Specific Test IV. Exploring Equivariant Neural Networks

Task: Use an Equivariant Neural Network of your choice to build a robust and efficient model for binary classification or unsupervised anomaly detection on the provided dataset. In the case of unsupervised anomaly detection, train your model to learn the distribution of the provided strong lensing images with no substructure. Please implement your approach in PyTorch or Keras and discuss your strategy.

Dataset: https://drive.google.com/file/d/16Y1taQoTeUTP5rGpB0tuPZ_S30acvnqr/view?usp=sharing

Dataset Description: A set of simulated strong gravitational lensing images with and without substructure.

Evaluation Metrics: ROC curve (Receiver Operating Characteristic curve) and AUC score (Area Under the ROC Curve)\

Downloading the data:

```
from google.colab import drive
drive.mount('/content/gdrive')
!tar --extract --file '/content/gdrive/MyDrive/lenses.tgz'
print('Extraction done.')
```

```
Mounted at /content/gdrive
Extraction done.
```

Setting up imports:

```
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import auc, roc_curve
```

Extracting the data from the lense images:

```
X_data = []
Y_data = []

#substructure data
```

```
sub = os.listdir('/content/lenses/sub')
for i in sub:
    img = cv2.imread('/content/lenses/sub/' + i)
    img = img / 255.0
    X_data.append(img)
    Y_data.append(1)

#no-substructure data
no_sub = os.listdir('/content/lenses/no_sub')
for i in no_sub:
    img = cv2.imread('/content/lenses/no_sub/' + i)
    img = img / 255.0
    X_data.append(img)
    Y_data.append(0)
```

Shuffling to introduce randomness in the data:

```
data = list(zip(X_data, Y_data))
np.random.shuffle(data)
X_data, Y_data = zip(*data)

#delete to free redundant space
del data

X_data = np.array(X_data)
Y_data = np.array(Y_data)
```

Visualising the images belonging to the two classes:

```
classes = ["Substructured", "No substructure"]
plt.figure(figsize=(15, 15))
for i in range(16):
    plt.subplot(4, 4, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    num = np.random.randint(0, len(X_data))
    plt.imshow(X_data[num])
    plt.xlabel(classes[Y_data[num]])
plt.show()
```

Splitting the data into training and validation:

```
X_train , X_test, Y_train, Y_test = train_test_split(X_data, Y_data, test_size=0.2, random_st
X_data.shape, Y_data.shape
```

```
((10000, 150, 150, 3), (10000,))
```

Deleting large variables to free up memory:

```
del X_data, Y_data
del img, no_sub, sub
```

Defining the model and compiling:

```
model = tf.keras.applications.ResNet50V2(
    include_top=True,
    input_shape=(150, 150, 3),
    weights=None,
    classes=2,
    classifier_activation='softmax'
)

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

model.summary()
```

```
 conv5_block2_2_pad (ZeroPaddin  (None, 7, 7, 512)   0           ['conv5_block2_1_rel
 g2D)

 conv5_block2_2_conv (Conv2D)   (None, 5, 5, 512)    2359296     ['conv5_block2_2_pad

 conv5_block2_2_bn (BatchNormal  (None, 5, 5, 512)   2048        ['conv5_block2_2_conv
 ization)

 conv5_block2_2_relu (Activatio  (None, 5, 5, 512)   0           ['conv5_block2_2_bn[0
 n)

 conv5_block2_3_conv (Conv2D)   (None, 5, 5, 2048)   1050624     ['conv5_block2_2_rel

 conv5_block2_out (Add)         (None, 5, 5, 2048)   0           ['conv5_block1_out[0
                                                                  'conv5_block2_3_conv

 conv5_block3_preact_bn (BatchN  (None, 5, 5, 2048)  8192        ['conv5_block2_out[0
 ormalization)

 conv5_block3_preact_relu (Acti  (None, 5, 5, 2048)  0           ['conv5_block3_preact
 vation)

 conv5_block3_1_conv (Conv2D)   (None, 5, 5, 512)    1048576     ['conv5_block3_preact
                                                                 ]

 conv5_block3_1_bn (BatchNormal  (None, 5, 5, 512)   2048        ['conv5_block3_1_conv
 ization)

 conv5_block3_1_relu (Activatio  (None, 5, 5, 512)   0           ['conv5_block3_1_bn[0
 n)

 conv5_block3_2_pad (ZeroPaddin  (None, 7, 7, 512)   0           ['conv5_block3_1_rel
 g2D)

 conv5_block3_2_conv (Conv2D)   (None, 5, 5, 512)    2359296     ['conv5_block3_2_pad

 conv5_block3_2_bn (BatchNormal  (None, 5, 5, 512)   2048        ['conv5_block3_2_conv
 ization)
```

```
conv5_block3_2_relu (Activatio  (None, 5, 5, 512)   0           ['conv5_block3_2_bn[0
n)

conv5_block3_3_conv (Conv2D)   (None, 5, 5, 2048)   1050624     ['conv5_block3_2_rel

conv5_block3_out (Add)         (None, 5, 5, 2048)   0           ['conv5_block2_out[0
                                                                 'conv5_block3_3_conv

post_bn (BatchNormalization)   (None, 5, 5, 2048)   8192        ['conv5_block3_out[0

post_relu (Activation)         (None, 5, 5, 2048)   0           ['post_bn[0][0]']

avg_pool (GlobalAveragePooling  (None, 2048)        0           ['post_relu[0][0]']
2D)

predictions (Dense)            (None, 2)            4098        ['avg_pool[0][0]']
```

Defining a callback to save the best weight for using it in ROC curve:

```
checkpoint_filepath = 'weights.{epoch:02d}-{val_loss:.2f}.h5'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    monitor='val_accuracy',
    mode='max',
    verbose=1,
    save_best_only=True
)
```

Fitting and training the model:

```
model.fit(
    X_train,
    Y_train,
    epochs=20,
    validation_data=(X_test, Y_test),
    callbacks=[model_checkpoint_callback]
)
```

```
Epoch 6: val_accuracy did not improve from 0.54800
250/250 [==============================] - 92s 370ms/step - loss: 0.1765 - accuracy: 
Epoch 7/20
250/250 [==============================] - ETA: 0s - loss: 0.1602 - accuracy: 0.9409
Epoch 7: val_accuracy did not improve from 0.54800
250/250 [==============================] - 92s 370ms/step - loss: 0.1602 - accuracy: 
Epoch 8/20
250/250 [==============================] - ETA: 0s - loss: 0.1434 - accuracy: 0.9501
Epoch 8: val_accuracy improved from 0.54800 to 0.97300, saving model to weights.08-0.
250/250 [==============================] - 94s 375ms/step - loss: 0.1434 - accuracy: 
```

```
Epoch 9/20
250/250 [==============================] - ETA: 0s - loss: 0.1260 - accuracy: 0.9538
Epoch 9: val_accuracy did not improve from 0.97300
250/250 [==============================] - 92s 368ms/step - loss: 0.1260 - accuracy: 
Epoch 10/20
250/250 [==============================] - ETA: 0s - loss: 0.1293 - accuracy: 0.9516
Epoch 10: val_accuracy did not improve from 0.97300
250/250 [==============================] - 92s 368ms/step - loss: 0.1293 - accuracy: 
Epoch 11/20
250/250 [==============================] - ETA: 0s - loss: 0.1029 - accuracy: 0.9628
Epoch 11: val_accuracy did not improve from 0.97300
250/250 [==============================] - 92s 369ms/step - loss: 0.1029 - accuracy: 
Epoch 12/20
250/250 [==============================] - ETA: 0s - loss: 0.0987 - accuracy: 0.9651
Epoch 12: val_accuracy did not improve from 0.97300
250/250 [==============================] - 92s 367ms/step - loss: 0.0987 - accuracy: 
Epoch 13/20
250/250 [==============================] - ETA: 0s - loss: 0.0896 - accuracy: 0.9680
Epoch 13: val_accuracy did not improve from 0.97300
250/250 [==============================] - 92s 369ms/step - loss: 0.0896 - accuracy: 
Epoch 14/20
250/250 [==============================] - ETA: 0s - loss: 0.0948 - accuracy: 0.9678
Epoch 14: val_accuracy did not improve from 0.97300
250/250 [==============================] - 92s 369ms/step - loss: 0.0948 - accuracy: 
Epoch 15/20
250/250 [==============================] - ETA: 0s - loss: 0.0784 - accuracy: 0.9724
Epoch 15: val_accuracy did not improve from 0.97300
250/250 [==============================] - 92s 369ms/step - loss: 0.0784 - accuracy: 
Epoch 16/20
250/250 [==============================] - ETA: 0s - loss: 0.0926 - accuracy: 0.9670
Epoch 16: val_accuracy did not improve from 0.97300
250/250 [==============================] - 92s 368ms/step - loss: 0.0926 - accuracy: 
Epoch 17/20
250/250 [==============================] - ETA: 0s - loss: 0.0803 - accuracy: 0.9739
Epoch 17: val_accuracy did not improve from 0.97300
250/250 [==============================] - 92s 369ms/step - loss: 0.0803 - accuracy: 
Epoch 18/20
250/250 [==============================] - ETA: 0s - loss: 0.0741 - accuracy: 0.9732
Epoch 18: val_accuracy did not improve from 0.97300
250/250 [==============================] - 92s 368ms/step - loss: 0.0741 - accuracy: 
Epoch 19/20
250/250 [==============================] - ETA: 0s - loss: 0.0706 - accuracy: 0.9737
Epoch 19: val_accuracy did not improve from 0.97300
250/250 [==============================] - 92s 368ms/step - loss: 0.0706 - accuracy: 
Epoch 20/20
250/250 [==============================] - ETA: 0s - loss: 0.0849 - accuracy: 0.9725
Epoch 20: val_accuracy did not improve from 0.97300
```

Delete to free up memory:

```
del X_train, Y_train
```

Predict on the validation data and load the best saved weight:

```
model.load_weights('weights.08-0.08.h5')
predictions = model.predict(X_test)
temp_predictions·=·[]
for·i·in·range(len(predictions)):
····k·=·np.argmax(predictions[i])
    temp_predictions.append(k)

temp_predictions = np.array(temp_predictions)
```

Plotting the ROC AUC curve:

```
fpr, tpr, thresholds = roc_curve(Y_test, temp_predictions)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

✓ 0s completed at 9:22 PM