Siamese Network

Load data:

In [ ]:

```
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)
```

```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
```

Build model CNN

In [ ]:

```
# Importing necessary modules and layers from TensorFlow Keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Input, BatchNo
rmalization, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K

# Defining input shape for the images
inp = Input(shape = (28,28,1))

# First Convolutional Neural Network block
cnn = Conv2D(filters = 32 , kernel_size = 3 ,activation ='relu')(inp)
cnn = BatchNormalization()(cnn)
pooling = MaxPooling2D(pool_size=(2,2))(cnn)
drop = Dropout(0.2)(pooling)

# Second Convolutional Neural Network block
cnn = Conv2D(filters = 64, kernel_size = 3, activation='relu')(drop)
cnn = BatchNormalization()(cnn)
pooling = MaxPooling2D(pool_size = (2,2))(cnn)
drop = Dropout(0.2)(pooling)

# Third Convolutional Neural Network block
cnn = Conv2D(filters = 128, kernel_size = 3, activation='relu')(drop)
cnn = BatchNormalization()(cnn)
pooling = MaxPooling2D(pool_size = (2,2))(cnn)
drop = Dropout(0.2)(pooling)

# Flattening the output for Dense layers
f = Flatten()(drop)
# First fully connected layer
fc1 = Dense(units = 256, activation ='relu')(f)
# Second fully connected layer
fc2 = Dense(units = 16,activation = 'relu')(fc1)
# Output layer for classification
out = Dense(units = 2)(fc2)

# Creating the CNN model for image feature extraction
cnn = Model(inputs = inp, outputs = out)

# Creating inputs for image pairs
img1 = Input(shape = (28,28,1))
img2 = Input(shape = (28,28,1))
```

```
# Obtaining feature vectors for both images using the CNN model
f1 = cnn(img1)
f2 = cnn(img2)

# Calculating the Euclidean distance between the two feature vectors
d = K.sqrt(K.sum(K.square(f1 - f2),axis = 1, keepdims = True))

# Creating the Siamese network model
model = Model(inputs = [img1,img2], outputs = d)

# Printing model summaries
model.summary()   # Summary of the Siamese network model
cnn.summary()     # Summary of the CNN model for image feature extraction

# Custom loss function definitions for model compilation
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)))

# Compiling the Siamese network model
model.compile(optimizer = 'adam', loss = loss1)
```

```
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to                     
==================================================================================================
 input_9 (InputLayer)           [(None, 28, 28, 1)]  0           []                               
                                                                                                  
 input_10 (InputLayer)          [(None, 28, 28, 1)]  0           []                               
                                                                                                  
 model_4 (Functional)           (None, 2)            130738      ['input_9[0][0]',                
                                                                  'input_10[0][0]']               
                                                                                                  
 tf.math.subtract_2 (TFOpLa     (None, 2)            0           ['model_4[0][0]',                
 mbda)                                                            'model_4[1][0]']                
                                                                                                  
 tf.math.square_2 (TFOpLamb     (None, 2)            0           ['tf.math.subtract_2[0][0]']     
 da)                                                                                              
                                                                                                  
 tf.math.reduce_sum_2 (TFOp     (None, 1)            0           ['tf.math.square_2[0][0]']       
 Lambda)                                                                                          
                                                                                                  
 tf.math.maximum_2 (TFOpLam     (None, 1)            0           ['tf.math.reduce_sum_2[0][0]']   
 bda)                                                                                             
                                                                                                  
 tf.math.sqrt_2 (TFOpLambda     (None, 1)            0           ['tf.math.maximum_2[
```

```
0][0]']
 )


==========================================================================================
=========
Total params: 130738 (510.70 KB)
Trainable params: 130290 (508.95 KB)
Non-trainable params: 448 (1.75 KB)
__________________________________________________________________________________________
_________
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_8 (InputLayer)        [(None, 28, 28, 1)]       0         
                                                                 
 conv2d_7 (Conv2D)           (None, 26, 26, 32)        320       
                                                                 
 batch_normalization_7 (Bat  (None, 26, 26, 32)        128       
 chNormalization)                                                
                                                                 
 max_pooling2d_7 (MaxPoolin  (None, 13, 13, 32)        0         
 g2D)                                                            
                                                                 
 dropout_6 (Dropout)         (None, 13, 13, 32)        0         
                                                                 
 conv2d_8 (Conv2D)           (None, 11, 11, 64)        18496     
                                                                 
 batch_normalization_8 (Bat  (None, 11, 11, 64)        256       
 chNormalization)                                                
                                                                 
 max_pooling2d_8 (MaxPoolin  (None, 5, 5, 64)          0         
 g2D)                                                            
                                                                 
 dropout_7 (Dropout)         (None, 5, 5, 64)          0         
                                                                 
 conv2d_9 (Conv2D)           (None, 3, 3, 128)         73856     
                                                                 
 batch_normalization_9 (Bat  (None, 3, 3, 128)         512       
 chNormalization)                                                
                                                                 
 max_pooling2d_9 (MaxPoolin  (None, 1, 1, 128)         0         
 g2D)                                                            
                                                                 
 dropout_8 (Dropout)         (None, 1, 1, 128)         0         
                                                                 
 flatten_2 (Flatten)         (None, 128)               0         
                                                                 
 dense_6 (Dense)             (None, 256)               33024     
                                                                 
 dense_7 (Dense)             (None, 16)                4112      
                                                                 
 dense_8 (Dense)             (None, 2)                 34        
                                                                 
=================================================================
Total params: 130738 (510.70 KB)
Trainable params: 130290 (508.95 KB)
Non-trainable params: 448 (1.75 KB)
_________________________________________________________________
```

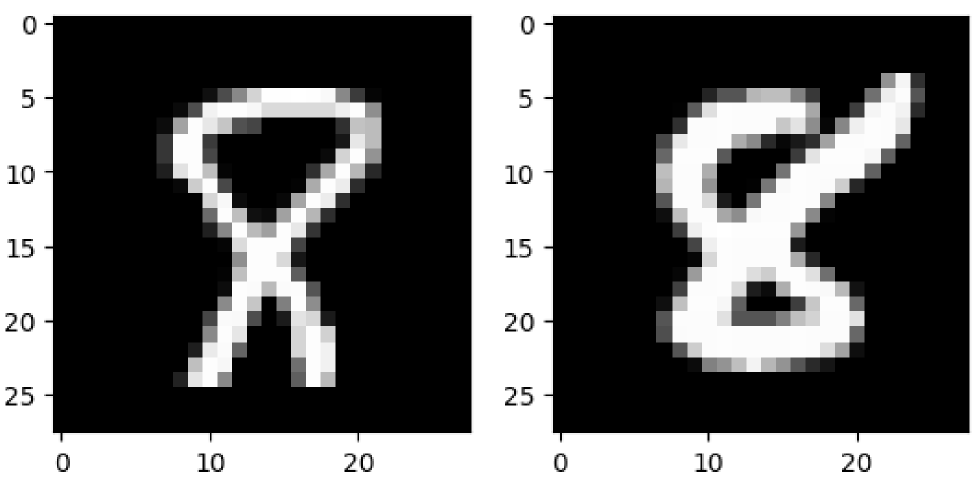
Make all pairs or other strategies; some innovation here

In [ ]:

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

# Generator function to yield batches of image pairs and their labels
def generator(X, y, k=8):
```

```
    unique_labels = np.unique(y)

    while True:
        X1 = []   # List to store first images in pairs
        X2 = []   # List to store second images in pairs
        y_batch = []   # List to store corresponding labels
        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 for positive example
                while i1 == i2:
                    i2 = np.random.choice(label_idx)

                # Create positive example
                X1.append(X[i1][:, :, None])   # Append first image
                X2.append(X[i2][:, :, None])   # Append second image
                y_batch.append(1.0)   # Assign label 1 for positive example

                # 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])   # Append first image
                X2.append(X[i2][:, :, None])   # Append second image
                y_batch.append(0.0)   # Assign label 0 for negative example

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

# For testing the generator
for pair, y in generator(X_test, y_test):
    print('Batch size: ', len(y))
    idx = np.random.choice(range(len(y)))   # Randomly select an index from the batch
    print(pair[0][idx].shape)
    print('Pair label:', y[idx])
    plt.subplot(121)
    plt.imshow(pair[0][idx].reshape(28, 28), cmap='gray')   # Display first image in pair
    plt.subplot(122)
    plt.imshow(pair[1][idx].reshape(28, 28), cmap='gray')   # Display second image in pai
r

    break   # Break after the first batch for testing purposes
```

```
Batch size:  160
(28, 28, 1)
Pair label: 1.0
```



Fit model

In [ ]:

```
history = model.fit(generator(X_train, y_train, k = 32),
                    steps_per_epoch = 10,
                    epochs = 50,
                    validation_data = generator(X_test, y_test, k = 8),
                    validation_steps = 5)
```

```
Epoch 1/50
10/10 [==============================] - 7s 344ms/step - loss: 0.4669 - val_loss: 0.4594
Epoch 2/50
10/10 [==============================] - 3s 332ms/step - loss: 0.2654 - val_loss: 0.4250
Epoch 3/50
10/10 [==============================] - 3s 344ms/step - loss: 0.2353 - val_loss: 0.4201
Epoch 4/50
10/10 [==============================] - 6s 710ms/step - loss: 0.1995 - val_loss: 0.4070
Epoch 5/50
10/10 [==============================] - 3s 358ms/step - loss: 0.1836 - val_loss: 0.3876
Epoch 6/50
10/10 [==============================] - 3s 318ms/step - loss: 0.1793 - val_loss: 0.3740
Epoch 7/50
10/10 [==============================] - 4s 414ms/step - loss: 0.1696 - val_loss: 0.3554
Epoch 8/50
10/10 [==============================] - 4s 466ms/step - loss: 0.1613 - val_loss: 0.3360
Epoch 9/50
10/10 [==============================] - 3s 320ms/step - loss: 0.1539 - val_loss: 0.3181
Epoch 10/50
10/10 [==============================] - 3s 335ms/step - loss: 0.1527 - val_loss: 0.3082
Epoch 11/50
10/10 [==============================] - 4s 403ms/step - loss: 0.1474 - val_loss: 0.2896
Epoch 12/50
10/10 [==============================] - 4s 450ms/step - loss: 0.1416 - val_loss: 0.2879
Epoch 13/50
10/10 [==============================] - 3s 363ms/step - loss: 0.1432 - val_loss: 0.2969
Epoch 14/50
10/10 [==============================] - 3s 321ms/step - loss: 0.1366 - val_loss: 0.2844
Epoch 15/50
10/10 [==============================] - 4s 412ms/step - loss: 0.1329 - val_loss: 0.3454
Epoch 16/50
10/10 [==============================] - 4s 433ms/step - loss: 0.1322 - val_loss: 0.3614
Epoch 17/50
10/10 [==============================] - 3s 349ms/step - loss: 0.1293 - val_loss: 0.3369
Epoch 18/50
10/10 [==============================] - 3s 341ms/step - loss: 0.1268 - val_loss: 0.3297
Epoch 19/50
10/10 [==============================] - 4s 432ms/step - loss: 0.1285 - val_loss: 0.3102
Epoch 20/50
10/10 [==============================] - 4s 459ms/step - loss: 0.1180 - val_loss: 0.3103
Epoch 21/50
10/10 [==============================] - 3s 331ms/step - loss: 0.1189 - val_loss: 0.3224
Epoch 22/50
10/10 [==============================] - 3s 335ms/step - loss: 0.1173 - val_loss: 0.3173
Epoch 23/50
10/10 [==============================] - 4s 448ms/step - loss: 0.1123 - val_loss: 0.2868
Epoch 24/50
10/10 [==============================] - 4s 390ms/step - loss: 0.1135 - val_loss: 0.2885
Epoch 25/50
10/10 [==============================] - 3s 382ms/step - loss: 0.1097 - val_loss: 0.2328
Epoch 26/50
10/10 [==============================] - 3s 362ms/step - loss: 0.1090 - val_loss: 0.2267
Epoch 27/50
10/10 [==============================] - 4s 460ms/step - loss: 0.1078 - val_loss: 0.1952
Epoch 28/50
10/10 [==============================] - 4s 364ms/step - loss: 0.1031 - val_loss: 0.1636
Epoch 29/50
10/10 [==============================] - 3s 344ms/step - loss: 0.1022 - val_loss: 0.1518
Epoch 30/50
10/10 [==============================] - 3s 340ms/step - loss: 0.1029 - val_loss: 0.1333
Epoch 31/50
10/10 [==============================] - 4s 484ms/step - loss: 0.1004 - val_loss: 0.1226
Epoch 32/50
10/10 [==============================] - 4s 375ms/step - loss: 0.1009 - val_loss: 0.1082
Epoch 33/50
10/10 [==============================] - 3s 345ms/step - loss: 0.0951 - val_loss: 0.1104
```

```
Epoch 34/50
10/10 [==============================] - 3s 374ms/step - loss: 0.0941 - val_loss: 0.0975
Epoch 35/50
10/10 [==============================] - 4s 481ms/step - loss: 0.0934 - val_loss: 0.0913
Epoch 36/50
10/10 [==============================] - 4s 370ms/step - loss: 0.0911 - val_loss: 0.0932
Epoch 37/50
10/10 [==============================] - 3s 328ms/step - loss: 0.0915 - val_loss: 0.0778
Epoch 38/50
10/10 [==============================] - 3s 345ms/step - loss: 0.0891 - val_loss: 0.0772
Epoch 39/50
10/10 [==============================] - 5s 504ms/step - loss: 0.0888 - val_loss: 0.0750
Epoch 40/50
10/10 [==============================] - 3s 344ms/step - loss: 0.0884 - val_loss: 0.0683
Epoch 41/50
10/10 [==============================] - 3s 350ms/step - loss: 0.0850 - val_loss: 0.0642
Epoch 42/50
10/10 [==============================] - 3s 374ms/step - loss: 0.0848 - val_loss: 0.0610
Epoch 43/50
10/10 [==============================] - 5s 532ms/step - loss: 0.0817 - val_loss: 0.0674
Epoch 44/50
10/10 [==============================] - 3s 335ms/step - loss: 0.0832 - val_loss: 0.0646
Epoch 45/50
10/10 [==============================] - 3s 373ms/step - loss: 0.0827 - val_loss: 0.0676
Epoch 46/50
10/10 [==============================] - 3s 342ms/step - loss: 0.0787 - val_loss: 0.0559
Epoch 47/50
10/10 [==============================] - 5s 535ms/step - loss: 0.0798 - val_loss: 0.0636
Epoch 48/50
10/10 [==============================] - 3s 350ms/step - loss: 0.0773 - val_loss: 0.0635
Epoch 49/50
10/10 [==============================] - 3s 346ms/step - loss: 0.0787 - val_loss: 0.0607
Epoch 50/50
10/10 [==============================] - 3s 349ms/step - loss: 0.0752 - val_loss: 0.0603
```

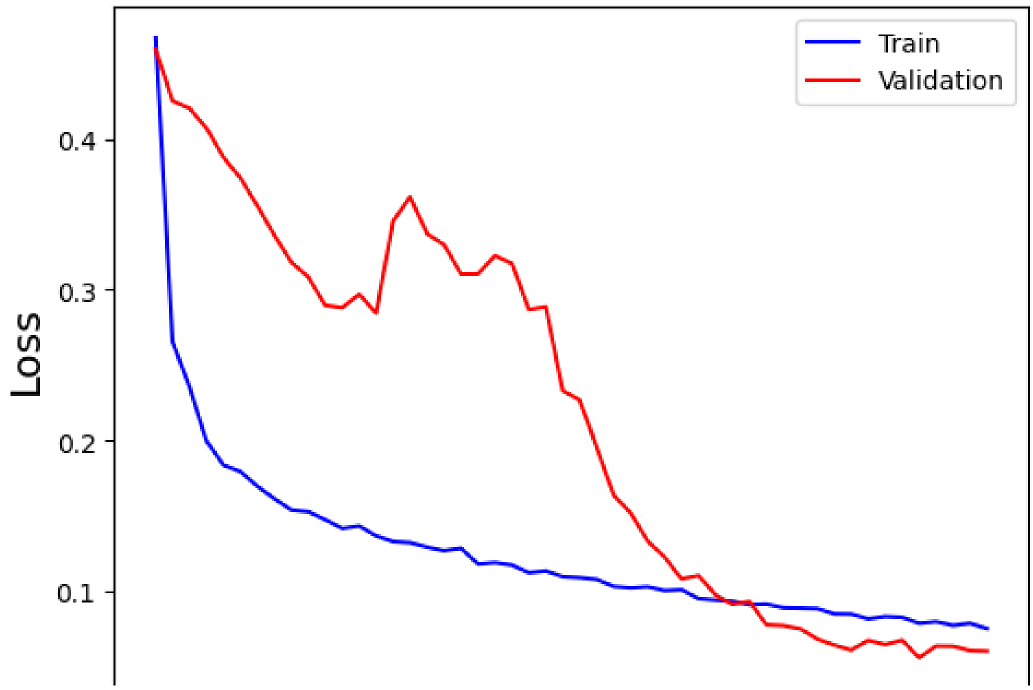
Visualize learning process

In [ ]:

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

Out[ ]:

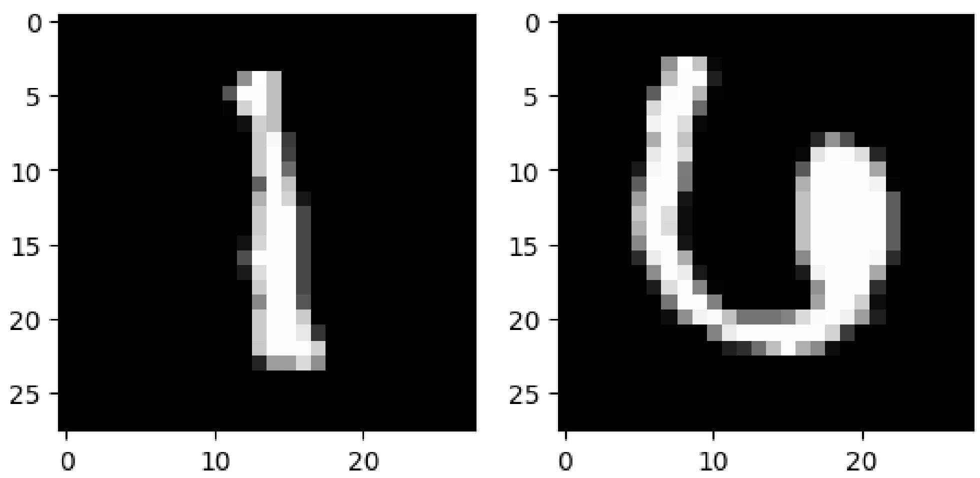
```
Text(0, 0.5, 'Loss')
```



In [ ]:

```
# Loop for generating predictions and visualizations based on the generator
for pair, y in generator(X_test, y_test):
    # Obtain predictions from the model for the image pairs
    y_pred = model.predict(pair)
    print('Batch_size: ', len(y))
    idx = np.random.choice(range(len(y)))  # Randomly select an index from the batch
    print('Pair label:', y[idx])  # Print label for the selected pair
    print('Distance:', y_pred[idx])  # Print the predicted distance for the pair

    # Extract features for both images in the pair using the CNN model
    f1 = cnn(pair[0])  # Features for the first image
    f2 = cnn(pair[1])  # Features for the second image

    # Calculate distance using the extracted features
    d = np.sqrt(np.sum((f1 - f2) ** 2, axis=1, keepdims=True))
    print('Distance by features:', d[idx])  # Print the distance calculated from features
for the pair

    # Visualize the pair of images
    plt.subplot(121)
    plt.imshow(pair[0][idx].reshape(28, 28), cmap='gray')  # Display the first image in
the pair
    plt.subplot(122)
    plt.imshow(pair[1][idx].reshape(28, 28), cmap='gray')  # Display the second image in
the pair

    break  # Break after processing the first batch for visualization purposes
```

```
5/5 [==============================] - 0s 5ms/step
Batch_size:  160
Pair label: 0.0
Distance: [1.2747759]
Distance by features: [1.2747766]
```



Visualize new feature space

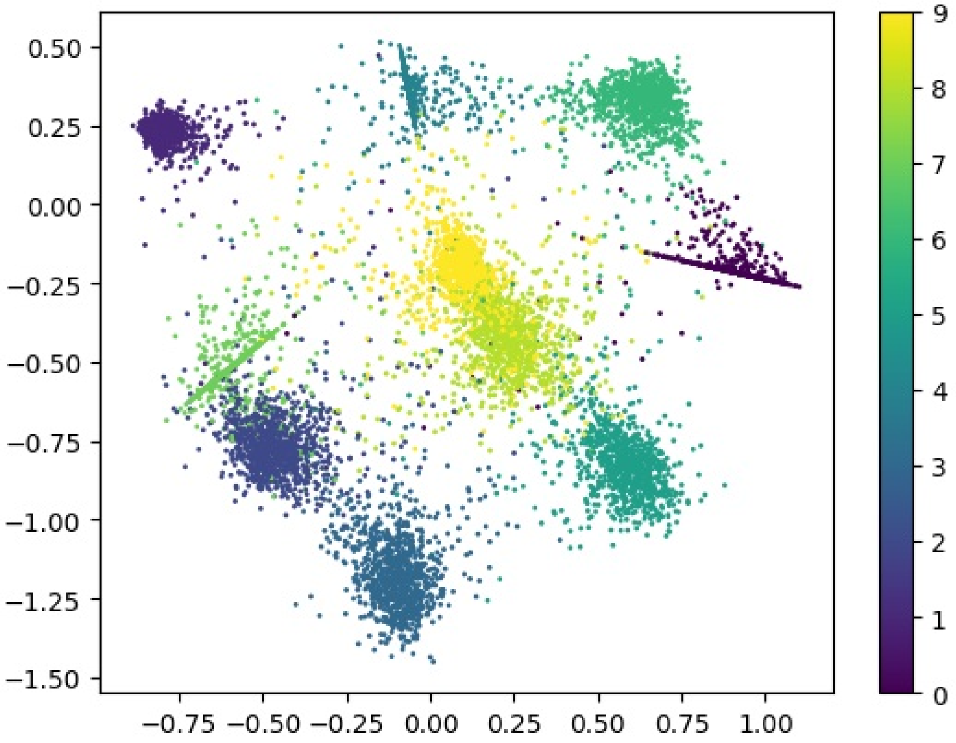
In [ ]:

```
# Using the CNN model to predict features for the test dataset after normalizing
f = cnn.predict(X_test / 255.)

# Creating a scatter plot to visualize the predicted features
p = plt.scatter(f[:, 0], f[:, 1], c=y_test, s=1)  # Scatter plot with x=f[:,0], y=f[:,1]
, color=y_test, point size=1
plt.colorbar(p)  # Adding a color bar to represent the classes or labels
```

```
313/313 [==============================] - 1s 3ms/step
```

Out[ ]:

```
<matplotlib.colorbar.Colorbar at 0x7ee5085da6e0>
```



Save model

In [ ]:

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

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: Y
ou are saving your model as an HDF5 file via `model.save()`. This file format is consider
ed legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model
.keras')`.
  saving_api.save_model(
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be bui
lt. `model.compile_metrics` will be empty until you train or evaluate the model.
```

Load model and test

In [ ]:

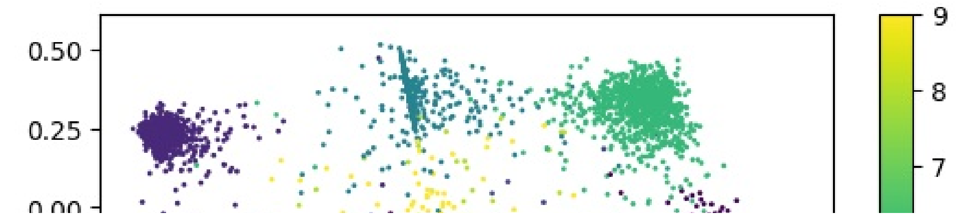
```
from tensorflow.keras.models import load_model
m = load_model('cnn_loss1.h5')

# Creating a scatter plot to visualize the test features
f1 = m.predict(X_test / 255.)
p = plt.scatter(f1[:,0],f1[:,1],c=y_test,s=1)
plt.colorbar(p)
```

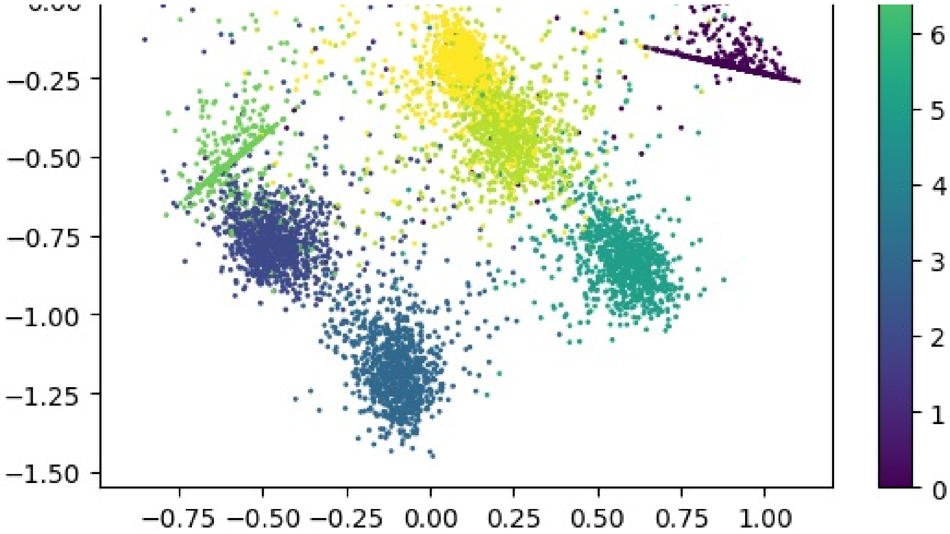
```
WARNING:tensorflow:No training configuration found in the save file, so the model was *no
t* compiled. Compile it manually.
```

```
313/313 [==============================] - 1s 2ms/step
```

Out[ ]:

```
<matplotlib.colorbar.Colorbar at 0x7ee5084b5360>
```





Visualize negative distance and positive distance

In [ ]:

```
# Initializing variables and lists for storing true and predicted labels
i = 0
y_true = []
y_pred = []

# Looping through the generator to calculate distances and collect labels
for pair, y in generator(X_test, y_test):
    # Extracting features for both images in the pair using the CNN model
    f1 = cnn(pair[0])
    f2 = cnn(pair[1])

    # Calculating distance using the extracted features
    d = np.sqrt(np.sum((f1 - f2) ** 2, axis=1, keepdims=True))

    # Adding calculated distances and true labels to the respective lists
    y_pred += list(d.ravel())  # Storing predicted distances
    y_true += list(y)  # Storing true labels

    i += 1  # Incrementing the iteration counter
    if i > 500:  # Breaking the loop after processing 500 iterations for testing purposes
        break
```

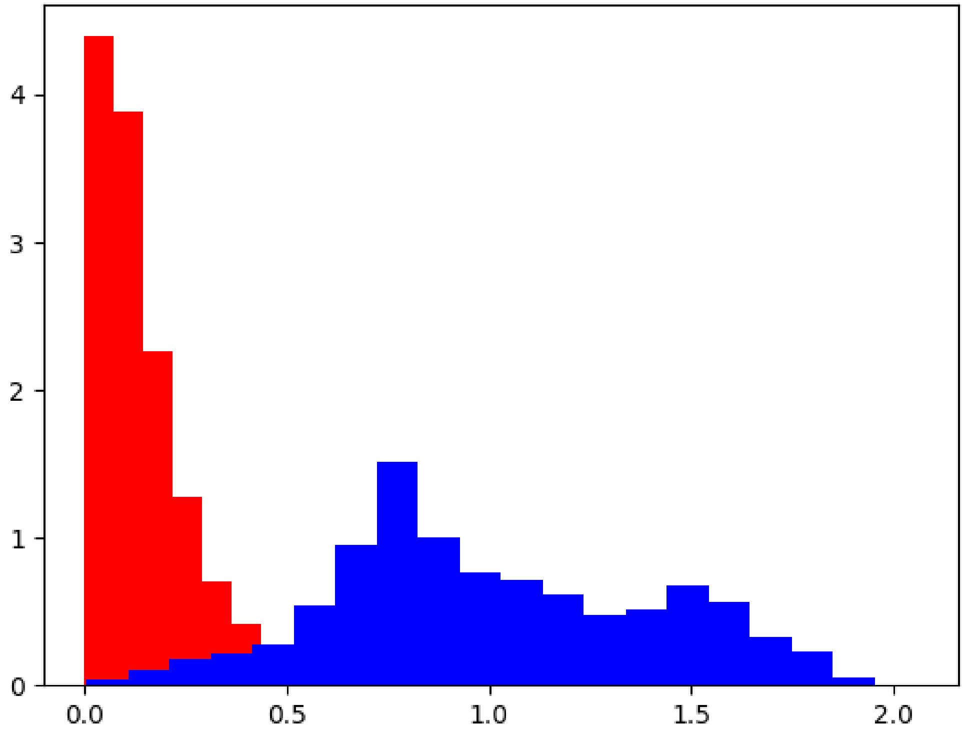
Plot histogram

In [ ]:

```
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)
```

Out[ ]:

```
(array([0.03871327, 0.10859194, 0.17798363, 0.21888825, 0.28219288,
        0.53395097, 0.94640557, 1.50933063, 0.99583195, 0.76525668,
        0.71047375, 0.61429927, 0.48135936, 0.52080246, 0.68223011,
        0.56073374, 0.33064543, 0.22521872, 0.05283509, 0.00292176]),
 array([0.00551254, 0.10798556, 0.21045859, 0.31293163, 0.41540465,
        0.5178777 , 0.62035072, 0.72282374, 0.82529676, 0.92776978,
        1.0302428 , 1.13271582, 1.23518884, 1.33766186, 1.440135  ,
        1.54260802, 1.64508104, 1.74755406, 1.85002708, 1.9525001 ,
        2.05497313]),
 <BarContainer object of 20 artists>)
```



Check report using sklearn

In [ ]:

```
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       0.96      0.92      0.94     40080
         1.0       0.92      0.96      0.94     40080

    accuracy                           0.94     80160
   macro avg       0.94      0.94      0.94     80160
weighted avg       0.94      0.94      0.94     80160
```