COMP5318-assignment2-notebook(2)

May 13, 2025

1 COMP4318/5318 Assignment 2: Image Classification

1.0.1 Group number: 283 , SID1: 540282735 , SID2: 540114883

This template notebook includes code to load the dataset and a skeleton for the main sections that should be included in the notebook. Please stick to this struture for your submitted notebook.

Please focus on making your code clear, with appropriate variable names and whitespace. Include comments and markdown text to aid the readability of your code where relevant. See the specification and marking criteria in the associated specification to guide you when completing your implementation.

1.1 Setup and dependencies

Please use this section to list and set up all your required libraries/dependencies and your plotting environment.

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪StratifiedGroupKFold, RandomizedSearchCV
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC

from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt


import keras_tuner
import tensorflow as tf
from tensorflow import keras

from sklearn.model_selection import ParameterGrid
import pandas as pd
import time
```

1.2 1. Data loading, exploration, and preprocessing

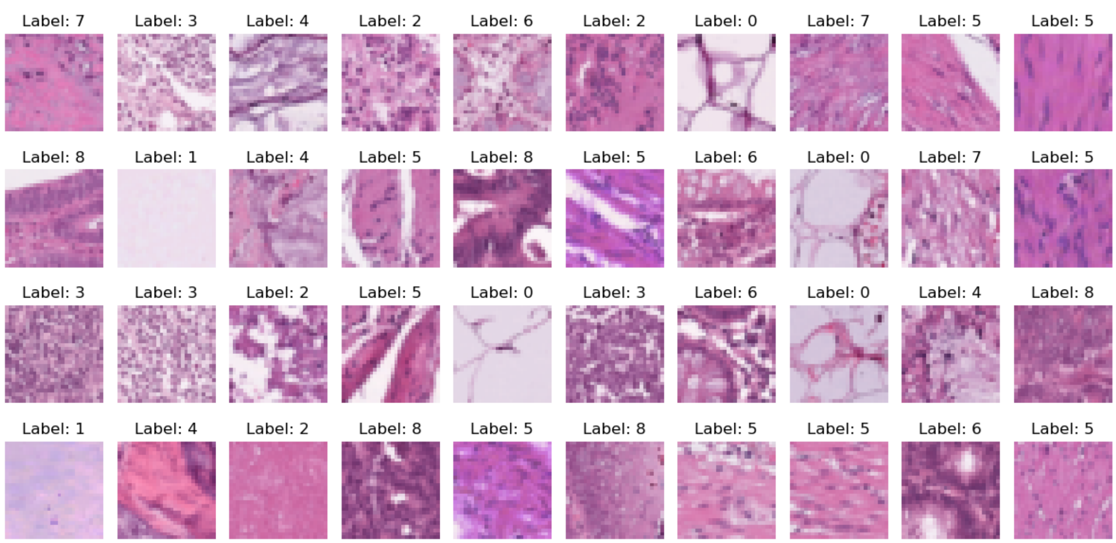
Code to load the dataset is provided in the following cell. Please proceed with your data exploration and preprocessing in the remainder of this section.

[4]:
```
# Load the dataset training and test sets as numpy arrays
# assuming Assignment2Data folder is present in the same directory
# as the notebook
X_train = np.load('Assignment2Data/X_train.npy')
y_train = np.load('Assignment2Data/y_train.npy')
X_test = np.load('Assignment2Data/X_test.npy')
y_test = np.load('Assignment2Data/y_test.npy')
```

[5]:
```
print(X_train.shape)
```

```
(32000, 28, 28, 3)
```

[6]:
```
def plot_examples(data, label, num_rows=4, num_cols=10):
    plt.figure(figsize=(num_cols * 1.2, num_rows * 1.5))
    for row in range(num_rows):
        for col in range(num_cols):
            index = num_cols * row + col
            plt.subplot(num_rows, num_cols, index + 1)
            plt.imshow(data[index])
            plt.title(f"Label: {label[index]}")
            plt.axis('off')
    plt.tight_layout()
    plt.show()

plot_examples(X_train, y_train)
```



[7]:
```
print(np.min(X_train), np.max(X_train))
```

```
0 255
```

[8]:
```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train.reshape(X_train.shape[0], -1)).
 ↪reshape(X_train.shape)
X_test_scaled = scaler.transform(X_test.reshape(X_test.shape[0], -1)).
 ↪reshape(X_test.shape)
```

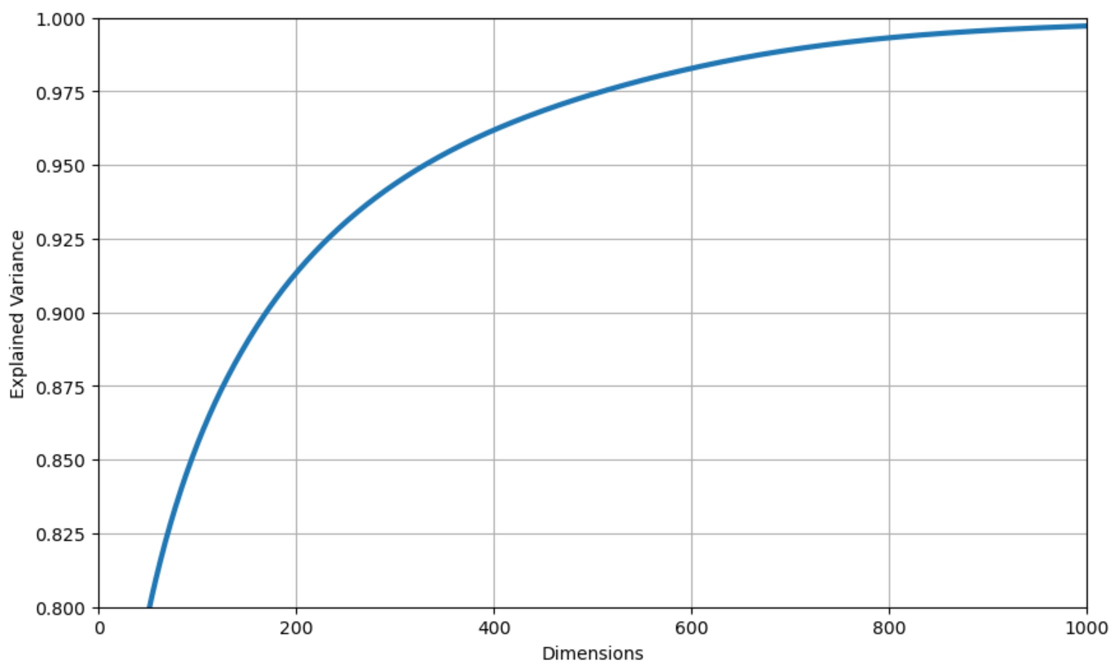
[9]:
```
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)
```

[10]:
```
print(X_train_flat.shape)
```

```
(32000, 2352)
```

[11]:
```
pca = PCA()
pca.fit(X_train_flat)
cumsum = np.cumsum(pca.explained_variance_ratio_)
```

[12]:
```
plt.figure(figsize=(10, 6))
plt.plot(cumsum, linewidth=3)
plt.axis([0, 1000, 0.8, 1])
plt.xlabel("Dimensions")
plt.ylabel("Explained Variance")
plt.grid(True)
plt.show()
```



[13]:
```
print("Number of components to retain 95% variance: ", np.argmax(cumsum >= 0.
↪95) + 1)
print("Number of components to retain 98% variance: ", np.argmax(cumsum >= 0.
↪98) + 1)
print("Number of components to retain 99% variance: ", np.argmax(cumsum >= 0.
↪99) + 1)
```

```
Number of components to retain 95% variance:  332
Number of components to retain 98% variance:  566
Number of components to retain 99% variance:  719
```

[14]:
```
pca = PCA(n_components=719)
X_train_pca = pca.fit_transform(X_train_flat)
X_test_pca = pca.transform(X_test_flat)
```

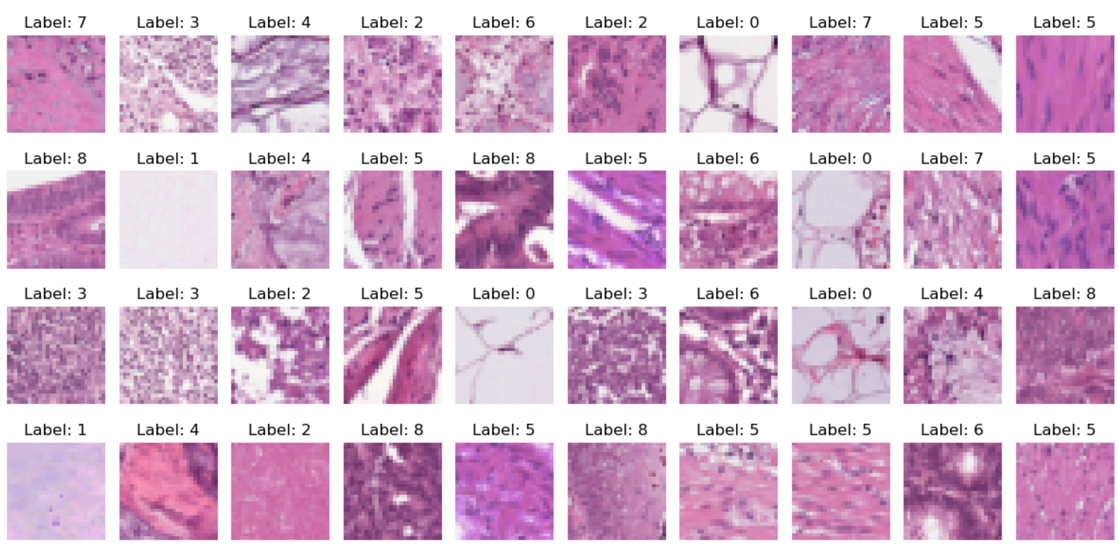
[15]:
```
print(X_train_pca.shape)
```

```
(32000, 719)
```

1.2.1 Examples of preprocessed data

Please print/display some examples of your preprocessed data here.

[16]:
```
plot_examples(X_train_scaled, y_train)
```



[17]:
```
print(X_train.shape)
print(X_test_pca.shape)
```

```python
print(X_train[0])
```

```
(32000, 28, 28, 3)
(8000, 719)
[[[199 129 181]
  [191 127 174]
  [179 118 173]
  …
  [209 136 184]
  [209 118 178]
  [205 122 180]]

 [[202 125 181]
  [194 127 182]
  [184 117 174]
  …
  [194 137 182]
  [162  95 154]
  [156  89 150]]

 [[165 102 156]
  [182 108 167]
  [190 115 174]
  …
  [202 138 188]
  [181 125 178]
  [163 102 159]]

 …

 [[208 124 182]
  [206 128 184]
  [217 123 181]
  …
  [217 128 185]
  [212 135 189]
  [207 137 190]]

 [[203 123 180]
  [194 127 182]
  [204 125 182]
  …
  [207 135 187]
  [205 137 190]
  [204 132 187]]

 [[202 122 180]
  [189 126 180]
```

```
[184 122 176]
…
[202 123 181]
[191 124 181]
[190 117 176]]]
```

[18]:
```
print(np.unique(y_train))
```

```
[0 1 2 3 4 5 6 7 8]
```

1.3 2. Algorithm design and setup

1.3.1 Algorithm of choice from first six weeks of course

[19]:
```
def bulid_svm(param):
    model = SVC(**param)
    return model
```

1.3.2 Fully connected neural network

[20]:
```
def build_mlp(hp):
    model = keras.Sequential([
        keras.layers.Input(shape=(X_train_pca.shape[1],)),
        keras.layers.Dense(units=hp.Int('units_1', min_value=128,␣
 ↪max_value=1024, step=64), activation='relu'),  # Updated range
        keras.layers.Dense(units=hp.Int('units_2', min_value=64, max_value=512,␣
 ↪step=32), activation='relu'),  # Updated range
        keras.layers.Dense(9, activation='softmax')
    ])

    model.compile(
        optimizer=keras.optimizers.Adam(hp.Choice('learning_rate',␣
 ↪values=[5e-4, 1e-3, 5e-3])),  # Updated values
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    return model
```

1.3.3 Convolutional neural network

[21]:
```
def build_cnn(hp):
    model = keras.Sequential([
        keras.layers.Input((28, 28, 3)),
        keras.layers.Conv2D(filters=hp.Int('conv_1_filters', min_value=32,␣
 ↪max_value=128, step=32),
                            kernel_size=hp.Choice('conv_1_kernel', values=[3,␣
 ↪5]),
```

```
                                activation='relu', padding='same'),
        keras.layers.MaxPooling2D(pool_size=2),
        keras.layers.Conv2D(filters=hp.Int('conv_2_filters', min_value=64,␣
↪max_value=256, step=64),
                                kernel_size=hp.Choice('conv_2_kernel', values=[3,␣
↪5]),
                                activation='relu', padding='same'),
        keras.layers.MaxPooling2D(pool_size=2),
        keras.layers.Flatten(),
        keras.layers.Dense(units=hp.Int('dense_units', min_value=128,␣
↪max_value=1024, step=128), activation='relu'),
        keras.layers.Dense(9, activation='softmax')
    ])

    model.compile(
        optimizer=keras.optimizers.Adam(hp.Choice('learning_rate',␣
↪values=[5e-4, 1e-3, 5e-3])),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    return model
```

1.4 3. Hyperparameter tuning

[22]:
```
X_small, _, y_small, _ = train_test_split(
    X_train_pca, y_train,
    train_size=6000,
    stratify=y_train,
    random_state=42
)


X_train_val, X_val, y_train_val, y_val = train_test_split(
    X_small, y_small,
    test_size=0.2,
    stratify=y_small,
    random_state=42
)
```

1.4.1 Algorithm of choice from first six weeks of course

[23]:
```
svm_param_grid = {
    'C': [1, 10],
    'kernel': ['rbf', 'linear', 'poly', 'sigmoid'],
    'gamma': ['scale']
}
```

```python
svm_search = RandomizedSearchCV(
    estimator=SVC(),
    param_distributions=svm_param_grid,
    n_iter=5,
    cv=3,
    n_jobs=-1,
    verbose=2
)

svm_search.fit(X_train_val, y_train_val)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
[CV] END …C=10, gamma=scale, kernel=sigmoid; total time=   6.0s
[CV] END …C=10, gamma=scale, kernel=sigmoid; total time=   6.1s
[CV] END …C=10, gamma=scale, kernel=sigmoid; total time=   6.0s
[CV] END …C=10, gamma=scale, kernel=linear; total time=   7.6s
[CV] END …C=10, gamma=scale, kernel=linear; total time=   7.6s
[CV] END …C=10, gamma=scale, kernel=linear; total time=   7.8s
[CV] END …C=10, gamma=scale, kernel=rbf; total time=   9.5s
[CV] END …C=10, gamma=scale, kernel=rbf; total time=   9.6s
[CV] END …C=1, gamma=scale, kernel=rbf; total time=   7.5s
[CV] END …C=1, gamma=scale, kernel=rbf; total time=   7.6s
[CV] END …C=10, gamma=scale, kernel=rbf; total time=   8.3s
[CV] END …C=1, gamma=scale, kernel=linear; total time=   6.4s
[CV] END …C=1, gamma=scale, kernel=linear; total time=   6.5s
[CV] END …C=1, gamma=scale, kernel=rbf; total time=   7.0s
[CV] END …C=1, gamma=scale, kernel=linear; total time=   5.4s
```

[23]:
```
RandomizedSearchCV(cv=3, estimator=SVC(), n_iter=5, n_jobs=-1,
                   param_distributions={'C': [1, 10], 'gamma': ['scale'],
                                        'kernel': ['rbf', 'linear', 'poly',
                                                   'sigmoid']},
                   verbose=2)
```

[24]:
```python
results_df = pd.DataFrame(svm_search.cv_results_)

results_df = results_df[[
    'params',
    'mean_test_score',
    'std_test_score',
    'rank_test_score'
]]


results_df = results_df.sort_values(by='mean_test_score', ascending=False)

print(results_df)
```

```python
print("Best parameters: ", svm_search.best_params_)
results_df.to_csv("svm_search_results.csv", index=False)
```

```
                                             params  mean_test_score  \
3      {'kernel': 'rbf', 'gamma': 'scale', 'C': 1}         0.591875
2     {'kernel': 'rbf', 'gamma': 'scale', 'C': 10}         0.578542
1  {'kernel': 'linear', 'gamma': 'scale', 'C': 10}         0.381250
4   {'kernel': 'linear', 'gamma': 'scale', 'C': 1}         0.381250
0 {'kernel': 'sigmoid', 'gamma': 'scale', 'C': 10}         0.282708

   std_test_score  rank_test_score
3        0.010206                1
2        0.010223                2
1        0.010607                3
4        0.010607                3
0        0.035858                5
Best parameters:  {'kernel': 'rbf', 'gamma': 'scale', 'C': 1}
```

1.4.2 Fully connected neural network

[25]:
```python
mlp_tuner = keras_tuner.RandomSearch(
    hypermodel=build_mlp,
    objective="val_accuracy",
    max_trials=10,  # Increased trials
    executions_per_trial=3,  # Increased executions
    overwrite=True,
    directory="result",
    project_name="mlp"
)

mlp_tuner.search(X_train_val, y_train_val, epochs=30, validation_data=(X_val,␣
 ↪y_val))
```

```
Trial 10 Complete [00h 00m 33s]
val_accuracy: 0.42249999443689984

Best val_accuracy So Far: 0.5099999904632568
Total elapsed time: 00h 04m 16s
```

[26]:
```python
print("Best MLP parameters:", mlp_tuner.get_best_hyperparameters()[0].values)

trials = list(mlp_tuner.oracle.trials.values())

results = [{
    **trial.hyperparameters.values,
    "val_accuracy": trial.metrics.get_last_value("val_accuracy"),
    "score": trial.score
} for trial in trials]
```

```
results_df = pd.DataFrame(results)
results_df = results_df.sort_values(by="val_accuracy", ascending=False)

print(results_df)
```

```
Best MLP parameters: {'units_1': 448, 'units_2': 480, 'learning_rate': 0.0005}
   units_1  units_2  learning_rate  val_accuracy     score
7      448      480         0.0005      0.510000  0.510000
2      320      352         0.0010      0.506111  0.506111
4      256      320         0.0010      0.496944  0.496944
8      960      416         0.0010      0.495833  0.495833
3      128      256         0.0010      0.485000  0.485000
0      128      192         0.0005      0.461111  0.461111
1      192       96         0.0005      0.454722  0.454722
5      704      320         0.0050      0.435833  0.435833
9      640      512         0.0050      0.422500  0.422500
6     1024      288         0.0050      0.367222  0.367222
```

1.4.3 Convolutional neural network

[27]:
```
cnn_tuner = keras_tuner.RandomSearch(
    hypermodel=build_cnn,
    objective='val_accuracy',
    max_trials=10,
    executions_per_trial=3,
    overwrite=True,
    directory="result",
    project_name="cnn"
)
```

[28]:
```
X_tune, _, y_tune, _ = train_test_split(X_train_scaled, y_train,␣
 ↪train_size=6000, stratify=y_train, random_state=42)

cnn_tuner.search(X_tune, y_tune, epochs=30, validation_split=0.2)
```

```
Trial 10 Complete [00h 22m 15s]
val_accuracy: 0.7322222391764323

Best val_accuracy So Far: 0.7463888923327128
Total elapsed time: 02h 23m 30s
```

[29]:
```
print("Best CNN parameters:", cnn_tuner.get_best_hyperparameters()[0].values)

cnn_trials = list(cnn_tuner.oracle.trials.values())
```

```python
cnn_results = [{
    **trial.hyperparameters.values,
    "val_accuracy": trial.metrics.get_last_value("val_accuracy"),
    "score": trial.score
} for trial in cnn_trials]


cnn_results_df = pd.DataFrame(cnn_results)
cnn_results_df = cnn_results_df.sort_values(by="val_accuracy", ascending=False)

print(cnn_results_df)
```

```
Best CNN parameters: {'conv_1_filters': 32, 'conv_1_kernel': 3,
'conv_2_filters': 128, 'conv_2_kernel': 3, 'dense_units': 768, 'learning_rate':
0.0005}
   conv_1_filters  conv_1_kernel  conv_2_filters  conv_2_kernel  dense_units  \
6              32              3             128              3          768
8             128              3              64              3          640
0              32              3             192              3          256
9             128              3             128              3          256
2              32              3              64              5          768
3              32              5              64              3          640
1              96              5              64              5          256
7              64              5              64              3          128
4              64              5             256              3          512
5              32              5             256              3          512

   learning_rate  val_accuracy     score
6         0.0005      0.746389  0.746389
8         0.0005      0.743889  0.743889
0         0.0010      0.735000  0.735000
9         0.0010      0.732222  0.732222
2         0.0010      0.716667  0.716667
3         0.0005      0.713889  0.713889
1         0.0005      0.702500  0.702500
7         0.0005      0.700000  0.700000
4         0.0010      0.690833  0.690833
5         0.0050      0.419444  0.419444
```

1.5 4. Final models

In this section, please ensure to include cells to train each model with its best hyperparmater combination independently of the hyperparameter tuning cells, i.e. don't rely on the hyperparameter tuning cells having been run.

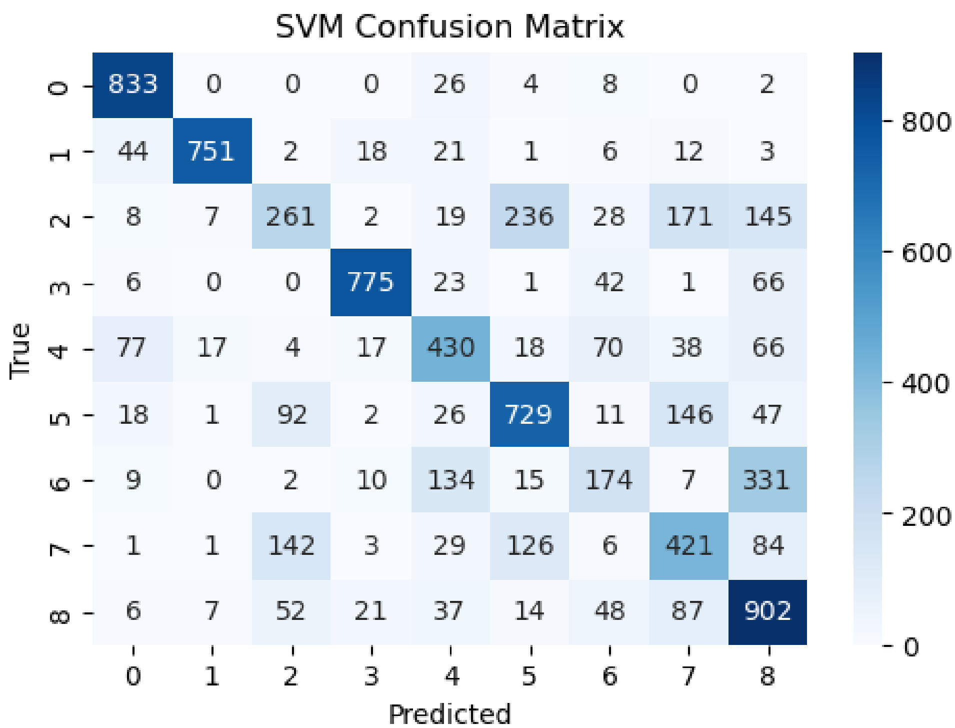
1.5.1 Algorithm of choice from first six weeks of course

[30]:
```
# the best parameters
best_svm = SVC(kernel='rbf', C=1, gamma='scale')

# train
start_time = time.time()
best_svm.fit(X_train_pca, y_train)
svm_train_time = time.time() - start_time
print(f"SVM training time: {svm_train_time:.2f} seconds")

svm_accuracy = best_svm.score(X_test_pca, y_test)
print(f"Best SVM Test Accuracy: {svm_accuracy:.3f}")
```

```
SVM training time: 237.36 seconds
Best SVM Test Accuracy: 0.659
```

[31]:
```
y_pred_svm = best_svm.predict(X_test_pca)

print("SVM Classification Report:")
print(classification_report(y_test, y_pred_svm))

plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, fmt='d',␣
↪cmap='Blues')
plt.title("SVM Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.95      0.89       873
           1       0.96      0.88      0.91       858
           2       0.47      0.30      0.36       877
           3       0.91      0.85      0.88       914
           4       0.58      0.58      0.58       737
           5       0.64      0.68      0.66      1072
           6       0.44      0.26      0.32       682
           7       0.48      0.52      0.50       813
           8       0.55      0.77      0.64      1174

    accuracy                           0.66      8000
   macro avg       0.65      0.64      0.64      8000
weighted avg       0.65      0.66      0.65      8000
```



1.5.2 Fully connected neural network

[32]:
```
# best parameter: units_1=576, units_2=448, learning_rate=0.001
mlp = keras.Sequential([
    keras.layers.Input(shape=(X_train_pca.shape[1],)),
    keras.layers.Dense(576, activation='relu'),
    keras.layers.Dense(448, activation='relu'),
    keras.layers.Dense(9, activation='softmax')
])

mlp.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])


start_time = time.time()
mlp.fit(X_train_pca, y_train, epochs=100, batch_size=64)
mlp_train_time = time.time() - start_time
print(f"MLP training time: {mlp_train_time:.2f} seconds")

_, mlp_accuracy = mlp.evaluate(X_test_pca, y_test)
print(f"Best MLP Test Accuracy: {mlp_accuracy:.3f}")
```

```
Epoch 1/100
```

```
500/500             2s 3ms/step -
accuracy: 0.4111 - loss: 14.5633
Epoch 2/100
500/500             1s 3ms/step -
accuracy: 0.6173 - loss: 1.0120
Epoch 3/100
500/500             1s 3ms/step -
accuracy: 0.7014 - loss: 0.8001
Epoch 4/100
500/500             1s 3ms/step -
accuracy: 0.7591 - loss: 0.6459
Epoch 5/100
500/500             1s 3ms/step -
accuracy: 0.7959 - loss: 0.5663
Epoch 6/100
500/500             1s 2ms/step -
accuracy: 0.8141 - loss: 0.5157
Epoch 7/100
500/500             1s 3ms/step -
accuracy: 0.8226 - loss: 0.5021
Epoch 8/100
500/500             1s 2ms/step -
accuracy: 0.8356 - loss: 0.4823
Epoch 9/100
500/500             1s 3ms/step -
accuracy: 0.8468 - loss: 0.4596
Epoch 10/100
500/500             1s 2ms/step -
accuracy: 0.8588 - loss: 0.4093
Epoch 11/100
500/500             1s 3ms/step -
accuracy: 0.8611 - loss: 0.4113
Epoch 12/100
500/500             1s 3ms/step -
accuracy: 0.8613 - loss: 0.4173
Epoch 13/100
500/500             1s 3ms/step -
accuracy: 0.8864 - loss: 0.3382
Epoch 14/100
500/500             1s 3ms/step -
accuracy: 0.8891 - loss: 0.3373
Epoch 15/100
500/500             1s 3ms/step -
accuracy: 0.8933 - loss: 0.3227
Epoch 16/100
500/500             1s 3ms/step -
accuracy: 0.8954 - loss: 0.3263
Epoch 17/100
```

```
500/500              1s 3ms/step -
accuracy: 0.9015 - loss: 0.3147
Epoch 18/100
500/500              1s 3ms/step -
accuracy: 0.9116 - loss: 0.2858
Epoch 19/100
500/500              1s 3ms/step -
accuracy: 0.9135 - loss: 0.2718
Epoch 20/100
500/500              1s 3ms/step -
accuracy: 0.9110 - loss: 0.2848
Epoch 21/100
500/500              1s 3ms/step -
accuracy: 0.9159 - loss: 0.2744
Epoch 22/100
500/500              1s 3ms/step -
accuracy: 0.9208 - loss: 0.2566
Epoch 23/100
500/500              1s 3ms/step -
accuracy: 0.9219 - loss: 0.2689
Epoch 24/100
500/500              1s 3ms/step -
accuracy: 0.9168 - loss: 0.2858
Epoch 25/100
500/500              1s 3ms/step -
accuracy: 0.9270 - loss: 0.2428
Epoch 26/100
500/500              1s 3ms/step -
accuracy: 0.9268 - loss: 0.2457
Epoch 27/100
500/500              1s 3ms/step -
accuracy: 0.9387 - loss: 0.2053
Epoch 28/100
500/500              1s 3ms/step -
accuracy: 0.9384 - loss: 0.1982
Epoch 29/100
500/500              1s 3ms/step -
accuracy: 0.9299 - loss: 0.2461
Epoch 30/100
500/500              1s 3ms/step -
accuracy: 0.9317 - loss: 0.2414
Epoch 31/100
500/500              1s 3ms/step -
accuracy: 0.9394 - loss: 0.2083
Epoch 32/100
500/500              1s 3ms/step -
accuracy: 0.9404 - loss: 0.2055
Epoch 33/100
```

```
500/500              1s 3ms/step -
accuracy: 0.9242 - loss: 0.2942
Epoch 34/100
500/500              1s 3ms/step -
accuracy: 0.9401 - loss: 0.2308
Epoch 35/100
500/500              1s 3ms/step -
accuracy: 0.9411 - loss: 0.2382
Epoch 36/100
500/500              1s 3ms/step -
accuracy: 0.9473 - loss: 0.2017
Epoch 37/100
500/500              1s 3ms/step -
accuracy: 0.9508 - loss: 0.1737
Epoch 38/100
500/500              1s 3ms/step -
accuracy: 0.9475 - loss: 0.1970
Epoch 39/100
500/500              1s 3ms/step -
accuracy: 0.9487 - loss: 0.1926
Epoch 40/100
500/500              1s 3ms/step -
accuracy: 0.9446 - loss: 0.2187
Epoch 41/100
500/500              1s 3ms/step -
accuracy: 0.9491 - loss: 0.1929
Epoch 42/100
500/500              1s 3ms/step -
accuracy: 0.9508 - loss: 0.1779
Epoch 43/100
500/500              1s 3ms/step -
accuracy: 0.9536 - loss: 0.1742
Epoch 44/100
500/500              1s 3ms/step -
accuracy: 0.9507 - loss: 0.1926
Epoch 45/100
500/500              1s 3ms/step -
accuracy: 0.9494 - loss: 0.1942
Epoch 46/100
500/500              1s 3ms/step -
accuracy: 0.9549 - loss: 0.1929
Epoch 47/100
500/500              1s 3ms/step -
accuracy: 0.9559 - loss: 0.1776
Epoch 48/100
500/500              1s 3ms/step -
accuracy: 0.9517 - loss: 0.2248
Epoch 49/100
```

```
500/500            1s 3ms/step -
accuracy: 0.9595 - loss: 0.1717
Epoch 50/100
500/500            1s 3ms/step -
accuracy: 0.9562 - loss: 0.1611
Epoch 51/100
500/500            1s 3ms/step -
accuracy: 0.9554 - loss: 0.1948
Epoch 52/100
500/500            1s 3ms/step -
accuracy: 0.9593 - loss: 0.1670
Epoch 53/100
500/500            1s 3ms/step -
accuracy: 0.9533 - loss: 0.2023
Epoch 54/100
500/500            1s 3ms/step -
accuracy: 0.9622 - loss: 0.1607
Epoch 55/100
500/500            1s 3ms/step -
accuracy: 0.9573 - loss: 0.1881
Epoch 56/100
500/500            1s 3ms/step -
accuracy: 0.9577 - loss: 0.1912
Epoch 57/100
500/500            1s 3ms/step -
accuracy: 0.9567 - loss: 0.2051
Epoch 58/100
500/500            1s 3ms/step -
accuracy: 0.9592 - loss: 0.1729
Epoch 59/100
500/500            1s 3ms/step -
accuracy: 0.9559 - loss: 0.1931
Epoch 60/100
500/500            1s 3ms/step -
accuracy: 0.9578 - loss: 0.1841
Epoch 61/100
500/500            1s 3ms/step -
accuracy: 0.9611 - loss: 0.1749
Epoch 62/100
500/500            1s 3ms/step -
accuracy: 0.9685 - loss: 0.1504
Epoch 63/100
500/500            1s 3ms/step -
accuracy: 0.9613 - loss: 0.1762
Epoch 64/100
500/500            1s 3ms/step -
accuracy: 0.9641 - loss: 0.1537
Epoch 65/100
```

```
500/500              1s 3ms/step -
accuracy: 0.9650 - loss: 0.1504
Epoch 66/100
500/500              1s 3ms/step -
accuracy: 0.9552 - loss: 0.2292
Epoch 67/100
500/500              1s 3ms/step -
accuracy: 0.9597 - loss: 0.1914
Epoch 68/100
500/500              1s 3ms/step -
accuracy: 0.9648 - loss: 0.1534
Epoch 69/100
500/500              1s 3ms/step -
accuracy: 0.9600 - loss: 0.2259
Epoch 70/100
500/500              1s 3ms/step -
accuracy: 0.9571 - loss: 0.2217
Epoch 71/100
500/500              1s 3ms/step -
accuracy: 0.9662 - loss: 0.1491
Epoch 72/100
500/500              1s 3ms/step -
accuracy: 0.9663 - loss: 0.1560
Epoch 73/100
500/500              1s 3ms/step -
accuracy: 0.9565 - loss: 0.2122
Epoch 74/100
500/500              1s 3ms/step -
accuracy: 0.9645 - loss: 0.1786
Epoch 75/100
500/500              1s 3ms/step -
accuracy: 0.9681 - loss: 0.1536
Epoch 76/100
500/500              1s 3ms/step -
accuracy: 0.9636 - loss: 0.1748
Epoch 77/100
500/500              1s 3ms/step -
accuracy: 0.9623 - loss: 0.1892
Epoch 78/100
500/500              1s 3ms/step -
accuracy: 0.9674 - loss: 0.1497
Epoch 79/100
500/500              1s 3ms/step -
accuracy: 0.9679 - loss: 0.1532
Epoch 80/100
500/500              1s 3ms/step -
accuracy: 0.9648 - loss: 0.1724
Epoch 81/100
```

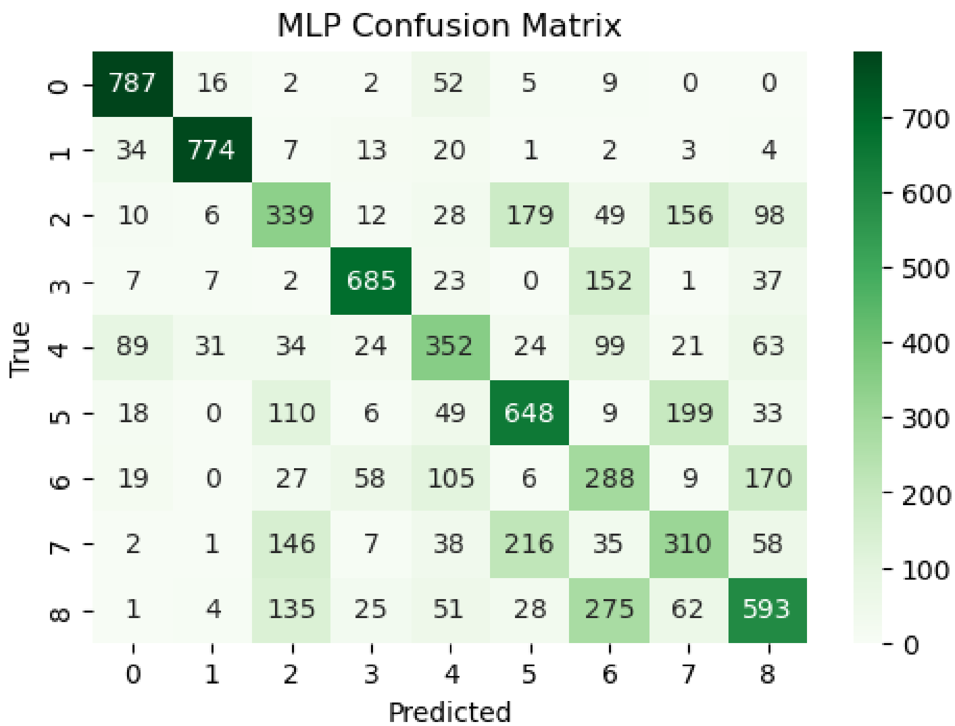
```
500/500 1s 3ms/step -
accuracy: 0.9588 - loss: 0.2350
Epoch 82/100
500/500 2s 3ms/step -
accuracy: 0.9649 - loss: 0.1778
Epoch 83/100
500/500 1s 3ms/step -
accuracy: 0.9668 - loss: 0.1702
Epoch 84/100
500/500 1s 3ms/step -
accuracy: 0.9713 - loss: 0.1293
Epoch 85/100
500/500 1s 3ms/step -
accuracy: 0.9633 - loss: 0.1961
Epoch 86/100
500/500 1s 3ms/step -
accuracy: 0.9642 - loss: 0.1752
Epoch 87/100
500/500 1s 3ms/step -
accuracy: 0.9683 - loss: 0.1596
Epoch 88/100
500/500 1s 3ms/step -
accuracy: 0.9646 - loss: 0.2019
Epoch 89/100
500/500 1s 3ms/step -
accuracy: 0.9690 - loss: 0.2252
Epoch 90/100
500/500 1s 3ms/step -
accuracy: 0.9689 - loss: 0.1564
Epoch 91/100
500/500 1s 3ms/step -
accuracy: 0.9679 - loss: 0.1601
Epoch 92/100
500/500 1s 3ms/step -
accuracy: 0.9698 - loss: 0.1635
Epoch 93/100
500/500 1s 3ms/step -
accuracy: 0.9675 - loss: 0.1750
Epoch 94/100
500/500 1s 3ms/step -
accuracy: 0.9711 - loss: 0.1490
Epoch 95/100
500/500 1s 3ms/step -
accuracy: 0.9679 - loss: 0.1710
Epoch 96/100
500/500 1s 3ms/step -
accuracy: 0.9696 - loss: 0.1661
Epoch 97/100
```

```
500/500              1s 3ms/step -
accuracy: 0.9655 - loss: 0.2215
Epoch 98/100
500/500              1s 3ms/step -
accuracy: 0.9653 - loss: 0.1888
Epoch 99/100
500/500              1s 3ms/step -
accuracy: 0.9707 - loss: 0.1536
Epoch 100/100
500/500              1s 3ms/step -
accuracy: 0.9730 - loss: 0.1553
MLP training time: 137.65 seconds
250/250              0s 625us/step -
accuracy: 0.5974 - loss: 9.7343
Best MLP Test Accuracy: 0.597
```

[33]:

```python
y_pred_mlp = mlp.predict(X_test_pca)
y_pred_labels_mlp = y_pred_mlp.argmax(axis=1)

print("MLP Classification Report:")
print(classification_report(y_test, y_pred_labels_mlp))


plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred_labels_mlp), annot=True, fmt='d',␣
 ↪cmap='Greens')
plt.title("MLP Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
250/250              0s 509us/step
MLP Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.90      0.86       873
           1       0.92      0.90      0.91       858
           2       0.42      0.39      0.40       877
           3       0.82      0.75      0.78       914
           4       0.49      0.48      0.48       737
           5       0.59      0.60      0.59      1072
           6       0.31      0.42      0.36       682
           7       0.41      0.38      0.39       813
           8       0.56      0.51      0.53      1174

    accuracy                           0.60      8000
   macro avg       0.59      0.59      0.59      8000
weighted avg       0.60      0.60      0.60      8000
```



1.5.3 Convolutional neural network

[34]:
```
# best parameter: {'conv_1_filters': 96, 'conv_1_kernel': 3, 'conv_2_filters':
↪64, 'conv_2_kernel': 3, 'dense_units': 640, 'learning_rate': 0.0005}
cnn = keras.Sequential([
    keras.layers.Input(shape=(28, 28, 3)),
    keras.layers.Conv2D(96, kernel_size=3, activation='relu', padding='same'),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Conv2D(64, kernel_size=3, activation='relu', padding='same'),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Flatten(),
    keras.layers.Dense(640, activation='relu'),
    keras.layers.Dense(9, activation='softmax')
])

cnn.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0005),
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])


start_time = time.time()
```

```
cnn.fit(X_train_scaled, y_train, epochs=100, batch_size=64)
cnn_train_time = time.time() - start_time
print(f"CNN training time: {cnn_train_time:.2f} seconds")


_, cnn_accuracy = cnn.evaluate(X_test_scaled, y_test)
print(f"Best CNN Test Accuracy: {cnn_accuracy:.3f}")
```

```
Epoch 1/100
500/500              18s 35ms/step -
accuracy: 0.3873 - loss: 1.6017
Epoch 2/100
500/500              17s 35ms/step -
accuracy: 0.6598 - loss: 0.9202
Epoch 3/100
500/500              18s 35ms/step -
accuracy: 0.7303 - loss: 0.7413
Epoch 4/100
500/500              18s 35ms/step -
accuracy: 0.7662 - loss: 0.6451
Epoch 5/100
500/500              18s 35ms/step -
accuracy: 0.7886 - loss: 0.5802
Epoch 6/100
500/500              18s 35ms/step -
accuracy: 0.8110 - loss: 0.5272
Epoch 7/100
500/500              18s 36ms/step -
accuracy: 0.8186 - loss: 0.4969
Epoch 8/100
500/500              18s 36ms/step -
accuracy: 0.8322 - loss: 0.4633
Epoch 9/100
500/500              18s 36ms/step -
accuracy: 0.8469 - loss: 0.4202
Epoch 10/100
500/500              18s 37ms/step -
accuracy: 0.8487 - loss: 0.4152
Epoch 11/100
500/500              18s 36ms/step -
accuracy: 0.8632 - loss: 0.3808
Epoch 12/100
500/500              18s 36ms/step -
accuracy: 0.8731 - loss: 0.3472
Epoch 13/100
500/500              18s 35ms/step -
accuracy: 0.8902 - loss: 0.3115
Epoch 14/100
```

```
500/500              18s 35ms/step -
accuracy: 0.8959 - loss: 0.2868
Epoch 15/100
500/500              18s 35ms/step -
accuracy: 0.9002 - loss: 0.2762
Epoch 16/100
500/500              18s 36ms/step -
accuracy: 0.9105 - loss: 0.2482
Epoch 17/100
500/500              18s 36ms/step -
accuracy: 0.9197 - loss: 0.2235
Epoch 18/100
500/500              18s 35ms/step -
accuracy: 0.9251 - loss: 0.2102
Epoch 19/100
500/500              18s 35ms/step -
accuracy: 0.9280 - loss: 0.1980
Epoch 20/100
500/500              18s 35ms/step -
accuracy: 0.9422 - loss: 0.1628
Epoch 21/100
500/500              18s 36ms/step -
accuracy: 0.9470 - loss: 0.1496
Epoch 22/100
500/500              18s 35ms/step -
accuracy: 0.9498 - loss: 0.1429
Epoch 23/100
500/500              18s 35ms/step -
accuracy: 0.9571 - loss: 0.1224
Epoch 24/100
500/500              18s 35ms/step -
accuracy: 0.9562 - loss: 0.1277
Epoch 25/100
500/500              18s 36ms/step -
accuracy: 0.9606 - loss: 0.1124
Epoch 26/100
500/500              18s 36ms/step -
accuracy: 0.9695 - loss: 0.0927
Epoch 27/100
500/500              18s 36ms/step -
accuracy: 0.9782 - loss: 0.0716
Epoch 28/100
500/500              18s 36ms/step -
accuracy: 0.9768 - loss: 0.0705
Epoch 29/100
500/500              18s 36ms/step -
accuracy: 0.9833 - loss: 0.0564
Epoch 30/100
```

```
500/500              18s 36ms/step -
accuracy: 0.9842 - loss: 0.0512
Epoch 31/100
500/500              18s 36ms/step -
accuracy: 0.9804 - loss: 0.0602
Epoch 32/100
500/500              18s 36ms/step -
accuracy: 0.9866 - loss: 0.0454
Epoch 33/100
500/500              18s 36ms/step -
accuracy: 0.9859 - loss: 0.0447
Epoch 34/100
500/500              18s 36ms/step -
accuracy: 0.9891 - loss: 0.0420
Epoch 35/100
500/500              18s 36ms/step -
accuracy: 0.9759 - loss: 0.0699
Epoch 36/100
500/500              18s 36ms/step -
accuracy: 0.9866 - loss: 0.0436
Epoch 37/100
500/500              18s 36ms/step -
accuracy: 0.9947 - loss: 0.0218
Epoch 38/100
500/500              18s 36ms/step -
accuracy: 0.9881 - loss: 0.0397
Epoch 39/100
500/500              18s 36ms/step -
accuracy: 0.9780 - loss: 0.0689
Epoch 40/100
500/500              18s 36ms/step -
accuracy: 0.9941 - loss: 0.0219
Epoch 41/100
500/500              18s 36ms/step -
accuracy: 0.9883 - loss: 0.0361
Epoch 42/100
500/500              18s 36ms/step -
accuracy: 0.9909 - loss: 0.0289
Epoch 43/100
500/500              18s 36ms/step -
accuracy: 0.9855 - loss: 0.0458
Epoch 44/100
500/500              18s 36ms/step -
accuracy: 0.9899 - loss: 0.0306
Epoch 45/100
500/500              18s 36ms/step -
accuracy: 0.9842 - loss: 0.0520
Epoch 46/100
```

```
500/500             18s 36ms/step -
accuracy: 0.9911 - loss: 0.0299
Epoch 47/100
500/500             18s 36ms/step -
accuracy: 0.9885 - loss: 0.0363
Epoch 48/100
500/500             18s 36ms/step -
accuracy: 0.9925 - loss: 0.0261
Epoch 49/100
500/500             18s 36ms/step -
accuracy: 0.9843 - loss: 0.0485
Epoch 50/100
500/500             18s 36ms/step -
accuracy: 0.9974 - loss: 0.0107
Epoch 51/100
500/500             18s 36ms/step -
accuracy: 0.9827 - loss: 0.0502
Epoch 52/100
500/500             18s 36ms/step -
accuracy: 0.9984 - loss: 0.0075
Epoch 53/100
500/500             18s 37ms/step -
accuracy: 0.9931 - loss: 0.0225
Epoch 54/100
500/500             18s 36ms/step -
accuracy: 0.9926 - loss: 0.0260
Epoch 55/100
500/500             18s 36ms/step -
accuracy: 0.9980 - loss: 0.0082
Epoch 56/100
500/500             18s 36ms/step -
accuracy: 0.9862 - loss: 0.0449
Epoch 57/100
500/500             18s 36ms/step -
accuracy: 0.9985 - loss: 0.0076
Epoch 58/100
500/500             18s 36ms/step -
accuracy: 0.9855 - loss: 0.0445
Epoch 59/100
500/500             18s 36ms/step -
accuracy: 0.9945 - loss: 0.0160
Epoch 60/100
500/500             18s 36ms/step -
accuracy: 0.9893 - loss: 0.0322
Epoch 61/100
500/500             18s 36ms/step -
accuracy: 0.9992 - loss: 0.0038
Epoch 62/100
```

```
500/500            18s 36ms/step -
accuracy: 0.9965 - loss: 0.0115
Epoch 63/100
500/500            18s 36ms/step -
accuracy: 0.9906 - loss: 0.0281
Epoch 64/100
500/500            18s 36ms/step -
accuracy: 0.9961 - loss: 0.0138
Epoch 65/100
500/500            18s 36ms/step -
accuracy: 0.9959 - loss: 0.0132
Epoch 66/100
500/500            18s 37ms/step -
accuracy: 0.9984 - loss: 0.0067
Epoch 67/100
500/500            18s 36ms/step -
accuracy: 0.9813 - loss: 0.0596
Epoch 68/100
500/500            18s 36ms/step -
accuracy: 0.9998 - loss: 0.0012
Epoch 69/100
500/500            18s 37ms/step -
accuracy: 0.9998 - loss: 0.0012
Epoch 70/100
500/500            18s 37ms/step -
accuracy: 0.9947 - loss: 0.0173
Epoch 71/100
500/500            18s 37ms/step -
accuracy: 0.9985 - loss: 0.0080
Epoch 72/100
500/500            18s 37ms/step -
accuracy: 0.9961 - loss: 0.0150
Epoch 73/100
500/500            18s 37ms/step -
accuracy: 0.9874 - loss: 0.0368
Epoch 74/100
500/500            18s 37ms/step -
accuracy: 0.9994 - loss: 0.0033
Epoch 75/100
500/500            18s 37ms/step -
accuracy: 0.9985 - loss: 0.0067
Epoch 76/100
500/500            18s 36ms/step -
accuracy: 0.9905 - loss: 0.0310
Epoch 77/100
500/500            18s 36ms/step -
accuracy: 1.0000 - loss: 9.7236e-04
Epoch 78/100
```

```
500/500              18s 36ms/step -
accuracy: 1.0000 - loss: 4.2210e-04
Epoch 79/100
500/500              18s 36ms/step -
accuracy: 0.9790 - loss: 0.0649
Epoch 80/100
500/500              18s 36ms/step -
accuracy: 0.9938 - loss: 0.0226
Epoch 81/100
500/500              18s 36ms/step -
accuracy: 0.9995 - loss: 0.0021
Epoch 82/100
500/500              18s 36ms/step -
accuracy: 0.9912 - loss: 0.0306
Epoch 83/100
500/500              18s 36ms/step -
accuracy: 0.9999 - loss: 0.0012
Epoch 84/100
500/500              18s 36ms/step -
accuracy: 0.9991 - loss: 0.0034
Epoch 85/100
500/500              18s 36ms/step -
accuracy: 0.9876 - loss: 0.0406
Epoch 86/100
500/500              18s 36ms/step -
accuracy: 0.9997 - loss: 0.0021
Epoch 87/100
500/500              18s 37ms/step -
accuracy: 0.9998 - loss: 0.0013
Epoch 88/100
500/500              18s 37ms/step -
accuracy: 0.9779 - loss: 0.0729
Epoch 89/100
500/500              18s 37ms/step -
accuracy: 0.9995 - loss: 0.0027
Epoch 90/100
500/500              19s 38ms/step -
accuracy: 0.9899 - loss: 0.0307
Epoch 91/100
500/500              19s 37ms/step -
accuracy: 0.9996 - loss: 0.0024
Epoch 92/100
500/500              18s 37ms/step -
accuracy: 0.9972 - loss: 0.0094
Epoch 93/100
500/500              18s 37ms/step -
accuracy: 0.9948 - loss: 0.0181
Epoch 94/100
```

```
500/500                18s 36ms/step -
accuracy: 0.9998 - loss: 0.0018
Epoch 95/100
500/500                18s 37ms/step -
accuracy: 0.9994 - loss: 0.0029
Epoch 96/100
500/500                18s 37ms/step -
accuracy: 0.9875 - loss: 0.0367
Epoch 97/100
500/500                18s 37ms/step -
accuracy: 0.9981 - loss: 0.0071
Epoch 98/100
500/500                18s 37ms/step -
accuracy: 0.9989 - loss: 0.0052
Epoch 99/100
500/500                19s 37ms/step -
accuracy: 0.9875 - loss: 0.0371
Epoch 100/100
500/500                18s 37ms/step -
accuracy: 0.9999 - loss: 9.8921e-04
CNN training time: 1810.31 seconds
250/250                1s 5ms/step -
accuracy: 0.8536 - loss: 0.9121
Best CNN Test Accuracy: 0.857
```

[35]:

```python
y_pred_cnn = cnn.predict(X_test_scaled)
y_pred_labels_cnn = y_pred_cnn.argmax(axis=1)


print("CNN Classification Report:")
print(classification_report(y_test, y_pred_labels_cnn))


plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred_labels_cnn), annot=True, fmt='d',␣
 ↪cmap='Oranges')
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
250/250                1s 5ms/step
CNN Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.99      0.96       873
           1       0.98      0.97      0.98       858
           2       0.81      0.71      0.76       877
```

3	0.95	0.96	0.95	914
4	0.83	0.87	0.85	737
5	0.83	0.84	0.83	1072
6	0.80	0.79	0.80	682
7	0.67	0.68	0.67	813
8	0.86	0.88	0.87	1174
accuracy			0.86	8000
macro avg	0.85	0.85	0.85	8000
weighted avg	0.86	0.86	0.86	8000