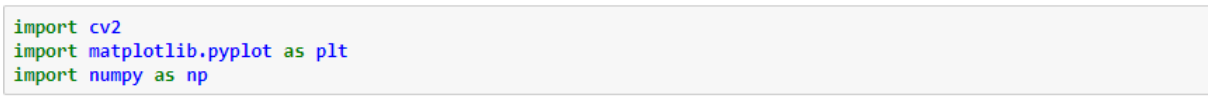
Monte Carlo Simulation

Aim

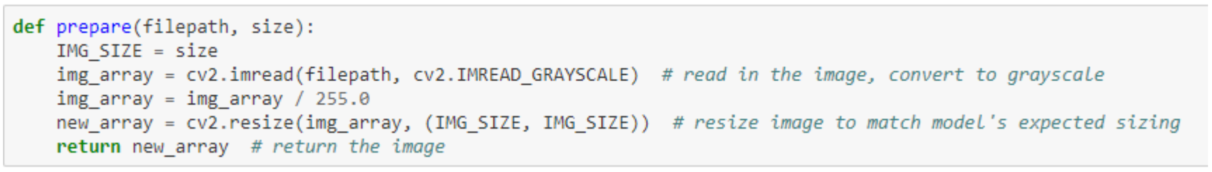
To simulate light scattering in biological tissue using Monte Carlo Simulation.

The Code

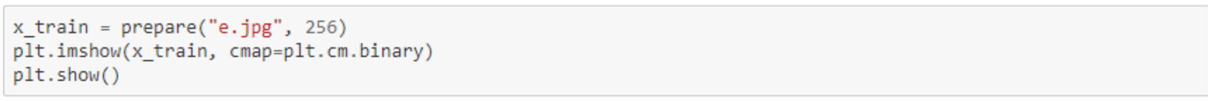
https://github.com/SayarGitHub/Monte_Carlo_Light_Scattering



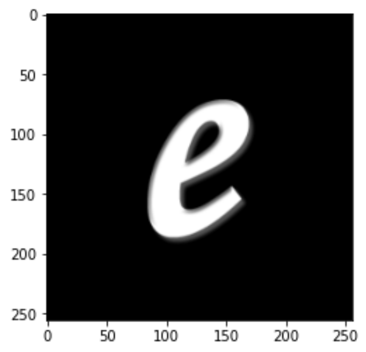
```python
import cv2
import matplotlib.pyplot as plt
import numpy as np
```



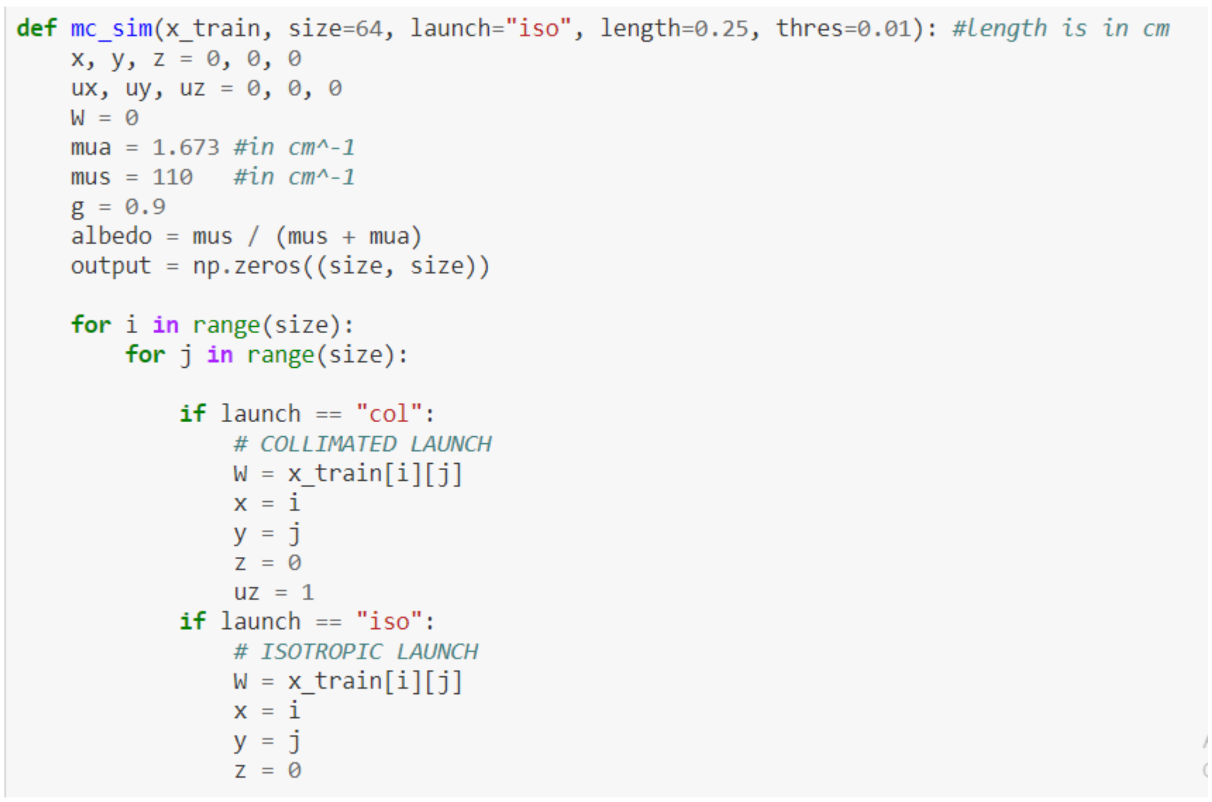
```python
def prepare(filepath, size):
    IMG_SIZE = size
    img_array = cv2.imread(filepath, cv2.IMREAD_GRAYSCALE)  # read in the image, convert to grayscale
    img_array = img_array / 255.0
    new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize image to match model's expected sizing
    return new_array  # return the image
```



```python
x_train = prepare("e.jpg", 256)
plt.imshow(x_train, cmap=plt.cm.binary)
plt.show()
```



The letter 'e' acts as the obstacle, and the surrounding is the source of light. The letter is plotted in the x-y plane. The biological tissue is imagined to be infinite in the x-y plane but limited in depth along the z-axis. This depth can be defined by the user when using the simulator. The source object is converted to grayscale and normalized to 0-1.



```python
def mc_sim(x_train, size=64, launch="iso", length=0.25, thres=0.01): #length is in cm
    x, y, z = 0, 0, 0
    ux, uy, uz = 0, 0, 0
    W = 0
    mua = 1.673 #in cm^-1
    mus = 110   #in cm^-1
    g = 0.9
    albedo = mus / (mus + mua)
    output = np.zeros((size, size))

    for i in range(size):
        for j in range(size):

            if launch == "col":
                # COLLIMATED LAUNCH
                W = x_train[i][j]
                x = i
                y = j
                z = 0
                uz = 1
            if launch == "iso":
                # ISOTROPIC LAUNCH
                W = x_train[i][j]
                x = i
                y = j
                z = 0
```

The input parameters-

x_train: the input image
size: the size of the input image (default = 64px)
launch: type of launch (default = ‘isotropic’)
length: length of the biological tissue (default = 0.25cm)
thres: the threshold value below which photon propagation will be stopped (default = 0.01)

User-defined initializations-

x,y,z: the coordinates
ux, uy, uz: the direction cosines.
mua: absorption coefficient of tissue
mus: scattering coefficient of tissue
g: anisotropy of scattering

```
        if launch == "col":
            # COLLIMATED LAUNCH
            W = x_train[i][j]
            x = i
            y = j
            z = 0
            uz = 1
        if launch == "iso":
            # ISOTROPIC LAUNCH
            W = x_train[i][j]
            x = i
            y = j
            z = 0

            costheta = 2 * np.random.rand() - 1
            sintheta = np.sqrt(1 - costheta ** 2)
            psi = 2 * np.pi * np.random.rand()
            ux = sintheta * np.cos(psi)
            uy = sintheta * np.sin(psi)
            uz = costheta
```

For ‘collimated launch,’ all the photons are launched in the z-direction. Whereas for ‘isotropic launch’ ϴ varies from -π to π and Φ varies from 0 to 2π. This is where the Monte Carlo simulation enters. The probability of the launch being in the hemisphere is 1.

```
        while length > z > -0.1:
            # HOP
            s = -np.log(np.random.rand()) / (mua + mus)
            x += s * ux
            y += s * uy
            z += s * uz

            # DROP
            absorb = W * (1 - albedo)
            W -= absorb
            if W <= thres:
                break
```

Some backward propagation is allowed in the hope that the ray might turn back towards positive z-direction. The value of s is derived from a probability distribution whose integral from 0 to infinity is 1. Hence the ‘hop’ step propagates the ray along each coordinate for a random distance.

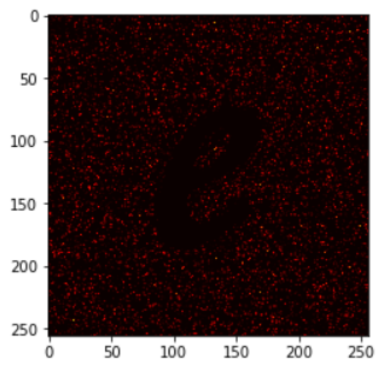
The ‘drop’ step simply reduces the intensity until the threshold.

```
                # SPIN/SCATTER
                rnd = np.random.rand()
                if g == 0:
                    costheta = 2 * rnd - 1
                else:
                    temp = (1 - g ** 2) / (1 - g + 2 * rnd * g)
                    costheta = (1 + g ** 2 - temp ** 2) / (2 * g)
                sintheta = np.sqrt(1 - costheta ** 2)

                psi = 2 * np.pi * np.random.rand()
                cospsi = np.cos(psi)
                if psi < np.pi:
                    sinpsi = np.sqrt(1 - cospsi ** 2)
                else:
                    sinpsi = -np.sqrt(1 - cospsi ** 2)

                if (1 - abs(uz)) <= 10 ** -12:
                    uxx = sintheta * cospsi
                    uyy = sintheta * sinpsi
                    uzz = costheta * np.sign(uz)
                else:
                    temp = np.sqrt(1 - uz ** 2)
                    uxx = sintheta * (ux * uz * cospsi - uy * sinpsi) / temp + ux * costheta
                    uyy = sintheta * (uy * uz * cospsi + ux * sinpsi) / temp + uy * costheta
                    uzz = -sintheta * cospsi * temp + uz * costheta

                ux = uxx
                uy = uyy
                uz = uzz

            x = int(np.round(x))
            y = int(np.round(y))

            if z >= length:
                if 0 <= x < size:
                    if 0 <= y < size:
                        output[x][y] = W
    return output
```

The photon is scattered into a new trajectory according to some scattering function. The two angles of scattering are Ѳ and Ф, the deflection and azimuthal scattering angles, respectively. The most commonly used function for the deflection angle Ѳ is the Henyey-Greenstein (HG) function (1941), which was proposed for describing the scattering of light from distant galaxies by galactic dust. Here we use the Monte Carlo sampling.

```
output = mc_sim(x_train,length=0.5,size=256) #takes approximately 11 minutes for length 0.5 and size 256
```

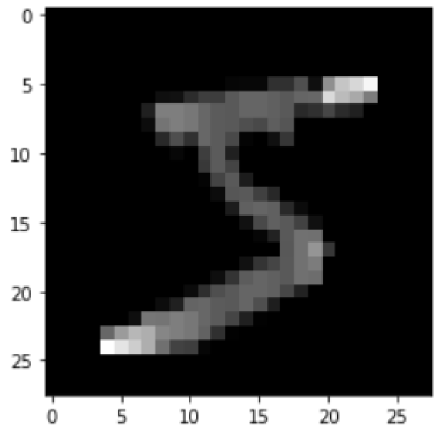
```
plt.imshow(output, cmap = 'hot')
plt.savefig("e_0.5_256.jpg", dpi =1200)
plt.show()
```



This how the simulated image looks like.

Are these images usable?

```
for i in range(len(x_train)):
    x_train[i] = 1 - x_train[i]
for i in range(len(x_test)):
    x_test[i] = 1 - x_test[i]
plt.imshow(x_train[0], cmap = plt.cm.binary)
plt.show()
```

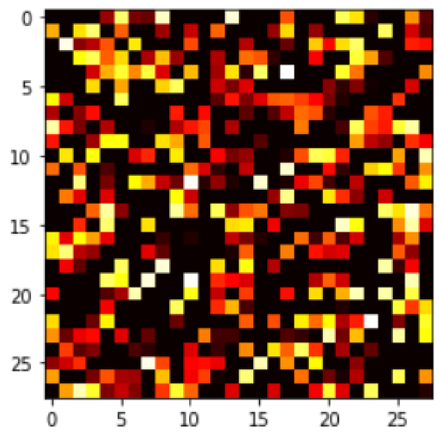


The MNIST dataset was normalized and used. Note the images are very low resolution (28X28).

```
for i in range(len(x_train)):
    x_train[i] = mc_sim(x_train[i],length=0.1, size=28)
    print((i+1)*100/len(x_train),"%")
```

```
for i in range(len(x_test)):
    x_test[i] = mc_sim(x_test[i],length=0.1, size=28)
    print((i+1)*100/len(x_test),"%")
```

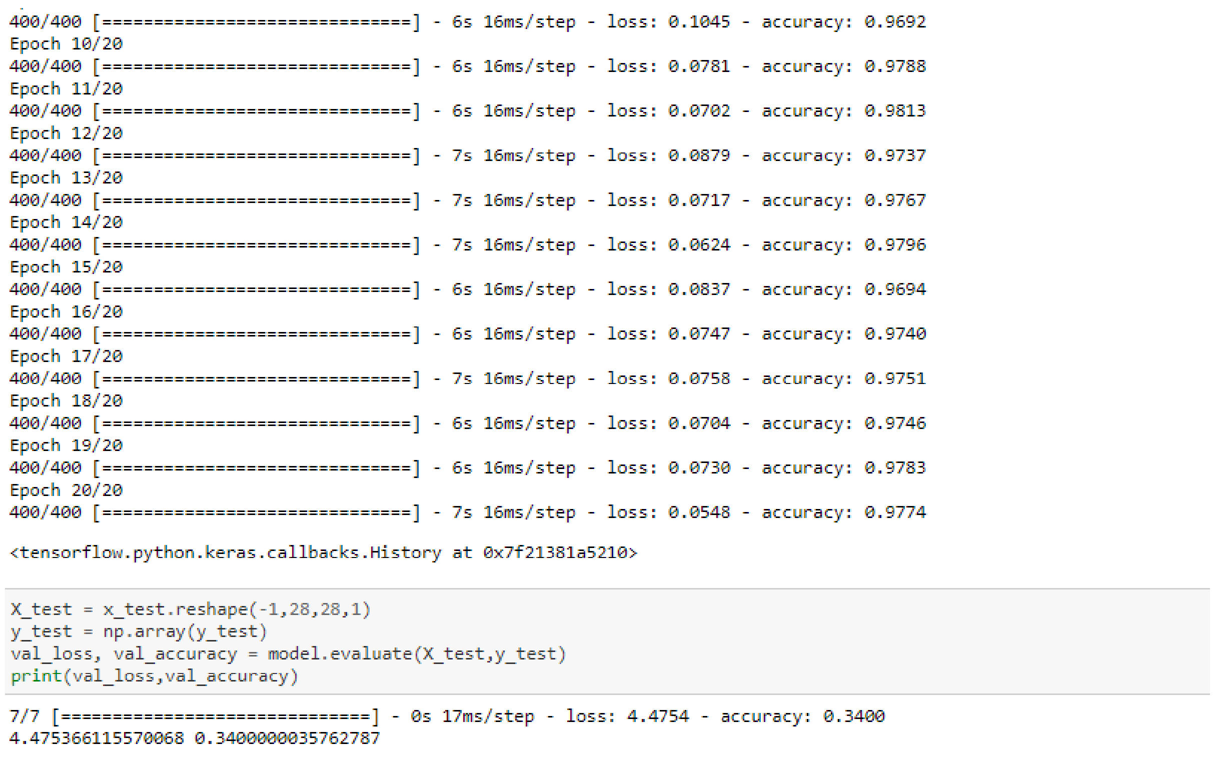
```
plt.imshow(x_train[0], cmap='hot')
plt.show()
```



After the simulation, the ‘5’ looks like this.

```
from keras.layers.advanced_activations import LeakyReLU
X = x_train.reshape(-1,28,28,1)
y = np.array(y_train)

dense_layer = 1
layer_size = 64
conv_layer = 2

model = Sequential()

model.add(Conv2D(layer_size, (3, 3), input_shape=X.shape[1:],padding="same"))
model.add(LeakyReLU(alpha=0.3))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

for l in range(conv_layer-1):
  model.add(Conv2D(layer_size, (3, 3),padding="same"))
  model.add(LeakyReLU(alpha=0.3))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(0.2))

model.add(Flatten())

for l in range(dense_layer):
    model.add(Dense(128))
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dropout(0.2))

model.add(Dense(10))
model.add(Activation('softmax'))


model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(X, y,
          batch_size=5,
          epochs=20)
```

A simple convolutional neural network was used with 1 dense layer and 2 convolutional layers. This network was trained with 2000 (28X28) scattered images using the LeakyReLU activation function for 20 epochs using the adam optimizer.



```
400/400 [==============================] - 6s 16ms/step - loss: 0.1045 - accuracy: 0.9692
Epoch 10/20
400/400 [==============================] - 6s 16ms/step - loss: 0.0781 - accuracy: 0.9788
Epoch 11/20
400/400 [==============================] - 6s 16ms/step - loss: 0.0702 - accuracy: 0.9813
Epoch 12/20
400/400 [==============================] - 7s 16ms/step - loss: 0.0879 - accuracy: 0.9737
Epoch 13/20
400/400 [==============================] - 7s 16ms/step - loss: 0.0717 - accuracy: 0.9767
Epoch 14/20
400/400 [==============================] - 7s 16ms/step - loss: 0.0624 - accuracy: 0.9796
Epoch 15/20
400/400 [==============================] - 6s 16ms/step - loss: 0.0837 - accuracy: 0.9694
Epoch 16/20
400/400 [==============================] - 6s 16ms/step - loss: 0.0747 - accuracy: 0.9740
Epoch 17/20
400/400 [==============================] - 7s 16ms/step - loss: 0.0758 - accuracy: 0.9751
Epoch 18/20
400/400 [==============================] - 6s 16ms/step - loss: 0.0704 - accuracy: 0.9746
Epoch 19/20
400/400 [==============================] - 6s 16ms/step - loss: 0.0730 - accuracy: 0.9783
Epoch 20/20
400/400 [==============================] - 7s 16ms/step - loss: 0.0548 - accuracy: 0.9774

<tensorflow.python.keras.callbacks.History at 0x7f21381a5210>
```

```
X_test = x_test.reshape(-1,28,28,1)
y_test = np.array(y_test)
val_loss, val_accuracy = model.evaluate(X_test,y_test)
print(val_loss,val_accuracy)
```

```
7/7 [==============================] - 0s 17ms/step - loss: 4.4754 - accuracy: 0.3400
4.475366115570068 0.3400000035762787
```

The model achieved 34% accuracy with limited data.

Future Plans

1. Multithreading the simulator for better performance.
2. With the availability of faster computational resources, a larger dataset can be generated using much higher resolution images.
3. With the availability of faster computational computing resources, a more complex and robust neural network can be used.
4. After satisfactory accuracy in identification, the network can be modified to perform segmentation and reconstruction of scattered images.

References

1. Chapter 5 in “Optical-Thermal Responses of Laser-Irradiated Tissue,” 2nd Edition, 2009, eds. A.J. Welch, M.J.C. van Gemert, publ. Springer
2. “Radiative Transfer Theory” in EECS 730, Winter 2009 by K. Sarabandi
3. “Light scattering study of tissues” by V V Tuchin
4. Li, Haicheng et al. “Monte Carlo simulation of light scattering in tissue for the design of skin-like optical devices.” *Biomedical optics express* vol. 10,2 868-878. 24 Jan. 2019, doi:10.1364/BOE.10.000868
5. “Introduction to Linear Algebra” by Gilbert Strang
6. “Deep Learning” by Ian Goodfellow and Yoshua Bengio and Aaron Courville