1.Give a summary of RNN and identify atleast one application , identify its topology and explain

RNN :

Recurrent neural networks (RNN) is widely utilized in speech recognition and natural language processing. Recurrent neural networks recognize the sequential properties of data and use patterns to forecast the following likely situation.

Deep learning and building models that imitate neuron activity in the human brain both use RNNs. They are particularly effective in applications where context is critical to predicting an outcome. They differ from other artificial neural networks in that they process a sequence of data that informs the final output using feedback loops. These feedback loops allow for the persistence of information. This effect is sometimes referred to.

HOW IT WORKS:

Artificial neural networks are interconnected data processing components that are loosely designed to function in the same way as the human brain. They are made up of layers of artificial neurons, or network nodes, that may process information and send it to other nodes in the network. The nodes are linked by edges or weights, which determine the strength of a signal and the network's overall output.

In some circumstances, artificial neural networks process information from input to output in a single direction. Convolutional neural networks, which underpin image recognition systems, are examples of "feed-forward" neural networks. RNNs, on the other hand, can be layered to process data in both directions.

RNNs can process data from the initial input to the final output, like feed-forward neural networks. RNNs use feedback loops, such as backpropagation via time, during the computational process to loop information back into the grid, unlike feed-forward neural networks. RNNs can process sequential and temporal data because of the connections made by this between inputs.

An RNN with a truncated input sequence limits the amount of time steps in the input sequence is known as a truncated backpropagation through a time neural network.

Applications of Recurrent Neural Networks:

- Prediction problems
- Machine Translation
- Speech Recognition
- Language Modelling and Generating Text
- Video Tagging
- Generating Image Descriptions
- Text Summarization
- Call Center Analysis
- Face detection,
- OCR Applications as Image Recognition
- Other applications also

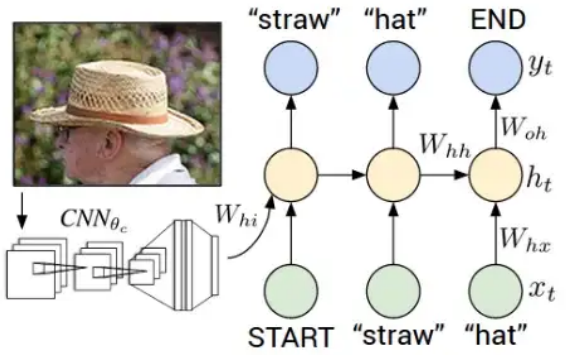
captioning images

The task of creating a textual description for an image is known as "image captioning," and it is a very intriguing job.

then this

The output will be a series or sequence of words from a single input, an image. Although the image in this case might be a constant size,

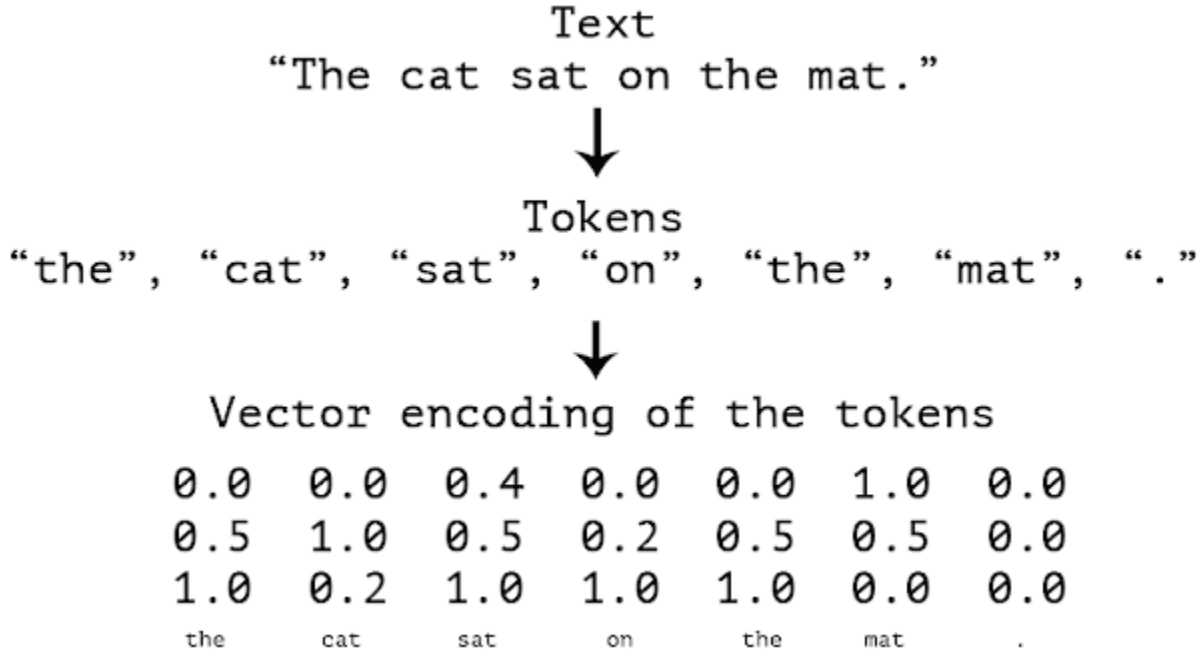
Network Topology



Recurrent Neural Networks (RNNs) in Computer Vision: Image Captioning

To do this, we need to use 2 different neural networks: a CNN and an RNN. Dataset

Pre-processing:The words are converted into tokens through a process of creating what are called word embeddings.



(source)

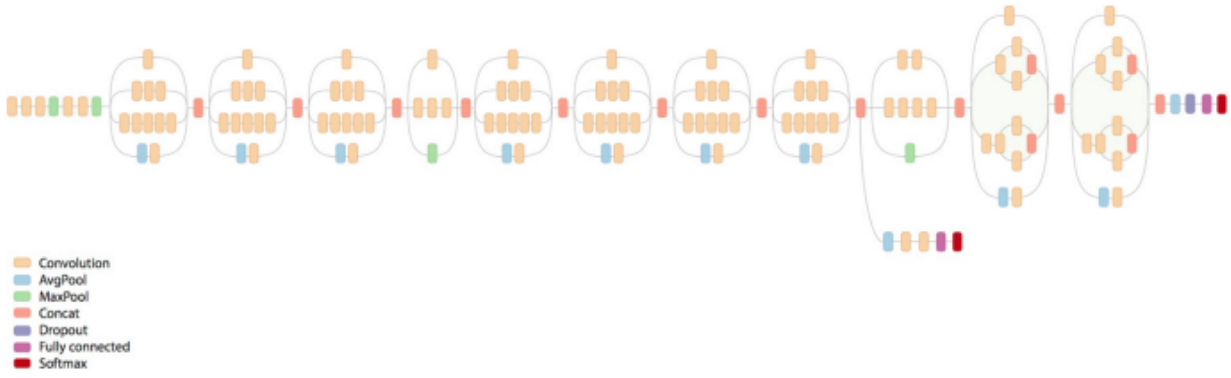
The process to convert an image into words/token is as follows:

- Take an image as an input and embed it
- Condition the RNN on that embedding
- Predict the next token given a START input token
- Use the predicted token as an input at the next time step
- Iterate until you predict an END token

TL;DR — We have images and sentences for each. Sentences are converted into vectors.

Encoder

The encoder is a convolutional neural network named Inception V3. This is a popular architecture for image classification.

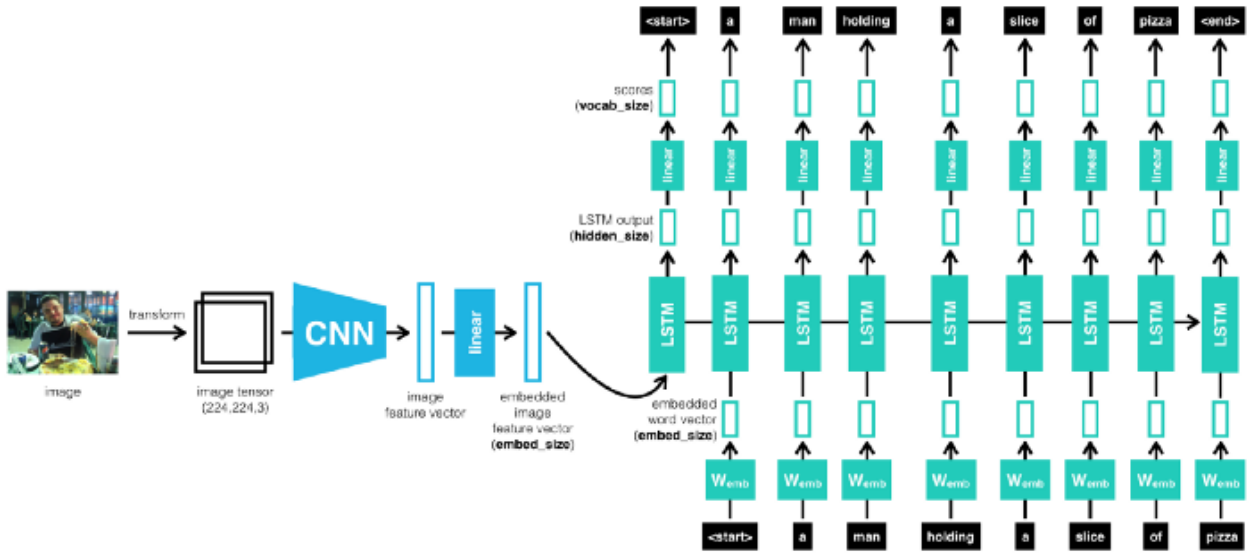


Inception v3 (source)

As you can see, the fully-connected layer is cropped with the inside function call. This means that we directly use the convolutional features and we don't activate them for a particular purpose (classification, regression, etc.).

Decoder

The decoder part of the model is using a recurrent neural networks and LSTM cells to generate the captions.



Essentially, the CNN output is adapted and fed to an RNN that learns to generate the words.

First, you might notice the vertical layers This is what a recurrent neural network produces. Every vertical layer is trying to predict the next word given the image.

The first layer will take the embedded image and predict “start”; then “man” is predicted, so the RNN will write “a man”; other tags are then generated, such as “pizza”.

We then learn how to say that a man holds a slice of pizza. The features are used and we try to correlate that with our captions.

In order to get a long-term memory, the RNN type is full of LSTM cells (Long Short-Term Memory) that can keep the state of a word. For example, `a man holding ___ beer` could be understood as `a man holding his beer` so the notion of masculinity is preserved here.

Finally, the horizontal layers are, like in deep learning, neural net layers. We could even stack more of these.

The decoder part first uses word embeddings

We first define a `Decoder` class and two placeholders. In TensorFlow, a placeholder is used to feed data into a model when training. We'll have one placeholder for image embedding and one for the sentences.

Then, we define our functions:

- `img_embed_to_bottleneck` will reduce the number of parameters.
- `img_embed_bottleneck_to_h0` will convert the previously retrieved image embedding into the initial LSTM cell
- `word_embed` will create a word embedding layer: the length of the vocabulary (all existing words)
- The next part creates an LSTM cell of a few hundred units
- Finally, the network must predict words. We call these predictions logits, and we thus need to convert the LSTM output into logits

- `token_logits_bottleneck` converts the LSTM to a logits bottleneck. That reduces the model complexity
- `token_logits` converts the bottleneck features into logits using a `Dense()` layer
- We can then condition our LSTM cell on the image embeddings placeholder.
- We embed all the tokens but the last
- Then, we create a dynamic RNN and calculate token logits for all the hidden states. We'll use this with the ground truth
- We create a loss mask that will take the value 1 for real tokens and 0 otherwise
- Finally, we compute a cross-entropy loss, generally used for classification. This loss is used to compare the `flat_ground_truth` to the `flat_token_logits` (prediction).