EN4553 – Deep Learning for Vision

University of Moratuwa, Sri Lanka

Assignment 02

Name	Index No
Anuradha A.K.	200041E
Chandrasiri Y.U.K.K.	200087A
Kannangara N.V.	200285E
Ranamukage R.R.A.	200505G

Question 01

Broadcasting:

In NumPy, broadcasting is a feature that allows arithmetic operations on arrays with different shapes. It enables smaller arrays to be automatically expanded to match the shape of larger arrays, allowing for efficient and concise computations without the need for explicit looping.

When carrying out operations on two arrays, NumPy compares their shapes element by element, beginning with the trailing dimensions. Two dimensions are considered compatible if they are equal, or one of them is 1. Then the smaller array broadcasts across the larger array so that they have compatible shapes.

If neither condition is met, a `ValueError` is raised to indicate that the shapes are incompatible.

Example:

```
a = np.array ( [ 1.0, 2.0, 3.0 ] )
b = 2.0
result = a * b
print(result)
```

Output: [2. 4. 6.]

In this example, the scalar **b** is broadcasted to match the shape of array **a**, resulting in element-wise multiplication

Numpy Broadcasting Rules:

When operating on two arrays, NumPy compares their shapes element-wise, starting from the trailing dimensions. Two dimensions are compatible if they are equal or if one of them is 1. If these conditions are not met, a ValueError is raised.

Operation	a.shape	b.shape	Output shape
a + b	(256, 256, 3)	(3,)	(256, 256, 3)
a – b	(10, 1, 6, 1)	(9, 1, 7)	(10, 9, 6, 7)
a * b	(2,)	(4,)	Error
a / b	(4, 1)	(8 , 5, 3)	Error
a - b	(5, 3, 2)	(5, 1, 2)	(5, 3, 2)
a - b.mean()	(128, 128, 3)	(256, 256, 5)	(128, 128, 3)
a - b.mean(axis=(1, 2))	(3, 256, 256)	(3, 256, 256)	Error
a - b.mean(axis=(1, 2), keepdims=True)	(3, 256, 256)	(3, 256, 256)	(3, 256, 256)
np.matmul(a, b)	(6, 5, 3)	(3, 4)	(6, 5, 4)

Question 02

Python function to compute the pairwise squared Euclidean distances between the rows of the two matrices is as follows:

```
def pairwise_squared_euclidean_distance(X, Y):
    # Compute squared norm of each row in X and reshape to column vector
    X_norm_squared = np.sum(X ** 2, axis=1).reshape(-1, 1)

    # Compute squared norm of each row in Y and reshape to row vector
    Y_norm_squared = np.sum(Y ** 2, axis=1).reshape(1, -1)

    # Compute dot product between X and Y
    XY_dot_product = np.dot(X, Y.T)

    # Compute the squared Euclidean distance
    Z = X_norm_squared + Y_norm_squared - 2 * XY_dot_product

    return Z
```

Question 03

a) Accuracy on the test set for KNN-classification using Resnet50 as the pretrained network is as follows:

```
Accuracy of k-NN classification: 0.7503
```

b) Accuracy on the test set for linear-classification using InceptionV3 as the pretrained network is as follows:

```
90/90 ━━━━━━━━━━━━━━━━━━━━ 28s 149ms/step - accuracy: 0.9135 - loss: 0.4928

Test Loss: 0.3943

Test Accuracy: 0.9270
```

c) For this classification task, we also use InceptionV3. To improve accuracy, we applied techniques like horizontal flipping, random zoom, Gaussian blur, and rotation. The model was designed as follows:

```
base_model = InceptionV3(weights='imagenet', include_top=False)

transfer_learning_arch = base_model.output
transfer_learning_arch =
GlobalAveragePooling2D()(transfer_learning_arch)
transfer_learning_arch = Dense(1024,
activation='relu')(transfer_learning_arch)
transfer_learning_arch = Dropout(0.4)(transfer_learning_arch)
transfer_learning_arch = Dense(512,
activation='relu')(transfer_learning_arch)
transfer_learning_arch = Dropout(0.4)(transfer_learning_arch)
predictions = Dense(101,
activation='softmax')(transfer_learning_arch)

transfer_learning_model = Model(inputs=base_model.input,
outputs=predictions)
```

Accuracy for the test set as follows:

```
90/90 ━━━━━━━━━━━━━━━━━━━━ 11s 51ms/step - accuracy: 0.7961 - loss: 1.4244
Test Loss: 1.4658
Test Accuracy: 0.7978
```

Note: More details about Question 2 and Question 3 can be found in the notebooks.