3 Important factors:

- **CLIP Architecture:**

Learning transferable visual models from natural language supervision and CLIP means Contrastive Language Image Pre-training. The trained model predicts which encoding of text and what text encoding corresponds with what encoding of what a visual encoding.

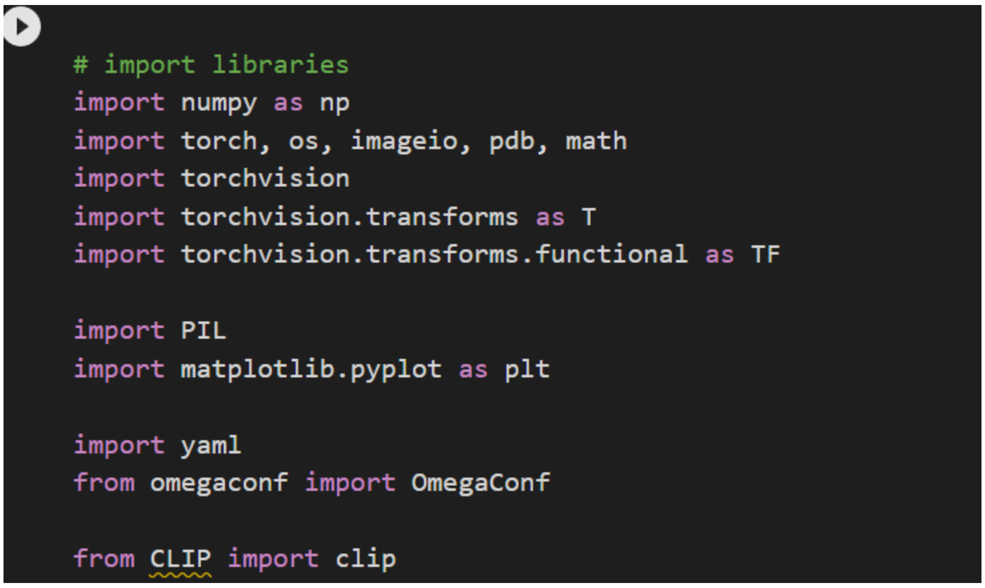
- **Taming Transformers:**

Type of generative architecture that can create and invent text to follow the text prompt with more and more text. This is a different kind of transformer that uses an architecture called VQGAN. It combines elements of convolutional architecture with GAN types of elements. It uses a codebook, works with patches. Creating the sequence of text elements is not very difficult but images have a very large no. of pixels which leads to the creation of long sequences in transformers.

- **Optimization Process:**

Take the text phrase and pass it through the CLIP architecture to encode it. And get that encoding in 512 numbers (encoding of the Architecture, understanding of CLIP architecture of that text). Do the same thing with the image, but instead of sending the image as it is, augment it, rotate it, move it or create crops of it (20, 30, 40).

Importing Libraries:



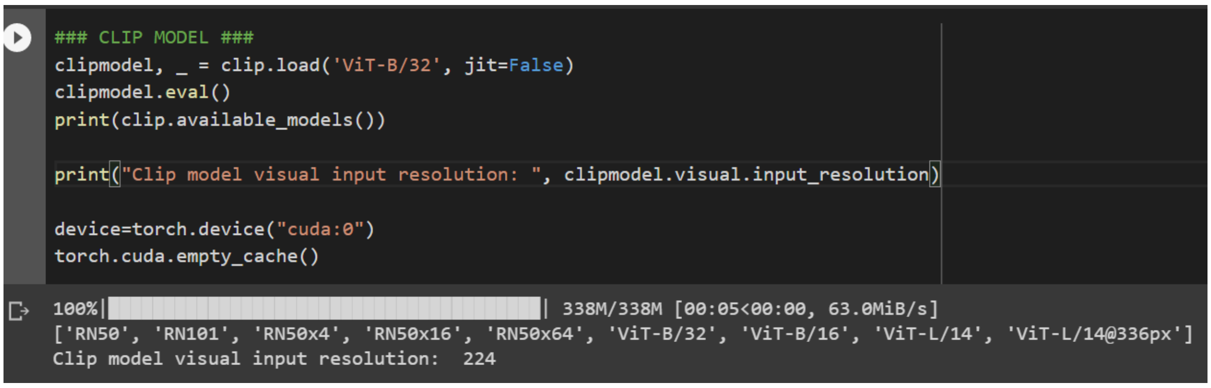
```python
# import libraries
import numpy as np
import torch, os, imageio, pdb, math
import torchvision
import torchvision.transforms as T
import torchvision.transforms.functional as TF

import PIL
import matplotlib.pyplot as plt

import yaml
from omegaconf import OmegaConf

from CLIP import clip
```

Here we are importing the all libraries which will come to use and to most essential libraries are clip and torchvision.transforms.

CLIP stands for Contrastive Language-Image Pre-training. CLIP (Contrastive Language-Image Pre-Training) is a neural network trained on a variety of (image, text) pairs. It can be instructed in natural language to predict the most relevant text snippet, given an image, without directly optimizing for the task.

The torchvision package consists of popular datasets, model architectures, and common image transformations for computer vision.

Selecting Model for CLIP:

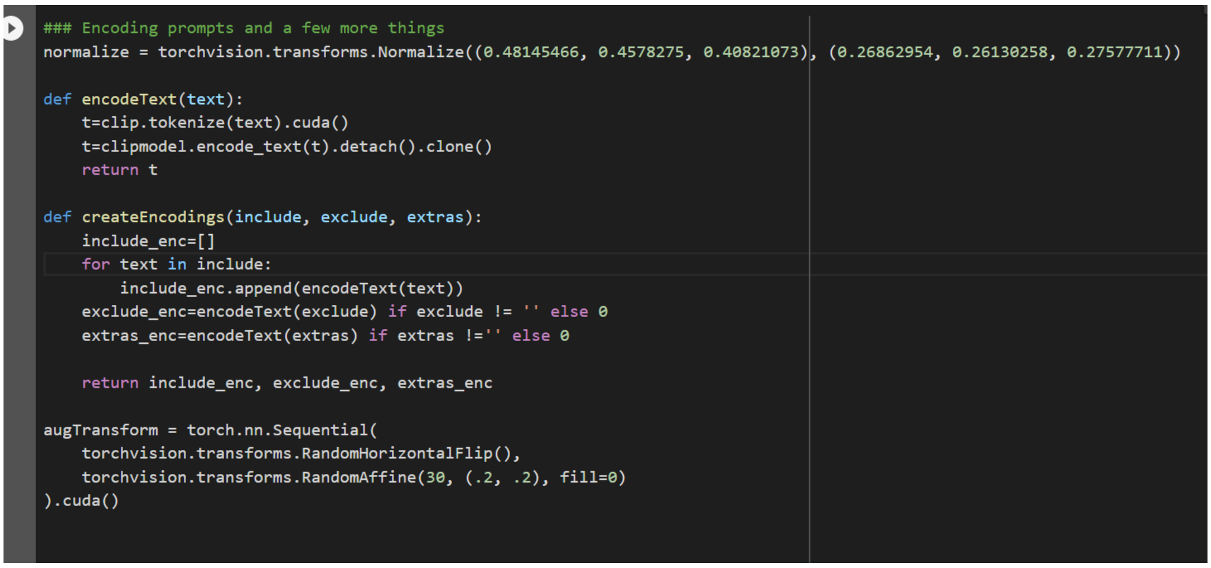


```
### CLIP MODEL ###
clipmodel, _ = clip.load('ViT-B/32', jit=False)
clipmodel.eval()
print(clip.available_models())

print("Clip model visual input resolution: ", clipmodel.visual.input_resolution)

device=torch.device("cuda:0")
torch.cuda.empty_cache()
```

```
100%|████████████████████████████████████████| 338M/338M [00:05<00:00, 63.0MiB/s]
['RN50', 'RN101', 'RN50x4', 'RN50x16', 'RN50x64', 'ViT-B/32', 'ViT-B/16', 'ViT-L/14', 'ViT-L/14@336px']
Clip model visual input resolution:  224
```

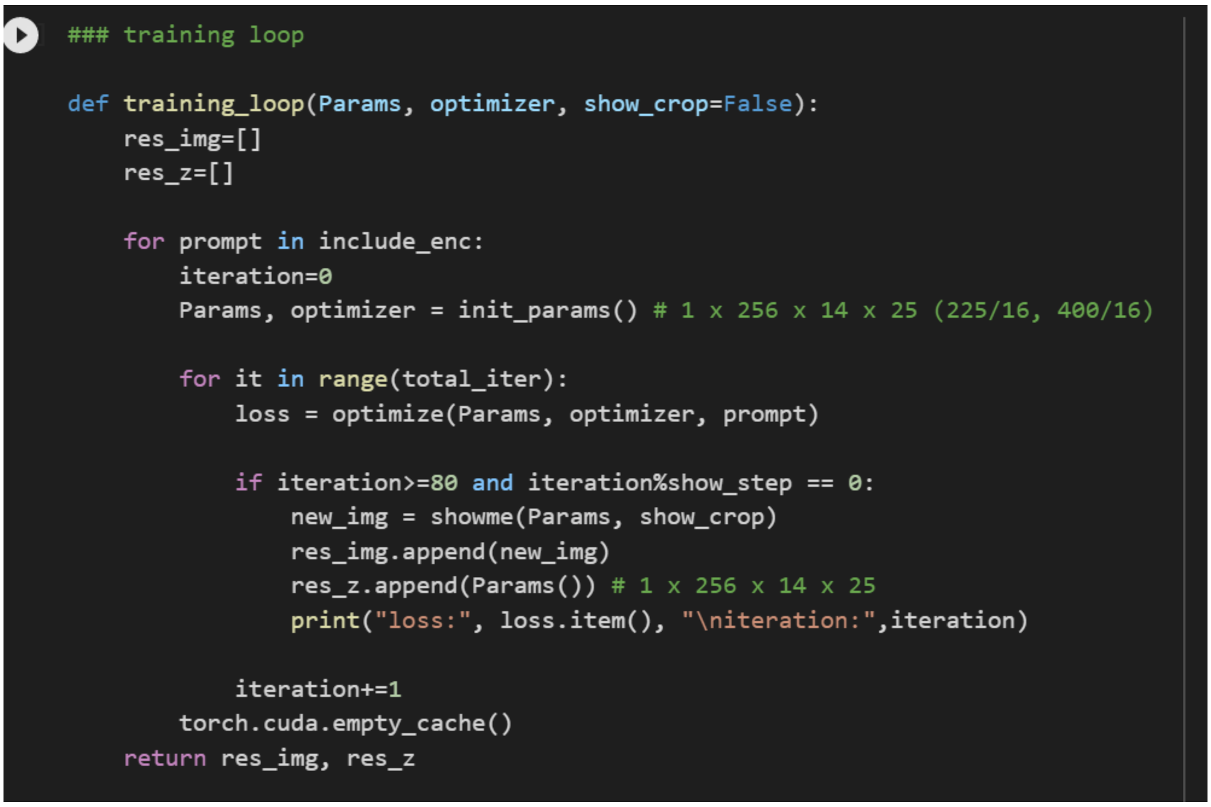
As you can see here we are loading the model of CLIP, there are different models but here we have choose ViT-B/32 because it is easy to use in term of working and takes less computational efforts.ViT-B-32 is used to map text and images to a shared vector space.

Encodings:



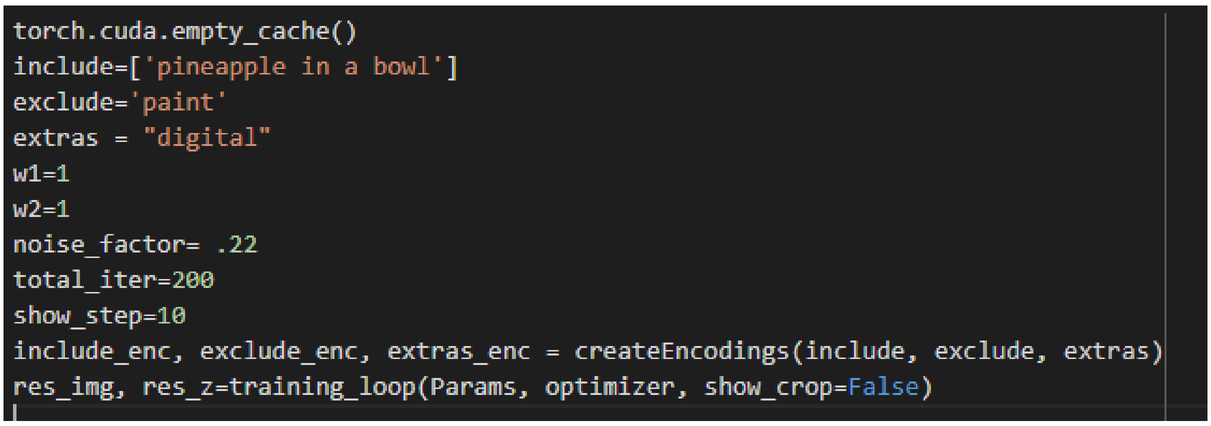
```
### Encoding prompts and a few more things
normalize = torchvision.transforms.Normalize((0.48145466, 0.4578275, 0.40821073), (0.26862954, 0.26130258, 0.27577711))

def encodeText(text):
    t=clip.tokenize(text).cuda()
    t=clipmodel.encode_text(t).detach().clone()
    return t

def createEncodings(include, exclude, extras):
    include_enc=[]
    for text in include:
        include_enc.append(encodeText(text))
    exclude_enc=encodeText(exclude) if exclude != '' else 0
    extras_enc=encodeText(extras) if extras !='' else 0

    return include_enc, exclude_enc, extras_enc

augTransform = torch.nn.Sequential(
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.RandomAffine(30, (.2, .2), fill=0)
).cuda()
```

Training the Model:

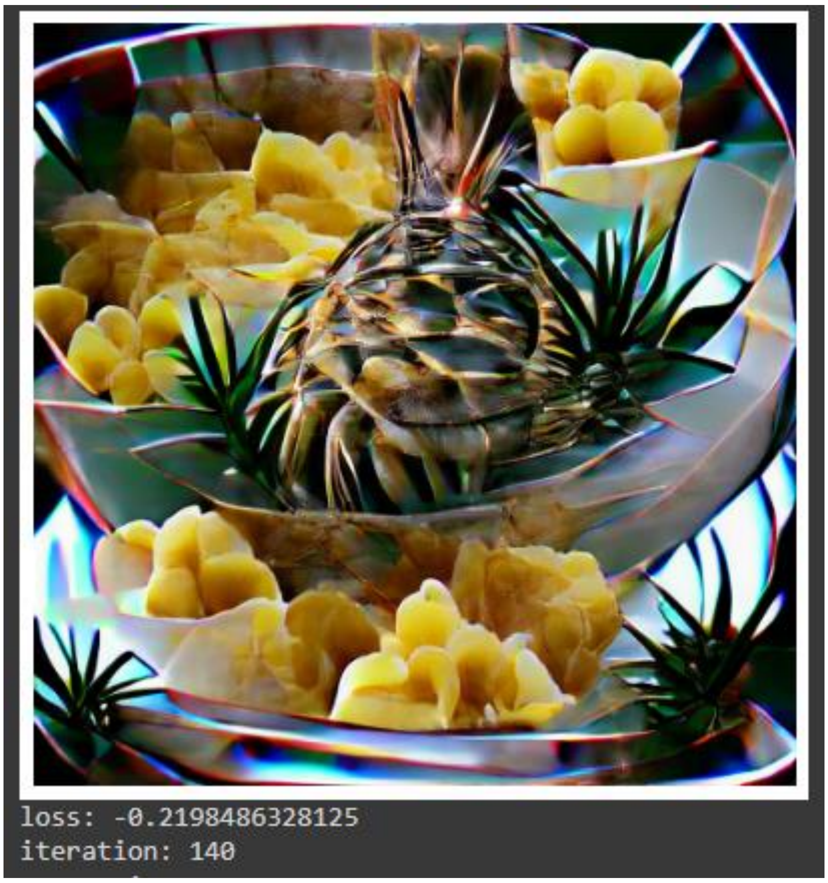


```
### training loop

def training_loop(Params, optimizer, show_crop=False):
    res_img=[]
    res_z=[]

    for prompt in include_enc:
        iteration=0
        Params, optimizer = init_params() # 1 x 256 x 14 x 25 (225/16, 400/16)

        for it in range(total_iter):
            loss = optimize(Params, optimizer, prompt)

            if iteration>=80 and iteration%show_step == 0:
                new_img = showme(Params, show_crop)
                res_img.append(new_img)
                res_z.append(Params()) # 1 x 256 x 14 x 25
                print("loss:", loss.item(), "\niteration:",iteration)

            iteration+=1
        torch.cuda.empty_cache()
    return res_img, res_z
```

Text Prompts:



```
torch.cuda.empty_cache()
include=['pineapple in a bowl']
exclude='paint'
extras = "digital"
w1=1
w2=1
noise_factor= .22
total_iter=200
show_step=10
include_enc, exclude_enc, extras_enc = createEncodings(include, exclude, extras)
res_img, res_z=training_loop(Params, optimizer, show_crop=False)
```



```
loss: -0.2198486328125
iteration: 140
```