final

April 29, 2024

[6]:
```
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from sklearn.model_selection import train_test_split
```

[8]:
```
# Load the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.
 ↪load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515             0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880             1s
0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
5148/5148             0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4422102/4422102             0s
0us/step
```

[9]:
```
# Normalize pixel values to be in the range [0, 1]
train_images = train_images / 255.0
test_images = test_images / 255.0
```

[10]:
```
# Split the dataset into training and testing sets
train_images, val_images, train_labels, val_labels =␣
 ↪train_test_split(train_images, train_labels, test_size=0.2, random_state=42)
```

[11]:
```
from tensorflow.keras import layers, models
```

[13]:
```
# Define the input shape
input_shape = (28, 28, 1)

# Define the CNN architecture
model_cnn = models.Sequential([
```

```
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

[14]:
```
# Compile the model
model_cnn.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

[15]:
```
# Reshape the input data for CNN (add a channel dimension)
train_images_cnn = train_images.reshape((train_images.shape[0], 28, 28, 1))
val_images_cnn = val_images.reshape((val_images.shape[0], 28, 28, 1))
```

[16]:
```
# Train the model
history_cnn = model_cnn.fit(train_images_cnn, train_labels, epochs=10,␣
 ↪batch_size=64,
                             validation_data=(val_images_cnn, val_labels))
```

```
Epoch 1/10
750/750              8s 10ms/step -
accuracy: 0.7150 - loss: 0.8106 - val_accuracy: 0.8495 - val_loss: 0.4085
Epoch 2/10
750/750              8s 10ms/step -
accuracy: 0.8661 - loss: 0.3720 - val_accuracy: 0.8771 - val_loss: 0.3309
Epoch 3/10
750/750              7s 10ms/step -
accuracy: 0.8866 - loss: 0.3136 - val_accuracy: 0.8898 - val_loss: 0.2982
Epoch 4/10
750/750              8s 10ms/step -
accuracy: 0.9010 - loss: 0.2743 - val_accuracy: 0.8942 - val_loss: 0.2911
Epoch 5/10
750/750              7s 10ms/step -
accuracy: 0.9103 - loss: 0.2486 - val_accuracy: 0.8987 - val_loss: 0.2772
Epoch 6/10
750/750              7s 10ms/step -
accuracy: 0.9184 - loss: 0.2233 - val_accuracy: 0.8951 - val_loss: 0.2862
Epoch 7/10
750/750              7s 10ms/step -
accuracy: 0.9258 - loss: 0.2065 - val_accuracy: 0.9053 - val_loss: 0.2580
Epoch 8/10
750/750              7s 10ms/step -
accuracy: 0.9311 - loss: 0.1852 - val_accuracy: 0.9038 - val_loss: 0.2620
```

```
Epoch 9/10
750/750             7s 10ms/step -
accuracy: 0.9364 - loss: 0.1707 - val_accuracy: 0.9043 - val_loss: 0.2645
Epoch 10/10
750/750             7s 10ms/step -
accuracy: 0.9409 - loss: 0.1578 - val_accuracy: 0.9120 - val_loss: 0.2542
```

[17]:
```python
# Evaluate the model on validation data
val_loss, val_acc = model_cnn.evaluate(val_images_cnn, val_labels)
print("Validation accuracy:", val_acc)
```

```
375/375             1s 2ms/step -
accuracy: 0.9140 - loss: 0.2482
Validation accuracy: 0.9120000004768372
```

[18]:
```python
# Reshape the test data
test_images_cnn = test_images.reshape((test_images.shape[0], 28, 28, 1))
```

[19]:
```python
# Evaluate the model on test data
test_loss, test_acc = model_cnn.evaluate(test_images_cnn, test_labels)
print("Test accuracy:", test_acc)
```

```
313/313             1s 2ms/step -
accuracy: 0.9018 - loss: 0.2820
Test accuracy: 0.9039999842643738
```

[21]:
```python
# Define the DNN architecture
model_dnn = models.Sequential([
    layers.Input(shape=(28, 28)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(10, activation='softmax')
])
```

[22]:
```python
# Compile the DNN model
model_dnn.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

[23]:
```python
# Train the DNN model
history_dnn = model_dnn.fit(train_images, train_labels, epochs=10,
 ↪batch_size=64,
                            validation_data=(val_images, val_labels))
```

```
Epoch 1/10
750/750             1s 1ms/step -
```

```
accuracy: 0.7023 - loss: 0.8469 - val_accuracy: 0.8462 - val_loss: 0.4300
Epoch 2/10
750/750              1s 943us/step -
accuracy: 0.8388 - loss: 0.4513 - val_accuracy: 0.8595 - val_loss: 0.3907
Epoch 3/10
750/750              1s 906us/step -
accuracy: 0.8546 - loss: 0.4076 - val_accuracy: 0.8637 - val_loss: 0.3669
Epoch 4/10
750/750              1s 1ms/step -
accuracy: 0.8618 - loss: 0.3808 - val_accuracy: 0.8697 - val_loss: 0.3576
Epoch 5/10
750/750              1s 997us/step -
accuracy: 0.8709 - loss: 0.3575 - val_accuracy: 0.8750 - val_loss: 0.3463
Epoch 6/10
750/750              1s 982us/step -
accuracy: 0.8726 - loss: 0.3464 - val_accuracy: 0.8777 - val_loss: 0.3302
Epoch 7/10
750/750              1s 934us/step -
accuracy: 0.8798 - loss: 0.3332 - val_accuracy: 0.8765 - val_loss: 0.3459
Epoch 8/10
750/750              1s 889us/step -
accuracy: 0.8815 - loss: 0.3265 - val_accuracy: 0.8778 - val_loss: 0.3247
Epoch 9/10
750/750              1s 905us/step -
accuracy: 0.8875 - loss: 0.3075 - val_accuracy: 0.8813 - val_loss: 0.3237
Epoch 10/10
750/750              1s 916us/step -
accuracy: 0.8877 - loss: 0.3005 - val_accuracy: 0.8838 - val_loss: 0.3202
```

[24]:

```python
# Evaluate the DNN model on validation data
val_loss_dnn, val_acc_dnn = model_dnn.evaluate(val_images, val_labels)
print("Validation accuracy (DNN):", val_acc_dnn)
```

```
375/375              0s 281us/step -
accuracy: 0.8862 - loss: 0.3210
Validation accuracy (DNN): 0.8837500214576721
```

[25]:

```python
# Evaluate the DNN model on test data
test_loss_dnn, test_acc_dnn = model_dnn.evaluate(test_images, test_labels)
print("Test accuracy (DNN):", test_acc_dnn)
```

```
313/313              0s 281us/step -
accuracy: 0.8793 - loss: 0.3455
Test accuracy (DNN): 0.8777999877929688
```

[26]:

```python
# Define the MLP architecture
model_mlp = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),
```

```python
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(10, activation='softmax')
])
```

[27]:
```python
# Compile the MLP model
model_mlp.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

[28]:
```python
# Train the MLP model
history_mlp = model_mlp.fit(train_images, train_labels, epochs=10,␣
 ↪batch_size=64,
                            validation_data=(val_images, val_labels))
```

```
Epoch 1/10
750/750              2s 2ms/step -
accuracy: 0.6327 - loss: 1.0226 - val_accuracy: 0.8369 - val_loss: 0.4456
Epoch 2/10
750/750              1s 2ms/step -
accuracy: 0.8260 - loss: 0.4989 - val_accuracy: 0.8544 - val_loss: 0.3912
Epoch 3/10
750/750              1s 2ms/step -
accuracy: 0.8438 - loss: 0.4419 - val_accuracy: 0.8673 - val_loss: 0.3727
Epoch 4/10
750/750              1s 2ms/step -
accuracy: 0.8552 - loss: 0.4119 - val_accuracy: 0.8682 - val_loss: 0.3602
Epoch 5/10
750/750              1s 2ms/step -
accuracy: 0.8601 - loss: 0.3947 - val_accuracy: 0.8641 - val_loss: 0.3765
Epoch 6/10
750/750              1s 2ms/step -
accuracy: 0.8655 - loss: 0.3769 - val_accuracy: 0.8716 - val_loss: 0.3507
Epoch 7/10
750/750              1s 2ms/step -
accuracy: 0.8703 - loss: 0.3643 - val_accuracy: 0.8735 - val_loss: 0.3400
Epoch 8/10
750/750              1s 2ms/step -
accuracy: 0.8678 - loss: 0.3582 - val_accuracy: 0.8760 - val_loss: 0.3327
Epoch 9/10
750/750              1s 2ms/step -
accuracy: 0.8761 - loss: 0.3427 - val_accuracy: 0.8820 - val_loss: 0.3284
Epoch 10/10
750/750              1s 2ms/step -
```

```
accuracy: 0.8761 - loss: 0.3432 - val_accuracy: 0.8808 - val_loss: 0.3278
```

[29]:
```
# Evaluate the MLP model on validation data
val_loss_mlp, val_acc_mlp = model_mlp.evaluate(val_images, val_labels)
print("Validation accuracy (MLP):", val_acc_mlp)
```

```
375/375              0s 429us/step -
accuracy: 0.8815 - loss: 0.3267
Validation accuracy (MLP): 0.8807500004768372
```

[30]:
```
# Evaluate the MLP model on test data
test_loss_mlp, test_acc_mlp = model_mlp.evaluate(test_images, test_labels)
print("Test accuracy (MLP):", test_acc_mlp)
```

```
313/313              0s 409us/step -
accuracy: 0.8741 - loss: 0.3505
Test accuracy (MLP): 0.8748999834060669
```

[31]:
```
#As committee
import numpy as np
```

[32]:
```
# Get the predicted probabilities for each model on the test data
probabilities_cnn = model_cnn.predict(test_images)
probabilities_dnn = model_dnn.predict(test_images)
probabilities_mlp = model_mlp.predict(test_images)
```

```
313/313              1s 2ms/step
313/313              0s 283us/step
313/313              0s 374us/step
```

[33]:
```
# Average the predicted probabilities across all three models
ensemble_probabilities = (probabilities_cnn + probabilities_dnn +␣
 ↪probabilities_mlp) / 3
```

[34]:
```
# Get the class predictions by selecting the class with the highest probability
ensemble_predictions = np.argmax(ensemble_probabilities, axis=1)
```

[35]:
```
# Evaluate the ensemble on the test data
test_accuracy_ensemble = np.mean(ensemble_predictions == test_labels)
print("Test accuracy (Ensemble):", test_accuracy_ensemble)
```

```
Test accuracy (Ensemble): 0.9045
```

[37]:
```
# Report individual model accuracies
print("Individual Model Accuracies:")
print("CNN Model Accuracy:", test_acc)
print("DNN Model Accuracy:", test_acc_dnn)
print("MLP Model Accuracy:", test_acc_mlp)

# Report final accuracy of the committee
```

```
print("\nFinal Accuracy of the Committee:")
print("Ensemble Model Accuracy:", test_accuracy_ensemble)
```

```
Individual Model Accuracies:
CNN Model Accuracy: 0.9039999842643738
DNN Model Accuracy: 0.8777999877929688
MLP Model Accuracy: 0.8748999834060669

Final Accuracy of the Committee:
Ensemble Model Accuracy: 0.9045
```

[ ]:
```
#Q2A How are the diverse deep-learning models formed in your project, please␣
↪explain? (10points)
```

[73]:
```
print("CNN (Convolutional Neural Network):")
print("CNNs tend to be very good in capturing spatial hierarchies in images. In␣
↪my implmentation of CNNs it consists of convolutional layers followed by␣
↪max-pooling layers and dense layers. The convulutional layers extract␣
↪certain features from the images through filters.\n ")
print("DNN (Deep Neural Networks):")
print("DNNs consist of connected layers and are effective when learning␣
↪patterns that are complex in data. In my implementation it is a simple feed␣
↪forward network with multiple hidden layers. Each layer connects with␣
↪another layer, this allows it to learn relations that are non-linear.\n ")
print("MLP (Multi-Layer perceptron):")
print("They are feedforward networks with multiple layers of nodes, they can be␣
↪used for both classification and regression. In my impelmentation it has␣
↪multiple hidden layers with ReLU activation functions and a softmax output␣
↪layer for classification.\n")
```

```
CNN (Convolutional Neural Network):
CNNs tend to be very good in capturing spatial hierarchies in images. In my
implmentation of CNNs it consists of convolutional layers followed by max-
pooling layers and dense layers. The convulutional layers extract certain
features from the images through filters.

DNN (Deep Neural Networks):
DNNs consist of connected layers and are effective when learning patterns that
are complex in data. In my implementation it is a simple feed forward network
with multiple hidden layers. Each layer connects with another layer, this allows
it to learn relations that are non-linear.

MLP (Multi-Layer perceptron):
They are feedforward networks with multiple layers of nodes, they can be used
for both classification and regression. In my impelmentation it has multiple
hidden layers with ReLU activation functions and a softmax output layer for
classification.
```

[ ]:
```python
#2B How are the three diverse deep-learning models combined? (20 points).
print("Done on Submission PDF")
```

[66]:
```python
#Q2C
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score

# Calculate metrics for individual models
metrics_individual = {
    "Model": ["CNN", "DNN", "MLP"],
    "Accuracy": [test_acc, test_acc_dnn, test_acc_mlp]
}

# Calculate precision, recall, and F1-score for each individual model
for model, model_name in zip([model_cnn, model_dnn, model_mlp], ["CNN", "DNN",␣
↪"MLP"]):
    predictions = np.argmax(model.predict(test_images), axis=1)
    metrics_individual[model_name + " Precision"] =␣
↪precision_score(test_labels, predictions, average='weighted')
    metrics_individual[model_name + " Recall"] = recall_score(test_labels,␣
↪predictions, average='weighted')
    metrics_individual[model_name + " F1-Score"] = f1_score(test_labels,␣
↪predictions, average='weighted')

# Calculate metrics for the committee
ensemble_predictions = np.argmax(ensemble_probabilities, axis=1)
metrics_committee = {
    "Model": ["Ensemble"],
    "Accuracy": [test_accuracy_ensemble],
    "Precision": [precision_score(test_labels, ensemble_predictions,␣
↪average='weighted')],
    "Recall": [recall_score(test_labels, ensemble_predictions,␣
↪average='weighted')],
    "F1-Score": [f1_score(test_labels, ensemble_predictions,␣
↪average='weighted')]
}

print("\nFinal Metrics of Individual Models:")
print(pd.DataFrame(metrics_individual))

print("\nFinal Metrics of the Committee:")
print(pd.DataFrame(metrics_committee))
```

```
313/313             1s 2ms/step
313/313             0s 265us/step
313/313             0s 399us/step
```

```
Final Metrics of Individual Models:
  Model  Accuracy  CNN Precision  CNN Recall  CNN F1-Score  DNN Precision  \
0   CNN    0.9905       0.906548       0.904       0.90468       0.877663
1   DNN    0.8778       0.906548       0.904       0.90468       0.877663
2   MLP    0.8749       0.906548       0.904       0.90468       0.877663

   DNN Recall  DNN F1-Score  MLP Precision  MLP Recall  MLP F1-Score
0      0.8778      0.875762       0.876199      0.8749      0.875076
1      0.8778      0.875762       0.876199      0.8749      0.875076
2      0.8778      0.875762       0.876199      0.8749      0.875076

Final Metrics of the Committee:
      Model  Accuracy  Precision  Recall  F1-Score
0  Ensemble    0.9045   0.904322  0.9045  0.904302
```

[ ]:
```
print("Proper tabulated answers on pdf")
```