A Survey of Convolution and Recurrent Neural Networks in a Merge Encoder-Decoder Architecture for Image Captioning

Roxane Martin, Quyen Ngo, Conway Xu
Professor Krishnagopal
Math 156: Machine Learning

Abstract. This project aims to develop an image caption generation model that uses a pre-trained convolutional neural network (CNN) as an image encoder and a recurrent neural network (RNN) as a text encoder, followed by a concatenating decoder. The model generates captions for input images by learning the relationship between image features and corresponding textual descriptions. Cosine proximity and BLEU scores are used to evaluate the accuracy of generated captions by measuring the similarity between generated and ground-truth captions. Different pre-trained CNNs such as InceptionV3, EfficientNetB0, and ConvNeXtTiny are used to extract image features, and the architectures of the RNNs are adjusted to optimize the model's performance. The results show that while the model's performance does not see significant change when varying the size and choice of RNN layers, it is dependent on the choice of the pre-trained CNN. The final proposed model achieves a cosine similarity score of 0.7812 and 0.0479 BLEU accuracy on the Flickr8k dataset, which is implemented with an InceptionV3 CNN and a two-layer LSTM network.

Introduction. In recent times, there has been a growing interest in integrating feature extraction techniques from images with generative language models to solve a wide variety of computer vision tasks. One task that has gained significant traction is image captioning, which involves automatically generating natural language descriptions of an image's context. The primary objective of this task is to produce text that can be easily comprehended by human readers.

The importance of image captioning lies in the fact that visual data is extensively used in numerous fields, and automatic captioning can expedite many of the underlying processes. Image captioning finds potential applications in automating captioning products for sale in online catalogs, aiding individuals with visual impairments, or even social media, where location detection can be inferred from the background of an image. In this project, we aim to delve into this area of research and compare various model architectures in order to develop a model that can generate contextually relevant captions for the Flickr8k set of images. Our goal is to answer the question of which CNN-RNN combination produces the most accurate captions as evaluated by cosine similarity of word embeddings and BLEU score.

Background. Previous research has produced survey papers examining various image caption models, including Hodosh et al.'s (2013) comparison of various CNN-RNN models. They examined VGG-16, VGG-19, ResNet-50, ResNet-101, InceptionV3, Inception-ResNet-v2 [4].We expand on their survey, by choosing a range of more modern CNNs to compare: we use InceptionV3 as our base model, and then also evaluate EfficientNetB0 and ConvNeXtTiny. Additionally, we vary the RNN between a long-short-term memory network and a gated recurrent unit network to identify an effective pairing. The abundance of survey papers on image caption models exhibits the existence of numerous models for this task, with the encoder-decoder architecture being the most commonly used, along with a large number transformer models which incorporate attention mechanisms [9]. It is worthwhile to note that research on image classification models is undeniably relevant to image captioning tasks since those models are employed to acquire feature encodings.

Dataset. We use the Flickr8k dataset for training and testing our model, a widely used benchmark dataset for image captioning. Originally introduced in 2013 by researchers from the University of Illinois’s

computer science department, it contains 8,000 images, each of which is accompanied by five (often similar, but unique) captions [5]. The images depict a wide range of scenes, including people, animals, and landscapes with image captions written by human annotators that aim to provide a brief but comprehensive description of the central content of the image. However, at only 8,000 images, the Flickr8k dataset is relatively small compared to more recent datasets, like MSCOCO or Open Images, and as such likely does not encompass the full range of possible image content or caption vocabulary.

Theory. The general structure of our model combines an image-based model and a language-based model in the architecture of a *Merge Encoder-Decoder* model. A diagram of this architecture can be seen in Figure 1 [2].

As our image encoder, we use a convolutional neural network trained via transfer learning for image feature extraction. This has two interesting theoretical portions to discuss: CNNs' application to image feature extraction and the efficacy of transfer learning.

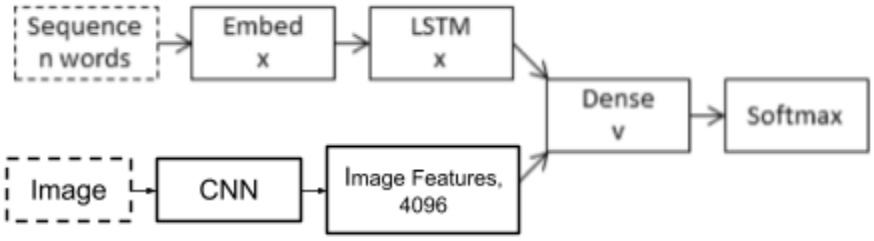


Figure 1: Merge Architecture for Encoder Decoder

CNNs operate by applying convolutional filters to subsamples of the input and performing element-wise multiplication and summation, resulting in a feature map that captures the edges, forms, and objects in an image. Using parameter sharing and local connections, CNNs are able to effectively learn from large datasets of images while keeping the number of parameters in the network tractable. They are translation-invariant which allows them to identify the same features regardless of their position in the input image [2]. The pooling layers alternating with convolutional layers serve to downsample the feature maps which limits the spatial dimensions of the input image while retaining the most important features, making them ideal for tasks that require feature extraction [1].

Transfer learning is particular common and useful in such feature extraction. As the large majority of computer vision tasks necessitate some sort of feature representation of images, there exist a multitude of very deep CNNs trained on large datasets–CNNs that would not be feasible to train on personal machines (or with the resources available to a college student). The concept of transfer learning is to employ these models, or portions of them, as a basis for models specialized to smaller tasks [6]. The specifics of the transfer learning we employed are discussed in the Methods section below, but generally, we use the intermediate feature representation found in powerful image classification models as our rich encoding on which to base caption generation.

The three image classification models we selected as our CNNs are InceptionV3, EfficientNetB0, and ConvNeXtTiny. These three models were chosen because they span a period of rapid improvement in image classification CNNs, having been released in 2016, 2019, and 2022. For all falling under the category of CNN, they have notable conceptual differences and each represented a novel architecture at the time of their publication. InceptionV3 uses a very deep and wide neural network architecture with multiple inception modules, which allows the network to capture multi-scale features at varying levels of abstraction [8]. EfficientNetB0 is the smallest model from the family of *EfficientNet*, which use compound scaling techniques to efficiently scale up the neural network while maintaining performance [10]. ConvNeXtTiny, likewise the smallest model from its family *ConvNeXt,* employs a hybrid

architecture with both convolutional and non-convolutional layers [7]. For the latter two models, we selected the models with the fewest parameters in their so-called-families as we are dealing with a small dataset and wish to prioritize efficiency and minimize overfitting. Diagrams of their individual architectures are suppressed here, but can be found in the Appendix.

As a text encoder, we use various recurrent neural networks to generate textual descriptions. Our first two models are long-short-term-memory networks (LSTMs), and our third model is a gated recurrent unit network (GRU). LSTMs are a special class of RNNs and as such perform best on tasks dealing with sequential data. Caption generation is perfomed by predicting one word at a time, conditioned on the previous words–a prime example of a sequential task [11]. The advantage of LSTM and GRUs over vanilla RNNs comes from its use of gated cells and memory units to selectively remember information from previous time steps, allowing it to maintain relevant features over longer sequences while avoiding retaining irrelevant detials [2]. The captions associated with each image are vectorized as word embeddings, paired with the feature representation of the image, and fed as input to the decoder.

The decoder itself is simply an addition layer that merges both encoded inputs, passes the merged encoding through a series of dense layers to predict the next word of the caption [2].

Evidently, for this encoder-decoder model to work well, certain assumptions must be met. Most rudimentary, the model requires captions to be in a single consistent language. The relevance of image features extracted by the CNN is central to accurate caption generation, followed closely by effective hyperparameter tuning. Larger models, like LSTMs and GRUs, have many tuneable hyperparameters that greatly impact model performance. There are also certain conditions under which the model may fail to generate accurate captions. For instance, ambiguity in image content or more complex images may make it difficult for the CNN to correctly capture relationships between objects. The size and quality of the training set may also negatively impact model performance. The decoder only has access to vocabulary that occurs in captions of training images, so it may generate repetitive or generic captions if the provided training vocabulary is not sufficiently large [1]. Similarly, inconsistency in image quality may make accurate feature extraction by the CNN unreliable. Our aim with our chosen dataset, use of transfer learning for our encoder, and hyperparamer choice for our decoder is to mitigate these negative effects and build a strong model.

Methods. As described above, our model architecture consists of a CNN and an RNN. Our objective is to investigate and evaluate various choices for each of these to identify the combination that achieves the best performance on the image captioning task at hand.

First, we will detail how we applied transfer learning to train our CNN. InceptionV3, EfficientNetB0, and ConvNeXtTiny are all image classification models which can be accessed via the Tensorflow Keras applications module and are each loaded with pre-trained weights learned from running on ImageNet, an image database organized according to the WordNet hierarchy. Each one of these three models has a distinct inner architecture, but generally, their last layer is fully-connected layer with a softmax activation function that serves to do the final classifying step for image classification problems. Since we are not concerned with image classification but rather are interested in the feature representations produced by these models, we drop the final layer by setting the include_top parameter to False [6].

In order to have our feature representation match the desired input to our decoder, we must add a pooling layer to the network. The decision to be made is between ‘avg’ and ‘max,’ meaning that global average pooling will be applied to the output of the last convolutional block or that global max pooling will be applied, respectively. We chose to implement an average pooling layer because average pooling

typically preserves more spatial information than max pooling, making it better suited for providing detailed information about different parts of the image [6]. Additionally, average pooling tends to work better when there is variability in the size and position of objects within the image, which is information we value within the context of an image processing task. Max pooling often discards information about the size and location of objects within an image which may make it more difficult for the decoder to generate accurate captions.

The resulting feature representations from InceptionV3, EfficientNetB0, ConvNeXtTiny have shapes of (, 2048), (,1280), and (,768) indicating the rich encoding is made up of those numbers of features. Dependant on the CNN, we adjust the input of our decoder to ensure vector sizing alignment without having to retrain the encoding model. By freezing the weights of the layers of the three pretrained models to avoid changing the learned features, we are able to use the weights as learned on the larger ImageNet dataset to extract rich features embeddings from the input images.

Once images are encoded as feature vectors of the correct dimension, the RNN is trained as a language model on these representations and the word embeddings of the captions associated with each image. To generate our word embeddings, we employed SpaCy, a software library for natural language processing [4]. Specifically, we use SpaCy's pretrained pipeline, created using word2vec algorithms and trained on data obtained from online sources such as Wikipedia and news websites. The training captions are passed as input, embedded via the SpaCy embedding matrix, and subsequently passed through one of three distinct implementations of RNN to our decoder.

The first implementation is a simpler model, consisting of a 0.5 dropout layer followed by a 256-unit LSTM layer. The specific architecture of our full model using this LSTM and the InceptionV3 CNN can be seen in Figure 2. The second implementation features a 0.2 dropout layer, followed by a two-layer stacked 256-unit LSTM architecture, which is expected to provide a better fit for the data, but may also be susceptible to overfitting. The third implementation employs a 0.2 dropout layer, followed by two stacked 256-units GRU layers. The corresponding diagrams for these latter models can be found in the Appendix.

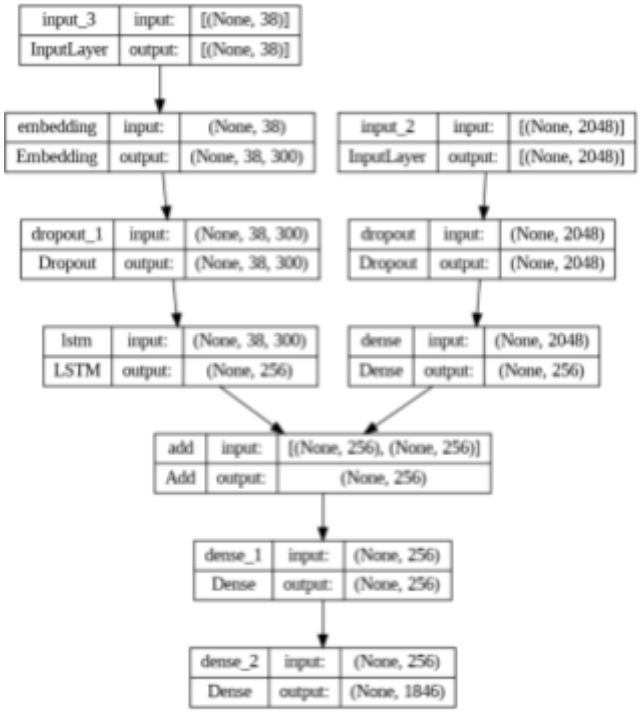


Figure 2: RNN Model 1: 1-Layer LSTM (with InceptionV3 as CNN)

The primary objective of our project is not to optimize the hyperparameters, but rather to determine the most effective combination of CNNs and RNNs. As such, the number of units in each RNN layer is held constant across all models to minimize the influence of hyperparameters on our experimental outcomes.

During model training, we utilize the built-in cosine proximity metric to evaluate the similarity of generated captions to training captions at each epoch. Once we have a trained model, we then compare the output captions with the ground-truth captions by computing the cosine

similarity between them. The cosine similarity metric is defined as A.B/(|A||B|), so values range between -1 and 1 where a value of -1 indicates completely opposite captions, while a value of 1 indicates exactly similar captions.

To evaluate the accuracy of our models' predictions, we also employ the BLEU (bilingual evaluation understudy) metric. Unlike the cosine similarity metric, BLEU does not depend on vectorized word embeddings. Instead, it splits reference and generated captions into n-grams, i.e. sequential phrases of length n (We use n=1,2,3, and 4 in our experiments). BLEU generates a score based on the number of matching n-grams between the reference and generated captions and then scales that score based on the length of the generated caption in comparison to the actual captions [9]. While BLEU is a widely-used metric for evaluating captioning models, we note that it is completely dependent on the specific words of the reference and generated captions. Therefore, it may be the case that the generated caption captures the correct context, but unless the wording is exact, the BLEU score may be very low.

Results. When considering the results to highlight in our report, instead of presenting all nine permutations of CNN-RNN models, we decided it would be best to showcase a truncated subset of the nine total models. First, we ran each of the RNN models (1-layer LSTM, 2-layer LSTM, 2-layer GRU) in conjunction with our base CNN encoder, InceptionV3, to choose the most accurate RNN.

The cosine proximity and loss of models 1, 2, and 3 when paired with the InceptionV3 CNN as those values change over 10 epochs can be seen in Figure 3. These values were calculated for both the training set, labeled in the legend as 'train-n' for the nth model, and for a validation set, labeled 'test-n' (See Appendix for training parameters). As the final cosine similarity values of the three models have very little disparity, we turn to the BLEU score to choose the best image captioner.

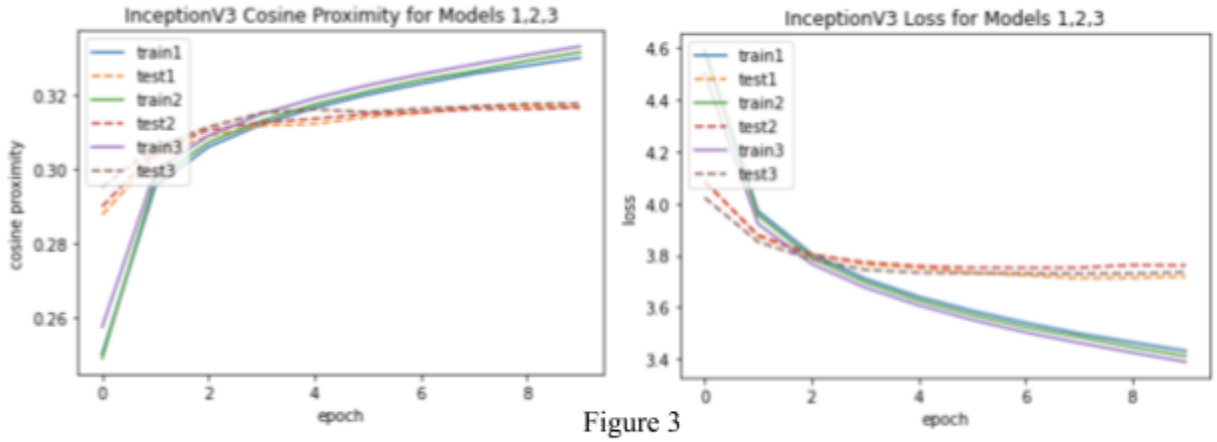


Figure 3

Figure 4 displays the BLEU scores for the three models and indicates that, by a small margin, the 2-layer LSTM model (model2) has a higher average BLEU score over 500 images.

After having made this selection, we ran the most performant RNN (2-layer LSTM) with the other two CNN models (EfficientNetB0 and ConvNeXtTiny). In Figure 5, we once again see cosine proximity and loss over 10 epochs, but this time of the training and test evaluations of InceptionV3, EfficientNetB0, and ConvNeXtTiny paired with model2.

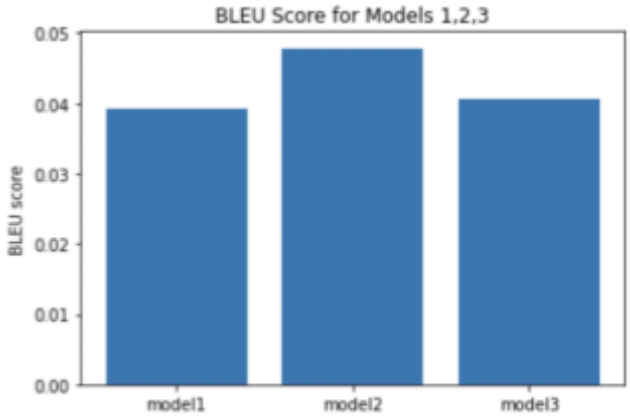


Figure 4

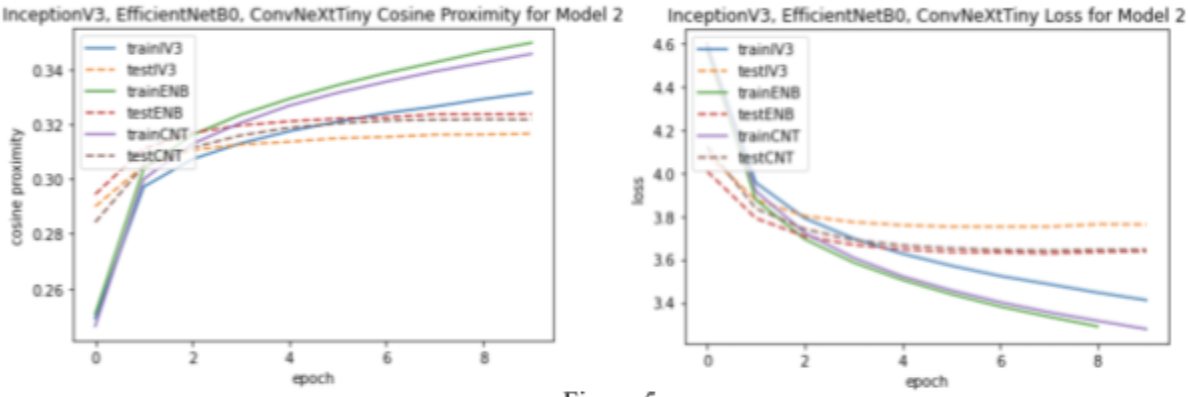


Figure 5

Even more so when only using the 2-layer LSTM, the BLEU scores for the three CNNs are extremely similar. Figure 6 shows us there is a slight advantage to using the InceptionV3 CNN when generating image captions on our specific dataset. This could be accorded to the fact that the InceptionV3 model has the highest number of image features, and therefore the richest encoding. Since many of the captions we are comparing against when using the BLEU metric use very specific language, the InceptionV3 is likely being rewarded for maintaining specific image features rather than generalizing more, as EfficientNet0 and ConvNeXtTiny's smaller feature representations may tend to do.

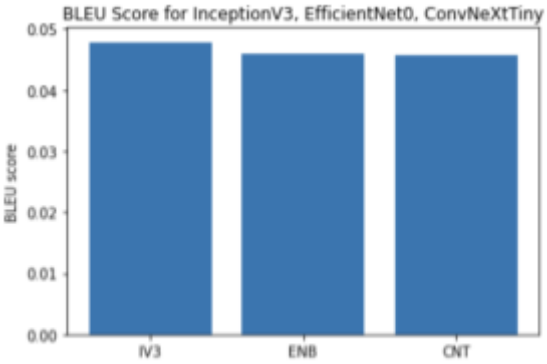


Figure 6

There is a strong dichotomy between the context score, calculated via cosine similarity between word embeddings, and the BLEU score, calculated via n-gram comparisons between the actual and generated captions. Specifically, the context of generated captions for every model performs quite highly, indicating that most models are able to generate accurate caption in reference to the image itself. On the other hand, the models each gave fairly low BLEU scores of >0.05 (0.3 is considering the minimum score for "understandable" captions) [9]. That being said, the low BLEU scores are not an indictment against the performance of the model given the fact that higher BLEU scores are only representative of the similarities of the exact words and phrases being used between the generated and actual captions.

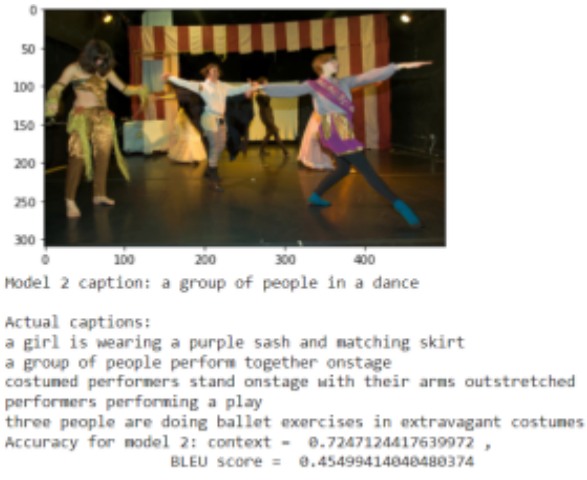


Figure 7

Figure 7 shows an example run of generating a caption with the InceptionV3 model and the 2-layaer LSTM for a random test image in our dataset (see Appendix for additional example). The context (cosine similarity) and the BLEU score are calculated according to the actual captions.

Conclusion. In short, for the Flickr8k dataset with our merge encoder-decoder model architecture, the InceptionV3 CNN along with the 2-layer LSTM performed the best at our image captioning task as measured by the BLEU metric. Differences in performance were minimal, potentially due to too few training epochs or too large of a training batch size, so in order to come to a stronger conclusion, more research is necessary.

As I was responsible for the CNN and transfer learning portion of our project, I became very comfortable with reading and implementing code from the Keras documentation. I also learned to read papers and code about transfer learning as well as survey papers about varying image classification models. From those papers, I learned how to freeze training for specific layers while retraining others, and how to harness intermediate feature representations of at-scale models. Interwoven in all of this, I learned how to slowly adjust working preprocessing and model-defining code to fit my design, while ensuring that all tensor sizing was preserved.

One promising future direction is incorporating transformer mechanisms with attention. The model we describe has a fixed-length representation of the input image which can impose limitations on the model's ability to capture the most relevant parts of the image; the advantage of attention mechanisms is that they allow the model to focus on specific parts of the image as it generates each word in the caption. The attention mechanism works by calculating attention weights for each region of the image based on its relevance to the word currently being generated by the LSTM decoder. These weights are subsequently used to weight the features extracted by the CNN encoder, allowing the model to attend to the most relevant regions of the image [3]. This approach has had success with image captioning tasks, so given more time, would certainly be the first extension of this project.

Data availability. We used the Flickr8k captioned dataset, which is publicly available on Kaggle. It is originally from “Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics” by Hodosh, M., et al., a paper published in 2013 by researchers at the University of Illinois.

Contributions. The project was a collaborative effort with each individual making significant contributions. Roxane implemented the initial code of the base model and conducted a literature review on various CNNs such as InceptionV3, EfficientNetB0, and ConvNeXtTiny. She implemented the three CNNs she selected as image encoders and adjusted the decoder to correspond to the various necessary input sizes. As for report writing, she was responsible for writing the abstract, theory, background, and dataset description portion of the report, and contributed to the method portion on CNNs. Quyen reorganized the code into clear sections, simplifying it to contain only necessary code. Her main contribution was researching and implementing three versions of the RNN language decoders. She worked with Roxane on adding the testing dataset for validation to the data generator. Additionally, she wrote a function to calculate the overall accuracy of each model and a function generated captions for random images. Conway's contributions involved refactoring the preprocessing stage for legibility and simplicity, researching and implementing SpaCy NLP model for word embeddings, and implementing cosine similarity for word embeddings and BLEU score for final predicted captions. Lastly, all team members worked on the result section by running one to two models on their personal Google CoLab and reporting the results. The team also collaborated on creating slides for the presentation.

References

[1] Alok, Priyanshu. "Image Captioning using Keras." OpenGenus IQ. OpenGenus Foundation, accessed March 10, 2023, https://iq.opengenus.org/image-captioning-using-keras/.

[2] Brownlee, J. "How to Develop a Deep Learning Model to Automatically Describe Photographs in Python." Machine Learning Mastery, 2019, https://machinelearningmastery.com/caption-generation-inject-merge-architectures-encoder-decoder-model/.

[3] Chordia, S., Pawar, Y., Kulkarni, S., Toradmal, U., Suratkar, S. “Attention Is All You Need to Tell: Transformer-Based Image Captioning.” Advances in Distributed Computing and Machine Learning. Lecture Notes in Networks and Systems, vol 427, 2022. Springer, Singapore. https://doi.org/10.1007/978-981-19-1018-0_52

[4] Explosion AI. "SpaCy 101: Everything you need to know." spaCy Usage Documentation. Explosion AI, accessed March 10, 2023, https://spacy.io/usage/spacy-101#annotations.

[5] Hodosh, M., et al. "Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics." Journal of Artificial Intelligence Research, vol. 47, 2013, pp. 853–899, https://doi.org/10.1613/jair.3994.

[6] Keras Team. "Guide to Transfer Learning & Fine-Tuning." Keras API Documentation. Keras, accessed March 10, 2023, https://keras.io/guides/transfer_learning/.

[7] Liu, Zhuang et al. "A ConvNet for the 2020s." arXiv, 2022, doi: 10.48550/ARXIV.2201.03545, https://arxiv.org/abs/2201.03545.

[8] Szegedy, Christian et al. "Rethinking the Inception Architecture for Computer Vision." arXiv, 2015, doi: 10.48550/ARXIV.1512.00567, https://arxiv.org/abs/1512.00567.

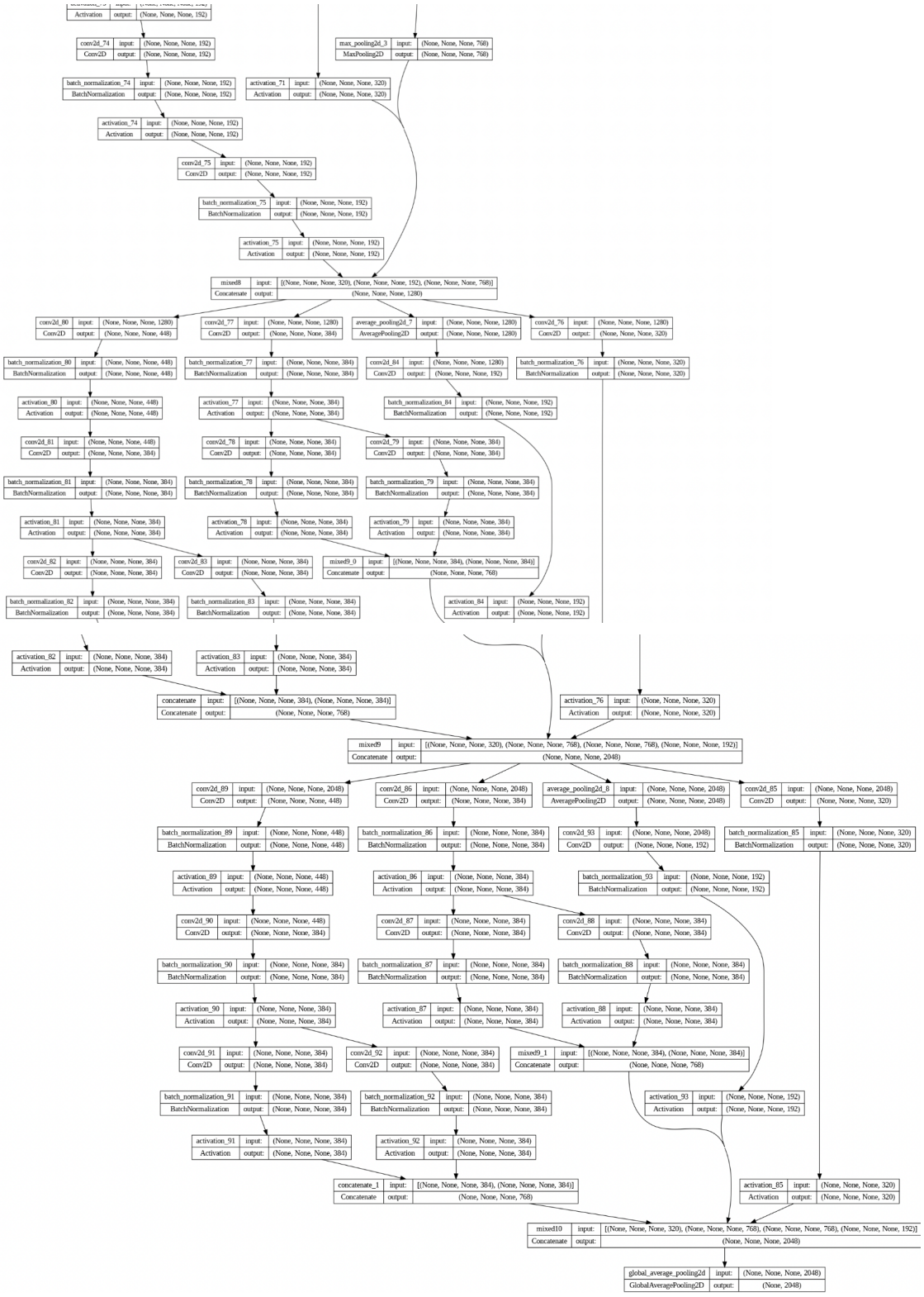
[9] Suresh, K.R., Jarapala, A., and Sudeep, P.V. "Image Captioning Encoder–Decoder Models Using CNN-RNN Architectures: A Comparative Study." Circuits Syst Signal Process, vol. 41, 2022, pp. 5719–5742, https://doi.org/10.1007/s00034-022-02050-2.

[10] Tan, Mingxing, and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." arXiv preprint arXiv:1905.11946 (2019), https://arxiv.org/abs/1905.11946.

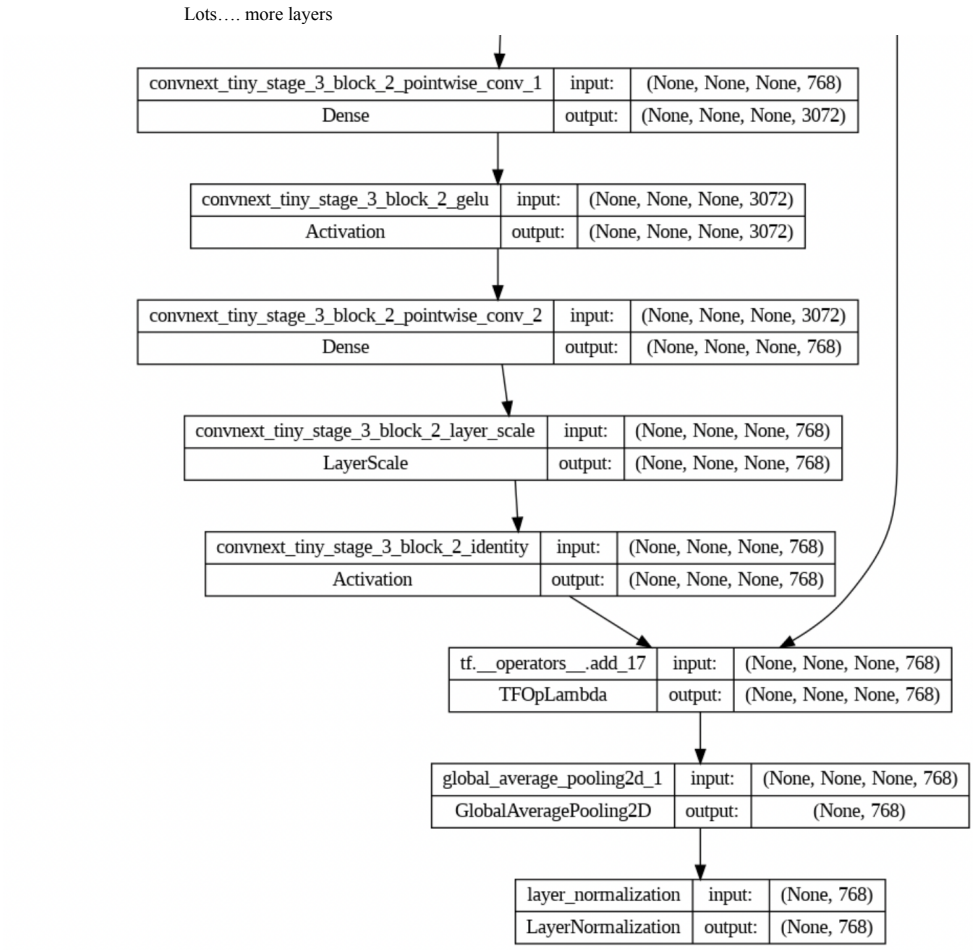
[11] Zhu, Haoran et al. "Generative Text-to-Image Synthesis via Word-to-Word Translation." arXiv preprint arXiv:1910.03940 (2019), https://arxiv.org/abs/1910.03940.

Appendix

InceptionV3 architecture is a very, very deep neural network. The entirety of its model diagram will not fit in this appendix, but the following is the top section of its layers, ending in the average pooling layer discussed in the Methods section.



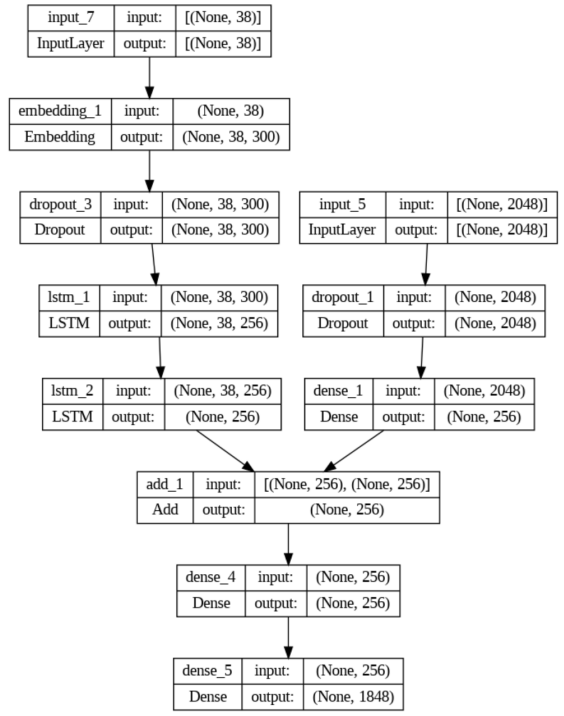
While less deep than InceptionV3, ConvNeXtTiny is still a very complex CNN. Once again the entirety of its model diagram will not fit in this appendix, but the following is the top section of its layers, ending in the average pooling layer and a final layer normalization.



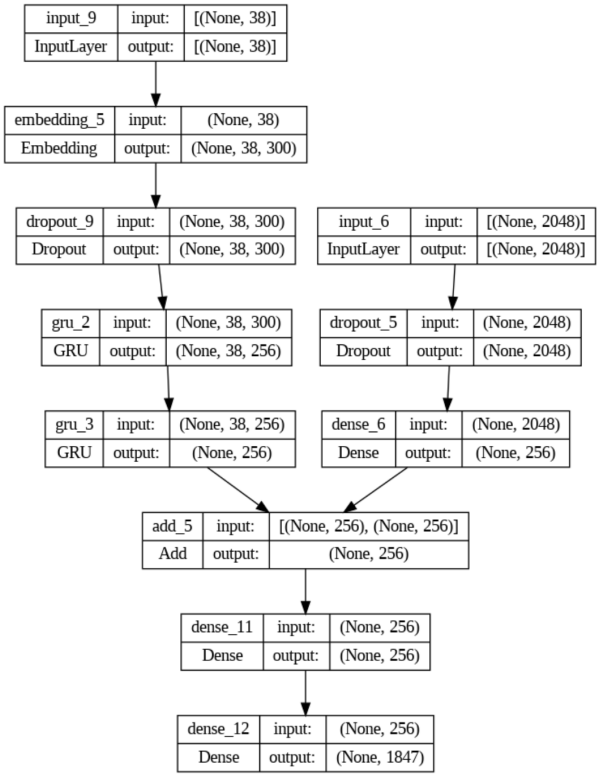
.

The diagram of EfficientNetB0 is not included, as the InceptionV3 and ConvNeXtTiny diagrams sufficiently demonstrate the depth and complexity of the pretrained CNNs.

RNN Model 2: 2-Layer LSTM



RNN Model 3: GRU



Each one of these model diagrams, as well as those included in the body of the paper is generated using the Keras plot_model function.

Figures 3 and 5 were generated by running an aggregation of the saved histories from training each one of the specified models, and plotting them all in a single matplotlib graph. The models were compiled using categorical cross, entropy loss and Adam as an optimizer. They were trained for 10 epochs with batch size of 32 and a steps-per-epoch of the typical size len(training_set)// batch_size, as specified in the code block below.

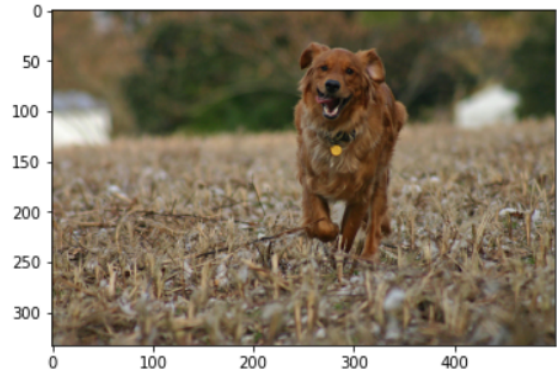
```
epochs = 10
number_pics_per_batch = 32
steps = len(train_captions)//number_pics_per_batch
val_steps = len(test_captions)//number_pics_per_batch

def train_model(model):
	model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['cosine_proximity'])

	generator = data_generator(train_captions, train_features, word_to_index,
max_length_caption, number_pics_per_batch)
	val_generator = data_generator(test_captions, test_features, word_to_index,
max_length_caption, number_pics_per_batch)
```

```
history = model.fit_generator(generator, steps_per_epoch=steps,
                validation_data=val_generator, validation_steps=val_steps,
                epochs=epochs, verbose=1)
```

Figures 4 and 6 were generated by average the BLEU score for 500 randomly selected images for each of the specified models and feeding the inputs to a matplotlib bar graph. Specifics about which cells to run to reproduce the plots can be found in the texts and comments of our code. Likewise, instructions on how to use the trained models to generate an image caption for a random image can be found in our code.

Below is an additional example of a successful run for image caption generation of a random test image.

```
Flicker8k_Dataset/69189650_6687da7280.jpg
```




```
Model 2 caption: a brown dog is running through a path path

Actual captions:
a brown dog is running through a brown field
a brown dog is running through the field
a brown dog with a collar runs in the dead grass with his tongue hanging out to the side
a brown dog with his tongue wagging as he runs through a field
a dog running in the grass


Accuracy for model 2: context =  0.8524840773457459 ,
                 BLEU score =  0.7259795291154771
```