train_set_x_orig => (209, 64, 64, 3)

train_set_y => (1, 209)

---

test_set_x_orig => (50, 64, 64, 3)

test_set_y => (1, 50)

---

m_train = train_set_x_orig.shape[0] => 209

m_test = test_set_x_orig.shape[0] => 50

num_px = train_set_x_orig.shape[1] => 64
(height, width each image)

You get X

$A = \sigma(w^T X + b)$

$J = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log(a^{(i)}) + (1-y^{(i)}) (\log(1-a^{(i)}$

$\frac{dJ}{dw} = \frac{1}{m} X (\underbrace{A-Y}_{dz})^T$

$\frac{dJ}{db} = \frac{1}{m} \sum_{i=1}^{m} \underbrace{(a^{(i)} - y^{(i)})}_{dz^{(i)}}$

---

flatten. (50, 64, 64, 3) => test_set_x_orig -> trs.
(209, 64, 64, 3) => train_set_x_orig => trs

```
train_set_x_flatten = train_set_x_orig.reshape(trs.shape[1] * trs.shape[2] * 3, trs.shape[0])
test_set_x_flatten = test_set_x_orig.reshape(trs.shape[1] * trs.shape[2] * trs.shape[3], trs.shape[0])
```

train_set_x_flatten = (12288, 209) test_set_x_flatte = (12288, 50) train_set_y = (1, 209) .. test (1, 50)

```
def sigmoid(z):
    return = 1/(1+np.exp(-z))
```

```
def initialize_with_zeros(dim):
    w = np.zeros((dim, 1))
    b = 0
```

$w = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ (dim)

```
    b = 0
    return w, b
```

```
def propagate(w, b, X, Y):
    m = X.shape[1] => 209, 50 -> kaç eder resim sayısı
    A = sigmoid(np.dot(w.T, X) + b) => [0.0.0] * [1 2 3 ...]^T + 0 => 0
    cost = (-1/m) * np.sum((Y * np.log(A) + (1-Y) * (np.log(1-A))), axis=1)
    dw = 1/m * (np.dot(X, (A-Y).T)) => dw = X . dz   (dz = A-Y)
    db = 1/m * np.sum((A-Y), axis=1)   db = dz
    grads = {"dw": dw,
             "db": db}
    return grads, cost
```

```
def optimize(w,b, X,Y, num_iterations, learning_rate, print_cost = False):

    costs = []

    for i in range(num_iterations):

        grads, cost = propagate(w,b,X,Y)

        dw = grads["dw"]
        db = grads["db"]

        w = w - learning_rate * dw
        b = b - learning_rate * db

        if i % 100 == 0:
            costs.append(cost)

    params = {"w": w,
              "b": b}

    grads = {"dw": dw,
             "db": db}

    return params, grads, costs
```

```
def predict(w,b,X):

    m = X.shape[1]
    Y_prediction = np.zeros((1,m))
    w = w.reshape(X.shape[0], 1)

    A = sigmoid(np.dot(w.T, X) + b)

    for i in range(A.shape[1]):
        if (A[0,i] > 0.5):
            Y_prediction[0,i] = 1
        else:
            Y_prediction[0,i] = 0

    return Y_prediction
```

$w = \begin{bmatrix} 0,11 \\ 0,23 \end{bmatrix}$ w.reshape(X.shape[0], 1)

$X = \begin{bmatrix} 1 & -1 & -3 \\ 1 & 2 & 0,1 \end{bmatrix}$

X.shape[0] = 2

$w^T \cdot X$ → dot

$[0,11 \;\; 0,23] \begin{bmatrix} 1 & -1 & -3 \\ 1 & 2 & 0,1 \end{bmatrix}$

1×2 · 2×3 → 1×3-Matrix

```
def model(X_train, Y_train, X_test, Y_test, num_iterations=2000, learning_rate=0.5,
print_cost=False):

    w, b = initialize_with_zeros(X_train.shape[0])

    w = parameters["w"]
    b = parameters["b"]

Y_prediction_test = predict(w, b, X_test)
Y_prediction_train = predict(w, b, X_train)
```
