Facial Recognition

February 11, 2026

1 M8 Project

1.0.1 Justin Stutler

In this project you will compare the performance of LDA and SVM for face recognition. You will use the Olivetti faces dataset, which contains 400 64x64 images from 40 different subjects, and your task is to discover the identity of a given face image. Some of these images are illustrated below.

An initial version of the code with the problem specification (below) and a report template are available (at the bottom). Deliverables are the final code (non-functioning code is worth 0 points) and the comparison report.

Solve the task above using: - LDA (20pts) - SVM (20pts)

For LDA: - Visualize a 2D representation of the faces in the dataset (20pts)

Split the dataset so that the first 5 images per subject are used for training, and the last 5 images are used for testing. If you need a validation set, use part of your training data. Compare the performance of LDA and SVM in terms of: - Average F-Score (15pts) - Confusion matrix (15pts) - Visualize the individuals with highest confusion and check if they look alike (10pts)

2 Implementation

You are free to change the code below as needed.

[1]:
```
from sklearn.datasets import fetch_olivetti_faces
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
# Get the dataset

faces = fetch_olivetti_faces()

_, img_height, img_width = faces.images.shape

print(faces.images.shape)
```

```
(400, 64, 64)
```

[3]:
```
# Split the dataset

N_IDENTITIES = len(np.unique(faces.target)) # how many different individuals␣
 ↪are in the dataset
GALLERY_SIZE = 8                            # use the first GALLERY_SIZE images␣
 ↪per individual for training, the rest for testing

gallery_indices = []
probe_indices = []
for i in range(N_IDENTITIES):
    indices = list(np.where(faces.target == i)[0])
    gallery_indices += indices[:GALLERY_SIZE]
    probe_indices += indices[GALLERY_SIZE:]

x_train = faces.images[gallery_indices].reshape(-1, img_height*img_width) #␣
 ↪vectorize train images
y_train = faces.target[gallery_indices]
x_test = faces.images[probe_indices].reshape(-1, img_height*img_width)    #␣
 ↪vectorize test images
y_test = faces.target[probe_indices]

print(x_train.shape, x_test.shape)
```

```
(320, 4096) (80, 4096)
```

[4]:
```
# Visualize image sets
def show_images(imgs, num_rows, num_cols):
    assert len(imgs) == num_rows*num_cols

    full = None
    for i in range(num_rows):
        row = None
        for j in range(num_cols):
            if row is None:
                row = imgs[i*num_cols+j].reshape(img_height, img_width)*255.0
            else:
                row = np.concatenate((row, imgs[i*num_cols+j].
 ↪reshape(img_height, img_width)*255.0), axis=1)
        if full is None:
            full = row
        else:
            full = np.concatenate((full, row), axis=0)

    f = plt.figure(figsize=(num_cols, num_rows))
    plt.imshow(full, cmap='gray')
    plt.axis('off')
    plt.show()
```

```
print('TRAINING')
show_images(x_train, N_IDENTITIES, GALLERY_SIZE)
print('TESTING')
show_images(x_test, N_IDENTITIES, 10 - GALLERY_SIZE)
```

```
TRAINING
```

TESTING

3 LDA

3.0.1 Imports

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

3.0.2 Confusion Matrix Function

[6]:
```
def plot_confusion(cm, title):
    plt.rcParams["figure.figsize"] = (15,15)
    fig, ax = plt.subplots()
    c= ax.matshow(cm, cmap='copper')
    fig.colorbar(c, ax=ax)
    for i in range(len(cm)):
        for j in range(len(cm[0])):
            text = ax.text(j, i, cm[i, j], ha='center', va='center', color='w',␣
 ↪fontsize=10, weight='bold')
    plt.xticks(range(40))
    plt.yticks(range(40))
    plt.title(title)
    plt.show()
```

3.0.3 Shape

[7]:
```
print("x_train shape", x_train.shape)
print("y_train shape", y_train.shape)
```

```
x_train shape (320, 4096)
y_train shape (320,)
```

3.0.4 2D LDA Visualization

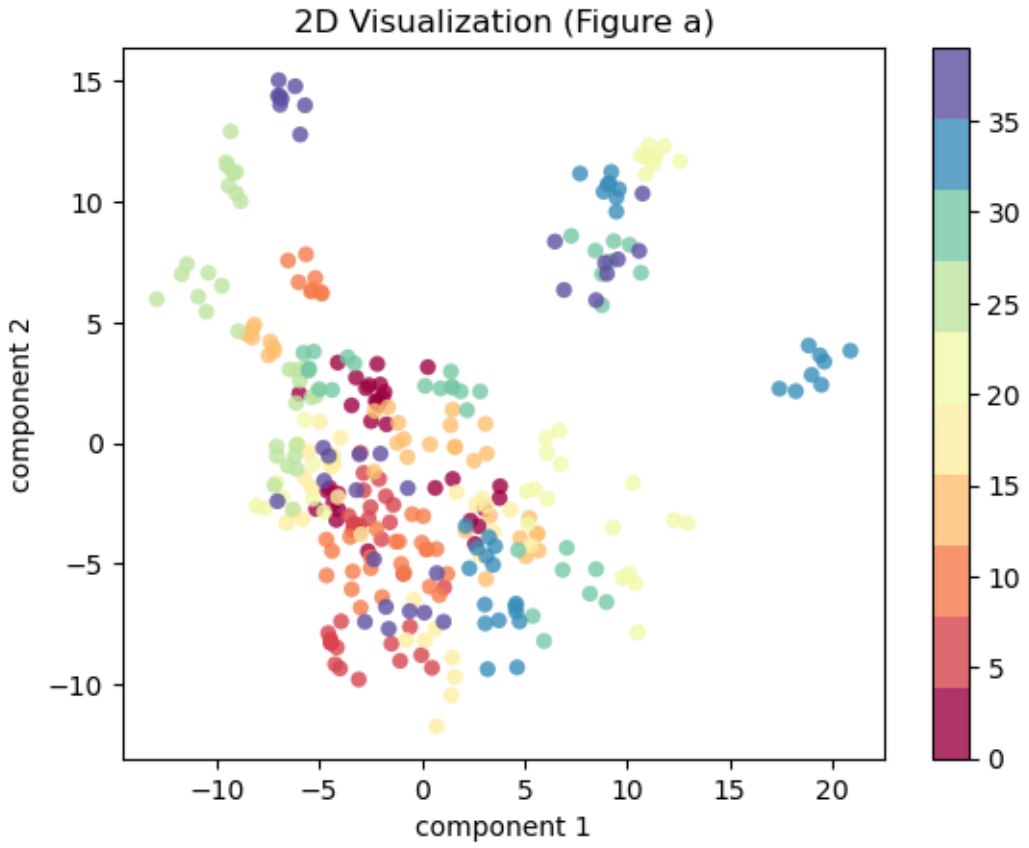
[8]:
```
lda = LDA(n_components=2)  # project from 4096 to 2 dimensions
X_train_projected = lda.fit_transform(x_train, y_train)
print(x_train.shape)
print(X_train_projected.shape)

print(lda.explained_variance_ratio_)
```

```
(320, 4096)
(320, 2)
```

```
[0.14880966 0.12554616]
```

[9]:
```
plt.scatter(X_train_projected[:, 0], X_train_projected[:, 1], c=y_train,␣
 ↪edgecolor='none', alpha=0.8, cmap=plt.cm.get_cmap('Spectral', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.title('2D Visualization (Figure a)')
plt.colorbar();
```

```
C:\Users\justi\AppData\Local\Temp\ipykernel_86316\464587426.py:1:
MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib
3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  plt.scatter(X_train_projected[:, 0], X_train_projected[:, 1], c=y_train,
edgecolor='none', alpha=0.8, cmap=plt.cm.get_cmap('Spectral', 10))
```

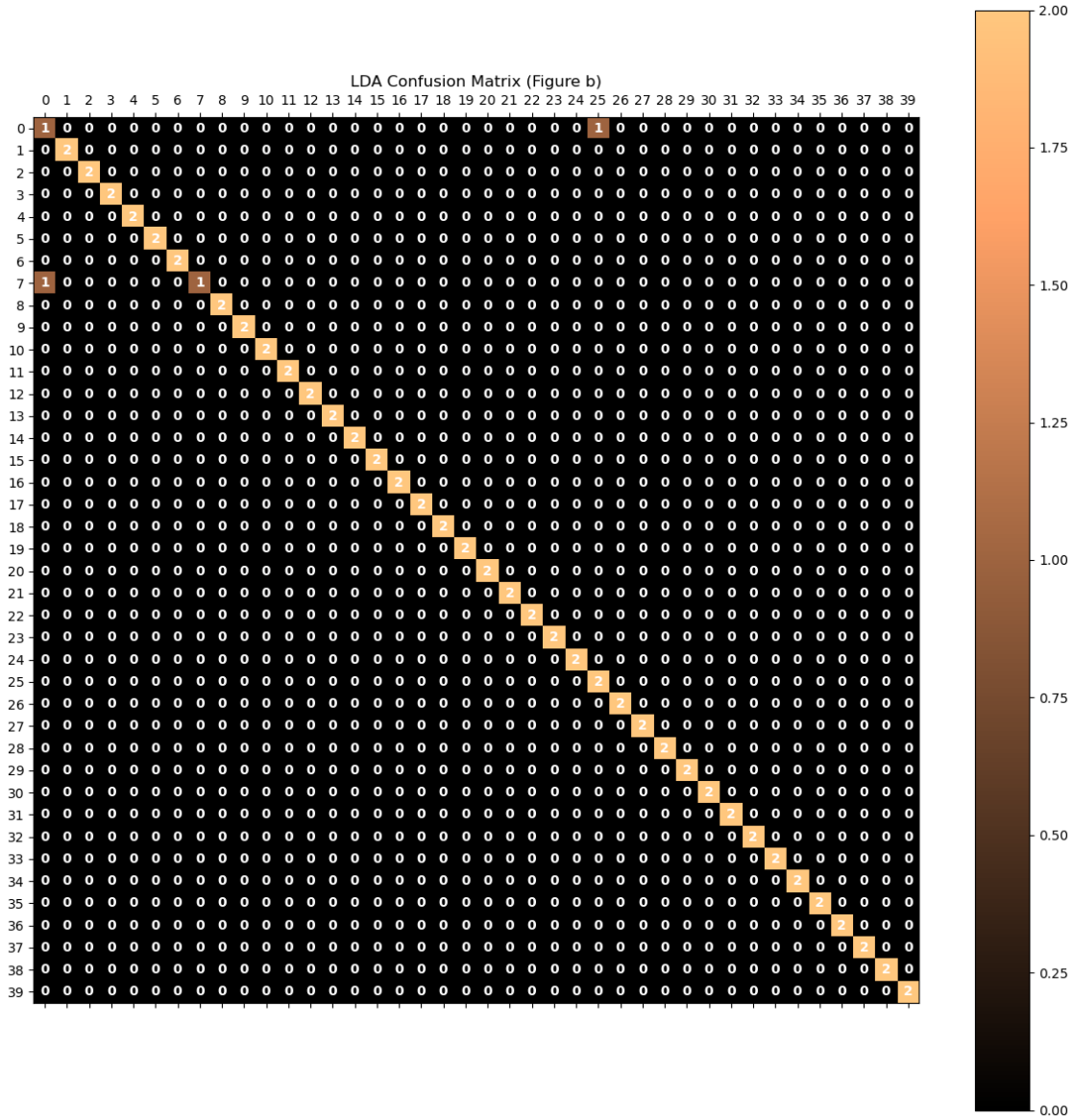


3.0.5 Confusion Matrix and F-score

[10]:
```
lda = LDA().fit(x_train, y_train)
y_pred = lda.predict(x_test)

# min_score used to track highest confusion
lda_min_score = float('inf')
lda_min_score_class = float('inf')

# compute F-score for each class
fscores = []
for j in range(40):
    positives = len(y_pred[np.where(y_test == j)])
    true_positives = len(y_pred[np.where((y_pred == j) & (y_test == j))])
    false_positives = len(y_pred[np.where((y_pred == j) & (y_test != j))])

    if true_positives+false_positives == 0:
        precision = 0
    else:
        precision = true_positives/(true_positives+false_positives)
    recall = true_positives/positives

    if precision+recall == 0:
        fscore = 0
    else:
        fscore = (2.0*precision*recall)/(precision+recall)
    fscores.append(fscore)
    if lda_min_score > fscore:
        lda_min_score = fscore
        lda_min_score_class = j

    print('Class: {} / F-score: {}'.format(j, fscore))

print('Average F-score:', np.mean(fscores))
cm = confusion_matrix(y_test, y_pred)
plot_confusion(cm, "LDA Confusion Matrix (Figure b)")
print('Class: {} / has the minimum F-score: {}'.format(lda_min_score_class,␣
 ↪lda_min_score))
```

```
Class: 0 / F-score: 0.5
Class: 1 / F-score: 1.0
Class: 2 / F-score: 1.0
Class: 3 / F-score: 1.0
Class: 4 / F-score: 1.0
Class: 5 / F-score: 1.0
Class: 6 / F-score: 1.0
Class: 7 / F-score: 0.6666666666666666
Class: 8 / F-score: 1.0
```

```
Class: 9 / F-score: 1.0
Class: 10 / F-score: 1.0
Class: 11 / F-score: 1.0
Class: 12 / F-score: 1.0
Class: 13 / F-score: 1.0
Class: 14 / F-score: 1.0
Class: 15 / F-score: 1.0
Class: 16 / F-score: 1.0
Class: 17 / F-score: 1.0
Class: 18 / F-score: 1.0
Class: 19 / F-score: 1.0
Class: 20 / F-score: 1.0
Class: 21 / F-score: 1.0
Class: 22 / F-score: 1.0
Class: 23 / F-score: 1.0
Class: 24 / F-score: 1.0
Class: 25 / F-score: 0.8
Class: 26 / F-score: 1.0
Class: 27 / F-score: 1.0
Class: 28 / F-score: 1.0
Class: 29 / F-score: 1.0
Class: 30 / F-score: 1.0
Class: 31 / F-score: 1.0
Class: 32 / F-score: 1.0
Class: 33 / F-score: 1.0
Class: 34 / F-score: 1.0
Class: 35 / F-score: 1.0
Class: 36 / F-score: 1.0
Class: 37 / F-score: 1.0
Class: 38 / F-score: 1.0
Class: 39 / F-score: 1.0
Average F-score: 0.9741666666666667
```




```
Class: 0 / has the minimum F-score: 0.5
```

3.0.6 Visualize Faces with Highest Confusion

[11]:
```
# copy confusion matrix
lda_cm = cm.copy()
# remove correct predictions by setting diag to 0
np.fill_diagonal(lda_cm, 0)
# find row and col index of highest confusion
max_conf = np.max(lda_cm)
r, c = np.where(lda_cm == max_conf)
# display all highest confused pairs of faces
```

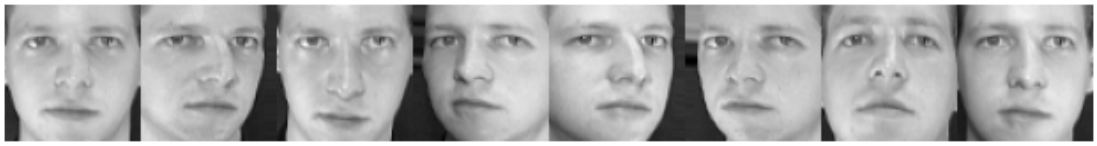
```
for i in range(len(r)):
    actual_face = r[i]
    predicted_face = c[i]
    count = cm[actual_face, predicted_face] # counts confusion
    # print highest confusion
    print(f'Highest confusion: Class {actual_face} was predicted as Class␣
 ↪{predicted_face} {count} times.')
    # print figure c
    print(f'\n(Figure c', (i+1),')')
    print(f'\nVisualizing training images for comparison:')
    print(f'Actual Class {actual_face}')
    actual_face_images = x_train[np.where(y_train == actual_face)]
    show_images(actual_face_images, num_rows=1, num_cols=8)
    print(f'Predicted Class {predicted_face}')
    predicted_face_images = x_train[np.where(y_train == predicted_face)]
    show_images(predicted_face_images, num_rows=1, num_cols=8)
```

```
Highest confusion: Class 0 was predicted as Class 25 1 times.

(Figure c 1 )

Visualizing training images for comparison:
Actual Class 0
```

```
Predicted Class 25
```

```
Highest confusion: Class 7 was predicted as Class 0 1 times.

(Figure c 2 )

Visualizing training images for comparison:
Actual Class 7
```

```
Predicted Class 0
```



4 SVM

4.0.1 Imports

[12]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
```

[13]:
```
print("x_train shape", x_train.shape)
print("y_train shape", y_train.shape)
```

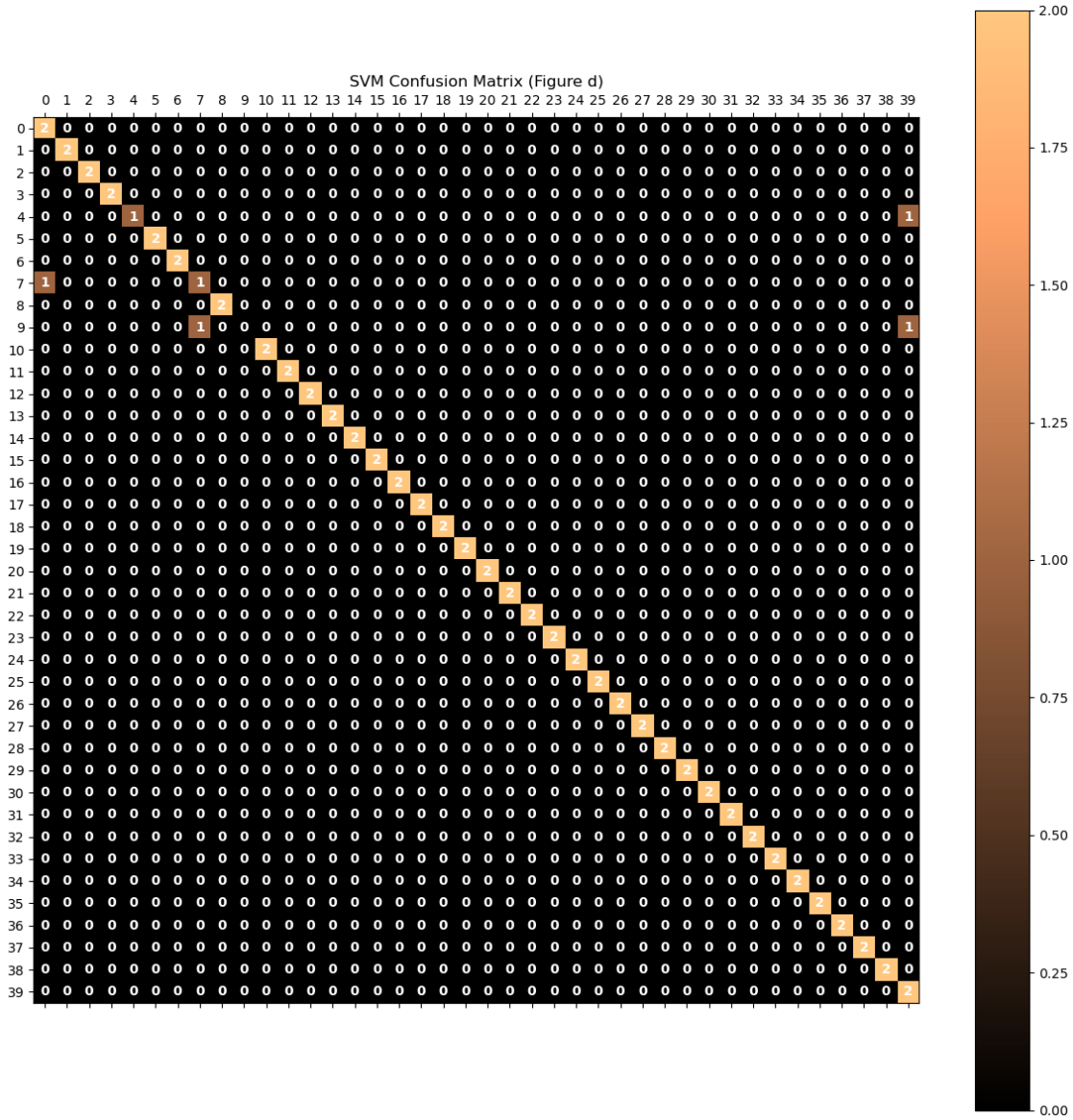
```
x_train shape (320, 4096)
y_train shape (320,)
```

[14]:
```
svm = SVC(kernel='linear')
svm.fit(x_train, y_train)
y_pred = svm.predict(x_test)

# min_score used to track highest confusion
svm_min_score = float('inf')
svm_min_score_class = float('inf')

# compute F-score for each class
fscores = []
for j in range(40):
    positives = len(y_pred[np.where(y_test == j)])
    true_positives = len(y_pred[np.where((y_pred == j) & (y_test == j))])
    false_positives = len(y_pred[np.where((y_pred == j) & (y_test != j))])
```

```
    if true_positives+false_positives == 0:
        precision = 0
    else:
        precision = true_positives/(true_positives+false_positives)
    recall = true_positives/positives

    if precision+recall == 0:
        fscore = 0
    else:
        fscore = (2.0*precision*recall)/(precision+recall)
    fscores.append(fscore)
    # get min f score
    if svm_min_score > fscore:
        svm_min_score = fscore
        svm_min_score_class = j

    print('Class: {} / F-score: {}'.format(j, fscore))

print('Average F-score:', np.mean(fscores))
cm = confusion_matrix(y_test, y_pred)
plot_confusion(cm, "SVM Confusion Matrix (Figure d)")
print('Class: {} / has the minimum F-score: {}'.format(svm_min_score_class,␣
 ↪svm_min_score))
```

```
Class: 0 / F-score: 0.8
Class: 1 / F-score: 1.0
Class: 2 / F-score: 1.0
Class: 3 / F-score: 1.0
Class: 4 / F-score: 0.6666666666666666
Class: 5 / F-score: 1.0
Class: 6 / F-score: 1.0
Class: 7 / F-score: 0.5
Class: 8 / F-score: 1.0
Class: 9 / F-score: 0
Class: 10 / F-score: 1.0
Class: 11 / F-score: 1.0
Class: 12 / F-score: 1.0
Class: 13 / F-score: 1.0
Class: 14 / F-score: 1.0
Class: 15 / F-score: 1.0
Class: 16 / F-score: 1.0
Class: 17 / F-score: 1.0
Class: 18 / F-score: 1.0
Class: 19 / F-score: 1.0
Class: 20 / F-score: 1.0
Class: 21 / F-score: 1.0
Class: 22 / F-score: 1.0
Class: 23 / F-score: 1.0
```

```
Class: 24 / F-score: 1.0
Class: 25 / F-score: 1.0
Class: 26 / F-score: 1.0
Class: 27 / F-score: 1.0
Class: 28 / F-score: 1.0
Class: 29 / F-score: 1.0
Class: 30 / F-score: 1.0
Class: 31 / F-score: 1.0
Class: 32 / F-score: 1.0
Class: 33 / F-score: 1.0
Class: 34 / F-score: 1.0
Class: 35 / F-score: 1.0
Class: 36 / F-score: 1.0
Class: 37 / F-score: 1.0
Class: 38 / F-score: 1.0
Class: 39 / F-score: 0.6666666666666666
Average F-score: 0.9408333333333335
```




```
Class: 9 / has the minimum F-score: 0
```

4.0.2 Visualize Faces with the Highest Confusion

[15]:
```
# copy confusion matrix
svm_cm = cm.copy()
# remove correct predictions by setting diag to 0
np.fill_diagonal(svm_cm, 0)
# find row and col index of highest confusion
max_conf = np.max(svm_cm)
r, c = np.where(svm_cm == max_conf)
# display all highest confused pairs of faces
```

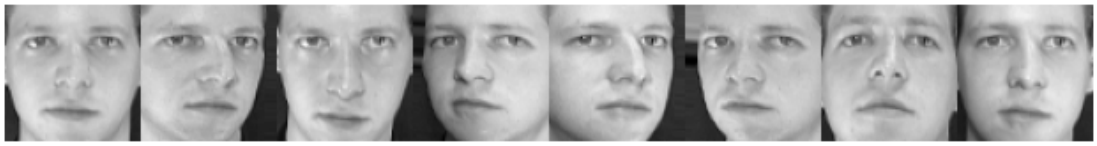
```
for i in range(len(r)):
    actual_face = r[i]
    predicted_face = c[i]
    count = cm[actual_face, predicted_face] # counts confusion
    # print highest confusion
    print(f'Highest confusion: Class {actual_face} was predicted as Class␣
 ↪{predicted_face} {count} times.')
    # print figure e
    print(f'\n(Figure e',(i+1),')')
    print(f'\nVisualizing training images for comparison:')
    print(f'Actual Class {actual_face}')
    actual_face_images = x_train[np.where(y_train == actual_face)]
    show_images(actual_face_images, num_rows=1, num_cols=8)
    print(f'Predicted Class {predicted_face}')
    predicted_face_images = x_train[np.where(y_train == predicted_face)]
    show_images(predicted_face_images, num_rows=1, num_cols=8)
```

```
Highest confusion: Class 4 was predicted as Class 39 1 times.

(Figure e 1 )

Visualizing training images for comparison:
Actual Class 4
```

```
Predicted Class 39
```

```
Highest confusion: Class 7 was predicted as Class 0 1 times.

(Figure e 2 )

Visualizing training images for comparison:
Actual Class 7
```

Predicted Class 0



Highest confusion: Class 9 was predicted as Class 7 1 times.

(Figure e 3)

Visualizing training images for comparison:
Actual Class 9

Predicted Class 7

Highest confusion: Class 9 was predicted as Class 39 1 times.

(Figure e 4)

Visualizing training images for comparison:
Actual Class 9

```
Predicted Class 39
```

5 CNN

Imports

[16]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

Create Validation Set from Training Set

[17]:
```
from sklearn.model_selection import train_test_split

# 1. Reshape the original flattened data back into images
# Reshapes from (N, 4096) -> (N, 64, 64, 1)
x_train_images = x_train.reshape(-1, 64, 64, 1)
x_test_images = x_test.reshape(-1, 64, 64, 1)

# 2. Create the Validation Set (Splitting the training data)
# Stratify ensures you keep the same distribution of faces in train and val
x_train_cnn, x_val_cnn, y_train_cnn, y_val_cnn = train_test_split(
    x_train_images, y_train,
    test_size=0.2,
    random_state=42,
    stratify=y_train
)

# 3. Verify shapes (Optional but recommended)
print("Training Shape:", x_train_cnn.shape) # Should be (approx 256, 64, 64, 1)
print("Validation Shape:", x_val_cnn.shape) # Should be (approx 64, 64, 64, 1)
```

```
Training Shape: (256, 64, 64, 1)
Validation Shape: (64, 64, 64, 1)
```

[18]:
```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(filters=6, kernel_size=(5, 5),␣
 ↪activation='relu', input_shape=(64, 64, 1))) # shape of image
model.add(tf.keras.layers.MaxPool2D())
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=(5, 5),␣
 ↪activation='relu'))
model.add(tf.keras.layers.MaxPool2D())
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=120, activation='relu'))
model.add(tf.keras.layers.Dense(units=84, activation='relu'))
model.add(tf.keras.layers.Dense(units=40, activation = 'softmax')) # 40 classes:
 ↪ individuals

print(model.summary())
```

```
C:\Users\justi\anaconda3\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 60, 60, 6)	156
max_pooling2d (MaxPooling2D)	(None, 30, 30, 6)	0
conv2d_1 (Conv2D)	(None, 26, 26, 16)	2,416
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 16)	0
flatten (Flatten)	(None, 2704)	0
dense (Dense)	(None, 120)	324,600

```
 dense_1 (Dense)                      (None, 84)                  ␣
↪10,164

 dense_2 (Dense)                      (None, 40)                  ␣
↪3,400


 Total params: 340,736 (1.30 MB)

 Trainable params: 340,736 (1.30 MB)

 Non-trainable params: 0 (0.00 B)
```

```
None
```

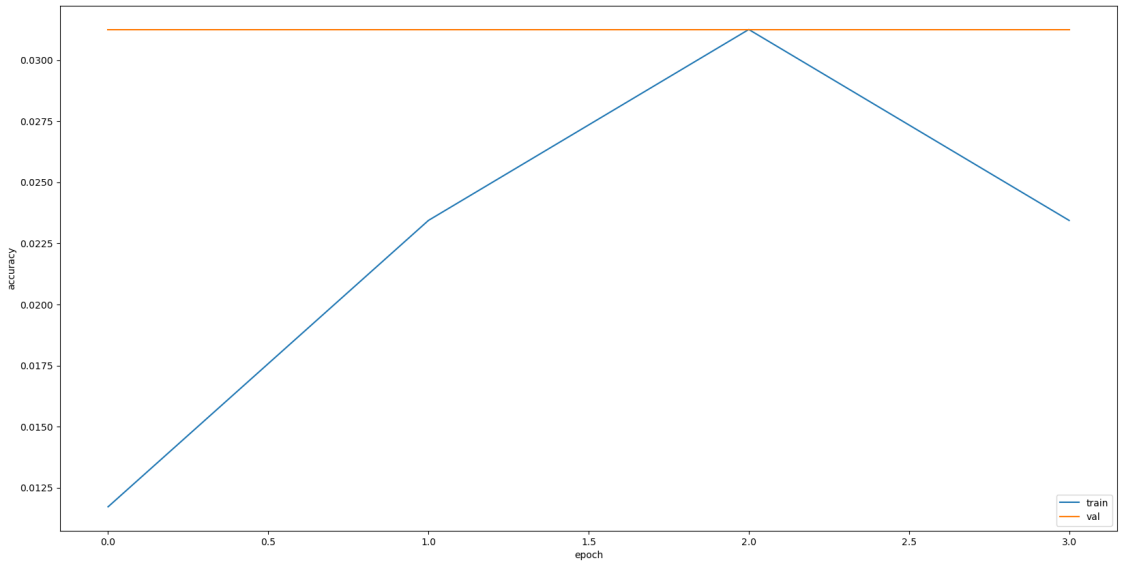
[19]:
```
learning_rate = 0.001
loss_function = 'sparse_categorical_crossentropy'
batch_size = 32
num_epochs = 50
```

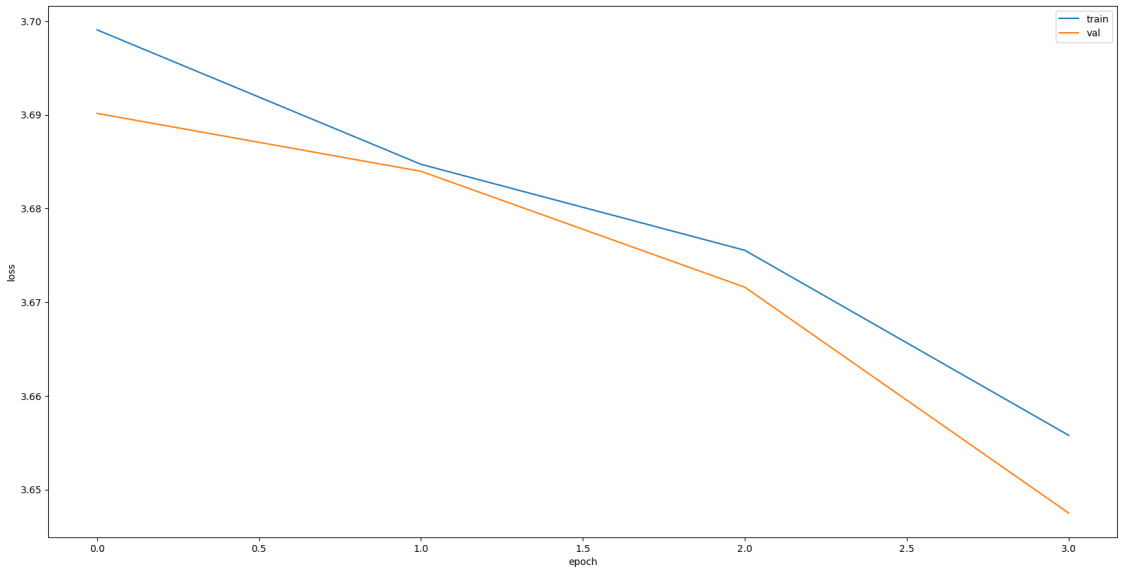
[20]:
```
# training configuration
optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss=loss_function, metrics=['accuracy'])

early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',␣
 ↪mode='max', patience=3, verbose=1)
model_save = tf.keras.callbacks.ModelCheckpoint(filepath='best_model.keras',␣
 ↪monitor='val_accuracy', save_best_only=True)

#train
history = model.fit(
    x=x_train_cnn,
    y=y_train_cnn,
    epochs=num_epochs,
    batch_size=batch_size,
    validation_data=(x_val_cnn, y_val_cnn),
    callbacks=[early_stop, model_save]
)
```

```
Epoch 1/50
8/8             3s 91ms/step -
accuracy: 0.0117 - loss: 3.6991 - val_accuracy: 0.0312 - val_loss: 3.6901
Epoch 2/50
8/8             0s 32ms/step -
accuracy: 0.0234 - loss: 3.6847 - val_accuracy: 0.0312 - val_loss: 3.6840
```

```
Epoch 3/50
8/8             0s 31ms/step -
accuracy: 0.0312 - loss: 3.6755 - val_accuracy: 0.0312 - val_loss: 3.6716
Epoch 4/50
8/8             0s 38ms/step -
accuracy: 0.0234 - loss: 3.6558 - val_accuracy: 0.0312 - val_loss: 3.6475
Epoch 4: early stopping
```

[21]:
```
plt.rcParams["figure.figsize"] = (20,10)

# accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='lower right')
plt.show()

# loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
```





[22]:
```
# restore weights from the best training point
model.load_weights('best_model.h5')

scores = model.evaluate(x_train_cnn, y_train_cnn, verbose=0)
print('TRAINING SET\nLoss:', scores[0], '\nAccuracy:', scores[1])

scores = model.evaluate(x_val_cnn, y_val_cnn, verbose=0)
print('\nVALIDATION SET\nLoss:', scores[0], '\nAccuracy:', scores[1])

scores = model.evaluate(x_test_images, y_test, verbose=0)
print('\nTEST SET\nLoss:', scores[0], '\nAccuracy:', scores[1])
```

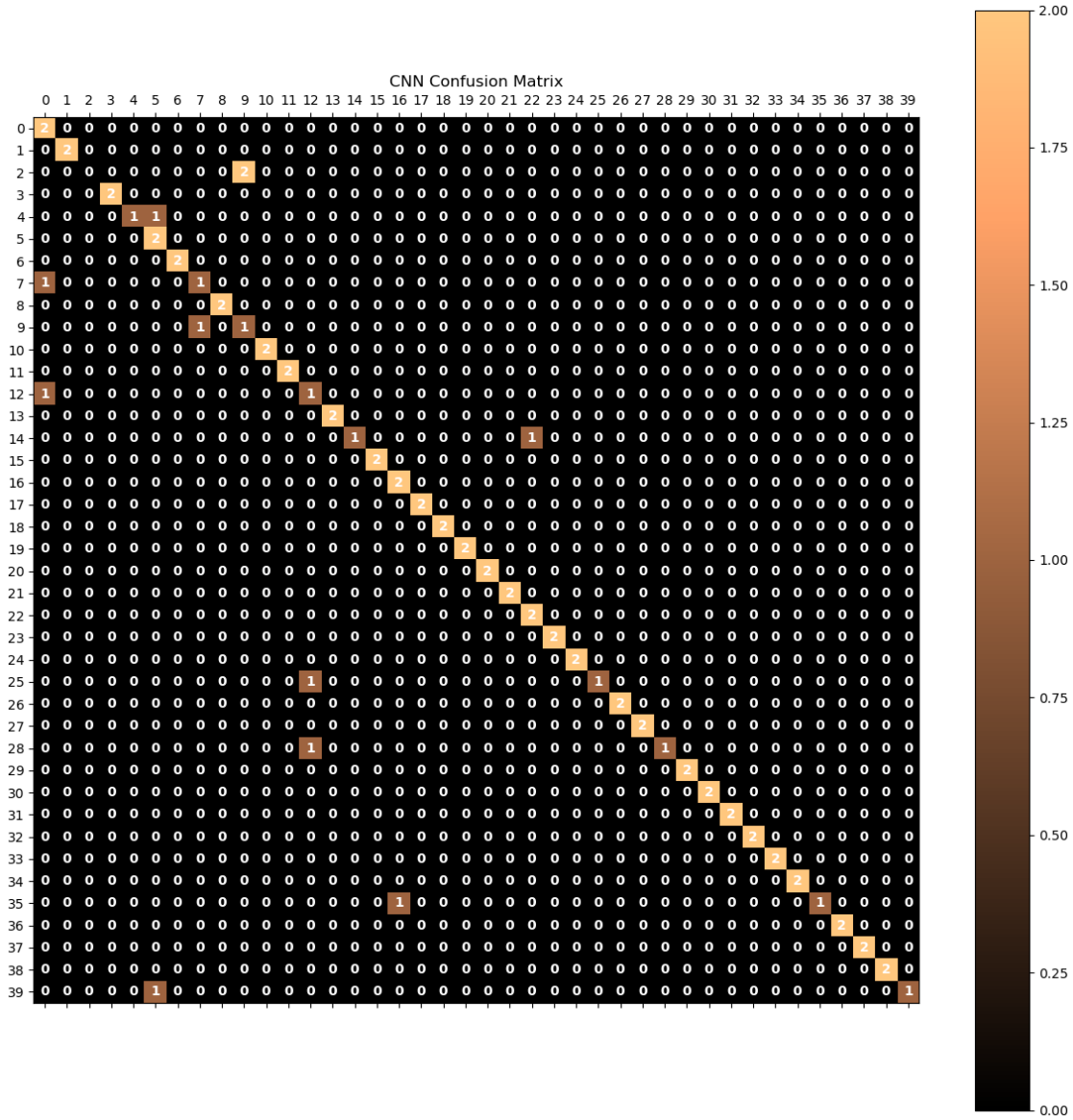
```
TRAINING SET
Loss: 0.02793259173631668
Accuracy: 1.0

VALIDATION SET
Loss: 0.5169829726219177
Accuracy: 0.875

TEST SET
Loss: 0.8993827104568481
Accuracy: 0.862500011920929
```

5.0.1 Plot Confusion Matrix for CNN and F-Score

[23]:
```
# get predictions using the reshaped images
# model returns probabilities, so we take the argmax to get the class integer
predictions = model.predict(x_test_images)
y_pred = np.argmax(predictions, axis=1)

# min_score used to track highest confusion
cnn_min_score = float('inf')
cnn_min_score_class = float('inf')

# compute f-score for each class
fscores = []
for j in range(40):
    positives = len(y_pred[np.where(y_test == j)])
    true_positives = len(y_pred[np.where((y_pred == j) & (y_test == j))])
    false_positives = len(y_pred[np.where((y_pred == j) & (y_test != j))])

    if true_positives+false_positives == 0:
        precision = 0
    else:
        precision = true_positives/(true_positives+false_positives)

    # avoid division by zero if positives is 0 (though unlikely in this dataset)
    if positives == 0:
        recall = 0
    else:
        recall = true_positives/positives

    if precision+recall == 0:
        fscore = 0
    else:
        fscore = (2.0*precision*recall)/(precision+recall)

    fscores.append(fscore)

    # get min f score
    if cnn_min_score > fscore:
        cnn_min_score = fscore
        cnn_min_score_class = j

    print('Class: {} / F-score: {}'.format(j, fscore))

print('Average F-score:', np.mean(fscores))

# plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plot_confusion(cm, "CNN Confusion Matrix")
```

```
print('Class: {} / has the minimum F-score: {}'.format(cnn_min_score_class,␣
 ↪cnn_min_score))
```

```
3/3             0s 76ms/step
Class: 0 / F-score: 0.6666666666666666
Class: 1 / F-score: 1.0
Class: 2 / F-score: 0
Class: 3 / F-score: 1.0
Class: 4 / F-score: 0.6666666666666666
Class: 5 / F-score: 0.6666666666666666
Class: 6 / F-score: 1.0
Class: 7 / F-score: 0.5
Class: 8 / F-score: 1.0
Class: 9 / F-score: 0.4
Class: 10 / F-score: 1.0
Class: 11 / F-score: 1.0
Class: 12 / F-score: 0.4
Class: 13 / F-score: 1.0
Class: 14 / F-score: 0.6666666666666666
Class: 15 / F-score: 1.0
Class: 16 / F-score: 0.8
Class: 17 / F-score: 1.0
Class: 18 / F-score: 1.0
Class: 19 / F-score: 1.0
Class: 20 / F-score: 1.0
Class: 21 / F-score: 1.0
Class: 22 / F-score: 0.8
Class: 23 / F-score: 1.0
Class: 24 / F-score: 1.0
Class: 25 / F-score: 0.6666666666666666
Class: 26 / F-score: 1.0
Class: 27 / F-score: 1.0
Class: 28 / F-score: 0.6666666666666666
Class: 29 / F-score: 1.0
Class: 30 / F-score: 1.0
Class: 31 / F-score: 1.0
Class: 32 / F-score: 1.0
Class: 33 / F-score: 1.0
Class: 34 / F-score: 1.0
Class: 35 / F-score: 0.6666666666666666
Class: 36 / F-score: 1.0
Class: 37 / F-score: 1.0
Class: 38 / F-score: 1.0
Class: 39 / F-score: 0.6666666666666666
Average F-score: 0.8558333333333333
```



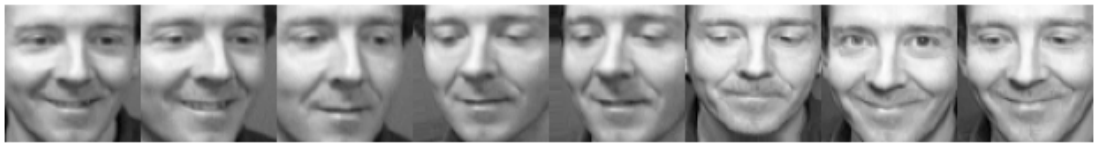

```
Class: 2 / has the minimum F-score: 0
```

5.0.2 Visualize Faces with the Highest Confusion

[24]:
```
# copy confusion matrix
cnn_cm = cm.copy()

# remove correct predictions by setting diag to 0
np.fill_diagonal(cnn_cm, 0)

# find row and col index of highest confusion
max_conf = np.max(cnn_cm)
```

```
r, c = np.where(cnn_cm == max_conf)

# display all highest confused pairs of faces
for i in range(len(r)):
    actual_face = r[i]
    predicted_face = c[i]
    count = cm[actual_face, predicted_face] # counts confusion

    # print highest confusion
    print(f'highest confusion: class {actual_face} was predicted as class␣
 ↪{predicted_face} {count} times.')

    # print figure label
    print(f'\n(figure f {i+1})')

    print(f'\nvisualizing training images for comparison:')
    print(f'actual class {actual_face}')

    # filter original flattened x_train images by class
    actual_face_images = x_train[np.where(y_train == actual_face)]
    show_images(actual_face_images, num_rows=1, num_cols=8)

    print(f'predicted class {predicted_face}')
    predicted_face_images = x_train[np.where(y_train == predicted_face)]
    show_images(predicted_face_images, num_rows=1, num_cols=8)
```

```
highest confusion: class 2 was predicted as class 9 2 times.

(figure f 1)

visualizing training images for comparison:
actual class 2
```



```
predicted class 9
```

6 Report template

6.1 Experimental results

Training description:
3 models are trained to classify the 40 faces provided in the data.
Model 1 utilizes LDA, Model 2 utilzes SVM, while Model 3 utilizes CNN.
A 2D plot is constructed for the LDA model. (Figure a)
The confusion matrix for each algorithm is created and analyzed. (Figures b,d)
The 8 faces from the training data of the highest confused pair for each algorithm are displayed. (Figures c, e)

Results (Best to Worst):
SVM: Average F-score: 0.9005794205794206
LDA: Average F-score: 0.8967577561327562
CNN: Average F-score: 0.8558333333333333

Analysis:
SVM performed to best with ~ 90.06% accuracy.
LDA performed almost identically to SVM at ~89.68% accuracy.
CNN performed slightly worse than SVM and LDA at ~85.58% accuracy.
SVM and LDA reported Class: 16 has the minimum F-score of 0.33333333333333337.
CNN reported Class: 2 has the minimum F-score of 0.

Highest Confusion:
LDA and SVM resulted in a tie between two pairs for the highest confusion, while CNN resulted in one pair with the highest confusion.
LDA:
Class 2 was predicted as Class 22 2 times.
Class 3 was predicted as Class 22 2 times.

LDA misclassified faces from class 2 and 3 as class 22. I would not confuse the faces from class 2 and class 22, as they do not look similar enough to me, however some of the faces from class 2 and class 3 do appear similar enough for me to confuse if both of the people were complete strangers to me.

SVM:
Class 2 was predicted as Class 39 2 times.
Class 16 was predicted as Class 0 2 times.

I would not confuse the faces from class 2 and 39 or from class 16 and class 0, as they do not appear similar enough to me.

CNN:
Class 2 was predicted as Class 9 2 times.

I would not confuse the faces from class 2 and class 9, as they do not appear similar to me.