In [38]:

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

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.moun
t("/content/gdrive", force_remount=True).
```

In [39]:

```
!unzip /content/gdrive/Shareddrives/nhan_dien_dac_diem_khuon_mat/emotions_data.zip -d "/
content/"
```

```
Archive:  /content/gdrive/Shareddrives/nhan_dien_dac_diem_khuon_mat/emotions_data.zip
replace /content/images/images/train/angry/0.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: N
```

Import thư viện và data

In [40]:

```
import glob
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sn
import pandas as pd
import cv2
```

In [41]:

```
train_dir = '/content/images/train'
test_dir = '/content/images/validation'
categories = ["angry", "disgust", "fear", "happy", "neutral", "sad", "surprise" ]
img_size = 64
```

In [42]:

```
X_train_label = []
X_train_list = []
X_test_label = []
X_test_list = []
```

Load data và preprocessing

- Tập train

In [43]:

```
angry_train_list = glob.glob(train_dir+"/"+categories[0]+"/*")
disgust_train_list = glob.glob(train_dir+"/"+categories[1]+"/*")
fear_train_list = glob.glob(train_dir+"/"+categories[2]+"/*")
happy_train_list = glob.glob(train_dir+"/"+categories[3]+"/*")
neutral_train_list = glob.glob(train_dir+"/"+categories[4]+"/*")
sad_train_list = glob.glob(train_dir+"/"+categories[5]+"/*")
surprise_train_list = glob.glob(train_dir+"/"+categories[6]+"/*")
for name in angry_train_list:
  X_train_label.append(0)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_train_list.append((img))
for name in disgust_train_list:
  X_train_label.append(1)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_train_list.append((img))
for name in fear_train_list:
  X_train_label.append(2)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_train_list.append((img))
for name in happy_train_list:
  X_train_label.append(3)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_train_list.append((img))
for name in neutral_train_list:
  X_train_label.append(4)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_train_list.append((img))
for name in sad_train_list:
  X_train_label.append(5)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_train_list.append((img))
for name in surprise_train_list:
  X_train_label.append(6)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_train_list.append((img))
```

- Tập test

In [44]:

```
angry_test_list = glob.glob(test_dir+"/"+categories[0]+"/*")
disgust_test_list = glob.glob(test_dir+"/"+categories[1]+"/*")
fear_test_list = glob.glob(test_dir+"/"+categories[2]+"/*")
happy_test_list = glob.glob(test_dir+"/"+categories[3]+"/*")
neutral_test_list = glob.glob(test_dir+"/"+categories[4]+"/*")
sad_test_list = glob.glob(test_dir+"/"+categories[5]+"/*")
surprise_test_list = glob.glob(test_dir+"/"+categories[6]+"/*")
for name in angry_test_list:
  X_test_label.append(0)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_test_list.append((img))
for name in disgust_test_list:
  X_test_label.append(1)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_test_list.append((img))
for name in fear_test_list:
  X_test_label.append(2)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_test_list.append((img))
for name in happy_test_list:
  X_test_label.append(3)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_test_list.append((img))
for name in neutral_test_list:
  X_test_label.append(4)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_test_list.append((img))
for name in sad_test_list:
  X_test_label.append(5)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_test_list.append((img))
for name in surprise_test_list:
  X_test_label.append(6)
  img = cv2.imread(name)
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img,(img_size,img_size))
  X_test_list.append((img))
```

In [45]:

```
X_train = np.array(X_train_list)
y_train = np.array(X_train_label)
X_test = np.array(X_test_list)
y_test = np.array(X_test_label)
```

Duỗi vector, chuẩn hoá input

In [46]:

```
X_train_scaled = np.array([x.ravel()/255. for x in X_train])
X_test_scaled = np.array([x.ravel()/255. for x in X_test])
```

Chuẩn hoá output

In [47]:

```
classes = np.unique(y_train)

y_train_onehot = np.zeros((len(y_train), len(classes)))
for i in range(len(y_train)):
    y_train_onehot[i, y_train[i]] = 1

y_test_onehot = np.zeros((len(y_test), len(classes)))
for i in range(len(y_test)):
    y_test_onehot[i, y_test[i]] = 1
```

In [48]:

```
print(y_train.shape)
print(y_train_onehot.shape)
```

```
(28821,)
(28821, 7)
```

In [49]:

```
print(y_test_onehot.shape)
```

```
(7066, 7)
```

Huấn luyện dữ liệu

In [50]:

```
def predict(X,w):
  h = np.dot(X,w)
  h = np.float64(h)
  softmax = np.exp(h)
  y_pred = softmax/np.sum(softmax, axis = 1, keepdims = True)
  return y_pred
def loss(X,w,y):
  y_pred = predict(X,w)
  theta = 1e-4
  l = -np.sum(y*np.log(y_pred)+theta)/y.shape[0]
  return l
def grad(X,w,y):
  y_pred = predict(X,w)
  delta = y_pred - y
  return np.dot(X.T,delta)/X.shape[0]
def gradient_descent(X,y,lr = 5e-3,epochs = 300):
  w = np.zeros((X.shape[1],y.shape[1]))
  losses = []
  for i in range(epochs):
    w -= lr*grad(X,w,y)
    l = loss(X,w,y)
    losses.append(l)
  print(f'Loss cross entropy: {loss(X,w,y)}')
  return losses,w
# def sgd(X, y, lr=1e-5, epochs=100, batch_size=32):
#     w = np.zeros((X.shape[1], y.shape[1]))
#     for epoch in range(epochs):
#         indices = np.random.permutation(X.shape[0])
#         X = X[indices]
#         y = y[indices]
#         for i in range(0, X.shape[0], batch_size):
#             X_batch = X[i:i+batch_size]
#             y_batch = y[i:i+batch_size]
#             # grad = np.dot(X_batch.T, predict(X_batch, w) - y_batch) / X_batch.shape[
0]
#             w -= lr * grad(X_batch,w,y_batch)
#         print(f"Epoch {epoch+1}/{epochs} loss: {loss(X, w, y)}")
#     return w
```

Tính loss

In [51]:

```
losses,w = gradient_descent(X_train_scaled, y_train_onehot)
# w = sgd(X_train_scaled, y_train_scaled)
```

```
Loss cross entropy: 1.7316008706438177
```

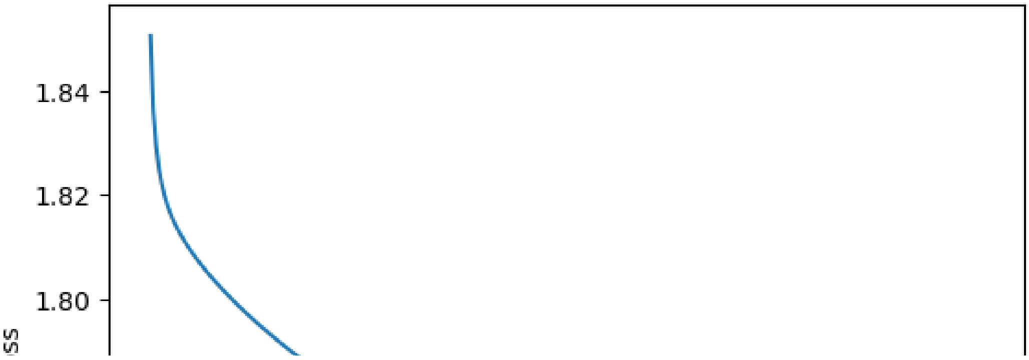
Vẽ đồ thị Loss ban đầu

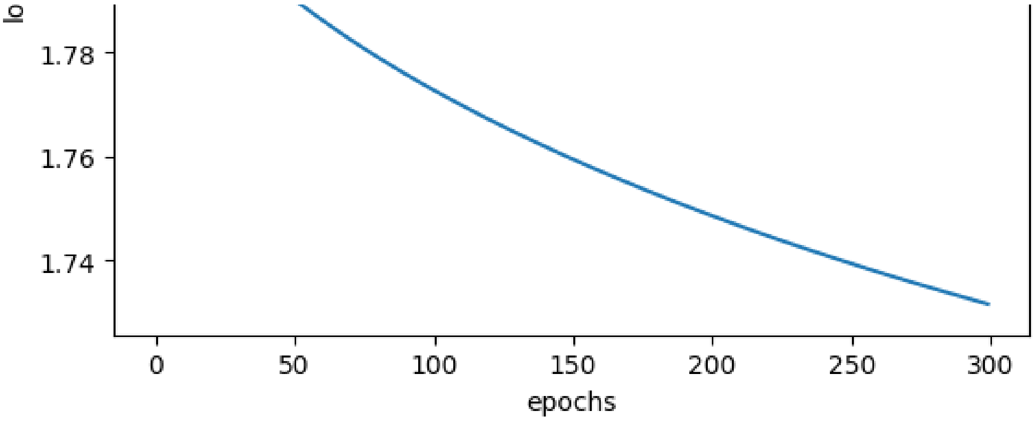
In [52]:

```
plt.plot(losses)
plt.xlabel("epochs")
plt.ylabel("loss")
```

Out[52]:

```
Text(0, 0.5, 'loss')
```





In [53]:

```
# Tính y dự đoán
y_pred_onehot = predict(X_test_scaled, w)
# Tìm y sao cho hiệu quả lớn nhất
y_pred = np.argmax(y_pred_onehot, axis=1)
```

Hàm đánh giá

In [54]:

```
def evaluation(y_test, y_pred):
    # Tính accuracy
    accuracy = np.mean(y_pred == y_test)

    # Tính precision và recall
    precision = np.zeros(len(classes))
    recall = np.zeros(len(classes))
    for i in range(len(classes)):
        tp = np.sum((y_pred == i) & (y_test == i))
        tn = np.sum((y_pred != i) & (y_test != i))
        fp = np.sum((y_pred == i) & (y_test != i))
        fn = np.sum((y_pred != i) & (y_test == i))
    precision = tp / (tp + fp) if (tp + fp) != 0 else 0
    recall = tp / (tp + fn) if (tp + fn) != 0 else 0

    # Tính macro-averaged precision and recall
    macro_precision = np.mean(precision)
    macro_recall = np.mean(recall)
    print(f'tp = {tp}, tn = {tn}, fp = {fp}, fn = {fn}')
    print(f'Accuracy: {accuracy}, Precision: {macro_precision}, Recall: {macro_recall}')
```

Đánh giá accuracy, recall, precision

In [55]:

```
evaluation(y_test, y_pred)
```

```
tp = 172, tn = 6020, fp = 249, fn = 625
Accuracy: 0.32097367676195865, Precision: 0.4085510688836104, Recall: 0.21580928481806774
```

Hàm LogisticRegression, có bổ sung bias

In [56]:

```
class LogisticRegression:
    def __init__(self, lr = 1e-4, epochs = 10):
        self.lr = lr
        self.epochs = epochs
    # Hàm thêm bias vào tập X
    def addBias(self,X):
        bias = np.ones((X.shape[0],1))
        X = np.hstack((bias,X));
        return X
    # Hàm sigmoid
    def predict(self,X,w):
        h = np.dot(X,w)
        h = np.float64(h)
        softmax = np.exp(h)
        y_pred = softmax/np.sum(softmax, axis = 1, keepdims = True)
        return y_pred
    # Hàm loss
    def loss(self,X,w,y):
        y_pred = self.predict(X,w)
        theta = 1e-4
        return -np.sum(y*np.log(y_pred)+theta)/y.shape[0]
    # Hàm gradient
    def grad(self,X,w,y):
        y_pred = self.predict(X,w)
        delta = y_pred - y
        return np.dot(X.T,delta)/X.shape[0]
    # Hàm gradient descent
    def gradient_descent(self,X,y,lr = 5e-3, epochs = 300):
        w = np.zeros((X.shape[1], y.shape[1]))
        losses = []
        for i in range(epochs):
          w-= lr*self.grad(X,w,y)
          l = self.loss(X,w,y)
          losses.append(l)
        print(f'Loss cross entropy: {self.loss(X,w,y)}')
        return losses, w
```

Thêm bias vào tập train và tính loss

In [57]:

```
model = LogisticRegression()
X_train_scaled_bias = model.addBias(X_train_scaled)
X_test_scaled_bias = model.addBias(X_test_scaled)
losses,w = model.gradient_descent(X_train_scaled_bias,y_train_onehot)
```

```
Loss cross entropy: 1.7309707547591657
```

In [58]:

```
# Tính y dự đoán sau khi thêm bias
y_pred = model.predict(X_test_scaled_bias,w)
# Tìm giá trị làm cho hiệu quả lớn nhất
y_pred = np.argmax(y_pred_onehot, axis=1)
```

Đánh giá mô hình sau khi thêm bias

In [59]:

```
evaluation(y_test,y_pred)
```

```
tp = 172, tn = 6020, fp = 249, fn = 625
Accuracy: 0.32097367676195865, Precision: 0.4085510688836104, Recall: 0.21580928481806774
```

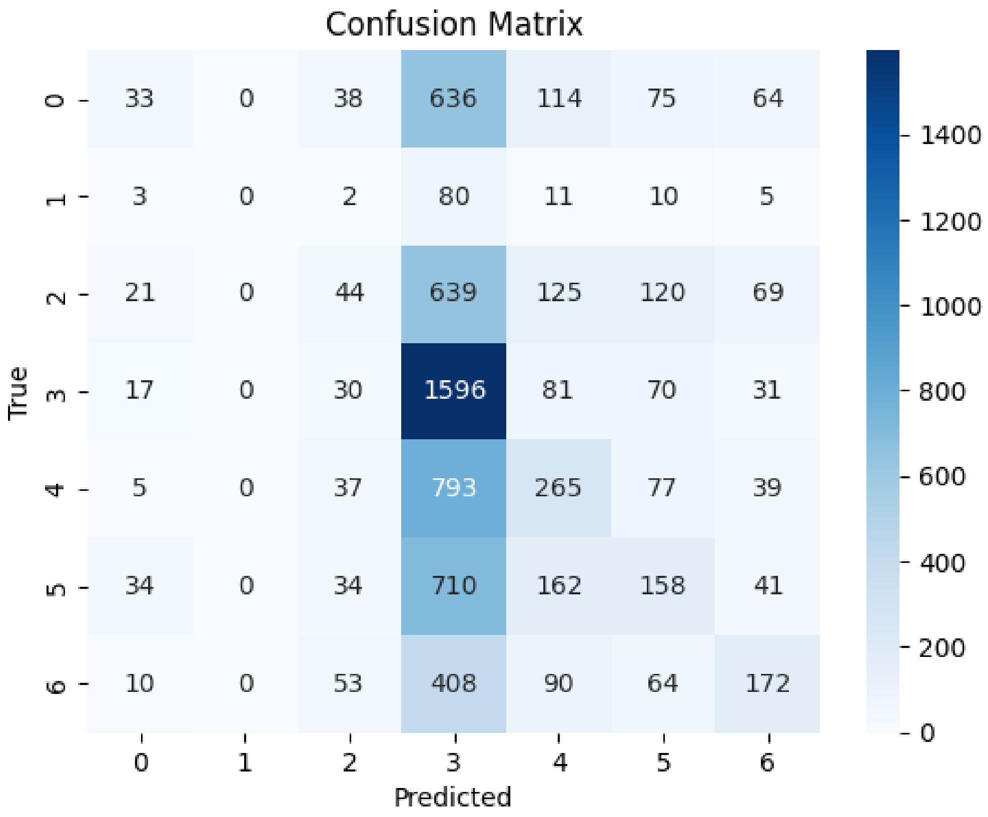
Vẽ confusion matrix dùng thư viện sklearn

In [60]:

```
import sklearn.metrics

# Tạo confusion matrix
cm = sklearn.metrics.confusion_matrix(y_test, y_pred)
```

```python
# Vẽ confusion matrix
plt.figure()
sn.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```


Confusion Matrix
True
Predicted