# Importando as bibliotecas para utilizar na Rede Neural Convolucional

- **Primeiro:** vamos importa a biblioteca numpy para fazer alguns cálculos científicos
- **Segundo:** vamos importar as bibliotecas os e csv para abrir o nosso arquivo para utilizarmos as nossas imagens.
- **Terceiro:** vamos importar as bibliotecas "skimage" para poder fazer manipulação na imagem
- **Quarto:** vamos importar o modelo "Sequential" para utilizar o keras como camada feed-foward.
- **Quindo** vamos importar a biblioteca "Matplotlib" para plotarmos a imagen que estamos utilizando.

```
import numpy as np
import os
import csv
from skimage import io, transform
import matplotlib.pyplot as plt
```

# Importando o google drive para montar e conseguir treinar as imagens

1. Será preciso importar a biblioteca *drive*;
2. Depois será preciso montar o arquivo para que possa utilizar o google drive como caminho

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfe

Enter your authorization code:
··········
Mounted at /content/drive/
```

# Após montar o drive vamos implementar algumas funções para podermos pegar as nossas imagens. Abaixo segue especificando cada uma delas:

1. **Abrir o arquivo csv:** para abrir o arquivo .csv implementamos uma função;
2. **Abrir as imagens:** para abrir as imagens implementamos uma função que tem como base o arquivo .csv
3. **Mudar o número das classes:** como as classes não começa com uma orde foi necessário escalar as imagens para que ficassem em uma escala entre zero até a classe mais alta que do zero ao 29;

```
def openCsv(wayFile):
    way_classes = []
    way_datas = []

    count = 0

    with open(wayFile, 'rb') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
        for row in spamreader:
            way_datas.append(row[0])
            way_classes.append(int(row[1]))
    return way_datas, way_classes

def openImage(way_path, way_image, width, height):
    X = []
    for i in range(0, len(way_image)):
        img = io.imread('%s%s' % (way_path, way_image[i]))
        img = img[...,:3]
        img = transform.resize(img,(width, height))
        X.append(img)
    return X

def map_classes(way_classes):
    m = {}
    y = np.zeros(len(way_classes))
    uc = np.unique(way_classes)

    for i in range(0, len(uc)):
        m[uc[i]] = i

    for i in range(0, len(way_classes)):
        y[i] = m[way_classes[i]]

    return y, m, uc
```

# Rodando as funções - openCsv(), openImage() e map_classes().

Nesta Seção será executado as funções que implementamos, para isso definiremos a altura e largura das imagens para que ambas ficam com tamanho padrão.

- **OBS.:** Transformaremos a lista X_data e y_layers em **array**; as imagens terão 229 de **largura** e 229 de **altura** pois a arquitetura que utilizaremos para que funcione de forma eficiente é necessário que utilizemos as imagens com está redimensão.

```
width = 100
height = 100

way_path = '/content/drive/My Drive/ColabNotebooks/A1/'
way_image, way_classes = openCsv('/content/drive/My Drive/ColabNotebooks/A1.csv')

X_datas = openImage(way_path, way_image, width, height)
y_layers, m, uc = map_classes(way_classes)

X_datas = np.asarray(X_datas)
y_layers = np.asarray(way_classes)
```

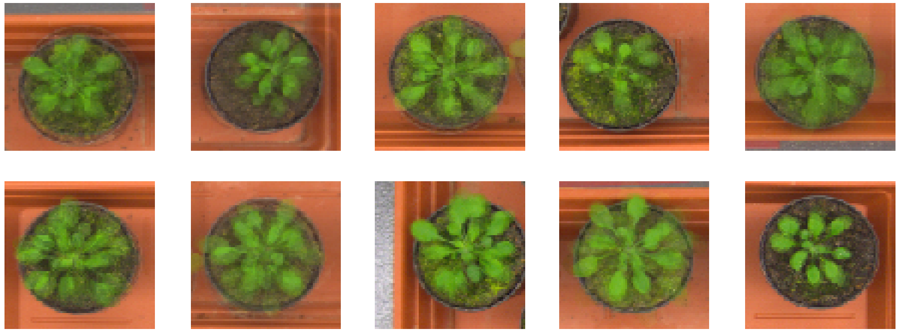
```
/usr/local/lib/python2.7/dist-packages/skimage/transform/_warps.py:84: UserWarning: The default mode, 'constant',
  warn("The default mode, 'constant', will be changed to 'reflect' in "
```

# Implementando o K-means

```
from sklearn.cluster import KMeans

X_datas = X_datas.reshape(X_datas.shape[0], 100*100*3)

kmeans = KMeans(n_clusters=len(uc), random_state=0)
clusters = kmeans.fit_predict(X_datas)
kmeans.cluster_centers_.shape

kmeans.cluster_centers_.shape
```

```
(10, 30000)
```

```
fig, ax = plt.subplots(2, 5, figsize=(8, 3))
centers = kmeans.cluster_centers_.reshape(len(uc), 100, 100, 3)
for axi, center in zip(ax.flat, centers):
    axi.set(xticks=[], yticks=[])
    axi.imshow(center, interpolation='nearest', cmap=plt.cm.binary)
```



```
data = X_datas
n_samples, n_features = data.shape
n_digits = len(uc)
labels = way_classes
```

```
print("n_digits: %d, \t n_samples %d, \t n_features %d"
      % (n_digits, n_samples, n_features))
```

```
n_digits: 10, 	 n_samples 128, 	 n_features 30000
```

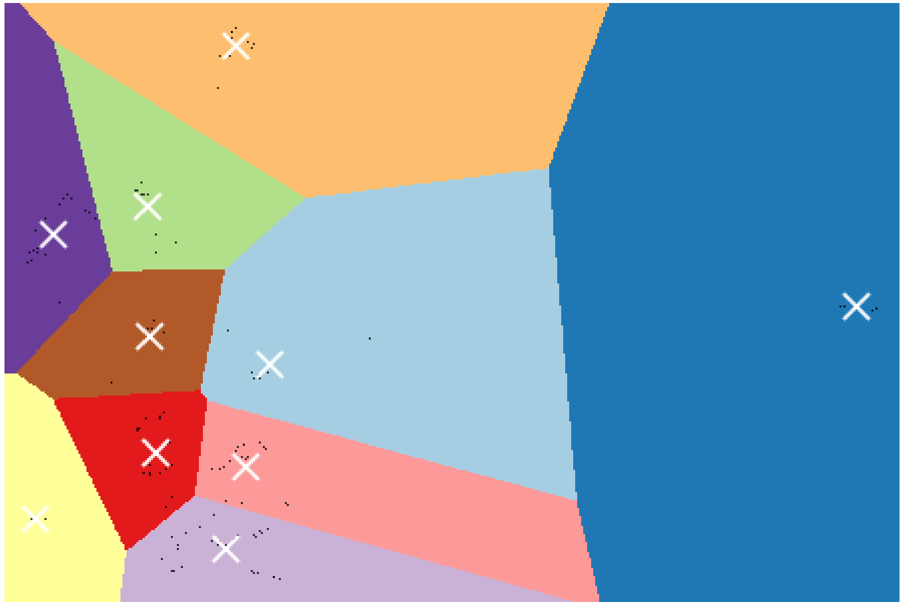
```
print(82 * '_')
print('init\t\ttime\tinertia\thomo\tcompl\tv-meas\tARI\tAMI\tsilhouette')


def bench_k_means(estimator, name, data):
    t0 = time()
    estimator.fit(data)
    print('%-9s\t%.2fs\t%i\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f'
          % (name, (time() - t0), estimator.inertia_,
             metrics.homogeneity_score(labels, estimator.labels_),
             metrics.completeness_score(labels, estimator.labels_),
             metrics.v_measure_score(labels, estimator.labels_),
             metrics.adjusted_rand_score(labels, estimator.labels_),
             metrics.adjusted_mutual_info_score(labels,  estimator.labels_),
             metrics.silhouette_score(data, estimator.labels_,
                                      metric='euclidean',
```

```
                                             sample_size=sample_size)))

    bench_k_means(KMeans(init='k-means++', n_clusters=n_digits, n_init=10),
                  name="k-means++", data=data)

    bench_k_means(KMeans(init='random', n_clusters=n_digits, n_init=10),
                  name="random", data=data)

    # in this case the seeding of the centers is deterministic, hence we run the
    # kmeans algorithm only once with n_init=1
    pca = PCA(n_components=n_digits).fit(data)
    bench_k_means(KMeans(init=pca.components_, n_clusters=n_digits, n_init=1),
                  name="PCA-based",
                  data=data)
    print(82 * '_')

    # #############################################################################
    # Visualize the results on PCA-reduced data

    reduced_data = PCA(n_components=2).fit_transform(data)
    kmeans = KMeans(init='k-means++', n_clusters=n_digits, n_init=10)
    kmeans.fit(reduced_data)

    # Step size of the mesh. Decrease to increase the quality of the VQ.
    h = .02     # point in the mesh [x_min, x_max]x[y_min, y_max].

    # Plot the decision boundary. For that, we will assign a color to each
    x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
    y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    # Obtain labels for each point in mesh. Use last trained model.
    Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure(1)
    plt.clf()
    plt.imshow(Z, interpolation='nearest',
               extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               cmap=plt.cm.Paired,
               aspect='auto', origin='lower')

    plt.plot(reduced_data[:, 0], reduced_data[:, 1], 'k.', markersize=2)
    # Plot the centroids as a white X
    centroids = kmeans.cluster_centers_
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='x', s=169, linewidths=3,
                color='w', zorder=10)
    plt.title('K-means clustering on the digits dataset (PCA-reduced data)\n'
              'Centroids are marked with white cross')
```

```
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

| init | time | inertia | homo | compl | v-meas | ARI | AMI | silhouette |
|---|---|---|---|---|---|---|---|---|
| k-means++ | 2.16s | 28761 | 0.398 | 0.344 | 0.369 | 0.143 | 0.239 | 0.261 |
| random | 1.89s | 29410 | 0.394 | 0.340 | 0.365 | 0.144 | 0.235 | 0.252 |
| PCA-based | 0.26s | 43598 | 0.164 | 0.303 | 0.213 | 0.043 | 0.063 | 0.066 |

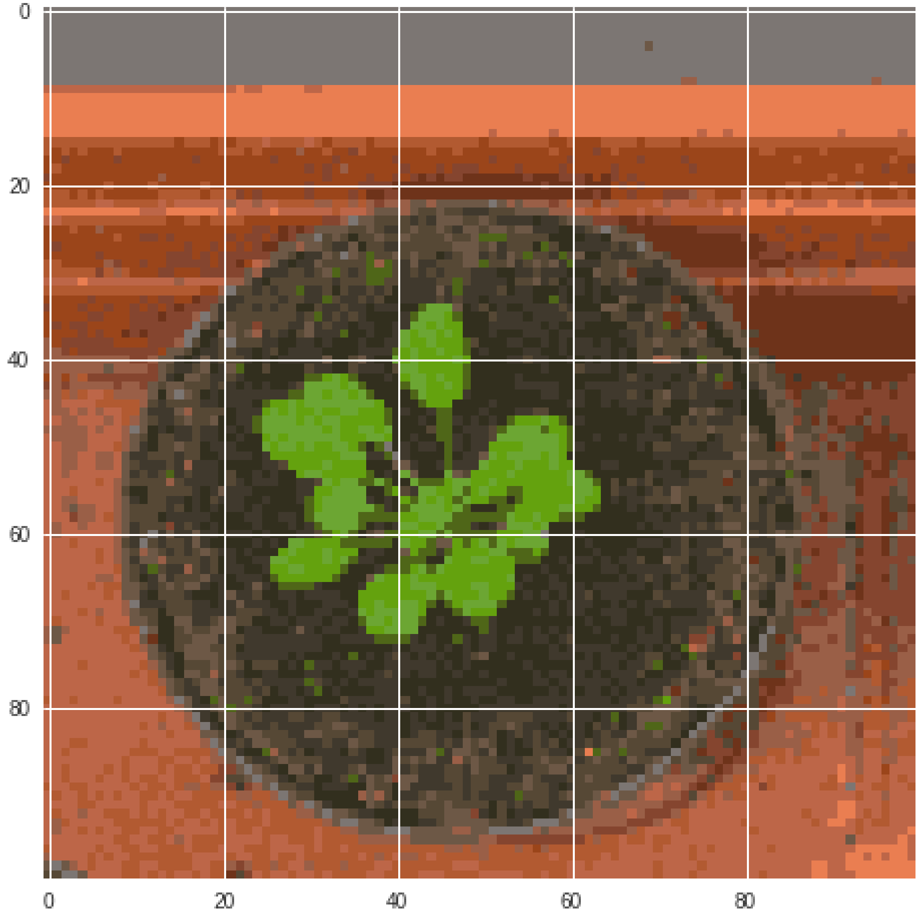
K-means clustering on the digits dataset (PCA-reduced data)
Centroids are marked with white cross



```
kmeans_cluster = KMeans(n_clusters=15)
kmeans_cluster.fit(image_2d)
cluster_centers = kmeans_cluster.cluster_centers_
cluster_labels = kmeans_cluster.labels_
```

```
plt.figure(figsize = (15,8))
plt.imshow(cluster_centers[cluster_labels].reshape(x, y, z))
```

```
<matplotlib.image.AxesImage at 0x7f6fcfa35890>
```




# Implementando o DBSCAN