Lab Assignment
21MIM10031

ANN

```
import numpy as np import tensorflow as tf from
tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist from
tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# Load dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize the images
train_images = train_images /
255.0 test_images = test_images /
255.0

# One-hot encode the labels train_labels =
to_categorical(train_labels, 10) test_labels =
to_categorical(test_labels, 10)

# Build the model
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten the input image
    Dense(128, activation='relu'),  # Hidden layer with 128 neurons and ReLU activation
    Dense(10, activation='softmax')  # Output layer with 10 neurons (one for each class) and softmax activation
])

# Compile the model
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model model.fit(train_images, train_labels, epochs=10,
validation_split=0.2)

# Evaluate the model test_loss, test_accuracy =
model.evaluate(test_images, test_labels) print(f'Test accuracy:
{test_accuracy}')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_d
super().__init__(**kwargs)
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8682 - loss: 0.4789 - val_accuracy: 0.9565 - val_loss: 0.1585
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9602 - loss: 0.1352 - val_accuracy: 0.9655 - val_loss: 0.1188
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.9745 - loss: 0.0863 - val_accuracy: 0.9725 - val_loss: 0.0953
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9826 - loss: 0.0599 - val_accuracy: 0.9733 - val_loss: 0.0924
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9872 - loss: 0.0469 - val_accuracy: 0.9733 - val_loss: 0.0945
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9898 - loss: 0.0331 - val_accuracy: 0.9744 - val_loss: 0.0941
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.9926 - loss: 0.0250 - val_accuracy: 0.9732 - val_loss: 0.0968
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9942 - loss: 0.0215 - val_accuracy: 0.9736 - val_loss: 0.0998
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9950 - loss: 0.0176 - val_accuracy: 0.9748 - val_loss: 0.0940
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9967 - loss: 0.0127 - val_accuracy: 0.9784 - val_loss: 0.0912
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9735 - loss: 0.1003
Test accuracy: 0.9765999913215637
```

CNN

```
import numpy as np
import tensorflow as tf
from
tensorflow.keras.models
import Sequential from
tensorflow.keras.layers
import Conv2D,
MaxPooling2D, Flatten,
Dense from
tensorflow.keras.datase
ts import mnist from
tensorflow.keras.utils
import to_categorical

# Load dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Reshape the images to include a channel dimension train_images =
train_images.reshape((train_images.shape[0], 28, 28, 1)) test_images =
test_images.reshape((test_images.shape[0], 28, 28, 1))

# Normalize the images
train_images = train_images /
255.0 test_images = test_images /
255.0

# One-hot encode the labels train_labels =
to_categorical(train_labels, 10) test_labels =
to_categorical(test_labels, 10)

# Build the model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),  # Convolutional layer with 32 filters and ReLU activation
MaxPooling2D((2, 2)),  # Max pooling layer
    Conv2D(64, (3, 3), activation='relu'),  # Convolutional layer with 64 filters and ReLU activation
    MaxPooling2D((2, 2)),  # Max pooling layer
    Flatten(),  # Flatten the output
    Dense(64, activation='relu'),  # Fully connected layer with 64 neurons and ReLU activation
    Dense(10, activation='softmax')  # Output layer with 10 neurons (one for each class) and softmax activation
])

# Compile the model
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model model.fit(train_images, train_labels, epochs=10,
validation_split=0.2)

# Evaluate the model test_loss, test_accuracy =
model.evaluate(test_images, test_labels) print(f'Test accuracy:
{test_accuracy}')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 46s 30ms/step - accuracy: 0.8863 - loss: 0.3634 - val_accuracy: 0.9824 - val_loss: 0.0598
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9839 - loss: 0.0519 - val_accuracy: 0.9838 - val_loss: 0.0553
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9881 - loss: 0.0353 - val_accuracy: 0.9861 - val_loss: 0.0486
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9925 - loss: 0.0247 - val_accuracy: 0.9893 - val_loss: 0.0355
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 83s 30ms/step - accuracy: 0.9944 - loss: 0.0180 - val_accuracy: 0.9898 - val_loss: 0.0367
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 81s 29ms/step - accuracy: 0.9951 - loss: 0.0143 - val_accuracy: 0.9873 - val_loss: 0.0477
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 83s 30ms/step - accuracy: 0.9952 - loss: 0.0135 - val_accuracy: 0.9908 - val_loss: 0.0396
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 81s 30ms/step - accuracy: 0.9972 - loss: 0.0086 - val_accuracy: 0.9886 - val_loss: 0.0458
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 45s 30ms/step - accuracy: 0.9978 - loss: 0.0073 - val_accuracy: 0.9899 - val_loss: 0.0471
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 84s 31ms/step - accuracy: 0.9981 - loss: 0.0058 - val_accuracy: 0.9893 - val_loss: 0.0545
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9859 - loss: 0.0545
Test accuracy: 0.9891999959945679
```

RNN

```
import numpy as np import tensorflow as tf from
tensorflow.keras.models import Sequential from
tensorflow.keras.layers import SimpleRNN, Dense
from tensorflow.keras.datasets import mnist from
tensorflow.keras.utils import to_categorical

# Load dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize the images
train_images = train_images /
255.0 test_images = test_images /
255.0

# One-hot encode the labels train_labels =
to_categorical(train_labels, 10) test_labels =
to_categorical(test_labels, 10)

# Build the model
model = Sequential([
    SimpleRNN(128, input_shape=(28, 28), activation='relu'),  # SimpleRNN layer with 128 units and ReLU activation
    Dense(10, activation='softmax')  # Output layer with 10 neurons (one for each class) and softmax activation
])

# Compile the model
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model model.fit(train_images, train_labels, epochs=10,
validation_split=0.2)

# Evaluate the model test_loss, test_accuracy =
model.evaluate(test_images, test_labels) print(f'Test accuracy:
{test_accuracy}')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argum
  super().__init__(**kwargs)
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 17s 10ms/step - accuracy: 0.6879 - loss: 0.9003 - val_accuracy: 0.9107 - val_loss: 0.3073
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.9285 - loss: 0.2417 - val_accuracy: 0.9365 - val_loss: 0.2046
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.9477 - loss: 0.1845 - val_accuracy: 0.9505 - val_loss: 0.1820
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 22s 11ms/step - accuracy: 0.9570 - loss: 0.1550 - val_accuracy: 0.9607 - val_loss: 0.1370
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.9609 - loss: 0.1383 - val_accuracy: 0.9596 - val_loss: 0.1435
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.9611 - loss: 0.1330 - val_accuracy: 0.9647 - val_loss: 0.1196
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.9641 - loss: 0.1285 - val_accuracy: 0.9695 - val_loss: 0.1103
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.9671 - loss: 0.1162 - val_accuracy: 0.9679 - val_loss: 0.1134
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.9703 - loss: 0.1047 - val_accuracy: 0.9703 - val_loss: 0.1051
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 20s 10ms/step - accuracy: 0.9705 - loss: 0.1034 - val_accuracy: 0.9685 - val_loss: 0.1131
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9617 - loss: 0.1366
Test accuracy: 0.9685999751091003
```

Autoencoder

```
import numpy as np import tensorflow as tf from tensorflow.keras.models import Model, Sequential from
tensorflow.keras.layers import Input, Dense, Flatten, Reshape, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.datasets import mnist from tensorflow.keras.utils import to_categorical

# Load dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Reshape the images to include a channel dimension train_images =
train_images.reshape((train_images.shape[0], 28, 28, 1)) test_images =
test_images.reshape((test_images.shape[0], 28, 28, 1))

# Normalize the images
train_images = train_images /
255.0 test_images = test_images /
255.0

# One-hot encode the labels train_labels =
to_categorical(train_labels, 10) test_labels =
to_categorical(test_labels, 10)

# Encoder input_img = Input(shape=(28, 28, 1)) x = Conv2D(32, (3,
3), activation='relu', padding='same')(input_img) x =
MaxPooling2D((2, 2), padding='same')(x) x = Conv2D(64, (3, 3),
activation='relu', padding='same')(x) encoded = MaxPooling2D((2,
2), padding='same')(x)

# Decoder x = Conv2D(64, (3, 3), activation='relu',
padding='same')(encoded) x = UpSampling2D((2, 2))(x) x = Conv2D(32,
(3, 3), activation='relu', padding='same')(x) x = UpSampling2D((2,
2))(x) decoded = Conv2D(1, (3, 3), activation='sigmoid',
padding='same')(x)

# Autoencoder autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam',
loss='binary_crossentropy')

# Train the autoencoder
autoencoder.fit(train_images,
train_images,                 epochs=50,
batch_size=256,
shuffle=True,
validation_split=0.2)

# Extract features using the encoder encoder =
Model(input_img, encoded) encoded_train_images =
encoder.predict(train_images) encoded_test_images =
encoder.predict(test_images)

# Flatten the encoded images for the classifier
encoded_train_images = encoded_train_images.reshape((encoded_train_images.shape[0], -1))
encoded_test_images = encoded_test_images.reshape((encoded_test_images.shape[0], -1))

# Build the classifier model
classifier = Sequential([
    Dense(64, activation='relu', input_shape=(encoded_train_images.shape[1],)),
    Dense(10, activation='softmax')
])

# Compile the classifier model
classifier.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the classifier model
classifier.fit(encoded_train_images,
train_labels,                epochs=50,
batch_size=256,                shuffle=True,
validation_split=0.2)

# Evaluate the classifier model test_loss, test_accuracy =
classifier.evaluate(encoded_test_images, test_labels) print(f'Test accuracy:
{test_accuracy}')
```

```
Epoch 1/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 144s 752ms/step - loss: 0.2721 - val_loss: 0.0861
Epoch 2/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 141s 749ms/step - loss: 0.0824 - val_loss: 0.0771
Epoch 3/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 141s 753ms/step - loss: 0.0755 - val_loss: 0.0738
Epoch 4/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 143s 756ms/step - loss: 0.0729 - val_loss: 0.0722
Epoch 5/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 201s 753ms/step - loss: 0.0712 - val_loss: 0.0713
Epoch 6/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 141s 750ms/step - loss: 0.0703 - val_loss: 0.0704
Epoch 7/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 143s 759ms/step - loss: 0.0696 - val_loss: 0.0697
Epoch 8/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 201s 752ms/step - loss: 0.0691 - val_loss: 0.0693
Epoch 9/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 141s 747ms/step - loss: 0.0684 - val_loss: 0.0686
Epoch 10/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 138s 734ms/step - loss: 0.0682 - val_loss: 0.0682
Epoch 11/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 144s 746ms/step - loss: 0.0677 - val_loss: 0.0679
Epoch 12/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 141s 742ms/step - loss: 0.0672 - val_loss: 0.0675
Epoch 13/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 140s 730ms/step - loss: 0.0669 - val_loss: 0.0671
Epoch 14/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 142s 733ms/step - loss: 0.0666 - val_loss: 0.0671
Epoch 15/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 143s 738ms/step - loss: 0.0664 - val_loss: 0.0667
Epoch 16/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 137s 725ms/step - loss: 0.0661 - val_loss: 0.0664
Epoch 17/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 142s 728ms/step - loss: 0.0660 - val_loss: 0.0662
Epoch 18/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 143s 736ms/step - loss: 0.0657 - val_loss: 0.0662
Epoch 19/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 140s 727ms/step - loss: 0.0655 - val_loss: 0.0658
Epoch 20/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 140s 715ms/step - loss: 0.0654 - val_loss: 0.0657
Epoch 21/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 143s 720ms/step - loss: 0.0653 - val_loss: 0.0656
Epoch 22/50
```