Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN4554 - Deep Learning for Vision

Assignment 02

Abeygunathilaka T. L. - 200003P

De Silva W. H. P. - 200114G

Haputhanthri H. H. A. M. - 200207U

Ranasingha A. S. N. - 200507N

Part 01

When broadcasting two arrays, NumPy compares their shapes, starting from the rightmost dimension to the left. Two dimensions are compatible when;

- They are equal, or
- One of them is 1

The number of dimensions need not be equal. In unequal dimension count cases, they are "stretched" to make the computation compatible.

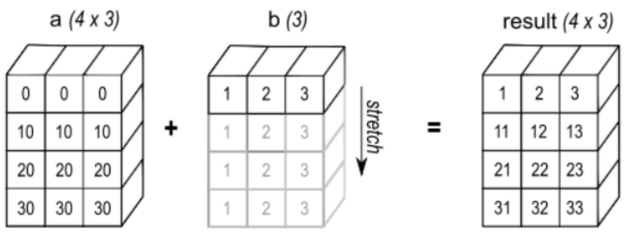


Figure 01: [Unequal dimension count]

I. a + b → (256,256,3)
II. a - b → (10,9,6,7)
III. a * b → Error
IV. a / b → Error
V. a - b → (5,3,2)
VI. a - b.mean() → (128,128,3)
VII. a - b.mean(axis=(1,2)) → Error
VIII. a - b.mean(axis=(1,2), keepdims=True) → (3,256,256)
IX. np.matmul(a,b) → (6,5,4)

Part 02

The formula for the squared Euclidean distance between two vectors x_i and y_j is given by;

$$||x_i - y_j||^2 = \sum_{k=1}^{d} x_{i,k}^{\ 2} + \sum_{k=1}^{d} y_{j,k}^{\ 2} - 2\sum_{k=1}^{d} x_{i,k}\, y_{j,k}$$

```python
import numpy as np

def pairwise_squared_euclidean3(X, Y):
    # Compute the squared norms for each row in X and Y using broadcasting
    X_norm = np.sum(X**2, axis=1)  # Shape (m,)
    Y_norm = np.sum(Y**2, axis=1)  # Shape (n,)

    # Compute the pairwise squared Euclidean distances using broadcasting and einsum
    Z = X_norm[:, np.newaxis] + Y_norm - 2 * np.einsum('ij,kj->ik', X, Y)

    # Ensure non-negative distances (can be slightly negative due to floating-point precision)
    Z = np.maximum(Z, 0)

    return Z
```

Results of the function for arbitrary x,y matrices,

```
x = np.array([[1,2,3],[2,3,4],[4,5,6]])
y = np.array([[2,4,5],[7,6,4],[1,1,1],[4,3,2]])
print(pairwise_squared_euclidean3(x, y))
```

✓ 0.0s

```
[[ 9 53  5 11]
 [ 2 34 14  8]
 [ 6 14 50 20]]
```

Part 03

(a)

Find optimal k using inbuilt functions

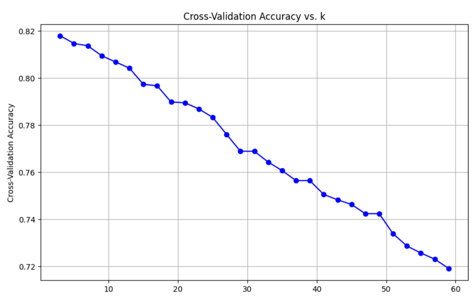
```
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, train_embeddings, train_labels, cv=3, scoring='accuracy')
    cv_scores.append(scores.mean())
optimal_k = k_values[np.argmax(cv_scores)]
print(f"Optimal value of k: {optimal_k}")
```

Find optimal k using custom functions and find accuracy

```
def pairwise_squared_euclidean_distance(X, Y):
    X_sq_norms = np.sum(X ** 2, axis=1).reshape(-1, 1)
    Y_sq_norms = np.sum(Y ** 2, axis=1).reshape(1, -1)
    cross_term = np.dot(X, Y.T)
    dists = X_sq_norms + Y_sq_norms - 2 * cross_term
    return dists
```

```
def knn_predict(X_train, train_labels, X_test, k=5):
    dists = pairwise_squared_euclidean_distance(X_test, X_train)
    num_test_samples = X_test.shape[0]
    y_pred = np.zeros(num_test_samples, dtype=int)
    for i in range(num_test_samples):
        nearest_neighbors = np.argsort(dists[i])[:k]
        nearest_labels = train_labels[nearest_neighbors]
        unique, counts = np.unique(nearest_labels, return_counts=True)
        y_pred[i] = unique[np.argmax(counts)]
    return y_pred
```

Performed 3-fold cross-validation to find the optimum value of k. The optimum k value is 3.



The accuracy on the test set is 84.16% when k = 3

(b) Linear classification

```python
# Load train and test embeddings
train_embeddings = np.load('/content/drive/MyDrive/Colab Notebooks/embeddings/train_embeddings.npy')
train_labels = np.load('/content/drive/MyDrive/Colab Notebooks/embeddings/train_labels.npy')
test_embeddings = np.load('/content/drive/MyDrive/Colab Notebooks/embeddings/test_embeddings.npy')
test_labels = np.load('/content/drive/MyDrive/Colab Notebooks/embeddings/test_labels.npy')

# Train a Logistic Regression model (equivalent to the last fully connected layer)
log_reg = LogisticRegression(solver='lbfgs', max_iter=1000)

# Fit the logistic regression model on the training embeddings
log_reg.fit(train_embeddings, train_labels)

# Predict on the test embeddings
test_predictions = log_reg.predict(test_embeddings)

# Calculate accuracy
accuracy = accuracy_score(test_labels, test_predictions)
print(f'Accuracy on the test set (Logistic Regression): {accuracy * 100:.2f}%')
```

```
Accuracy on the test set (Logistic Regression): 92.41%
```

(c) Part C is written in PyTorch separately in Kaggle

(i) Initiating the model

```python
class ImageModel(nn.Module):
    def __init__(self):
        super(ImageModel, self).__init__()
```

(ii) Forward function

```python
def forward(self, img):
    feats = self.features(img)
    x = self.flatten(feats)
    x = self.dense(x)

    out = F.softmax(x, dim=1)
    return out
```

(iii) Model training

```python
for (img, label) in train_loader:
    img, label = img.to(device), label.to(device)
    one_hot_labels = one_hot_encode(label)

    optimizer.zero_grad()

    pred = model(img)
    loss = criterion(pred, one_hot_labels)
    loss.backward()

    optimizer.step()
    batch_loss+= loss.item()

    with torch.no_grad():
        y_pred = torch.argmax(pred, dim=1)
        total += label.size(0)  # Update total count
        correct += (y_pred == label).sum().item()

train_loss.append(batch_loss/len(train_loader))
train_acc. append(correct/total)
```

(iv) Test accuracy evaluation

```python
with torch.no_grad():
    for (img, label) in test_loader:
        img, label = img.to(device), label.to(device)
        one_hot_labels = one_hot_encode(label)

        pred = model(img)
        loss = criterion(pred, one_hot_labels)
        batch_loss+=loss.item()

        y_pred = torch.argmax(pred, dim=1)
        total += label.size(0)  # Update total count
        correct += (y_pred == label).sum().item()

    test_loss.append(batch_loss/len(test_loader))
    test_acc. append(correct/total)
```

(v) Training accuracy logs

```
epoch 0 | 100 train_loss= 4.3217 , train_acc= 0.3190, test_loss: 4.5038, test_acc= 0.1210
epoch 1 | 100 train_loss= 4.2841 , train_acc= 0.3491, test_loss: 4.3238, test_acc= 0.3128
epoch 2 | 100 train_loss= 4.2683 , train_acc= 0.3645, test_loss: 4.2813, test_acc= 0.3546
epoch 3 | 100 train_loss= 4.2653 , train_acc= 0.3677, test_loss: 4.2614, test_acc= 0.3771
epoch 4 | 100 train_loss= 4.2584 , train_acc= 0.3731, test_loss: 4.3066, test_acc= 0.3332
epoch 5 | 100 train_loss= 4.2616 , train_acc= 0.3724, test_loss: 4.3070, test_acc= 0.3294
epoch 6 | 100 train_loss= 4.2397 , train_acc= 0.3928, test_loss: 4.2693, test_acc= 0.3674
epoch 7 | 100 train_loss= 4.2304 , train_acc= 0.4023, test_loss: 4.2426, test_acc= 0.3944
epoch 8 | 100 train_loss= 4.2261 , train_acc= 0.4058, test_loss: 4.2434, test_acc= 0.3930
epoch 9 | 100 train_loss= 4.2308 , train_acc= 0.4030, test_loss: 4.2600, test_acc= 0.3775
epoch 10 | 100 train_loss= 4.2233 , train_acc= 0.4089, test_loss: 4.2320, test_acc= 0.4044
epoch 11 | 100 train_loss= 4.2166 , train_acc= 0.4167, test_loss: 4.2232, test_acc= 0.4117
epoch 12 | 100 train_loss= 4.2121 , train_acc= 0.4220, test_loss: 4.2154, test_acc= 0.4207
epoch 13 | 100 train_loss= 4.2018 , train_acc= 0.4312, test_loss: 4.2304, test_acc= 0.4044
epoch 14 | 100 train_loss= 4.1997 , train_acc= 0.4329, test_loss: 4.2327, test_acc= 0.4055
epoch 15 | 100 train_loss= 4.1887 , train_acc= 0.4440, test_loss: 4.1905, test_acc= 0.4438
epoch 16 | 100 train_loss= 4.1759 , train_acc= 0.4573, test_loss: 4.1914, test_acc= 0.4431
epoch 17 | 100 train_loss= 4.1706 , train_acc= 0.4623, test_loss: 4.2019, test_acc= 0.4293
epoch 18 | 100 train_loss= 4.1721 , train_acc= 0.4632, test_loss: 4.1867, test_acc= 0.4487
epoch 19 | 100 train_loss= 4.1658 , train_acc= 0.4677, test_loss: 4.2015, test_acc= 0.4355
epoch 20 | 100 train_loss= 4.1611 , train_acc= 0.4699, test_loss: 4.1924, test_acc= 0.4380
```