# Reading Images:

1. Most common libraries PIL - Pillow, Matplotlib, Scikit-image, OpenCv
2. Sometimes Metadata of an image is important, **Ome tiff** formathas metadata. For e.g. How do we know if image is a satellite image and 1 pixel is 20km or image is a microscopic image and i pixel is 20nm

## PIllow:

1. Can be used for loading images and basic manipulation & processing, for advanced images OpenCV and Scikit-image can be used.
2. PIL.Image.open() is not an numpy
3. Functions / Modules learnt:
   1. .show()
   2. .format, .size, .mode
   3. .resize: can enlarge or reduce the size but does not maintain the aspect ratio
   4. .thumbnail: can only reduce the size, maintaining the aspect ratio
   5. np.asarray() - Converts to numpy array
   6. .crop((x,y,L,H)) - x,y indicate the starting coordinate and L,H indicate the length and height
   7. img.copy() creates a copy & Img.paste(copied\_img, (x,y)) pastes copied\_img on img starting at x,y
   8. .rotate(angle, expand = True), expand maintains the whole image

## Matplotlib:

1. Not a processing library, a plotting library for python. It can also help in visualizing images
2. Module Pyplot is a very useful plotting module.
3. Plots similar to MATLAB
4. Imports image as numpy array
5. Functions / Modules used
   1. imread()
   2. imshow()
   3. colorbar()

## Scikit-image

1. Image processing library
2. Used for Image segmentation, Colour space manipulation, Geometric segmentation, Analysis, Filtering, Feature extraction.
3. Useful for traditional ML tasks.
4. Imports image as numpy array
5. Images can be imported as float or int8 using modules
6. Functions / Modules learnt
   1. Io.imread
   2. Img\_as\_float, img\_as\_ubyte: img\_as\_float helps in keeping the range of pixels between 0 & 1 whereas astype(float) converts int number to float

## OpenCV:

1. Library of functions which are useful for computer vision and advanced image processing i.e. images and videos. For e.g. for Object detection, Facial recognition, OCR, Image segmentation, ANN etc.
2. By default, CV2 imports and handles images as BGR
3. Functions used, modules learnt
   1. cv2.cvtColor(image\_variable, converting function (e.g. BGR2RGB)
   2. cv2.imread(path, flag)
   3. cv2.waitKey() - Time for which image is displayed. 0 - showed until keyboard key is pressed

Key points:

1. Converting image pixels from int to float helps in maintaining the precision of image during transformations.
2. Sometimes Metadata of an image is important, **Ome tiff** formathave metadata. For e.g. How do we know if image is a satellite image and 1 pixel is 20 km or image is a microscopic image and i pixel is 20nm

Popular Deep Learning Architectures:

1. **AlexNet** (2012)
2. **CNNs**
3. **VGG** (2014)
4. **GoogleNet / Inception** (2014)
5. **ResNet, UNet** (2015)
6. **Xception** (2016)
7. **YOLO** (2016)
8. **EfficientNet** (2019)
9. Xception, MobileNet, DenseNet, EfficientNet, CapsNet, Attention Mechanism

Image Segmentation:

# Instance and Semantic Segmentation:

Semantic Segmentation: All objects in a particular class have same colour

Semantic Segmentation: All objects in a particular class have different colour

## Threshold segmentation:

We can classify images into background and object by using a single threshold or set multiple thresholds for multiple objects.

## 

# Entropy based sementation

1. Entropy measures the amound of randomness in the neighbourhood, it is calculated as entropy = -sum(p \* log2(p)), p is the probability of each intensity value in the image or a neighborhood window.

2. disk(r) determines neighbourood, such that randius of the pixel is less than distance of other pixel from the origin

3. threshold\_otsu considers image to have 2 elements background & foreground(object), it determines the threshold to seperate this using otsu method

4. Threshold\_Otsu maximises the binary class variance to determine the threshold i.e.

a. Initialise threshold as 0

b. For each threshold value, divide the histogram into two classes: pixels with intensity values below the threshold (background) and pixels with intensity values above or equal to the threshold (foreground).

c. Calculate the probabilities of each class by summing the normalized histogram values within each class.

d. Compute the means of each class by averaging the intensity values weighted by the corresponding probabilities.

e. Calculate the between-class variance using the following formula:

between-class variance = class1\_probability \* class2\_probability \* (class1\_mean - class2\_mean)^2

# Histogram based segmentation:

Gray level ranges - segmented  
After small amount of preprocessing, segmentation can be done

GANs and Diffusion:

1. Converting a noise to an image in a single step is difficult and thus GANs are not very appropriate so stable diffusion kind of iteratively removes noise from the image by passing it again and again through the model.
2. If we add the same noise again and again, it's a linear schedule. You can ramp up the amount of noise in each step, there are various methods. This method is schedule
3. We iterate by determining an estimate of image at t0 and noise at tn & then put tn-1 noise to estimate and again send back to model and get a new estimate t0 then we add tn-2 noise and repeat the process till noise t1 is put..
4. We embed our statement in GPT style transformer embedding & using classifier free guidance (i.e. we amplify the difference between noise predicted by text embedded input and without text embedding input) we improve our model.

# GANs - Pix2Pix:

1. O/p - 256 x 256 coloured images
2. 4 models : 2 Generator, 2 Discriminator
3. **Discriminator** :
   1. Image classifier to predict whether O/P image is real or fake
   2. PatchGan : 70 x 70 patches of input image
   3. Convolution - BatchNorm - LeakyRelU is used
   4. Instance Normalisation is used instead of Batch Normalisation
   5. Discriminator is updated slower than generator
   6. Loss is weighted 0.5 times so that model has half the effect on update to slow down the update
   7. We use 0.5 as Beta\_1 for ADAM. It tells about weightage to previous 10 gradients and the recent gradient for the update of weight. When 0.5, we give equal importance to both. it slows down the path to convergence i.e. these are decay rates at the end of each batch. Adam internally stores moving averages.
4. **Generator:**
   1. Encoder - Decoder architecture
   2. Image is first downsampled or encoded down, then resnet layers with skip connections are used to interpret these and then upsampled to original image size.
   3. The Resnet layer has 2 - 3x3 CNN layers. Output is then concatenated to input - channelwise. Reflection padding is used. Same padding results in 0 output and black pixels on the border
   4. For 256 x 256 images, 9 resnet blocks are used and for 128 x 128, 6 resnet blocks are used.
   5. Pixel values are in range [-1,1]
5. **Training:**
   1. **Discriminator** models are trained on real and generated images by labeling the images & generators are trained on the output of the discriminator.
   2. **Generator:**
      1. Generators are updated to minimize the loss for predictions where images are marked as real. I.e. They trained to generate images to better fit the target domain - Adversarial - MSE
      2. Cycle loss is also used to update generators i.e. when both generators are used, how effectively an image is regenerated. (Forward & Backward cycle loss) - MAE
      3. Identity loss is also used to update generators i.e. when an image from target domain is passed, it should not alter it - MAE
      4. I.e. 4 losses and 4 outputs are used to update the generator with the help of a composite model built only to train the generator.
      5. Composite model takes the generator to be trained, its corresponding discriminator, and other generators (trainable = False).
      6. **For Adversarial loss**, Discriminator is connected to composite model, Image from Target domain is used as input to composite model and generator is expected to return it unaltered - **Identity mapping**,

**Forward loss** - Generator is connected to another generator to reconstruct the source domain

**Backward loss -** Image from target domain, used for identity mapping is passed also from another generator and then from considered generator, to check the reconstructed version

**Summarise:** Composite model has 2 inputs from real images i.e. domain A and domain B & 4 outputs - identity mapping, forward cycle, backward cycle, discriminator output

**Weights for losses:** Cycle loss is 10 x Adversarial loss, And identity loss is always half the cycle loss.

* 1. **GANs** do not converge, there is equilibrium between generator and discriminator convergence

# UCV GAN:

Boosts cycleGAN performance by

1. Including a Visual Transformer (VIT) with generator i.e. (UNet with VIT bottleneck)
2. Change in Training : Instead of training with random weights, we pre train generators in self supervised way
3. Regularization of discriminator: Add Vanilla gradient discriminator with a gradient penalty term

# CGAN & cDCGAN:

CGAN or Conditional GAN is conditional generation of images based on class label.

GANs are usually trained on zero sum or adversarial manner

Labels can

1. Improve GAN: Stable training, faster training and improved image quality
2. Target generation: Generating images of a particular target

Best way to incorporate class labels is using an embedded layer followed by a fully connected layer with a linear activation that scales embedding to size of image & then concatenating it into image as an additional channel.

DCGAN (Deep Convolutional GAN) is an advancement of GAN which outlines training procedures that reliably result in stable training of GANs

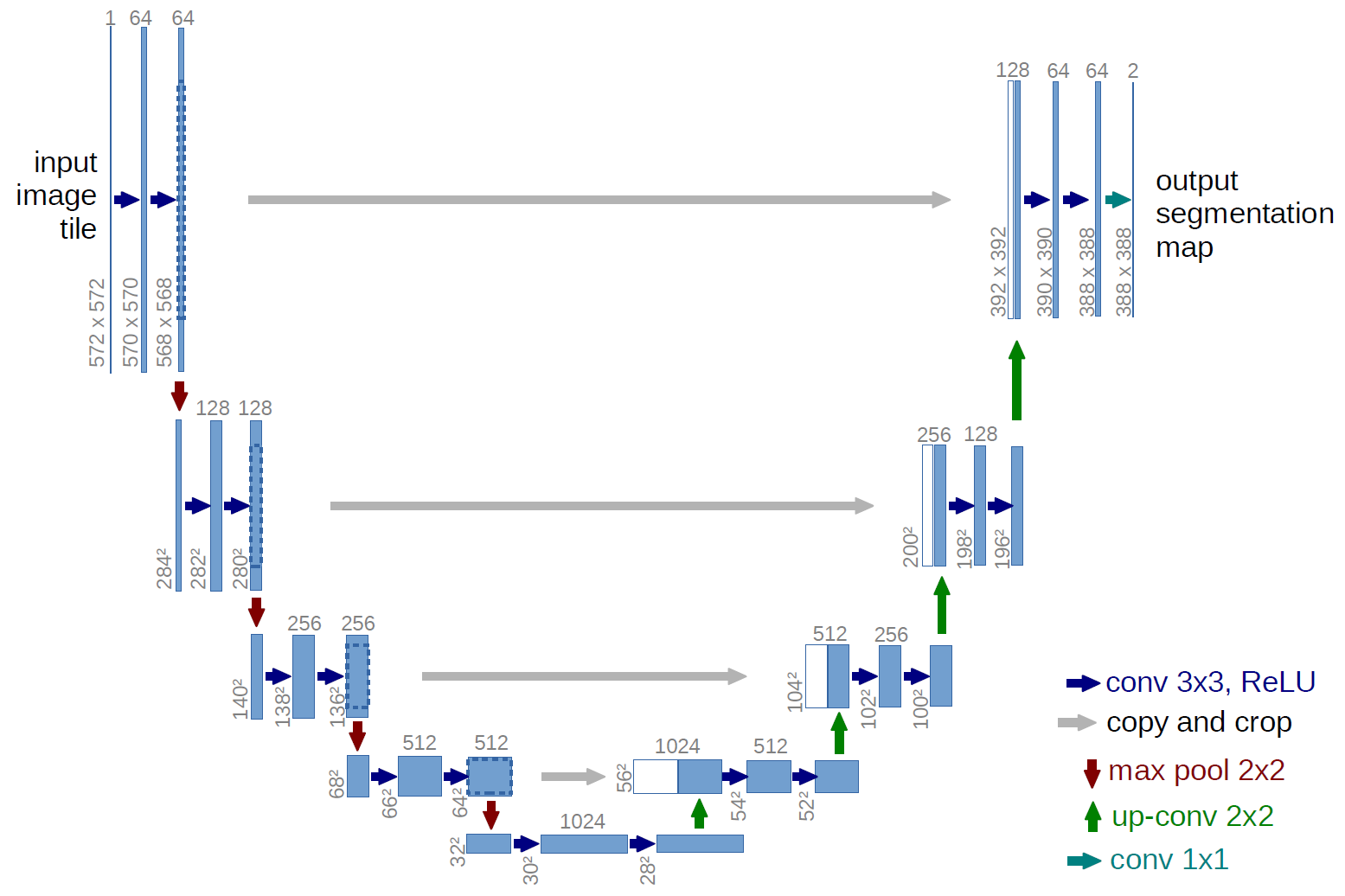
Diffusion models:

The Complete noise image has pixels sampled from Normal distribution.

In our case we have sprites (images of characters in 2D video games)

For each level of noise, Network should progressively denoise it to previous step i.e. less noise

UNET:



1. Used for Segmentation i.e. assigning label to each pixel
2. Fully Convolutional network - All layers are convolutional layers and no fully connected layer
3. Designed to learned from few training samples
4. U shaped encoder decoder architecture with skip connections. Encoder path captures the context and captures high level features & decoded path enables precise location and reconstructs the segmented image. Skip connections help in preserving spatial details
5. Skip connections feed the output by skipping some layers to the input of other layer
6. Context refers to surrounding information, High level features are meaningful characteristics of image.
7. For e.g. context is that buildings are tall structures, cars are on road etc & High level features are colors of these cars, shape

# Upsampling2D vs Conv2DTranspose:

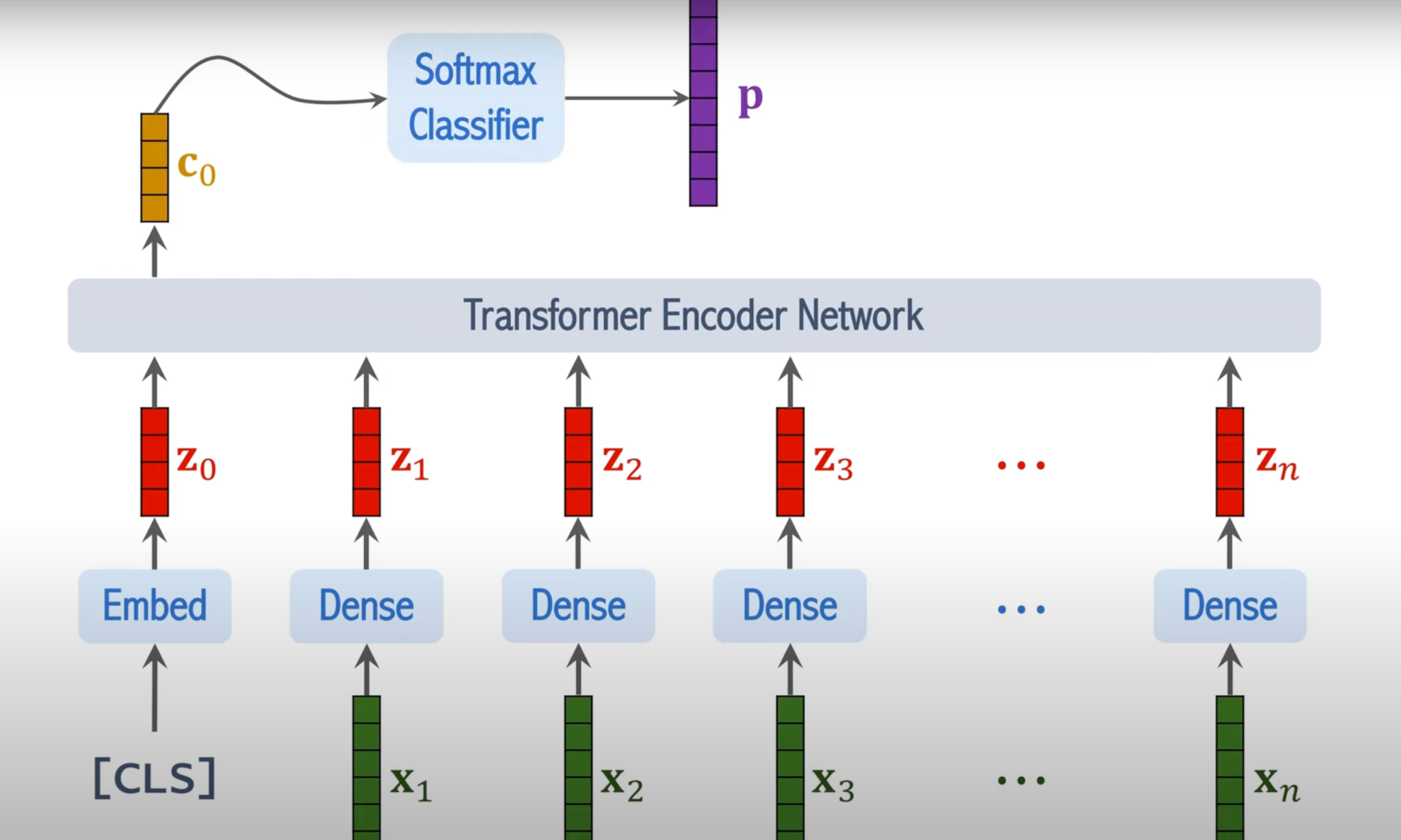
1. 2 types of layers to increase the dimension of arrays in keras
2. Upsampling2D is opposite of Max pooling where it repeats rows and columns of output
3. Conv2DTranspose performs up sampling and convolution
4. Conv2DTranspose has been reported to result in Checkerboard artifacts but not much info or data is available on this. It is a case of GAN as per a blog.

# Resnets:

1. Blocks having convolutional layers and skip connections are ResNets.
2. They help in handling vanishing gradients problem and preserve lost context.
3. Popular in classification and Segmentation.
4. Different ResNet architectures exist based on depth ResNet-18, ResNet-34, ResNet-50 etc

# Vision Transformer:

1. Used for classification & rules out CNN
2. Requires bigger dataset to outperform CNN
3. VIT converts image into patch and converts each patch tensor to vector.
4. Positional encoding i.e. (adding position information of a patch) improves the model.



1. X1,x2,...,xn are vectors of n patches, z1,z2,...,zn are linear transformations combined with positional encoding.
2. We also use CLS token for classification, Embedding layers is used for CLS to Z0 transformation
3. Only C0 is used to classify