CV LAB ASSIGNMENT-7

121CS0755
Rachana Dubey

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import accuracy_score
from keras.optimizers import Adam

from sklearn.ensemble import RandomForestClassifier
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated.
```

```
mnist = keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

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

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
X_train = x_train.reshape((-1, 28, 28, 1))
x_test = x_test.reshape((-1, 28, 28, 1))
```

..

.

a) **Model 1**

```
build_model_1 = keras.Sequential([
layers.Conv2D(32, (3, 3), strides=(2, 2), activation='relu', input_shape=(28, 28, 1)),
layers .MaxPooling2D((2, 2), strides=(1, 1)),
layers.Conv2D(16, (4, 4), strides=(2, 2), activation='relu'),
layers .MaxPooling2D((4, 4), strides=(2, 2)),
layers.Flatten(),
layers.Dense(8, activation='relu'),
layers.Dense(10, activation='softmax')

])

learning_rate = 0.001 # Adjust this value as needed

optimizer = Adam(learning_rate=learning_rate)

build_model_1.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy',metrics=['accuracy'])
build_model_1.summary()
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.com

WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Pl

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 13, 13, 32)        320       
                                                                 
 max_pooling2d (MaxPooling2  (None, 12, 12, 32)        0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 5, 5, 16)          8208      
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 1, 1, 16)          0         
 g2D)                                                            
                                                                 
 flatten (Flatten)           (None, 16)                0         
                                                                 
 dense (Dense)               (None, 8)                 136       
                                                                 
 dense_1 (Dense)             (None, 10)                90        
                                                                 
=================================================================
Total params: 8754 (34.20 KB)
Trainable params: 8754 (34.20 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
model1_accuracies = []
for i in range(5):

    print(f"Training model1 {i+1}...")
    build_model_1.fit(x_train, y_train, epochs=5,batch_size=128)

    print(f"Evaluating model1 {i+1}...")
    test_loss_m1, test_accuracy_m1 = build_model_1.evaluate(x_test, y_test, verbose=2)
    model1_accuracies.append(test_accuracy_m1)

mean_test_accuracy_m1 = np.mean(model1_accuracies)
print("Mean test accuracy of Model 1:", mean_test_accuracy_m1)
```

```
Training model1 1...
Epoch 1/5
469/469 [==============================] - 3s 6ms/step - loss: 0.1055 - accuracy: 0.9672
Epoch 2/5
469/469 [==============================] - 3s 6ms/step - loss: 0.1010 - accuracy: 0.9689
Epoch 3/5
469/469 [==============================] - 3s 6ms/step - loss: 0.1010 - accuracy: 0.9681
Epoch 4/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0985 - accuracy: 0.9686
Epoch 5/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0947 - accuracy: 0.9708
Evaluating model1 1...
313/313 - 0s - loss: 0.1294 - accuracy: 0.9627 - 396ms/epoch - 1ms/step
Training model1 2...
Epoch 1/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0947 - accuracy: 0.9706
Epoch 2/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0934 - accuracy: 0.9702
Epoch 3/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0919 - accuracy: 0.9710
Epoch 4/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0889 - accuracy: 0.9721
Epoch 5/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0888 - accuracy: 0.9727
Evaluating model1 2...
313/313 - 0s - loss: 0.1363 - accuracy: 0.9599 - 393ms/epoch - 1ms/step
Training model1 3...
```

```
Epoch 1/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0879 - accuracy: 0.9723
Epoch 2/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0860 - accuracy: 0.9735
Epoch 3/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0859 - accuracy: 0.9735
Epoch 4/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0848 - accuracy: 0.9732
Epoch 5/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0825 - accuracy: 0.9740
Evaluating model1 3...
313/313 - 0s - loss: 0.1681 - accuracy: 0.9515 - 404ms/epoch - 1ms/step
Training model1 4...
Epoch 1/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0823 - accuracy: 0.9736
Epoch 2/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0819 - accuracy: 0.9746
Epoch 3/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0791 - accuracy: 0.9753
Epoch 4/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0776 - accuracy: 0.9760
Epoch 5/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0766 - accuracy: 0.9760
Evaluating model1 4...
313/313 - 0s - loss: 0.1243 - accuracy: 0.9637 - 457ms/epoch - 1ms/step
Training model1 5...
Epoch 1/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0757 - accuracy: 0.9763
Epoch 2/5
469/469 [==============================] - 3s 7ms/step - loss: 0.0754 - accuracy: 0.9761
Epoch 3/5
```

```
# Extract the features from second Last fully connected Layer (having 8 neurons)
feature_extractor_m1 = keras.Model(inputs=build_model_1.inputs, outputs=build_model_1.layers[-2].output)

#predict the feature train and test
train_features_m1 = feature_extractor_m1.predict(x_train)
test_features_m1 = feature_extractor_m1.predict(x_test)

# Model features using Random Forest classifier
rf_classifier_m1 = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier_m1.fit(train_features_m1, y_train)

# Predict using the Random Forest classifier
predictions_m1 = rf_classifier_m1.predict(test_features_m1)

# Calculate accuracy
accuracy_m1 = accuracy_score(y_test, predictions_m1)
print("Accuracy of Random Forest classifier on extracted features model 1:", accuracy_m1)
```

```
1875/1875 [==============================] - 2s 1ms/step
313/313 [==============================] - 0s 1ms/step
Accuracy of Random Forest classifier on extracted features model 1: 0.9608
```

b) **Model 2**

```
build_model_2 = keras.Sequential([
layers.Conv2D(32, (3, 3), strides=(2, 2), activation='relu', input_shape=(28, 28, 1)),
layers .AveragePooling2D((2, 2), strides=(1, 1)),
layers.Conv2D(16, (4, 4), strides=(2, 2), activation='relu'),
layers .AveragePooling2D((4, 4), strides=(2, 2)),
layers.Flatten(),
layers.Dense(8, activation='relu'),
layers.Dense(10, activation='softmax')

])

learning_rate = 0.001 # Adjust this value as needed

optimizer = Adam(learning_rate=learning_rate)

build_model_2.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy',metrics=['accuracy'])
build_model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_2 (Conv2D)           (None, 13, 13, 32)        320       
                                                                 
 average_pooling2d (Average  (None, 12, 12, 32)        0         
 Pooling2D)                                                      
                                                                 
 conv2d_3 (Conv2D)           (None, 5, 5, 16)          8208      
                                                                 
 average_pooling2d_1 (Avera  (None, 1, 1, 16)          0         
 gePooling2D)                                                    
                                                                 
 flatten_1 (Flatten)         (None, 16)                0         
                                                                 
 dense_2 (Dense)             (None, 8)                 136       
                                                                 
 dense_3 (Dense)             (None, 10)                90        
                                                                 
=================================================================
Total params: 8754 (34.20 KB)
Trainable params: 8754 (34.20 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model2_accuracies = []
for i in range(5):

    print(f"Training model1 {i+1}...")
    build_model_2.fit(x_train, y_train, epochs=5,batch_size=128)

    print(f"Evaluating model1 {i+1}...")
    test_loss_m2, test_accuracy_m2 = build_model_2.evaluate(x_test, y_test, verbose=2)
    model2_accuracies.append(test_accuracy_m2)

mean_test_accuracy_m2 = np.mean(model2_accuracies)
print("Mean test accuracy of Model 1:", mean_test_accuracy_m2)
```

```
469/469 [==============================] - 3s 6ms/step - loss: 0.5481 - accuracy: 0.8246
Epoch 2/5
469/469 [==============================] - 3s 6ms/step - loss: 0.5118 - accuracy: 0.8368
Epoch 3/5
469/469 [==============================] - 3s 6ms/step - loss: 0.4794 - accuracy: 0.8484
Epoch 4/5
469/469 [==============================] - 3s 6ms/step - loss: 0.4512 - accuracy: 0.8584
Epoch 5/5
469/469 [==============================] - 3s 6ms/step - loss: 0.4292 - accuracy: 0.8665
Evaluating model1 2...
313/313 - 0s - loss: 0.3957 - accuracy: 0.8771 - 439ms/epoch - 1ms/step
Training model1 3...
Epoch 1/5
469/469 [==============================] - 3s 6ms/step - loss: 0.4067 - accuracy: 0.8747
Epoch 2/5
469/469 [==============================] - 3s 6ms/step - loss: 0.3818 - accuracy: 0.8827
Epoch 3/5
469/469 [==============================] - 3s 6ms/step - loss: 0.3644 - accuracy: 0.8897
Epoch 4/5
469/469 [==============================] - 3s 6ms/step - loss: 0.3448 - accuracy: 0.8959
Epoch 5/5
469/469 [==============================] - 3s 6ms/step - loss: 0.3286 - accuracy: 0.9000
Evaluating model1 3...
313/313 - 0s - loss: 0.3142 - accuracy: 0.9052 - 482ms/epoch - 2ms/step
Training model1 4...
Epoch 1/5
469/469 [==============================] - 3s 6ms/step - loss: 0.3147 - accuracy: 0.9052
Epoch 2/5
469/469 [==============================] - 3s 6ms/step - loss: 0.3020 - accuracy: 0.9090
Epoch 3/5
469/469 [==============================] - 3s 6ms/step - loss: 0.2908 - accuracy: 0.9131
Epoch 4/5
469/469 [==============================] - 3s 6ms/step - loss: 0.2810 - accuracy: 0.9154
Epoch 5/5
469/469 [==============================] - 3s 6ms/step - loss: 0.2740 - accuracy: 0.9171
Evaluating model1 4...
313/313 - 0s - loss: 0.2565 - accuracy: 0.9227 - 361ms/epoch - 1ms/step
Training model1 5...
Epoch 1/5
469/469 [==============================] - 3s 6ms/step - loss: 0.2662 - accuracy: 0.9203
Epoch 2/5
469/469 [==============================] - 3s 6ms/step - loss: 0.2579 - accuracy: 0.9226
```

```
    Mean test accuracy of Model 1: 0.8934599995613098
```

```
# Extract the features from second Last fully connected Layer (having 8 neurons)
feature_extractor_m2 = keras.Model(inputs=build_model_2.inputs, outputs=build_model_2.layers[-2].output)

#predict the feature train and test
train_features_m2 = feature_extractor_m2.predict(x_train)
test_features_m2 = feature_extractor_m2.predict(x_test)

# Model features using Random Forest classifier
rf_classifier_m2 = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier_m2.fit(train_features_m2, y_train)

# Predict using the Random Forest classifier
predictions_m2 = rf_classifier_m2.predict(test_features_m2)

# Calculate accuracy
accuracy_m2 = accuracy_score(y_test, predictions_m2)
print("Accuracy of Random Forest classifier on extracted features model 1:", accuracy_m2)
```

```
    1875/1875 [==============================] - 2s 1ms/step
    313/313 [==============================] - 0s 1ms/step
    Accuracy of Random Forest classifier on extracted features model 1: 0.9405
```

c) Model 3

```
train_stacked_features = np.hstack((train_features_m1, train_features_m2))
test_stacked_features = np.hstack((test_features_m1, test_features_m2))
```

```
# Model stacked features using Random Forest classifier
rf_classifier_stack = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier_stack.fit(train_stacked_features, y_train)

# Predict using the Random Forest classifier
predictions_m3 = rf_classifier_stack.predict(test_stacked_features)

# Calculate accuracy
accuracy_m3 = accuracy_score(y_test, predictions_m3)
print("Accuracy of Random Forest classifier on Stacked Features:", accuracy_m3)
```

```
    Accuracy of Random Forest classifier on Stacked Features: 0.9722
```

d) **Model 4**

```
# Extract the deep features from Model-1 and Model-2 stack the features horizontally, reduce the
#dimension to either 8, 10 or 12 using principal component analysis (PCA) and model the reduced features
#using a Random Forest classifier.
```

```
from sklearn.decomposition import PCA

pca = PCA(n_components=8) # use 8,18 or 12
pca.fit(train_stacked_features)

features_train_reduced = pca.transform(train_stacked_features)
features_test_reduced = pca.transform(test_stacked_features)
```

```
features_train_reduced.shape,features_test_reduced.shape
```

```
((60000, 8), (10000, 8))
```

```
 # Model stacked features using Random Forest classifier
rf_classifier_pca = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier_pca.fit(features_train_reduced, y_train)

# Predict using the Random Forest classifier
predictions_m4 = rf_classifier_pca.predict(features_test_reduced)

# Calculate accuracy

accuracy_m4 = accuracy_score(y_test, predictions_m4)
print("Accuracy of Random Forest classifier After feature reduction:", accuracy_m4)
```

```
Accuracy of Random Forest classifier After feature reduction: 0.9675
```

e) Analysis

Based on the provided specifications, Model-1 and Model-2 are similar in architecture but differ in the type of pooling layers used (maxpooling vs. average pooling). Model-3 combines the features extracted from both Model-1 and Model-2, while Model-4 further reduces the dimensionality using PCA before using a Random Forest classifier.

Model-4 might have an edge over the others if the dimensionality reduction via PCA effectively captures the most discriminative information while reducing noise.

Therefore, the best model would likely be Model-4 because it combines the features from both Model-1 and Model-2, reduces dimensionality using PCA, and then employs a Random Forest classifier for classification

Model 3 and model 4 has only slight varience in the accuracy hence both can be the best models precisely model 4 has best accurcy with 8,10,12 componets which has mostly same accuracy of 0.97.