NBA Image-to-Text Write-Up https://github.com/sebastianalgharaballi/NBAImagetoText

Introduction:

For my final project, I successfully deployed my own image-to-text deep learning model to successfully identify 15 input NBA players of my choice. I was inspired by the success of text-to-image models like DALL-E 2 and Stable Diffusion, and wondered if I could build a similar program that performs the opposite task of taking an image and turning it into a correct description. I began to dig deeper and see how I could potentially build an image-to-text system of my own. I found some project examples that used the Flickr8k dataset in order to train a model to label images correctly, but I wanted to do something unique and original for this project. Therefore, I combined my love for the NBA and growing passion for machine learning to create a dataset, model, and additional Python files that all successfully contributed to my end goal of predicting an NBA player based on an input player image. The problem I successfully solved was classifying over 1000 uncaptioned pictures of 15 different NBA players by running the data through many steps of code and printing output captions that the model produced, all in an effort to compare the results to the respective test images that I wished to caption.

Model:

In order to perform this task, I created a model that consisted of three parts: an encoder CNN, decoder RNN, and a mapping between the respective CNN and RNN. The encoder CNN is responsible for extracting useful information from the input image and encoding it into a compact representation. This involves passing the image through a series of convolutional and pooling layers, which extracts features at different scales and reduces the dimensionality of the

data. The output of the encoder CNN is a fixed-length feature vector $f \in R^d$, where "d" is the size of the feature vector. This is the vector that captures the most important information from the input image. Then, the decoder RNN is responsible for generating a natural language description of the image based on the encoded feature vector, which, for my project, is in the form of the name of the player featured in the input image. This involves passing the encoded feature vector through a series of recurrent layers, which use the information in the vector to generate a sequence of words that describes the image. The output of the decoder RNN can be represented mathematically as a sequence of words $y = (y1, y2, ..., yT)$, where "T" is the length of the generated sequence. Finally, the mapping between the encoder CNN and decoder RNN is used to connect the output of the CNN to the input of the RNN. This mapping involves a linear transformation that converts the fixed-length feature vector from the CNN into a sequence of vectors that can be fed into the RNN. This transformation can be represented mathematically as $h0 = Whf$, where $Wh \in R^{d\ x\ dh}$ is a weight matrix and "dh" is the size of the hidden state in the RNN. This allows the RNN to use the information from the CNN to generate a natural language description of the input image. All together, these three components form my complete image-to-text model. The encoder CNN extracts relevant information from the input image and encodes it into a compact representation "f", the decoder RNN uses this representation to generate a corresponding natural language description "y", and the mapping between the two allows the RNN to use the information from the CNN to generate an accurate description.

Datasets:

As previously mentioned, the majority of image-to-text models are trained with the Flickr8k dataset, a collection of 8000 images with a corresponding text file that pairs each image

ID with a description of that particular image[1]. I used a similar approach for my project, except with a self-made, smaller dataset composed of pictures of NBA players and an adjacent text file that houses each image ID and player name. The picture dataset includes 1140 different photos of 15 top NBA players in the league, and the captions text file is filled with my custom image IDs and each player those IDs correspond with. This dataset powers the overall training of my model and begins the encoder CNN process. Finally, I created a test images dataset, a file that contains 15 images (one of each player). These images were downloaded from Google and double checked to make sure that they were not included in the training dataset. This final dataset was used in my evaluation notebook, where I trained my model using my training dataset and, in parallel, compared the results after every epoch to the 15 input images in the test set. All in all, a single training point consisted of one player image and the corresponding player name in text, with the image serving as an input and the name serving as the desired output. There are 15 of these training points (one for each player in my dataset), which are presented to the model in order for it to go through the previously defined steps and eventually map the images with the correct description after training with an optimized learning rate and number of epochs.

Results:

I knew that a key area of evaluation for my project would be seeing how many correct answers my model gives when processing through the 15 photos. However, I needed more than just an eye test to determine the accuracy of my model. Two additional points of evaluation that I used were a cross-entropy loss function and an NLTK BLEU score. Cross-entropy loss is used in image-to-text models in consideration that the overall goal of the model is to predict a

[1] https://www.kaggle.com/datasets/adityajn105/flickr8k

probability distribution over a set of classes. This particular loss function is continuous and smooth, making it easy to optimize and perfect for classification tasks. It also penalizes confident predicted outputs that are incorrect. A BLEU score (Bilingual Evaluation Understudy) is a common metric used to evaluate performance in image-to-text models. It is calculated based on the precision of the predicted text, which is defined by the number of words (names, in my case) that also appear in the reference text, along with similarities in pronunciation, spelling, and syllabic features. This is a crucial metric since it provides a clear evaluation of the accuracy and completeness of generated captions from the model. A high BLEU score indicates that the model is generating captions that are accurate and closely match the reference captions. In general, the target number for a BLEU score is anything above 0.6[2].

Now that I had my three evaluation metrics finalized, I trained my model using a learning rate of 0.0003 and 100 epochs, which worked extremely well. The learning rate allowed my model to learn slowly and accurately, avoiding any signs of overfitting. This is evidenced by the model correctly identifying 12 out of 15 of my test images (80%). For my loss function, I utilized TensorBoard in order to visualize the changes in loss over the course of training and document the final loss value. That final value was recorded as 0.1379, which was much lower than the loss values achieved by the model on previous epochs. This was an excellent sign, because it meant that my model was learning over the course of training and minimizing error to its best capabilities[3]. Finally, after documenting the names that were produced by my model, my BLEU score was calculated as 0.6744, which is yet another excellent metric my model generated and follows with the goal of achieving a score greater than 0.6. While it was slightly surprising

[2] https://cloud.google.com/translate/automl/docs/evaluate
[3] https://tensorboard.dev/experiment/P8XyiqxtROOprDM0XGmXQQ/#scalars

that my BLEU score was not closer to 0.8, it makes sense when you consider that a BLEU score rates output labels based on the previously discussed linguistic similarities between those outputs and the correct captions. The three names that my model mislabeled are completely different from the correct names in terms of language. For example, my model predicted Anthony Davis as Stephen Curry, names that are not similar language-wise (besides being first and last names). This phenomenon makes a score of almost 0.7 even more remarkable, being that any incorrect output name is completely different from the wanted input name. Overall, my model provided outstanding results for my three evaluation methods.

Conclusion:

To conclude, my successful image-to-text NBA model demonstrates the potential of deep learning and machine learning techniques to accurately classify images and generate descriptive text. By training a model on a large dataset of NBA player images and reference captions, I was able to achieve high accuracy and produce output captions that matched the reference captions. This highlights the importance of using large, high-quality datasets in deep learning applications. I was able to achieve phenomenal performance after training my model on a comprehensive dataset of photos. Furthermore, this experiment also shows the potential of combining domain-specific knowledge with machine learning techniques. By using my passion for the NBA to create a unique dataset, model, and supporting Python files, I was able to produce a program that is tailored to a specific problem and domain. Overall, this experiment has shown that deep learning and machine learning can be counted on to solve complex processes like image classification. By using large, high-quality datasets and domain-specific knowledge, we can develop effective models that can provide valuable insights and solutions to real-world problems.