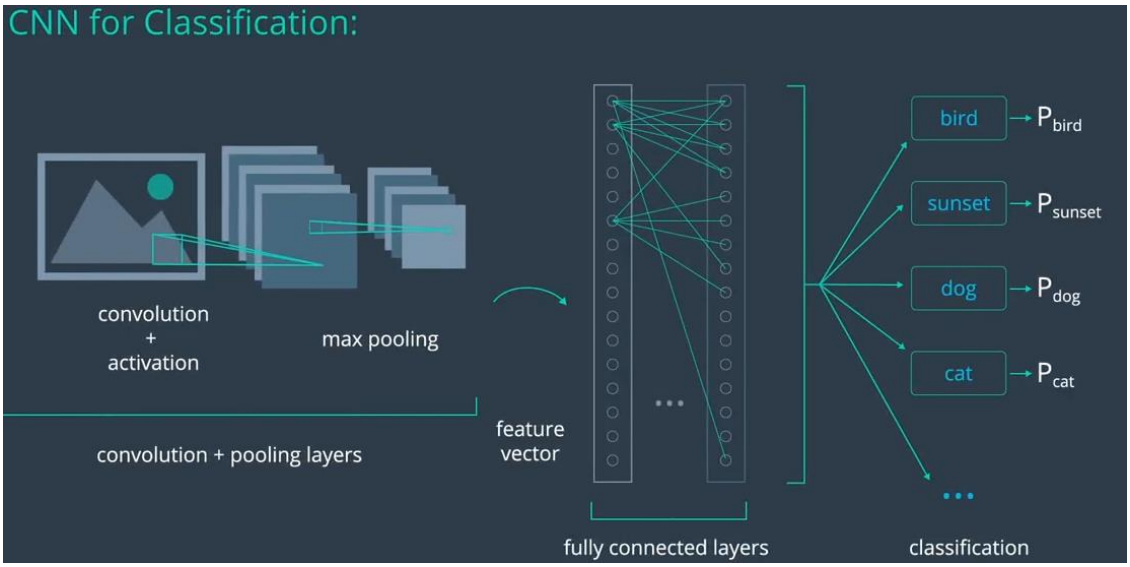
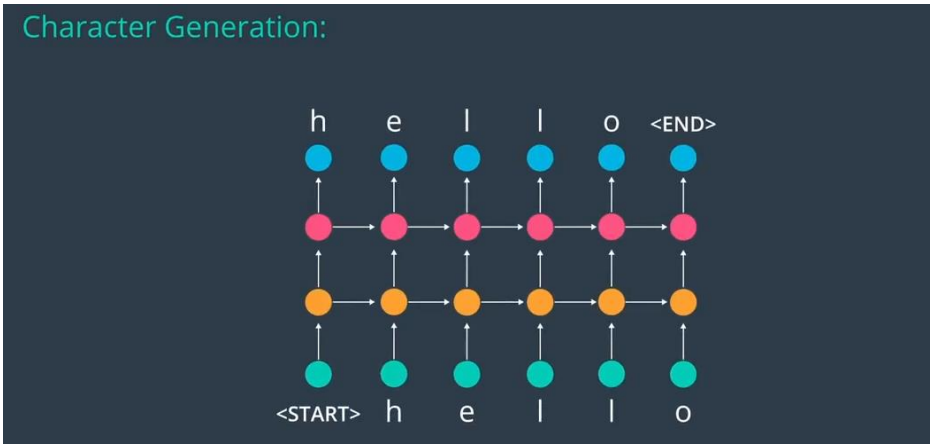
IMAGE CAPTIONING

We looked at Convolutional Neural Networks that are used for image classification and object Localizations.



And we looked at Recurrent Neural Networks mostly in the context of text generation.



You have seen how networks like LSTM can learn from sequential data, like a series of words of characters.

These network use hidden layers that over time link the output of one layer to the input of the next.

This creates a kind of memory loop that allows the network to learn from pervious information.

IMAGE CAPTIONING

In this section to create an automatic image captioning model that takes in an image as input output of sequence of text that describes the image.

Image captioning are used in a variety of application.

1- Image captions can be used to describes images to people who are blind or have low vision and who rely on sounds and texts to describes a scene. (In web development) it`s good practice to provide a description for any image that appears on the page so that an image can read or heard as opposed to just seen. This makes web content accessible.

Similarly, captions can be used to describe video in real time.

You can imagine a lot of use cases for which automatically generated captions will be really useful.

This lesson and the following project will be about creating a model that can be produce descriptive caption for an image.

Enabling everyone in the industry to create AI companies. For that, we need very capable Ai and image captioning is where we go from mere perception modules to ones with generative capabilities.

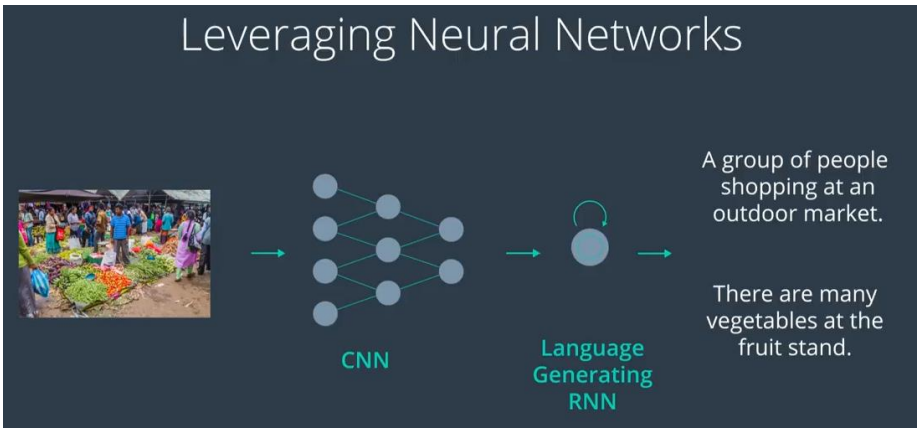
A Captioning model rely on two model main component,

1- **A CNN**
2- **RNN**

You have already learned about a variety of applications for convolutional and recurrent neural network separately, but captioning is all about merging the two to combine their most powerful attributes.

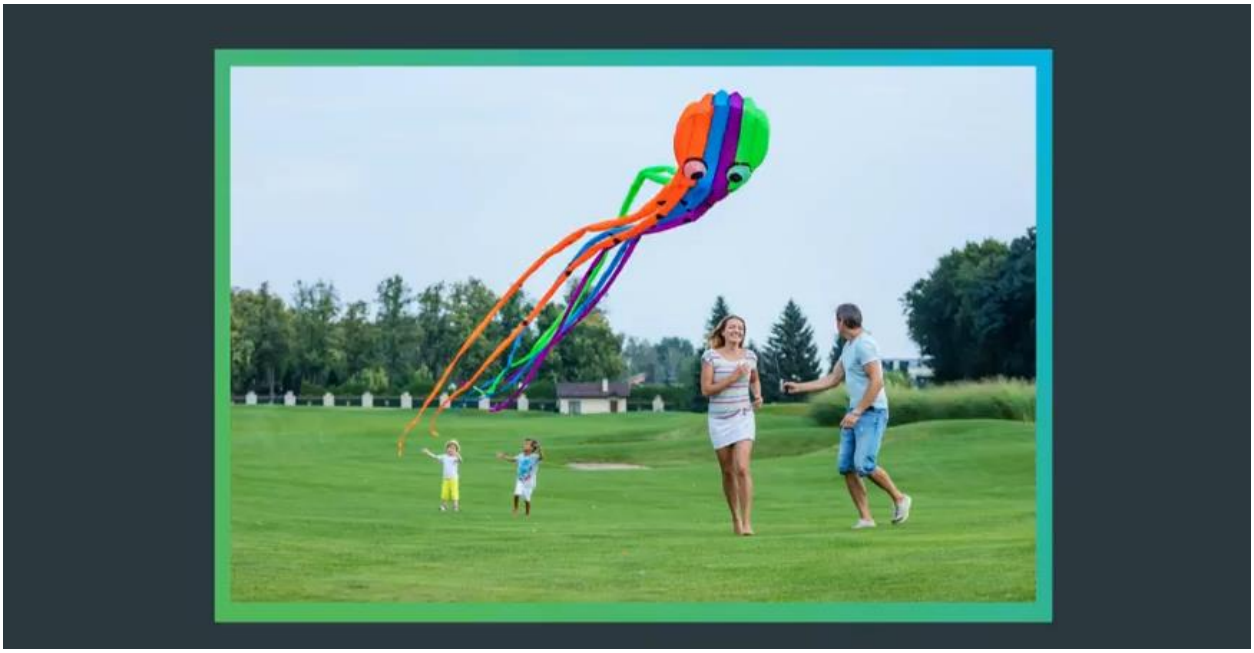
1- **CNN excel at preserving spatial information and images, and RNNs work well with any kind of sequential data, such as generating a sequence of words. So by merging the two you can get a model that can find patterns and images and then use that information to help to generate a description of those images.**

IMAGE CAPTIONING



First to know from image how to able to train, Your model will learn from a dataset composed of images, pair with captions that describe the content of the images.

Say you asking to write a caption that describe this image, How would you approach this task ?



1- First, **you might look at image and take notes of a bunch of different objects like different people and kites and the blue sky**.
2- **Then based on how those objects are placed in an image and their relationship to each other, you might think that these people are flying kites.**

They are in this big grassy area, so they may also be in a park.

After collecting these visual observations, you could put together a phrase that describe the image as,

“People flying kites in a park”.

You use a combination of spatial observation and sequential text descriptions to write a caption and this exactly the kind of flow we will aim to create in a captioning model that uses CNN and RNN architectures.

A common dataset that is used in training and captioning model is that COCO dataset.

COCO stands for common objects and contexts and it contains a large variety of images.

Each image has a set of about five associated captions, and you can see a few example of those captions here.

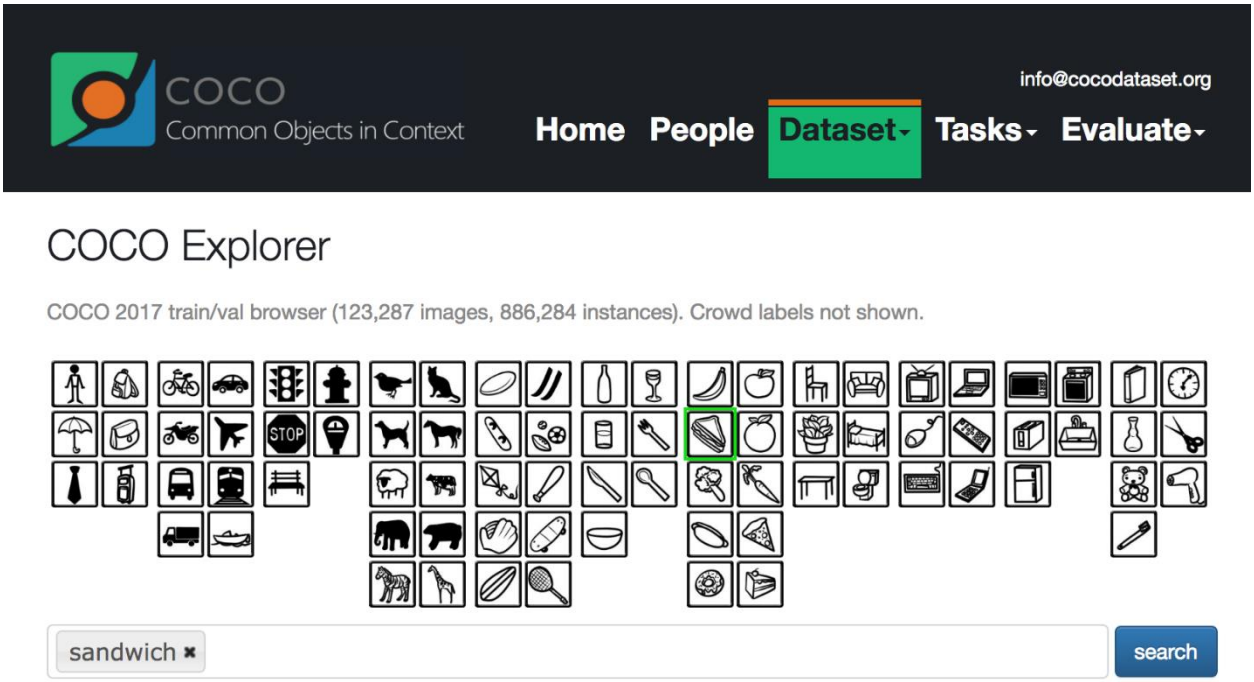
COCO Dataset

The COCO dataset is one of the largest, publicly available image datasets and it is meant to represent realistic scenes. What I mean by this is that COCO does not overly pre-process images, instead these images come in a variety of shapes with a variety of objects and environment/lighting conditions that closely represent what you might get if you compiled images from many different cameras around the world.

To explore the dataset, you can check out the dataset website.

Explore

Click on the explore tab and you should see a search bar that looks like the image below. Try selecting an object by it's icon and clicking search!



A sandwich is selected by icon.

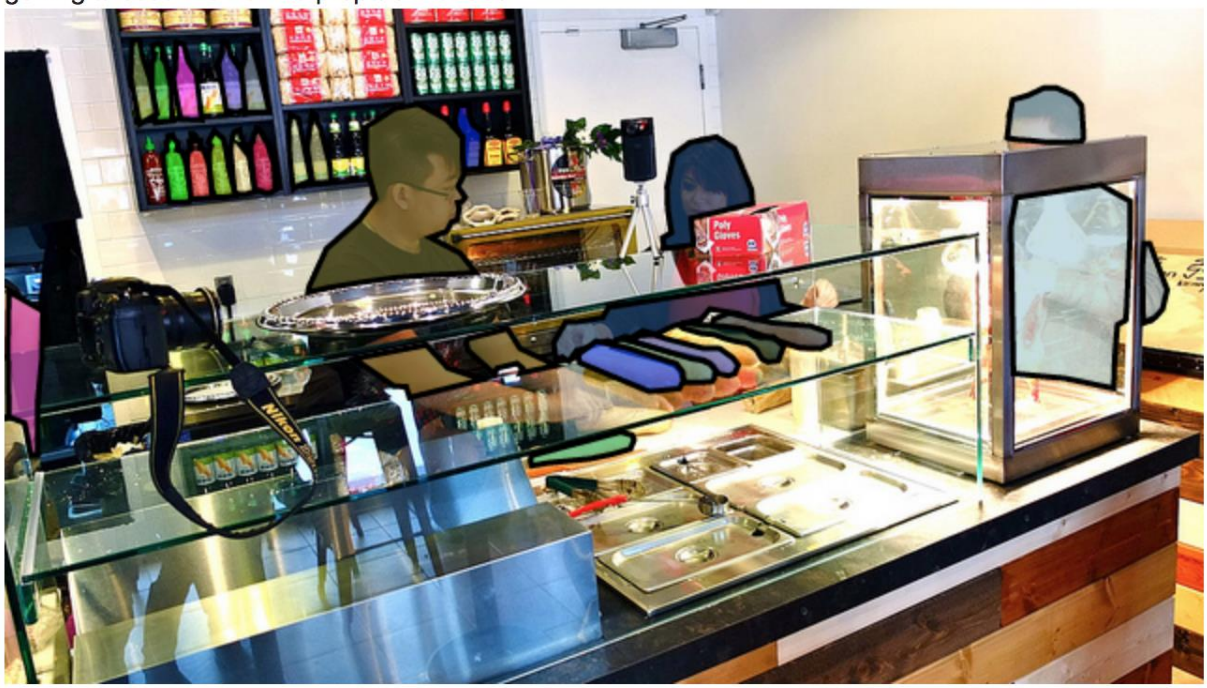
You can select or deselect multiple objects by clicking on their corresponding icon. Below are some examples for what a `sandwich` search turned up! You can see that the initial results show colored overlays over objects like sandwiches and people and the objects come in different sizes and orientations.

COCO sandwich detections

Captions

COCO is a richly labeled dataset; it comes with class labels, labels for segments of an image, *and* a set of captions for a given image. To see the captions for an image, select the text icon that is above the image in a toolbar. Click on the other options and see what the result is.

the counter of a restaurant with food displayed.
a store has their display of food with workers behind it
a few people that are out front of a cafe
a man prepares food behind a counter with two others.
getting a lesson to how to prepare the food.



Example captions for an image of people at a sandwich counter.

When we actually train our model to generate captions, we'll be using these images as input and sampling *one* caption from a set of captions for each image to train on.

IMAGE CAPTIONING

End-to-End we caption the model and image as input and output a text description of that image.

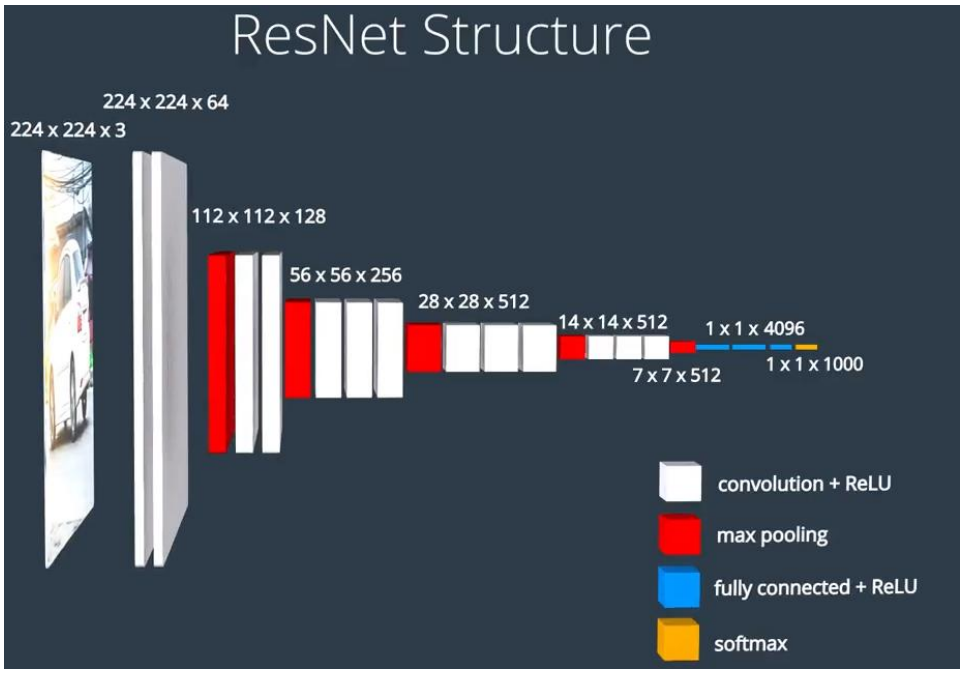
The input image will be processed by CNN and will connect the output of the CNN to the input of the RNN which will allow us to generate descriptive texts.

To see how this works, Let`s look at a particular example.

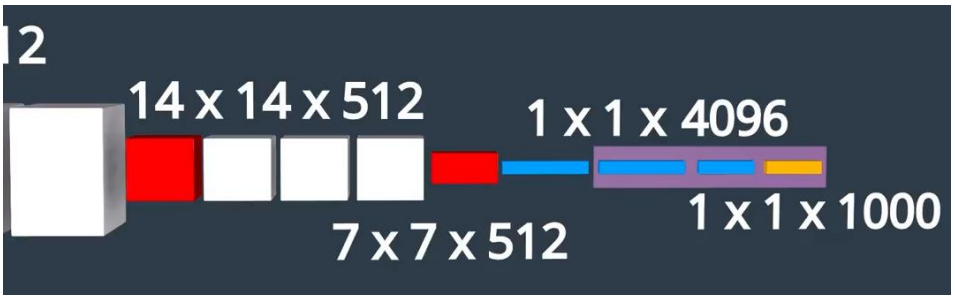
Say we have a training image and associated caption which is a man holding a slice of pizza.

Our goal is to train a network to generate this caption given the image as input.

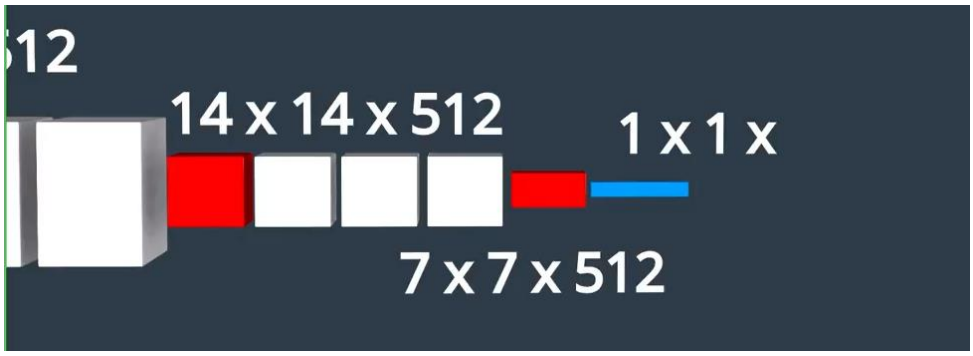
1- First, we feed this image into a CNN. **We will be using pre-trained network like VGG16 or Resnet.**



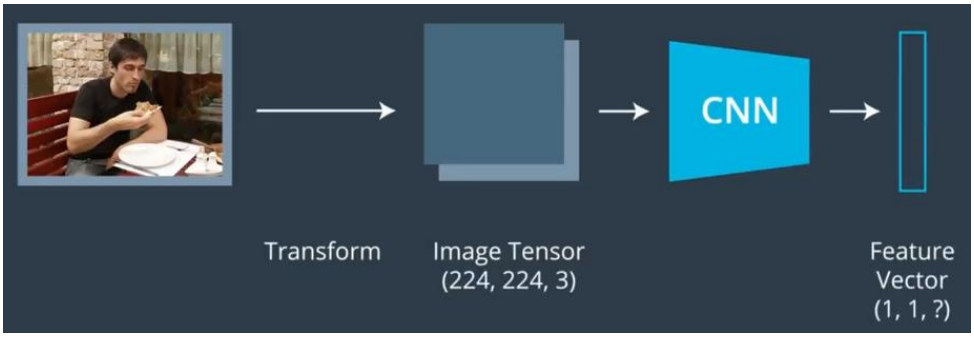
At the end of the networks, is SoftMax classifier that outputs a vector of class scores. But we don`t to classify the image, instead we want a set of features that represents the spatial content in the image.



To get kind of spatial information, we are going to remove the final fully connected layer that classifies the image and look at its earlier layer that distills the spatial information in the image.



Now, we are using the CNN as a feature extractor that compresses the huge amount of information contained in the original image into a smaller representation.



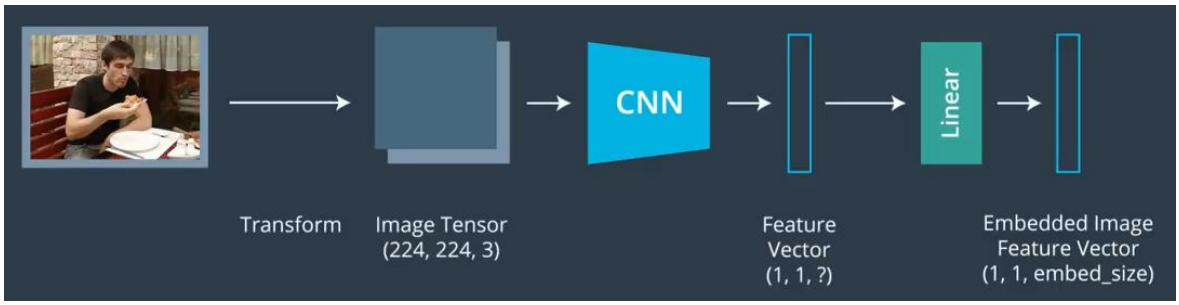
This CNN is often called the encoder because it encodes the content of the image into a smaller feature vector.

Then we can process this feature vector and use it as initial input to the following RNN.

IMAGE CAPTIONING

There are couple of way to engage CNN to RNN but in all cases the **feature vector** that`s extracted from the CNN will go through some processing steps, to be used as input to the first cell in the RNN.

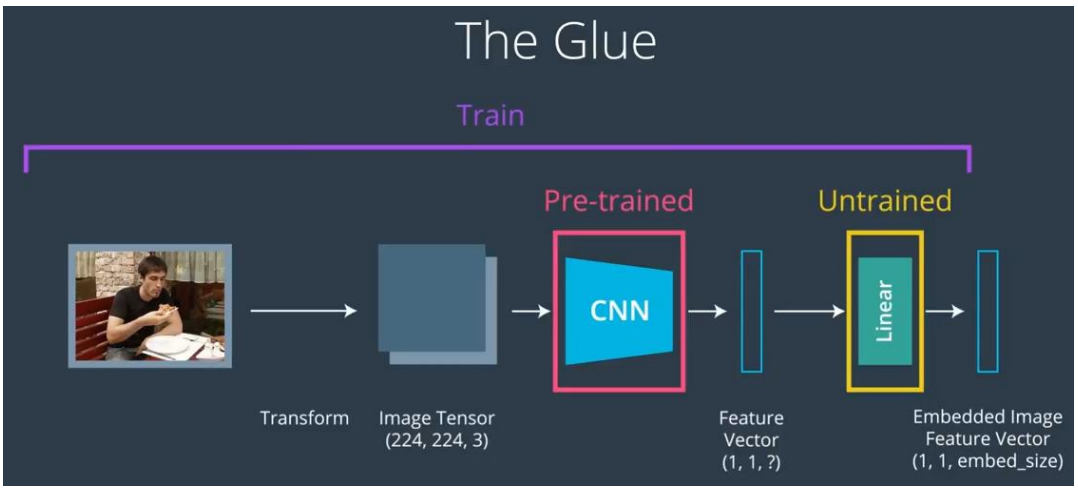
It can sometimes prove useful to parse a CNN output through an additional fully-connected or linear layer before using it as an input to the RNN.



This is similar to what you are seen in other transfer learning example.

The CNN we `re using in pretrain.

Adding an untrained linearly or at the end of it allow us to tweak only this final layer as we train the entire model to generate captions.



After we **extract a feature vector from** the CNN and process it, we can then use this as the initial input into our RNN and the job of the RNN is to decode the process feature vector and turn it into natural language. This portion of the network is often called decoder, and we will learn more about the caption Generation and training this decoder next.

IMAGE CAPTIONING

The RNN component the captioning network is trained on the captions in the COCO dataset.

We are aiming to train the RNN to predict the next word of a sentence based on previous words,

But, how exactly can it train on string data?

Neural network does not do well with strings. They need a well-defined numerical alpha to effectively perform back propagation and learn to produce similar output

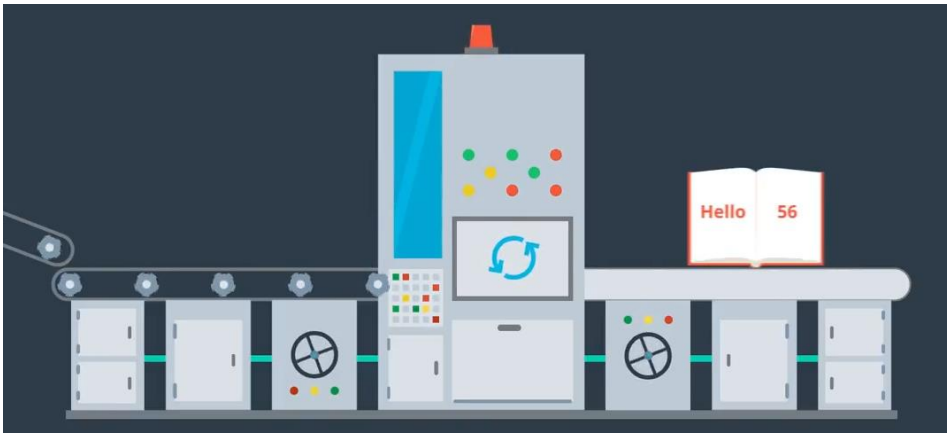
So, we have to transform the captioned associated with image into a list of tokenize words.

This tokenization turns any string into a list of integers.

So how does this tokenization work?

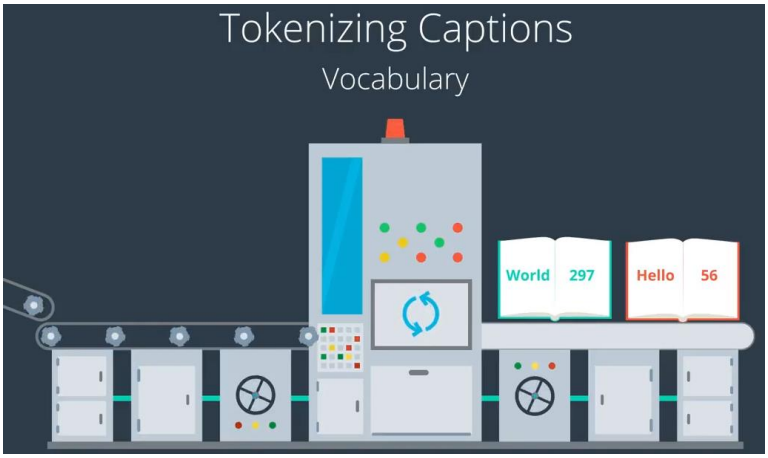
1- **First, we iterate through all of the training captions and create a dictionary that maps al unique words to a numerical index.**

2- **So, every word we come across, will have a corresponding integer value that we can find in this dictionary.**



The words in these dictionaries are referred to as our vocabulary.

The vocabulary typically also includes a few special tokens.



In this example, we will add two special tokens to our dictionary, a Start and End token that indicates the start and the end of a sentence.

<Start> <End>

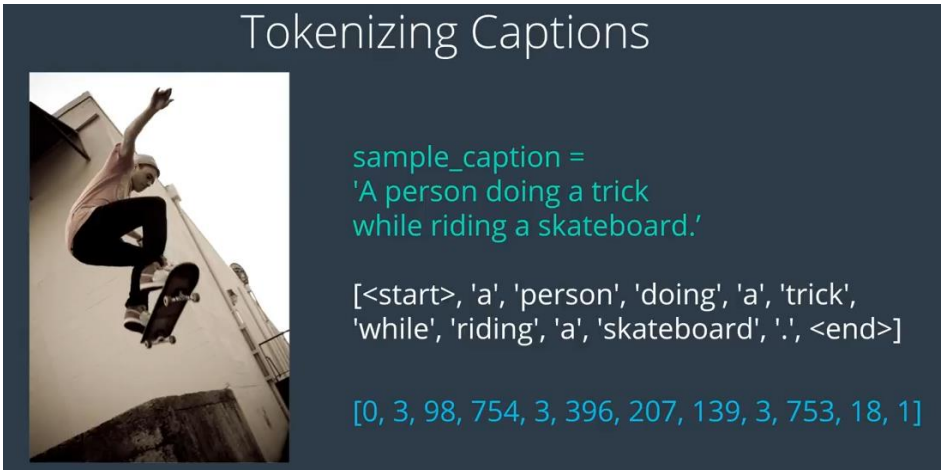
Now, the entire vocabulary is the length of the number of unique words in our training dataset plus two for the start and end tokens.

Let`s take a look at this sample caption.

'A person doing a trick on a rail while riding a skateboard.'

This caption is then converted into a list of tokens with the special start and the end tokens marking the beginning and end of the sentence.

This list of tokens is then turned into a list of integers which come from our dictionary that maps each distinct word in the vocabulary to an integer value.



This one step more before these words gets sent as input to an RNN and that`s the embedding layer, which transform each word in a caption into a vector of a desired consistent shape.

After this embedding step, we are finally to train an RNN that can predict the most likely next word in a sentence.

Tokenization

Token is fans transform similar

Usually, one that holds some meaning and is not typically split up any further. In case of natural language processing,

1- **Our tokens are usually individual words.**

So tokenization is simply splitting each sentence into a sequence of words.

dogs/are/the/best

The simplest way to do this is using the split method which returns a list of words.

Whitespace Tokenization

#split_text_into_token

```
Words = text.split()

Print(Words)
```

Note that it splits on whitespaces characters by default, which includes

1- **Regular space**
2- **Tabs**
3- **Newline**

It`s also smart about ignoring two or more whitespace characters in a sequence, so it doesn`t return blank strings.

But you can control all this using optional parameters.

```
the first time you see the second renaissance it may look boring  look at it at least twice and definitely watch part 2  it will change your view of the matrix  are the human people the ones who started the war  is ai a bad thing
```

→

```
['the', 'first', 'time', 'you', 'see', 'the', 'second', 'renaissance', 'it', 'may', 'look', 'boring', 'look', 'at', 'it', 'at', 'least', 'twice', 'and', 'definitely', 'watch', 'part', '2', 'it', 'will', 'change', 'your', 'view', 'of', 'the', 'matrix', 'are', 'the', 'human', 'people', 'the', 'ones', 'who', 'started', 'the', 'war', 'is', 'ai', 'a', 'bad', 'thing']
```

So far , we are only been using Python `s built-in functionality, but some of these operations are much easier to perform using library like NTLK , which stands for natural language toolkit.

The most common approach for splitting up texting NTLK is to use the word tokenized function from

`Nltk.tokenize`

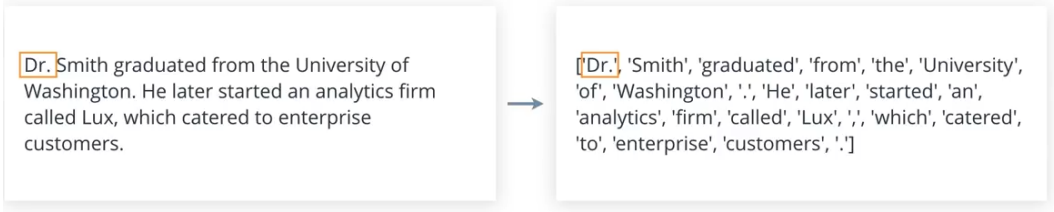
```
from nltk.tokenize import word_tokenize

# Split text into words using NLTK
words = word_tokenize(text)
print(words)
```

This perform the same tasks as split but is a little smarter Try passing in some raw text that has not been normalized.

You will notice that the punctuations are treated differently based on their position.

Here, the period after the title Doctor has been retained along with Dr as a single token.

As you can imagine, NTLK is using some rule or patterns to decide what to do with each punctuation.



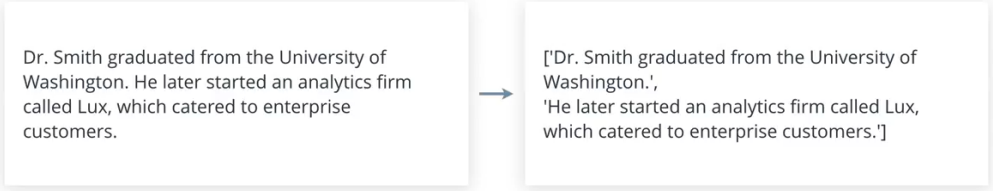
Sometimes, you may need to split text into sentence.

For instance, if you want to translate it.

You can achieve this with NTLK using sent tokenize.

Then you can split each sentence into words if needed.

```
from nltk.tokenize import sent_tokenize

# Split text into sentences
sentences = sent_tokenize(text)
print(sentences)
```



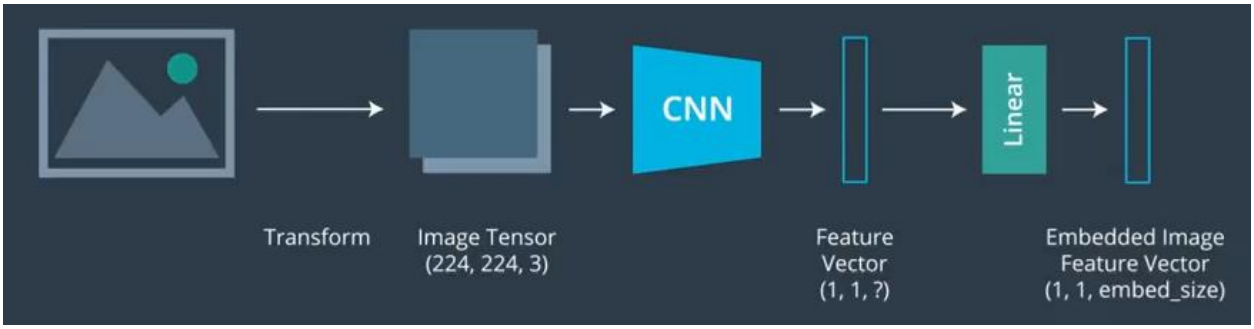
NTLK provides several other tokenizers, including a regular expression base tokenizer that you can use to remove punctuation and perform tokenization in a single step. And also, a tweet tokenizer that is aware of twitter handles, hash tags, and emoticons.

How to decoder given caption

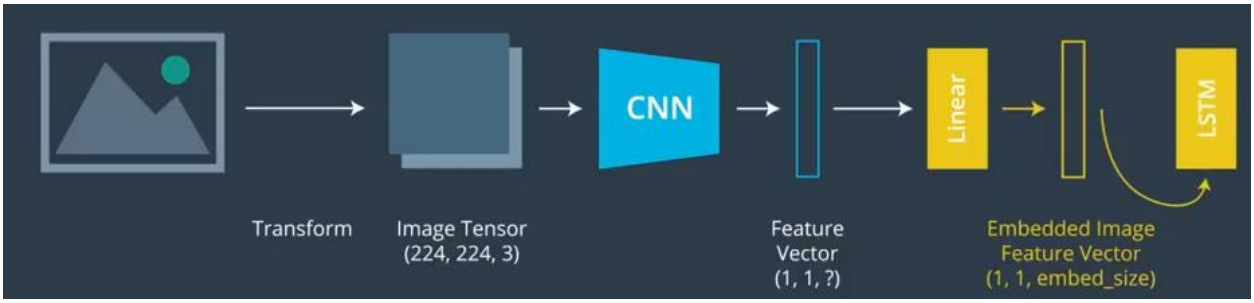
The decoder will be made of LSTM cells, which are good at remembering lengthy sequence of words.

Each LSTM cell is expecting to see the same shape of the input vector at each time-step.

The very first cell is connected to the output feature vector of the CNN encoder.



Previously, I mentioned an embedding layer that transformed each input word into a vector of a certain shape being feed as input to the RNN.



We need to apply the same transformation to the output feature vector of the CNN.

Once this feature vector is embedding into expected input shape, we can begin that RNN training process with **this as the first input.**

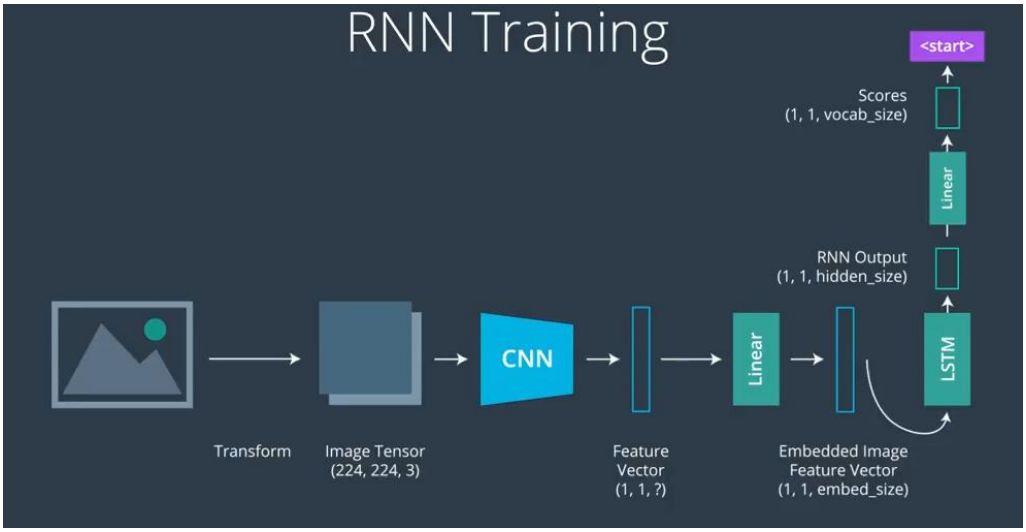
The input to the RNN for all future time steps, will be the individual words of the training caption.

So, at start of training, we have some input from our CNN, and LSTM cell with the initial state.

Now the R-net has two responsibilities.

1- **One, remember spatial information from the input feature vector.**
2- **And, two predict the next word**

We know that the first the very word it produces should always be the start token, and the next word should be those in the training caption.



For a caption, a man holding a slice of pizza.

We know that after the start token comes

1- A
2- Man
3- Holding
4- A
5- Slice
6- Of
7- Pizza

At every time step, we look at the current caption word as input and combine it with the hidden state of L-S-T-M cell to produce an output.

The output is then passed through a fully connected layer that produces a distribution that represents the most likely next word.

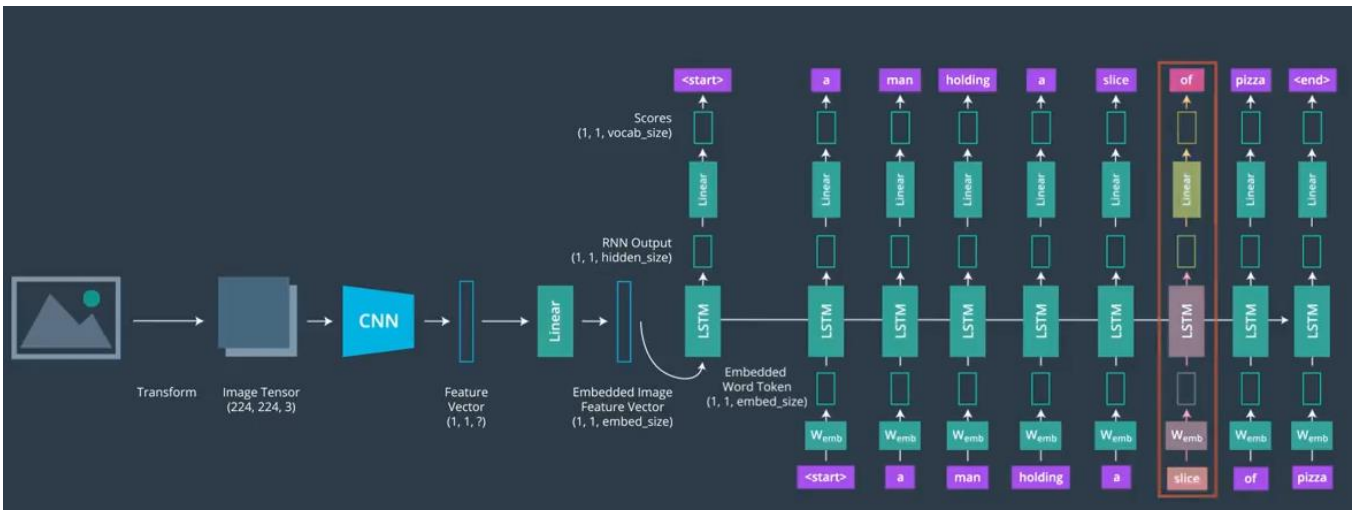


IMAGE CAPTIONING

This is like we have seen SoftMax apply to classification task. But in this case, it produces a **list of next words scores** instead of **a list of class scores**.

We feed the next word in the caption to the network, and so on, until we reach the end token.

The hidden state of an LSTM is a function of the input token to the LSTM, and previous state.

I will refer to this function as the **recurrence function**.

This **recurrence function** is defined by weights, and during the training process this model uses back propagation to update these weights until the L-S-T-M cells learn to produce the **correct next** word in the caption given the current input word.

As with most models, you can also take advantage of batching the training data.

The model updates its weights after each training batch with the batch size is, he numbers of image caption pairs sent through that network during a **single training step**.

Once the model is trained, it will have learned from many image caption pairs and should be able to generate captions for new image data.

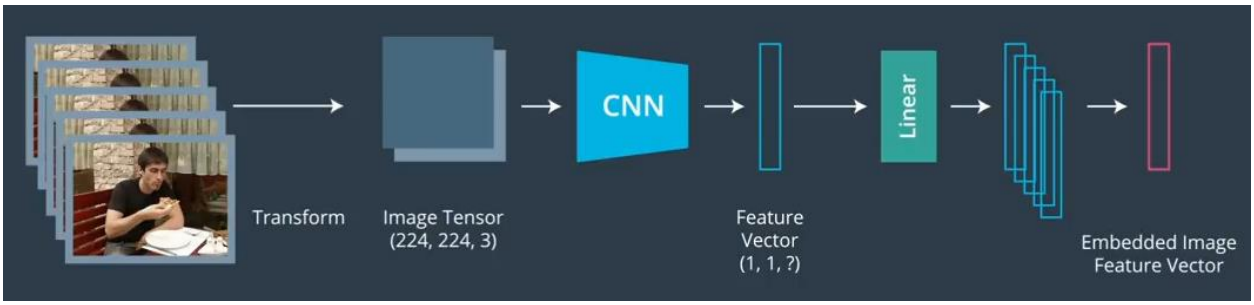
This caption network can also be video not just single images.

In case of video captioning, the only thing that has to change about this network architecture is feature extraction step that occurs between the CNN and the RNN.

The input to the pre-trained CNN will be a short video clip which can be through of as series of image frames.

For each image frame, the CNN will produce a feature vector but our RNN cannot accept multiple feature vector as input, so we have to merge all of these feature vectors into one that is representative of all image frames.

This produces a single average feature vector that represents the entire video clip.



Then we can proceed as usual, training the RNN, and then using single vector as the initial input and training on a set of tokenized video captions.

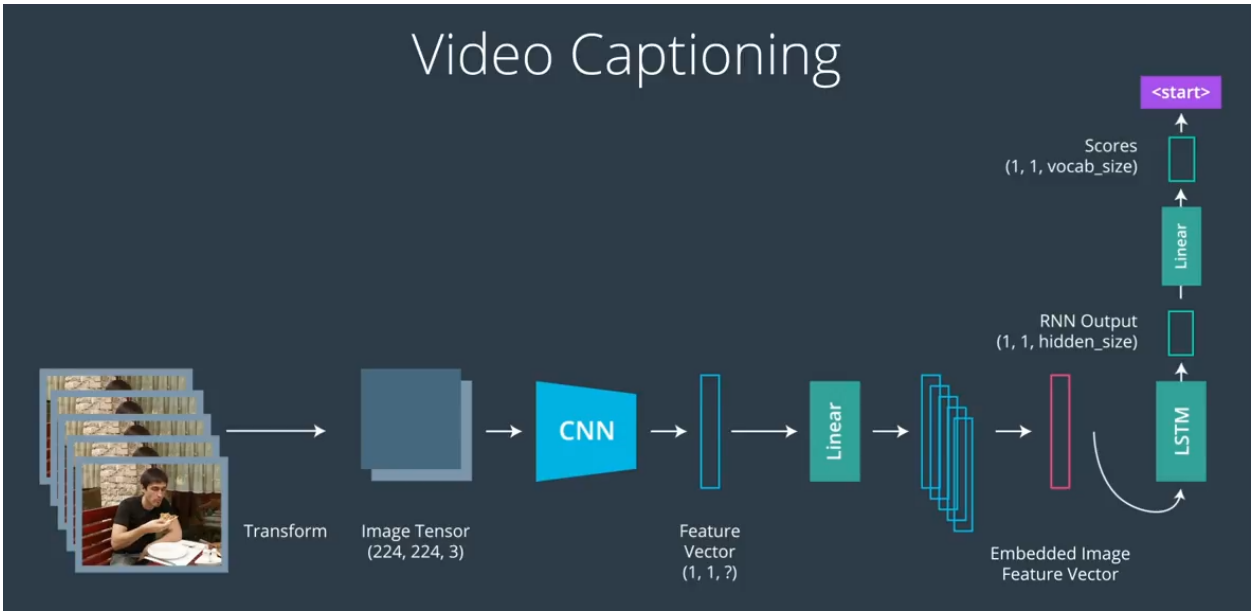


IMAGE CAPTIONING

Project !!!

Apply what you have learned to build and train your own captioning network.

You will be provided with some data pre-processing steps, and you main tasks will be about deciding how to train the RNN portion of the model.

1- **Keep in mind the type and shape of the input that each LSTM cell expect to see**
2- **And what shape the output takes,**
3- **And use the information to your decisions about optimization and hyperparameters.**