Siamese Network

Load data:

```
from tensorflow.keras.datasets.mnist import load_data

(X_train, y_train), (X_test, y_test) = load_data()

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
```

Build model CNN

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Input
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K

inp = Input(shape=(28, 28, 1))
x = Conv2D(filters=8, kernel_size=3, activation='relu')(inp)
x = MaxPooling2D(pool_size=(2,2))(x)

x = Flatten()(x)
x = Dense(units=32, activation='relu')(x)
x = Dense(units=2)(x)

cnn = Model(inputs=inp, outputs=x)

img1 = Input(shape=(28, 28, 1))
img2 = Input(shape=(28, 28, 1))

f1 = cnn(img1)
f2 = cnn(img2)

d = K.sqrt(K.sum(K.square(f1 - f2), axis=1, keepdims=True))

model =  Model(inputs=[img1, img2], outputs=d)
model.summary()
cnn.summary()

def loss(y_true, y_pred):
  proba = K.exp(-K.square(y_pred))
  return -K.mean(y_true * K.log(proba) + (1-y_true) * K.log(1-proba))

def loss1(y_true, y_pred):
  return K.mean(y_true * K.square(y_pred ) + (1-y_true) * K.square(K.maximum(1.0 - y_pred, 0)))

model.compile(optimizer='adam',loss=loss1)
```

Make all pairs or other strategies; some innovation here

```python
import numpy as np
from matplotlib import pyplot as plt

def generator(X, y, k=8):
  unique_labels = np.unique(y)

  while True:
    X1 = []
    X2 = []
    y_batch = []
    for label in unique_labels:
      label_idx = np.where(y == label)[0]
      other_labels = set(unique_labels) - {label}

      for i in range(k):
        i1 = np.random.choice(label_idx)
        i2 = np.random.choice(label_idx)
        # i1 must be different from i2
        while i1 == i2:
          i2 = np.random.choice(label_idx)
        # create positive example
        X1.append(X[i1][:, :, None])
        X2.append(X[i2][:, :, None])
        y_batch.append(1.0)

        # create negative example
        i1 = np.random.choice(label_idx)
        my_label = np.random.choice(list(other_labels))
        i2 = np.random.choice(list(np.where(y == my_label)[0]))
        X1.append(X[i1][:, :, None])
        X2.append(X[i2][:, :, None])
        y_batch.append(0.0)

    yield [np.array(X1) / 255., np.array(X2) / 255.], np.array(y_batch)

#For testing
for pair, y in generator(X_test, y_test):
  print('Batch size: ', len(y))
  idx = np.random.choice(range(len(y)))
  print(pair[0][idx].shape)
  print('Pair label:', y[idx])
  plt.subplot(121)
  plt.imshow(pair[0][idx].reshape(28, 28), cmap='gray')
  plt.subplot(122)
  plt.imshow(pair[1][idx].reshape(28, 28), cmap='gray')

  break
```

Fit model

```
history = model.fit(generator(X_train, y_train, k=32),
                    steps_per_epoch=5,
                    epochs=500,
                    validation_data=generator(X_test, y_test, k=8),
                    validation_steps=5)
```

Visualize learning process

```
plt.plot(history.history['loss'], label='Train', c='b')
plt.plot(history.history['val_loss'], label='Validation', c='r')
plt.legend()
plt.xlabel('Epochs', fontsize=16)
plt.ylabel('Loss', fontsize=16)
```

```
for pair, y in generator(X_test, y_test):
  y_pred = model.predict(pair)
  print('Batch size: ', len(y))
  idx = np.random.choice(range(len(y)))
  print('Pair label:', y[idx])
  print('Distance:', y_pred[idx])

  f1 = cnn(pair[0])
  f2 = cnn(pair[1])
  d = np.sqrt(np.sum((f1-f2)**2, axis=1, keepdims=True))
  print("Distance by features:", d[idx])

  plt.subplot(121)
  plt.imshow(pair[0][idx].reshape(28, 28), cmap='gray')
  plt.subplot(122)
  plt.imshow(pair[1][idx].reshape(28, 28), cmap='gray')
  break
```

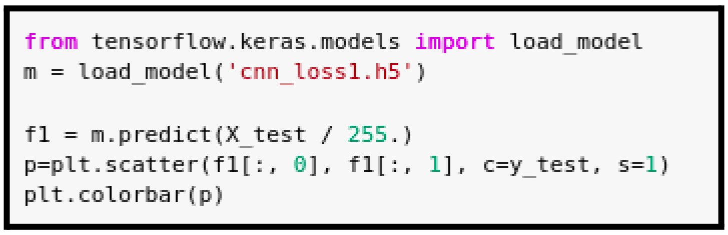
Visualize new feature space

```
f = cnn.predict(X_test/255.)
p=plt.scatter(f[:, 0], f[:, 1], c=y_test, s=1)
plt.colorbar(p)
```

Save model

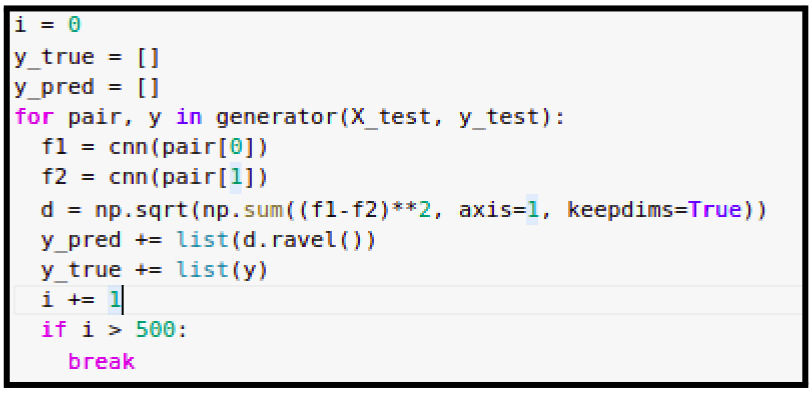
```
cnn.save('cnn_loss1.h5')
```

Load model and test



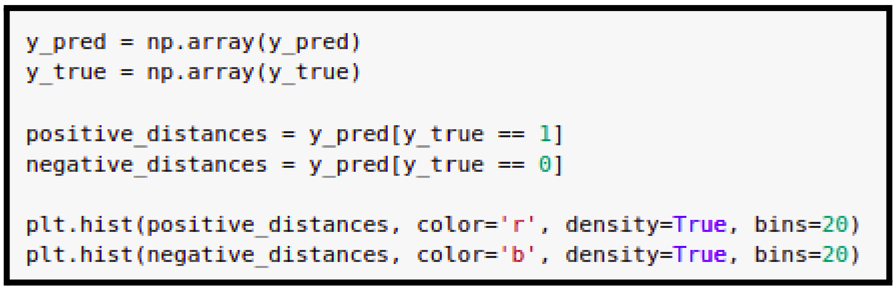
```python
from tensorflow.keras.models import load_model
m = load_model('cnn_loss1.h5')

f1 = m.predict(X_test / 255.)
p=plt.scatter(f1[:, 0], f1[:, 1], c=y_test, s=1)
plt.colorbar(p)
```

Visualize negative distance and positive distance



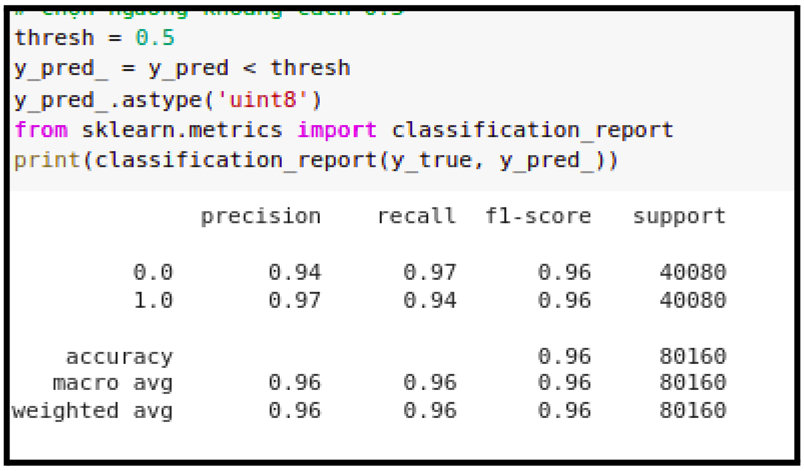
```python
i = 0
y_true = []
y_pred = []
for pair, y in generator(X_test, y_test):
  f1 = cnn(pair[0])
  f2 = cnn(pair[1])
  d = np.sqrt(np.sum((f1-f2)**2, axis=1, keepdims=True))
  y_pred += list(d.ravel())
  y_true += list(y)
  i += 1
  if i > 500:
    break
```

Plot histogram



```python
y_pred = np.array(y_pred)
y_true = np.array(y_true)

positive_distances = y_pred[y_true == 1]
negative_distances = y_pred[y_true == 0]

plt.hist(positive_distances, color='r', density=True, bins=20)
plt.hist(negative_distances, color='b', density=True, bins=20)
```

Check report using sklearn



```
thresh = 0.5
y_pred_ = y_pred < thresh
y_pred_.astype('uint8')
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred_))
```

```
              precision    recall  f1-score   support

         0.0       0.94      0.97      0.96     40080
         1.0       0.97      0.94      0.96     40080

    accuracy                           0.96     80160
   macro avg       0.96      0.96      0.96     80160
weighted avg       0.96      0.96      0.96     80160
```