task, is always presented. To address this question, we built an Encoder-Decoder network for Microsoft COCO image captioning using three distinct CNN networks - VGG16 ,[7] Resnet101 [5],[2] and InceptionV3 [9] - as encoders. We fine-tuned the ImageNet pre-trained weights of these CNNs on the Microsoft COCO dataset for a multi-label classification task to learn COCO dataset features. We compared the feature extraction quality of the pre-trained CNNs and the fine-tuned CNNs for image captioning tasks by training an LSTM decoder network on top of each CNN to generate captions. We also tested the initialization quality of the GloVe 6B 200 dimension embedding for all the Encoder-Decoder networks in this work. [5]
The results of this work provide valuable cost and benefit analysis of utilizing pre-trained CNN or embedding for image captioning tasks. By comparing the performance of pre-trained and fine-tuned CNNs, we aim to determine whether the additional training time required for fine-tuning is worth the improved performance for the target task.

1.1. Dataset

The data utilized in this project is from Microsoft COCO 2014 dataset,[4] which is a widely used benchmark dataset for object detection, segmentation and captioning tasks. It contains more than 330,000 images in total, with 2.5 million object instances labeled across 80 different object cate-

gories. The dataset is divided into three different sets: training, validation, and test. Training (82,783 images) and validation (40,504 images) datasets are in scope of our project; test dataset is not included given no annotation available. The dataset also includes human-labeled captions for each image, making it perfect for our captioning task.
For the CNN fine tuning part of the project. The 2014 Train and 2014 Val dataset was used for training and testing/validation of the fine tuned multilabel classification capability. For the image caption generation portion, the same 2014 Train/Val dataset was further spilled into the Train/Validation/Testset as described in Karpathy, Li(2015) works as the 2014 Test dataset didn't contain any caption label. [3]

2. Approach

In this work, we built an Encoder-Decoder network inspired by Xu et al. (2016),[11] with an example model structure shown in Figure 1. This network utilizes the VGG16 as a feature extractor, followed by an LSTM decoder to generate text captions. We adapted this model structure to allow the use of InceptionV3 and ResNet101 as feature extractors.
These three popular CNN models were chosen to cover a range of model sizes and architectures from the 2013-2015 period. InceptionV3 has approximately 23.9M parameters, ResNet101 has around 167.3M parameters, and VGG16 has 138M parameters. This variety provides a good representation of CNN feature extractors from that time. Moreover, these model sizes are suitable for training on current consumer-level hardware, such as the RTX3090, RTX2080, and GTX1080 used in this work, or cloud platforms like Google Cloud Computing with V100 GPUs.
Although the LSTM decoder can be challenging to train due to vanishing gradient problems, this specific model structure incorporates an attention mechanism that addresses this issue. Furthermore, the LSTM decoder is lighter than a transformer-based decoder like GPT-2, making training more manageable on our current hardware. On the other hand, the large size of the COCO dataset presents a challenge, as it contains a significant number of examples. To ensure that all models can be trained within a reasonable amount of time using our hardware, we carefully designed our experiments, taking into account the dataset's complexity and size.
In practice, we the Encoder-Decoder model and training pipeline from https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning.[10] Changed was made to replace legacy scipy.misc usage in the original code base and to allow adaptation of VGG16, Resnet101 and InceptionV3 for the encoder. Also a custom training pipeline for fine tuning VGG16, Resnet101 and InceptionV3 for COCO dataset multilabel classification was added. Details about each component of the model was shown in the following section.

2.1. Encoder

2.1.1 VGG16

VGG16 is a CNN that was developed in 2014 which is widely used for image classifications due to its strong performance and relatively simple architecture.[7] It consists of 16 layers, including 13 convolutional layers and 3 fully connected layers.(Figure 2) In our implementation, we removed the last 3 fully connected layers in the end of the original network by replacing them with a fully connected layer and a sigmoid layer. The goal of this updated design is to let the model emphasize on feature extraction instead of classification, which will be input into Decoder discussed in the next section.

2.1.2 Resnet101

ResNet-101 is a deep neural network architecture that was introduced in 2015 by researchers from Microsoft Research.[2] It is a variant of the ResNet family of architectures that are designed to address the problem of vanishing gradients in very deep neural networks. ResNet-101 has 101 layers and is composed of a series of residual blocks.(Figure 3) Each residual block contains two or three convolutional layers and a shortcut connection that bypasses the convolutional layers. The shortcut connection allows the gradients to flow more easily through the network, which helps to address the problem of vanishing gradients.

2.1.3 InceptionV3

The original paper reporting this model is "Rethinking the Inception Architecture for Computer Vision" in 2015.[9] An overview of this model architecture is shown Figure 4. In this work, the model with pre-trained weights before the final fully connected layer is applied to the COCO dataset. The final fully connected layer (FC) is adjusted to (in feature: 2048, out feature: 80) and the weights of this layer are fine tuned with the COCO dataset for multiple categories. The loss function used in the retraining is Binary Cross Entropy loss, applied with nn.BCELoss() from PyTorch module.

2.2. Decoder

The structure of the decoder is summarized in Figure 5 and is mainly constructed by two components: the LSTM and attention module. The attention module takes in the encoder features and the last hidden states of the LSTM as inputs to produce attention-weighted feature maps. These attention-weighted feature maps are then fed back to the LSTM with the last word generated by the LSTM as input to generate the next word. In the training process, teacher forcing was used instead of feeding the actual word generated by the LSTM. The generation stops once the ¡end¿ token is generated or text length reaches the maximal length. In the train and validation step, teacher forcing was used 100% of the time with BLEU-4 score calculated at each epoch validation. Note this validation BLEU-4 score is distorted by teacher forcing. In the final evaluation of captioning capability, beam search was used to produce the real BLEU-4 score of the model on the testing set. Also visualization of image gradient activation for each word generated were shown to show the quality of the caption.

2.3. Data and Train/Validation Pipeline

Data Pipeline is set up to automatically load COCO 2014 datasets for training and testing. They are pre-downloaded from COCO websource[1] which includes 3 folders: "train2014", "val2014" for images and "annotations_trainval2014" for annotations and instances. Annotations and instances are in Json format which COCO's API library "pycocotools.coco" is used to extract the "Annotations" and "Category ID" (Labels) for each Image.[4]

After data is extracted, a python script is set up to preprocess the data by converting into a format suitable to the training pipeline line. This involves resizing and cropping the images based on the corresponding encoder model of choice, normalizing the pixel values, and creating a mapping between the image files and their corresponding annotations. Image rotation is also performed as a data augmentation technique.

For the caption task with the entire Encoder-Decoder network, the train/validation/test described in Karpathy and Li(2015) was used.[3] A HDF5 file was created containing the entire train/test image dataset and corresponding JSON with the order and list of captions for each image in the file was used. In the training step, the HDF5 was read directly from the disk to reduce usage of memory during training.

3. Experiments and Results

3.1. Encoder Fine Tuning

3.1.1 VGG16

To be able to fine-tune VGG-16, pre-trained weights were used to transfer learning on the COCO 2014 dataset without freezing any layers. Different combinations of hyper-parameters were experimented to be able to find the best model with highest average validation accuracy and lowest loss as well as other qualitative metrics such as over-fitting and training time.
Due to the constraint on a GTX2080 , only batch sizes of 32 vs 64 were investigated for this model. Although, studies implied that large networks require larger batch sizes.[12] However, based on our experiment, batch size 32 actually has lower losses than batch size 64 given other parameters stay the same, this is because smaller batch sizes usually generalize well. Different optimizers were tested, Adam in general has lower losses compared to SGD which is due to it having a decreasing schedule of learning rate. When using SGD, higher momentum 0.9 is tested against 0.1. High momentum turns out to have better losses due to it helps SGD not to be trapped at saddle point. Ultimately, different learning rates were analyzed. Higher learning rate such as 0.1 turns to be overshooting which implies a much higher loss. Learning rate too small like 0.00001 trains much longer and does not yield the best outcome as well. The best case learning curve was shown in figure 6. All hyperparameter experiment discussed here are summarized in figure 7. Finally the best model from our experiment for VGG16 is using Adam with a learning rate 0.001.

3.1.2 Resnet101

To fine-tune ResNet101, pre-trained weights were utilized to transfer learning on the COCO 2014 dataset without freezing any layers. Various hyper-parameter combinations were tested like VGG-16 to identify the optimal model with the highest average validation accuracy and lowest loss.
Due to the constraint of local graphic card, the VM instance on the Google Cloud Console was set up with the required specifications and environment, installed the necessary software packages and downloaded COCO dataset. The trained Resnet-101 weights were saved to the instance and downloaded to the user's local machine for deployment. Learning curve of the best hyperiparameter selected was shown in figure 8. All result of hyperiparameter tuning on resnet101 training was summarized in figure 9.

3.1.3 InceptionV3

The Inception V3 model from Pytorch, with modified FC layer, is run with the Nvidia 2070 GPU. With preliminary experiments with the code and hardware, it is found that the training and validation loss changes with a much slower speed after 5 Epochs (Figure 10). In addition, the validation loss increases, indicating the overfitting with more epocs. Considering the learning performance, as well as the computation resource available, hyperparameters with Batch size of 64, Epoch of 5, learning rate varying from 0.0001 to 0.01, two optimizers (SGD and Adam), momentum (with SGD optimizer) of 0.9, are investigated to find the optimized hyperparameters. All result of hyperiparameter tuning on InceptionV3 training was summarized in figure 11.

3.2. Summary for Encoder Fine Tuned

The training loss, validation loss, and computation efficiency are quantitatively analyzed and compared among the three CNN models. These result was summarized in Figure 12. It can be seen from the sections above that the results from the models are comparable with lowest training and validation loss from the Inception V3 model, slightly better than the other two models. In summary, the Inception V3 model showed the lowest training and validation loss. The lead of InceptionV3 in the classification surrogate task is likely due to InceptionV3 architecture use of a convolution kernel of different size which enables it to better extract features from images at different scales.

3.3. Decoder

Two types of experiments were conducted on the encoder-decoder model. The first experiment compared the initialization ability of the pre-trained GloVe 6B 200-dimensional embeddings (GloVe) to random initialization (RNG). The second experiment compared the caption generation ability of the encoder fine-tuned on the COCO dataset (FT) to the encoder with ImageNet pre-trained weights (RNG). All encoder-decoder experiments were trained and tested on an RTX 3090 GPU with a batch size of 128. For all experiments, a learning rate of 1e-4, 20 epochs of training, and early stopping based on validation loss were used with the Adam optimizer. Example of a caption generated and corresponding attention maps for each word was shown in Figure 13. Each of the nine model configurations took about six hours to train, which was a significant time investment. We chose the adaptive learning rate optimizer, Adam, as it is less sensitive to the learning rate input than SGD, thus requiring less tuning and saving time. However, we acknowledge that we did not fully explore the learning rate space in this study, so the results presented here may be subject to local minima in the training optimization.The resulting learning curve of all model configuration training was shown in figure 14, 15, 16 for VGG16, Resnet101, InceptionV3 respectively. All experiment result was summarized in figure 17.

For the first experiment, we found that random embedding initialization achieved higher caption quality than the GloVe 6B initialization for all three tested encoder architectures. This was surprising, as pre-trained embeddings like GloVe are generally perceived as high-quality initializations that already capture semantic and syntactic relationships between words. However, as our results show, this is not always the case, as pre-trained GloVe embeddings are not specifically designed for image captioning tasks. In real-world applications, it is best practice to experiment with different initialization combinations to achieve the best results.

Regarding the second experiment on whether to fine-tune the encoder or not, the results were intriguing. When comparing the ImageNet weight model (RNG) to the COCO dataset fine-tuned model (FT), we observed mixed results. For VGG16, which has fewer filter outputs (512), fine-tuning with the multi-label classification improved caption quality. In contrast, for ResNet101 and InceptionV3, which both have 2048 filters, fine-tuning actually reduced the performance of the overall model. This contradicts the general rule of thumb of always fine-tuning, especially in cases like the COCO dataset where there is likely sufficient data to refine features without concern for overfitting due to the large sample size. Three competing explanations may account for these results: 1) VGG16 (512 filter map) is less complex than ResNet101 and InceptionV3 (2048 filter map), making fine-tuning VGG16 to a global minimum for the COCO dataset potentially easier than for ResNet101 or InceptionV3; 2) the COCO dataset may not provide enough examples for fine-tuning features extracted by ResNet101 and InceptionV3; 3) the default ImageNet weights may already be optimal for ResNet101 and InceptionV3, leaving no room for improvement.

In terms of the best case caption generation performance, Resnet101 outperformed others with VGG16 came in second place. This was a surprise as we showed that for the multilabel classification(surrogate task), InceptionV3 actually performed the best. This may potentially mean that the surrogate task we fined tuned encoder on does not fully align with our target task. Also the validation BLEU-4 score generated is systematically lower than the test score as teaching forcing was turned off for the evaluation. Without teacher forcing, the model seems to be more resilient to error than with teacher forcing. This showcases the need to implement schedule sampling in training.

In conclusion, our results suggest that for pre-trained CNNs like VGG16, InceptionV3, and ResNet101, using ImageNet weights for image captioning tasks is a safe choice. In real-world applications with smaller training sets than the COCO dataset, it may not be reasonable to consider fine-tuning the CNNs.

4. Future Work and Potential Improvement on Current Approach

4.1. Mulilabel Classifcation Surrogate Task

The surrogate task(multi-label classification) used in CNNs fine tuned may not be optimal for image caption purposes. Better surrogate tasks like image segmentation or object detection may be better surrogate tasks for use cases in this work as those tasks need location information of the object to be successful. In the future those surrogate tasks can be adapted.

4.2. Mismatch Between Training Hardward

In the encoder fine tuning step is done in this work. Due to the teamwork nature of the group project, three models weren't trained on the same hardware with large enough VRAM like the 3090. This limited our ability to explore the batch size space in training these models. Fine tuning these models may only be possible on hardware that enables larger batch size due to potential gradient noise associated with small batch. In the future we standardize the hardware used so batch size won't be an issue.

4.3. Features Map Size Bottleneck

The caption generation compatibility of this model used in this work is constrained by the bottom neck features map sizes which a pooling was used to standardize all configuration to be 7*7. In fact InceptionV3 can output larger size features map of 8*8. So if the larger features map is adopted, InceptionV3 based caption generation capability may improve. Also using a larger features maps pretrained CNN may also improve our caption ability, the original Xu paper actually use a Resnet152 for the larger features map.[11]

4.4. Maybe Transformer?

From an image caption generation standpoint, the Encoder-Decoder architecture we adapted from Xu et al. (2015) is great in 2015 but then the mighty transformer came out.[11] Using our current setup, even the best model only achieves a 0.3 BLEU-4 score in caption generation which is not bad but far from level like 0.5 which indicates a good resemblance to the label caption. Two improvements can be made: 1) Use a transformer base Decoder architect. 2) Implement a probabilistic or temperature based teaching forcing schedule in training instead of the current 100% teacher forcing in training.

5. Figures and Table

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 1

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 2, 6

[3] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015. 2, 3

[4] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. 1, 3

[5] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014. 1

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. 1

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 2, 6, 7

[8] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014. 1

[9] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 1, 2

[10] Sagar Vinodababu. Sgrvinod/a-pytorch-tutorial-to-image-captioning: Show, attend, and tell: A pytorch tutorial to image captioning. 2

[11] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015. 2, 5, 6

[12] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017. 3

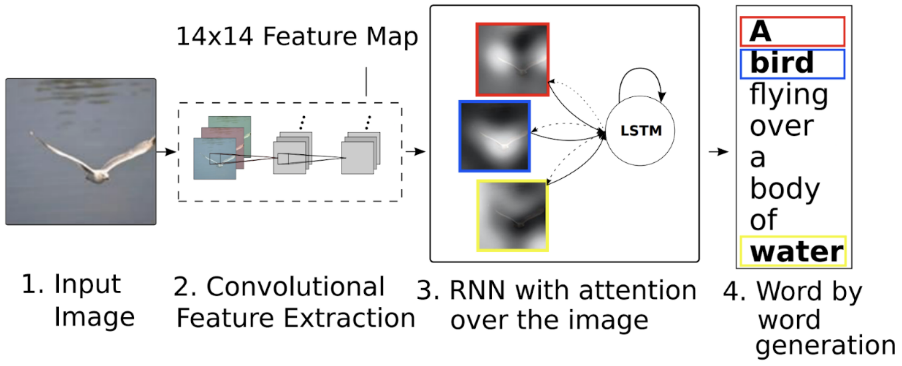

14x14 Feature Map
LSTM
A
bird
flying
over
a
body
of
water
1. Input Image
2. Convolutional Feature Extraction
3. RNN with attention over the image
4. Word by word generation

Figure 1. Encoder-Decoder Network for Image Captions[11]

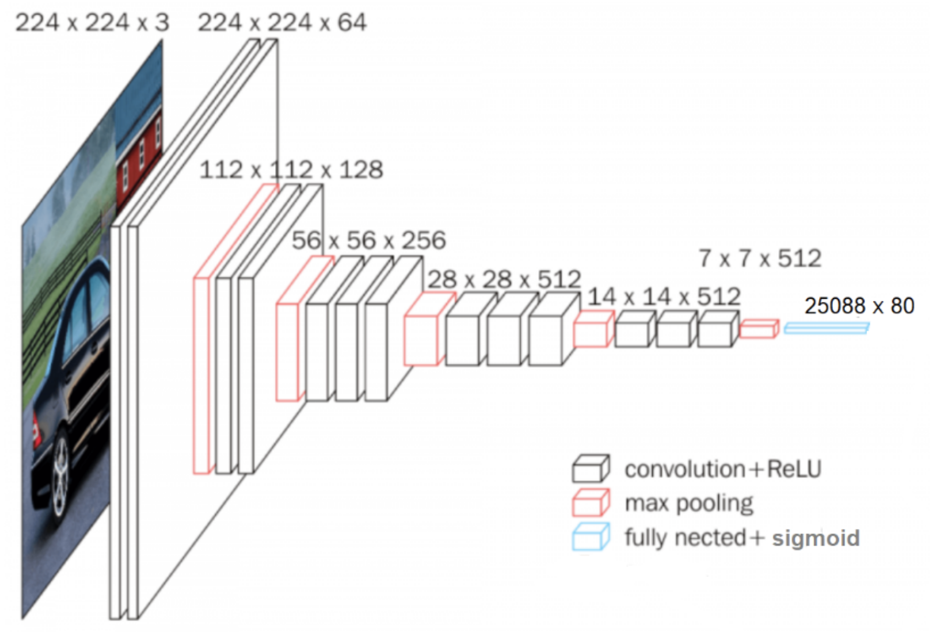


Figure 2. VGG16 Network Stcture[7]

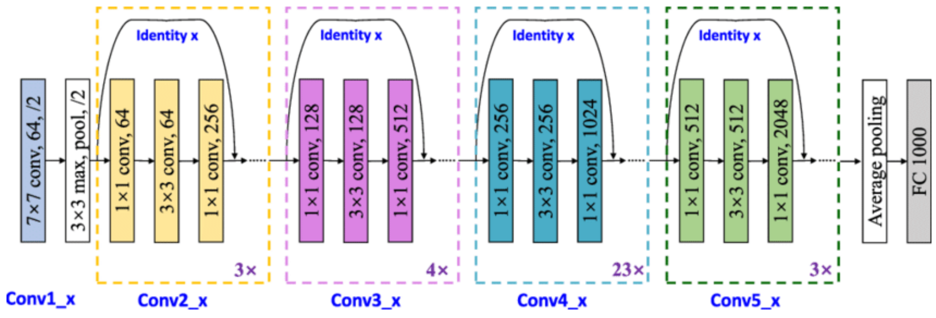


Figure 3. Resnet101 Network[2]

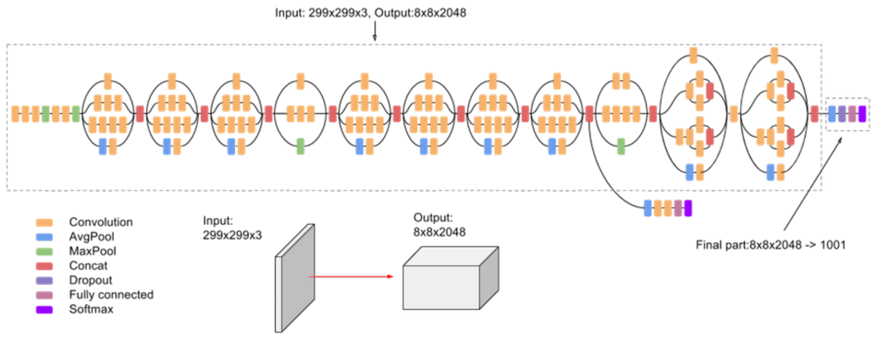


Figure 4. InceptionV3 Network[7]

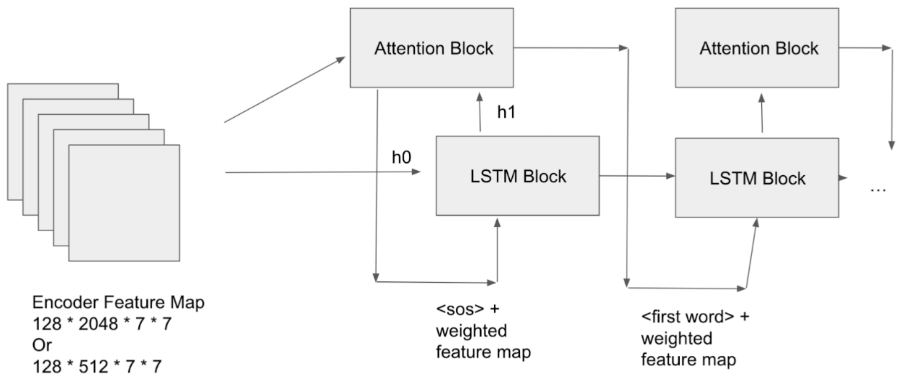


Figure 5. LSTM Decoder Network with Attention Mechanism. Note 2048 is the filter number of Resnet101/InceptionV3 and 512 is for VGG16.

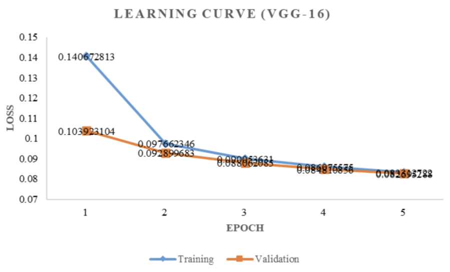


Figure 6. Learning Curve of the Best VGG16 Setting

Hyperparameters Experiment (VGG-16)							
Batch Size	64	64	64	32	32	32	32
Learning Rate	0.1	0.00001	0.0001	0.0001	0.0001	0.0001	0.0001
Epoch	10	10	5	10	5	5	5
Optimizer	Adam	SGD	SGD	SGD	SGD	SGD	Adam
Momentum	NA	0.9	0.9	0.9	0.9	0.1	NA
Training Loss	3.5128	0.1214	0.0913	0.0764	0.0833	0.1176	0.0548
Validation Loss	3.5253	0.1198	0.0901	0.07706	0.0826	0.1167	0.06310

Figure 7. Summary of all Hyperparameter Experimented in VGG16 Fine Tuned

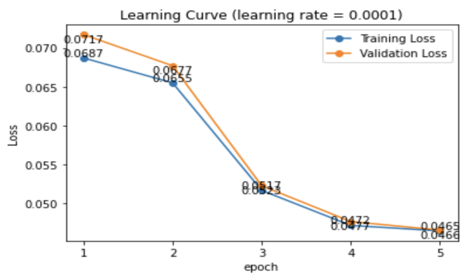


Figure 8. Learning Curve of the Best Resnet101 Setting

Hyperparameters Experiment (ResNet-101)					
Batch Size	64	64	64	64	64
Learning Rate	0.01	0.01	0.001	0.0001	0.0001
Epoch	5	5	5	5	5
Optimizer	SGD	Adam	Adam	Adam	SGD
Momentum	0.9	NA	NA	NA	0.9
Training Loss	0.1178	0.0995	0.0635	0.0465	0.1354
Validation Loss	0.1205	0.1102	0.0601	0.0467	0.1324

Figure 9. Summary of all Hyperparameter Experimented in Resnet101 Fine Tuned

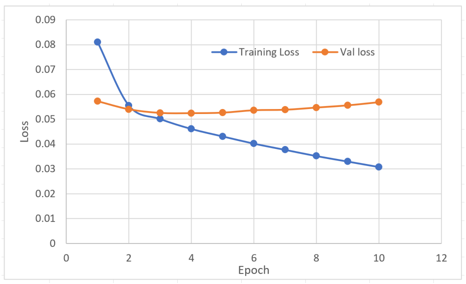


Figure 10. Learning Curve of Inception V3

Hyperparameters Experiment (Inception V3)						
Batch Size	64	64	64	64	64	64
Learning Rate	0.01	0.01	0.001	0.001	0.0001	0.0001
Epoch	5	5	5	5	5	5
Optimizer	SGD	Adam	SGD	Adam	SGD	Adam
Momentum	0.9	NA	0.9	NA	0.9	0.9
Training Loss	0.0644	0.1217	0.1128	0.0612	0.1689	0.0430
Validation Loss	0.0607	0.1179	0.1076	0.0651	0.1611	0.0524

Figure 11. Summary of all Hyperparameter Experimented in InceptionV3 Fine Tuned

Model	Training Loss	Validation Loss	Training Accuracy	Validation Accuracy	Training Run Time	Validation Run Time
VGG-16	0.0548	0.0631	0.9576	0.9465	330.33min	178.23min
ResNet-101	0.0465	0.0451	0.9799	0.9762	225.87min	153.37min
Inception-v3	0.1128	0.1076	0.9227	0.9265	255.32min	162.11min

Figure 12. Summary of Best Performing CNNs Classification Performance

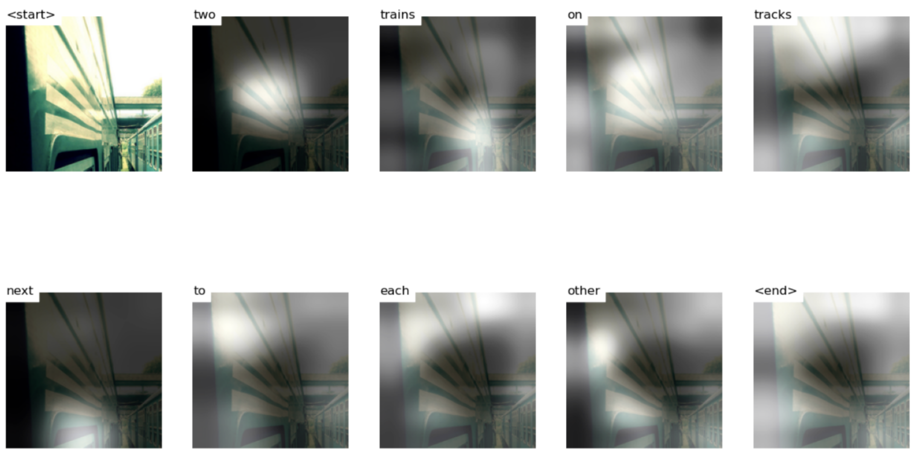


Figure 13. example of a generated image caption, with attention maps shown for each generated word. The attention maps highlight the regions of the image that the model focused on while generating each word in the caption

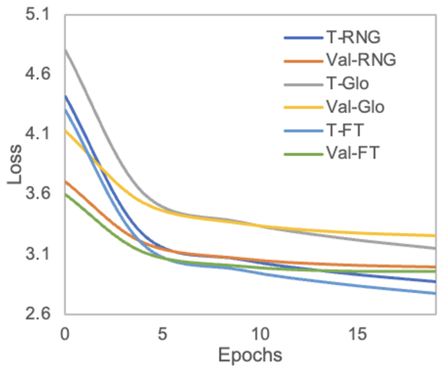


Figure 14. Learning Curve of all VGG16 base model. RNG, Glo, FT here stands for random embedding initationization with default CNN weight, GloVe embedding initationization with default CNN weight, random embedding initationization with fine tuned CNN weight repectively.

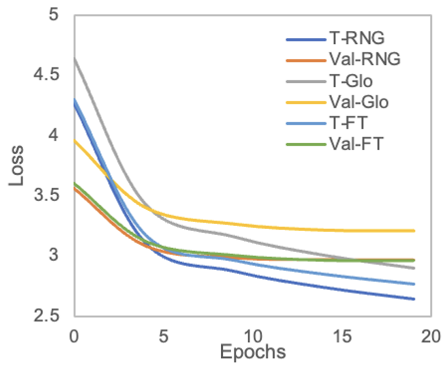


Figure 15. Learning Curve of all Resnet101 base model. RNG, Glo, FT here stands for random embedding initationization with default CNN weight, GloVe embedding initationization with default CNN weight, random embedding initationization with fine tuned CNN weight repectively

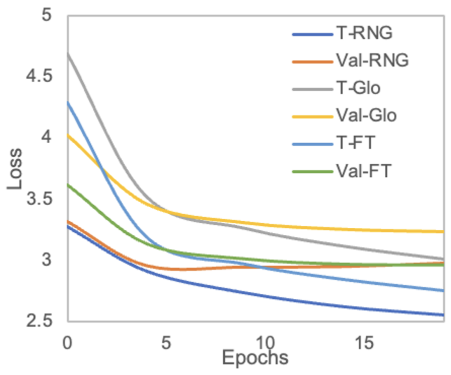
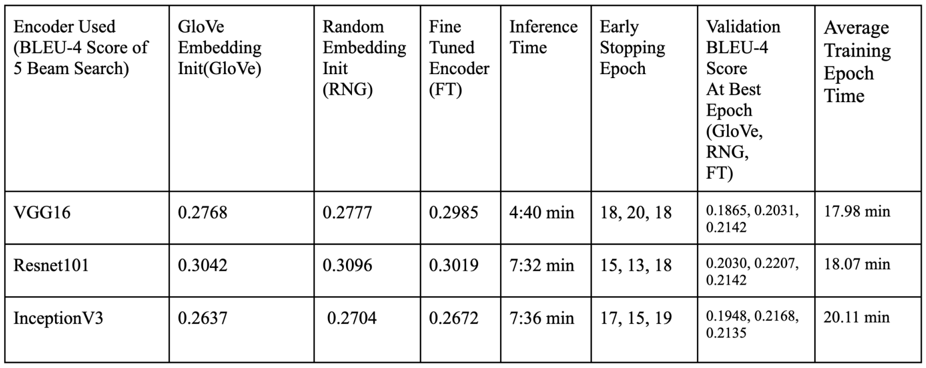


Figure 16. Learning Curve of all InceptionV3 base model. RNG, Glo, FT here stands for random embedding initationization with default CNN weight, GloVe embedding initationization with default CNN weight, random embedding initationization with fine tuned CNN weight repectively



Encoder Used (BLEU-4 Score of 5 Beam Search)	GloVe Embedding Init(GloVe)	Random Embedding Init (RNG)	Fine Tuned Encoder (FT)	Inference Time	Early Stopping Epoch	Validation BLEU-4 Score At Best Epoch (GloVe, RNG, FT)	Average Training Epoch Time
VGG16	0.2768	0.2777	0.2985	4:40 min	18, 20, 18	0.1865, 0.2031, 0.2142	17.98 min
Resnet101	0.3042	0.3096	0.3019	7:32 min	15, 13, 18	0.2030, 0.2207, 0.2142	18.07 min
InceptionV3	0.2637	0.2704	0.2672	7:36 min	17, 15, 19	0.1948, 0.2168, 0.2135	20.11 min

Figure 17. Summary of Results of all Caption Experiment Done

Student Name	Contributed Aspects	Details
Zhiqin He	Surrogate Task Data Loading and VGG16	Prepare Data Pipeline and training code for multilabel classifcation fine tuned of the encoder.Analyse result and prepare document.
Mingrui Sun	Image Caption Task Data and the Overall Model	Prepare Data Pipeline for caption task and training the whole model on different configuration. Analyse result and prepare latex document format.
Ruoshui Zhang	Resnet101 Training	Fined tuned Resnet101 on COCO dataset on Google Cloud Computing VM. Optimize training hyperparameters. Analyse result and prepare document.
Hansong Zeng	InceptionV3 Training	Fined tuned InceptionV3 on COCO dataset. Optimize training hyperparameters. Build modify InceptionV3 forward function to output the features map. Analyse result and prepare latex document format.

Table 1. Contributions of team members.