project-3-sml-310-pranav-avva

April 20, 2022

1 SML 310 - Project 3 - NLP and Neural Networks

Pranav Avva

[34]:
```
import numpy as np
import numpy.random as npr
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import gensim
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, datasets, models

%matplotlib inline
sns.set()
```

1.1 Problem 1 - NLP and Word2Vec

1.1.1 Part A

Please describe what is meant by a "vector embedding" of words in Word2Vec.

Simply put, "vector embedding" is a process by which a word is represented by a tuple of numbers, that is, a vector in some n-dimensional space. The precise vector is dependent on the model used (usually either CBOW or Skip-gram) and the context in which the word appears (the corpus). Because the corpus is often very large in practice, the vectors often exist in extremely high-dimensional space, upwards of 1000 dimensions.

Representing words with vectors enables us to compute the similarity of two words, and to find the most similar words to a given words. This metric is known as Cosine Similarity and is computed by taking the dot product of the two representative vectors of the words. Two very similar words will have a dot product close to 1, while two dissimilar words will have a dot product close to 0.

1.1.2 Part B

Please use the pre-trained gensim `glove-wiki-gigaword-50` Word2Vec model to determine reasonable synonyms for the following words: 1. **Tiger** 2. **Awesome** 3. **Song** 4. **Data**

[35]:
```
import gensim.downloader as api

corpus = api.load("glove-wiki-gigaword-50")
```

[36]:
```
corpus.most_similar("tiger")
```

[36]:
```
[('tigers', 0.7239238619804382),
 ('woods', 0.6852725148200989),
 ('warrior', 0.6822084784507751),
 ('ltte', 0.6664599776268005),
 ('wild', 0.6495701670646667),
 ('elephant', 0.6488101482391357),
 ('crocodile', 0.6469965577125549),
 ('leopard', 0.6459459066390991),
 ('eelam', 0.6417322754859924),
 ('warriors', 0.6396878361701965)]
```

[37]:
```
corpus.most_similar("awesome")
```

[37]:
```
[('unbelievable', 0.8638609051704407),
 ('amazing', 0.8620665669441223),
 ('incredible', 0.8470885753631592),
 ('fantastic', 0.8059698343276978),
 ('marvelous', 0.7899898886680603),
 ('terrific', 0.7802115678787231),
 ('phenomenal', 0.7445020079612732),
 ('truly', 0.7433545589447021),
 ('luck', 0.7240825295448303),
 ('damn', 0.7129141688346863)]
```

[38]:
```
corpus.most_similar("song")
```

[38]:
```
[('album', 0.9297007918357849),
 ('songs', 0.90098637342453),
 ('soundtrack', 0.8414768576622009),
 ('albums', 0.8228148221969604),
 ('pop', 0.8219155669212341),
 ('sings', 0.8201141357421875),
 ('tune', 0.8188669681549072),
 ('duet', 0.8186635971069336),
 ('remix', 0.8086214065551758),
 ('band', 0.8063263297080994)]
```

[39]:
```
corpus.most_similar("data")
```

[39]:
```
[('information', 0.8329989314079285),
 ('tracking', 0.8124602437019348),
 ('database', 0.8122304677963257),
 ('analysis', 0.7966611981391907),
 ('applications', 0.792366623878479),
 ('indicate', 0.7726755142211914),
 ('indicates', 0.7607491612434387),
 ('computer', 0.7605311274528503),
 ('indicating', 0.7578965425491333),
 ('user', 0.7567423582077026)]
```

Gensim believes that the most similar word to "tiger" is "tigers," however, we know that the plural form of a word isn't the best synonym. A more reasonable synonym that is also within the top 10 most similar words is "leopard" as it, like a tiger, is also a big cat, even though they don't live in the same general area.

Gensim provides much better synonyms for the other words. "Unbelievable" is an appropriate synonym for "awesome". "Soundtrack" is an appropriate synonym for "song". "Information" is an appropriate synonym for "data".

1.1.3 Part C

Please use the pre-trained gensim `glove-wiki-gigaword-50` Word2Vec models to determine reasonable answers for the following analogies:

1. **puppy : kitten :: dog : ?**
2. **freshman : sophomore :: junior : ?**
3. **brother : sister :: grandson : ?**

[40]:
```
# in the analogy A is to B as C is to X, given A, B, and C,
# to find X, we have B - A + C = X. In this equation, A, B, C, and X
# are the vectors of the corresponding words

def analogy(a, b, c):
    """
    a: word A
    b: word B
    c: word C
    """
    global corpus

    return corpus.most_similar(positive=[b, c], negative=[a], topn=1)


print("Analogy 1:", analogy("puppy", "kitten", "dog"))
print("Analogy 2:", analogy("freshman", "sophomore", "junior"),
      "the best answer is 'senior' but gensim does not include that word in the␣
 ↪top 50 analogy solutions")
# corpus.most_similar(positive=["sophomore", "junior"], negative=["freshman"],␣
 ↪topn=50)
print("Analogy 3:", analogy("brother", "sister", "grandson"))
```

```
Analogy 1: [('cat', 0.7508497834205627)]
Analogy 2: [('basketball', 0.7672479152679443)] the best answer is 'senior' but
gensim does not include that word in the top 50 analogy solutions
Analogy 3: [('granddaughter', 0.8704106211662292)]
```

1.2 Problem 2 - NLP and Topic Modeling

1.2.1 Part A

Please prepare the built-in "fake-news" corpus of text using the commands:

```
import gensim.downloader as api
corpus_data = api.load("fake-news")
docs = [x['text'] for x in corpus_data]
```

From the Gensim GitHub at https://github.com/RaRe-Technologies/gensim-data.

[41]:
```
fake_news = api.load("fake-news")
docs = [x["text"] for x in fake_news]
```

1.2.2 Part B

Please use Gensim to preprocess these documents by tokenizing and lemmatizing them and removing other small text/strings that you decide are not meaningful for NLP, as well as rare and scarce words.

You may find the following Gensim tutorial useful: https://radimrehurek.com/gensim/auto_examples/tutorials/run_lda.html

[42]:
```
from nltk.tokenize import RegexpTokenizer

# Split the documents into tokens.
tokenizer = RegexpTokenizer(r'\w+')
for idx in range(len(docs)):
    docs[idx] = docs[idx].lower()  # Convert to lowercase.
    docs[idx] = tokenizer.tokenize(docs[idx])  # Split into words.

# Remove numbers, but not words that contain numbers.
docs = [[token for token in doc if not token.isnumeric()] for doc in docs]

# Remove words that are only one character.
docs = [[token for token in doc if len(token) > 1] for doc in docs]
```

[43]:
```
from nltk.stem.wordnet import WordNetLemmatizer
import nltk

nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download("stopwords")
```

```
lemmatizer = WordNetLemmatizer()
docs = [[lemmatizer.lemmatize(token) for token in doc] for doc in docs]
```

```
[nltk_data] Downloading package wordnet to /Users/pranav/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /Users/pranav/nltk_data…
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/pranav/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[44]:
```
from gensim.models import Phrases

# Create bigrams and trigrams from docs.
# This adds words like saudi_arabia when they appear multiple
# times together

bigram = Phrases(docs, min_count=20)
for idx in range(len(docs)):
    for token in bigram[docs[idx]]:
        if '_' in token:
            # Token is a bigram, add to document.
            docs[idx].append(token)
```

1.2.3 Part C

Please use the Gensim Dictionary and dictionary.doc2bow to create a dictionary and a bag of words representation of your tokenized corpus.

[45]:
```
from gensim.corpora import Dictionary

# Create a dictionary representation of the documents.
dictionary = Dictionary(docs)

# Remove rare and common tokens.
dictionary.filter_extremes(no_below=20, no_above=0.4)

# remove stopwords
stopwords = nltk.corpus.stopwords.words('english')
dictionary.filter_tokens(stopwords)

# Bag-of-words representation of the documents.
corpus = [dictionary.doc2bow(doc) for doc in docs]
```

[46]:
```
print(f'Number of unique tokens: {len(dictionary)}')
print(f'Number of documents: {len(corpus)}')
```

```
Number of unique tokens: 18307
Number of documents: 12999
```

1.2.4 Part D

Please use the Gensim `LDAModel` to perform topic modelling of the corpus into 3 topics.

[47]:
```
# Train LDA model.
from gensim.models import LdaModel
from gensim.models.callbacks import PerplexityMetric

# Set training parameters.
num_topics = 3
chunksize = 2000
passes = 20
iterations = 400
update_every = 0
eval_every = None  # Don't evaluate model perplexity, takes too much time.

# Make a index to word dictionary.
temp = dictionary[0]  # This is only to "load" the dictionary.
id2word = dictionary.id2token

model = LdaModel(
    corpus=corpus,
    id2word=id2word,
    chunksize=chunksize,
    alpha='auto',
    eta='auto',
    iterations=iterations,
    num_topics=num_topics,
    passes=passes,
    update_every=update_every,
    eval_every=eval_every,
)
```

1.2.5 Part E

For each topic show the main words and use these to give a rough name to each topic.

[48]:
```
top_topics = model.top_topics(corpus, topn=10, dictionary=dictionary)

from pprint import pprint
pprint(top_topics)
```

```
[([(0.016787693, 'clinton'),
   (0.010307074, 'trump'),
   (0.0101102935, 'hillary'),
   (0.008381988, 'her'),
   (0.007379002, 'she'),
   (0.006318001, 'email'),
   (0.0062377807, 'election'),
   (0.0052118604, 'hillary_clinton'),
   (0.0048548966, 'fbi'),
   (0.0043788264, 'campaign')],
  -0.7599929250132453),
 ([(0.0058009042, 'u'),
   (0.005378236, 'trump'),
   (0.0044661122, 'war'),
   (0.0040123523, 'american'),
   (0.0037186246, 'government'),
   (0.00362009, 'russia'),
   (0.0034244682, 'country'),
   (0.003375615, 'our'),
   (0.0033430737, 'world'),
   (0.003051214, 'president')],
  -0.7825276207608365),
 ([(0.006382521, 'your'),
   (0.0061420063, 'de'),
   (0.004465713, 'our'),
   (0.0038123245, 'la'),
   (0.003577981, 'like'),
   (0.0035773022, 'do'),
   (0.0034581325, 'my'),
   (0.0031395683, 'them'),
   (0.0029864246, 'other'),
   (0.002848007, 'into')],
  -0.9083852593560271)]
```

- Topic 1: Miscellaneous words
- Topic 2: 2016 Election
- Topic 3: Trump presidency

1.3 Problem 3 - Data Classification

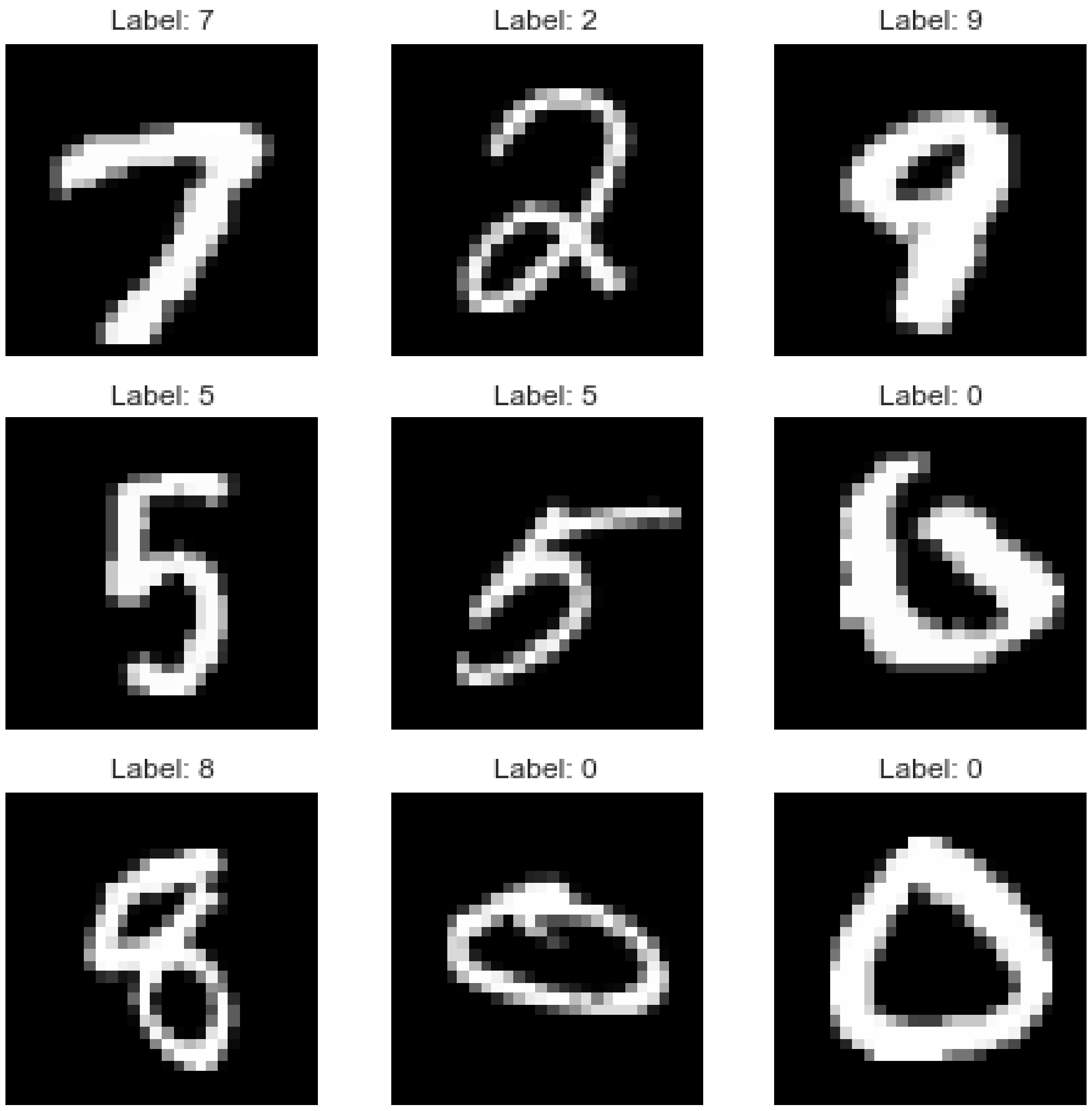
1.3.1 Part A

Please take the MNIST training dataset and split off the last 10,000 images as a validation data set. Then assign the other training images to a new training set.

[49]:
```python
from sklearn.model_selection import train_test_split

(X_train, y_train), (X_test, y_test) = datasets.mnist.load_data()

# normalize the data
X_train = X_train / 255.0
X_test = X_test / 255.0

# create train-validation split
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,␣
 ↪test_size=10000)
```

```python
# print the shape of all sets with the set name
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_valid shape:", X_valid.shape)
print("y_valid shape:", y_valid.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)

# show some training images with their labels
fig, ax = plt.subplots(3, 3, figsize=(8, 8))
for i in range(3):
    for j in range(3):
        ax[i, j].imshow(X_train[i * 5 + j], cmap='binary_r')
        ax[i, j].set_title(f"Label: {y_train[i * 5 + j]}")
        ax[i, j].axis('off')
```

```
X_train shape: (50000, 28, 28)
y_train shape: (50000,)
X_valid shape: (10000, 28, 28)
y_valid shape: (10000,)
X_test shape: (10000, 28, 28)
y_test shape: (10000,)
```



1.3.2 Part B

Please create a Neural Network with three dense hidden layers each having 32 nodes, and train it to classify the MNIST Data set over five epochs. For this please use your new training set and validation set respectively for model training and model performance reporting.

[50]:
```python
model = models.Sequential()

# create an input layer with 784 neurons
model.add(layers.Flatten(input_shape=(28, 28), name='input'))

# create 3 hidden dense layers with 32 neurons each
```

```
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(32, activation='relu'))

# create an output layer with 10 neurons
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (Flatten)             (None, 784)               0

 dense_8 (Dense)             (None, 32)                25120

 dense_9 (Dense)             (None, 32)                1056

 dense_10 (Dense)            (None, 32)                1056

 dense_11 (Dense)            (None, 10)                330

=================================================================
Total params: 27,562
Trainable params: 27,562
Non-trainable params: 0
_________________________________________________________________
```

[51]:
```
# compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# fit the model
history = model.fit(X_train, y_train, epochs=5, validation_data=(X_valid,␣
 ↪y_valid))
```

```
Epoch 1/5
1563/1563 [==============================] - 2s 1ms/step - loss: 0.3914 -
accuracy: 0.8820 - val_loss: 0.2346 - val_accuracy: 0.9297
Epoch 2/5
1563/1563 [==============================] - 2s 1ms/step - loss: 0.1762 -
accuracy: 0.9479 - val_loss: 0.1795 - val_accuracy: 0.9444
Epoch 3/5
1563/1563 [==============================] - 2s 1ms/step - loss: 0.1350 -
accuracy: 0.9600 - val_loss: 0.1783 - val_accuracy: 0.9470
Epoch 4/5
1563/1563 [==============================] - 2s 1ms/step - loss: 0.1139 -
```

```
accuracy: 0.9659 - val_loss: 0.1621 - val_accuracy: 0.9540
Epoch 5/5
1563/1563 [==============================] - 2s 1ms/step - loss: 0.0990 -
accuracy: 0.9697 - val_loss: 0.1509 - val_accuracy: 0.9574
```

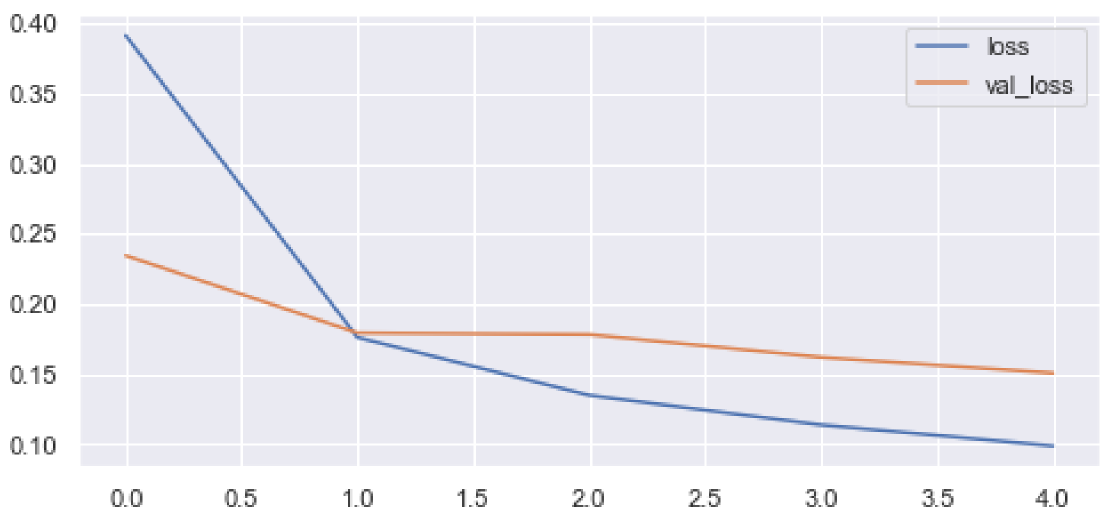
1.3.3 Part C

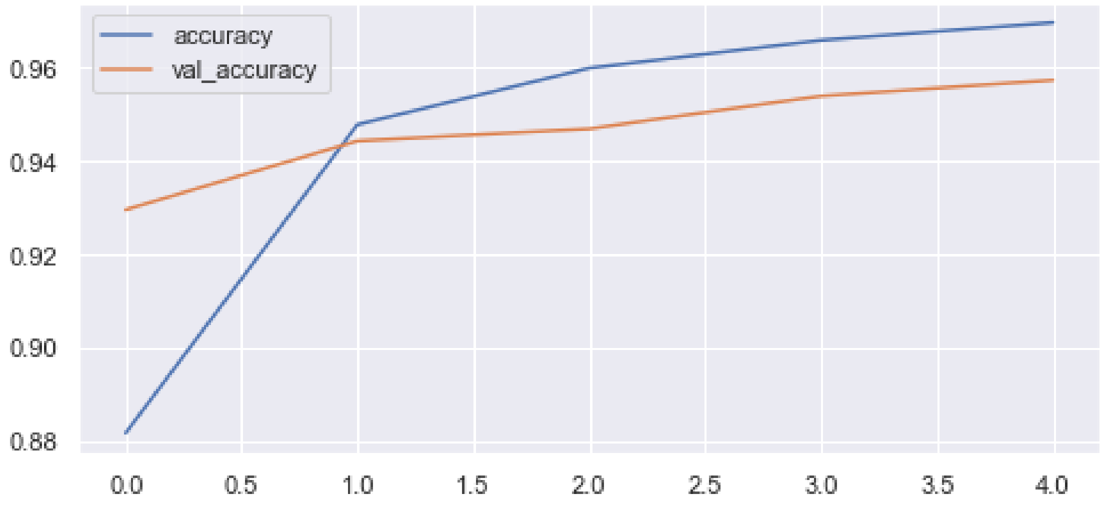
What is the performance of the model on the training and validation sets over the 5 epochs? Which of these do we expect is characteristic of model performance in new data?

The model's accuracy, validation accuracy, loss, and validation loss are presented in the plot below. The trained model's accuracy on the

[52]:
```python
# plot loss, val_loss on one axis and accuracy, val_accuracy on the other
fig, ax = plt.subplots(2, 1, figsize=(8, 8))
ax[0].plot(history.history['loss'], label='loss')
ax[0].plot(history.history['val_loss'], label='val_loss')
ax[0].legend()

ax[1].plot(history.history['accuracy'], label='accuracy')
ax[1].plot(history.history['val_accuracy'], label='val_accuracy')
ax[1].legend()
```

[52]:
```
<matplotlib.legend.Legend at 0x144a6a9a0>
```





```
[53]: from sklearn.metrics import classification_report

y_train_pred = model.predict(X_train)
y_valid_pred = model.predict(X_valid)

# print the classification report
print("Classification report for training set:")
print(classification_report(y_train, y_train_pred.argmax(axis=1)))

print("Classification report for validation set:")
print(classification_report(y_valid, y_valid_pred.argmax(axis=1)))
```

```
Classification report for training set:
              precision    recall  f1-score   support
```

```
           0       0.98      0.99      0.99      4924
           1       0.99      0.99      0.99      5660
           2       0.97      0.98      0.98      4990
           3       0.96      0.97      0.96      5097
           4       0.99      0.97      0.98      4817
           5       0.99      0.96      0.97      4542
           6       0.97      0.99      0.98      4951
           7       0.97      0.98      0.98      5248
           8       0.98      0.96      0.97      4812
           9       0.98      0.95      0.97      4959

    accuracy                           0.98     50000
   macro avg       0.98      0.98      0.98     50000
weighted avg       0.98      0.98      0.98     50000

Classification report for validation set:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       999
           1       0.98      0.98      0.98      1082
           2       0.93      0.97      0.95       968
           3       0.95      0.95      0.95      1034
           4       0.98      0.94      0.96      1025
           5       0.97      0.94      0.96       879
           6       0.93      0.99      0.96       967
           7       0.93      0.98      0.95      1017
           8       0.97      0.92      0.95      1039
           9       0.95      0.92      0.93       990

    accuracy                           0.96     10000
   macro avg       0.96      0.96      0.96     10000
weighted avg       0.96      0.96      0.96     10000
```

The following plot shows the confusion matrices for the training and validation sets. The main diagonal of the confusion matrix is removed to show where the errors lie, as the number of correct predictions far outnumbers incorrect predictions. There are a couple patterns in the confusion matrices of both the training and validation set:

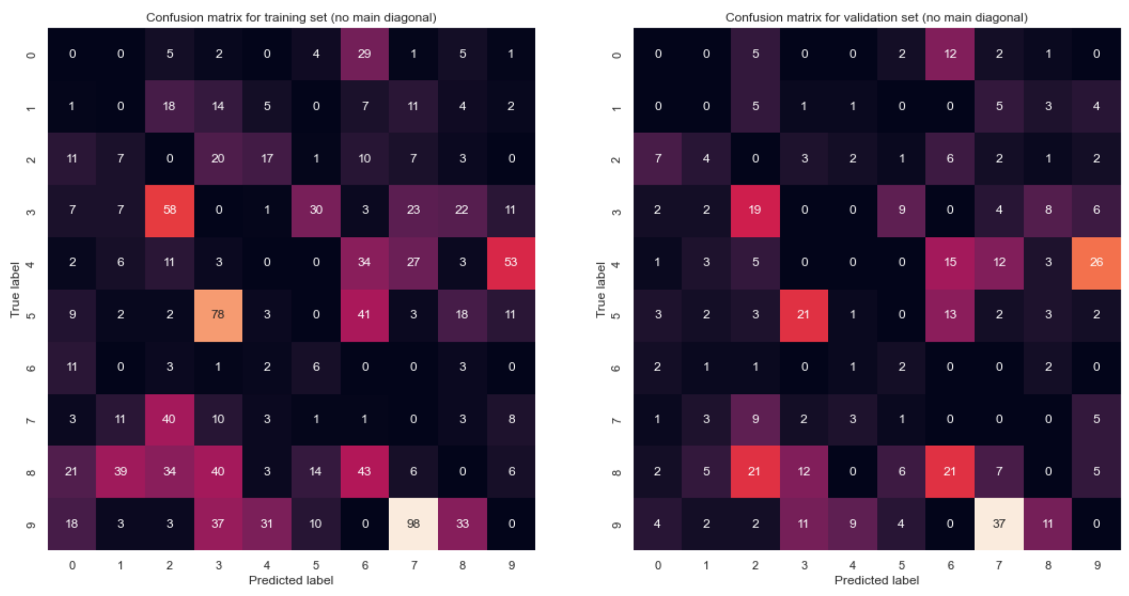
- 5s and 8s are often confused for 3s
- 6s and 4s are often confused for 4s
- 8s are often confused for 1s and 3s
- 2s and 9s are often confused for 7s

The above patterns may not be present in the following heatmaps due to effects of randomization in the splitting of data and the neural network solver.

[54]:
```
from sklearn.metrics import confusion_matrix
```

```
# confusion matrix with the main diagonal set to zero
train_confusion_matrix_no_diagonal = confusion_matrix(y_train, y_train_pred.
 ↪argmax(axis=1)) - np.diag(np.diag(confusion_matrix(y_train, y_train_pred.
 ↪argmax(axis=1))))
valid_confusion_matrix_no_diagonal = confusion_matrix(y_valid, y_valid_pred.
 ↪argmax(axis=1)) - np.diag(np.diag(confusion_matrix(y_valid, y_valid_pred.
 ↪argmax(axis=1))))

# imshow the confusion matrix for train and test sets
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 9))

sns.heatmap(train_confusion_matrix_no_diagonal, annot=True, ax=ax1, fmt='d',␣
 ↪cbar=False)
ax1.set_title('Confusion matrix for training set (no main diagonal)')
ax1.set_ylabel('True label')
ax1.set_xlabel('Predicted label')

sns.heatmap(valid_confusion_matrix_no_diagonal, annot=True, ax=ax2, fmt='d',␣
 ↪cbar=False)
ax2.set_title('Confusion matrix for validation set (no main diagonal)')
ax2.set_ylabel('True label')
ax2.set_xlabel('Predicted label')
```

```
[54]: Text(0.5, 57.5, 'Predicted label')
```



1.3.4 Part D

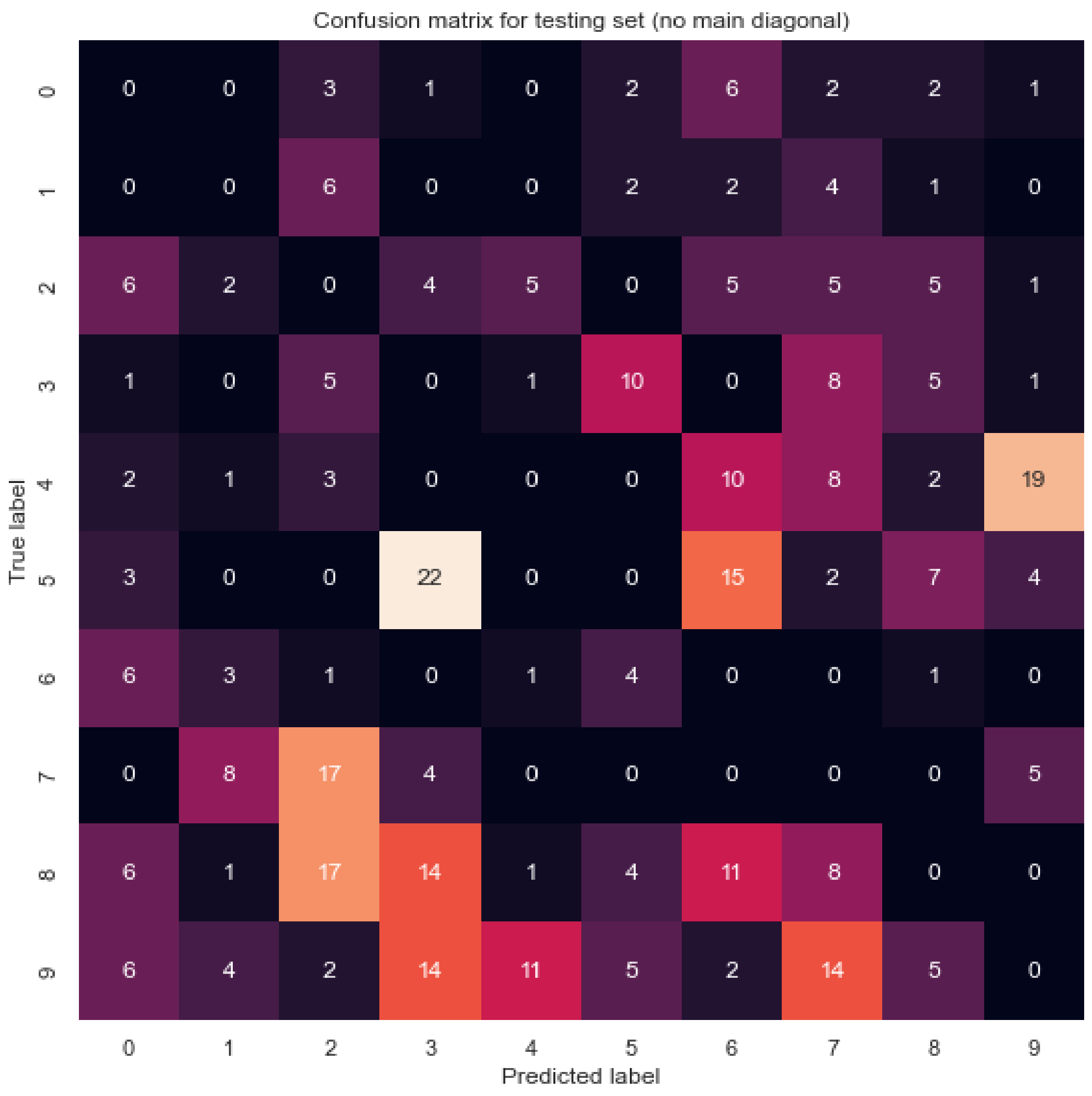
Check your model performance on the test set, and compare with your expectations in Part C.

[55]:
```
y_test_pred = model.predict(X_test)

print("Classification report for test set:")
print(classification_report(y_test, y_test_pred.argmax(axis=1)))
```

```
Classification report for test set:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       980
           1       0.98      0.99      0.99      1135
           2       0.95      0.97      0.96      1032
           3       0.94      0.97      0.96      1010
           4       0.98      0.95      0.97       982
           5       0.97      0.94      0.95       892
           6       0.95      0.98      0.97       958
           7       0.95      0.97      0.96      1028
           8       0.97      0.94      0.95       974
           9       0.97      0.94      0.95      1009

    accuracy                           0.96     10000
   macro avg       0.96      0.96      0.96     10000
weighted avg       0.96      0.96      0.96     10000
```

[56]:
```
test_confusion_matrix_no_diagonal = confusion_matrix(y_test, y_test_pred.
 ↪argmax(axis=1)) - np.diag(np.diag(confusion_matrix(y_test, y_test_pred.
 ↪argmax(axis=1))))

fig, ax = plt.subplots(1, 1, figsize=(9, 9))

sns.heatmap(test_confusion_matrix_no_diagonal, annot=True, ax=ax, fmt='d',␣
 ↪cbar=False)
ax.set_title('Confusion matrix for testing set (no main diagonal)')
ax.set_ylabel('True label')
ax.set_xlabel('Predicted label')
```

[56]:
```
Text(0.5, 57.5, 'Predicted label')
```



	0	1	2	3	4	5	6	7	8	9
0	0	0	3	1	0	2	6	2	2	1
1	0	0	6	0	0	2	2	4	1	0
2	6	2	0	4	5	0	5	5	5	1
3	1	0	5	0	1	10	0	8	5	1
4	2	1	3	0	0	0	10	8	2	19
5	3	0	0	22	0	0	15	2	7	4
6	6	3	1	0	1	4	0	0	1	0
7	0	8	17	4	0	0	0	0	0	5
8	6	1	17	14	1	4	11	8	0	0
9	6	4	2	14	11	5	2	14	5	0

There are similar patterns in the confusion matrix of the test set as well. The model's test performance is similar to the model's validation performance. With a consistent F1 score of above 0.95, the model is performing well for a simple feed-forward neural network with a shallow (low number of layers) and narrow (low number of nodes per layer) architecture.

It is likely possible to improve the model's performance by making the architecture more complicated and possibly including convolutional layers. Using a CNN architecture would allow the model to focus on the most important parts of the images

Testing a Convolutional Neural Network architecture

[79]:
```
model2 = models.Sequential()

# use convolutional layers to create a CNN
# input is images of size 28x28 pixels
model2.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,␣
 ↪1)))
model2.add(layers.MaxPooling2D((2, 2)))

model2.add(layers.Conv2D(64, (3, 3), activation='relu'))
model2.add(layers.MaxPooling2D((2, 2)))

model2.add(layers.Conv2D(64, (3, 3), activation='relu'))

# flatten and add dense layers
model2.add(layers.Flatten())
model2.add(layers.Dense(32, activation='relu'))

model2.add(layers.Dense(32, activation='relu'))

model2.add(layers.Dense(32, activation='relu'))
model2.add(layers.Dropout(0.4))

# output layer
model2.add(layers.Dense(10, activation='softmax'))

model2.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_12 (MaxPoolin  (None, 13, 13, 32)       0
 g2D)

 conv2d_19 (Conv2D)          (None, 11, 11, 64)        18496

 max_pooling2d_13 (MaxPoolin  (None, 5, 5, 64)         0
 g2D)

 conv2d_20 (Conv2D)          (None, 3, 3, 64)          36928

 flatten_6 (Flatten)         (None, 576)               0

 dense_32 (Dense)            (None, 32)                18464

 dense_33 (Dense)            (None, 32)                1056

 dense_34 (Dense)            (None, 32)                1056
 dropout_10 (Dropout)        (None, 32)                0

 dense_35 (Dense)            (None, 10)                330

=================================================================
Total params: 76,650
Trainable params: 76,650
Non-trainable params: 0
_________________________________________________________________
```

```
[80]: # early stopping
      from keras.callbacks import EarlyStopping

      early_stopping = EarlyStopping(monitor='val_accuracy', patience=7,␣
       ↪restore_best_weights=True)

      # compile the model
      model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
       ↪metrics=['accuracy'])
      history2 = model2.fit(X_train, y_train, epochs=50, validation_data=(X_valid,␣
       ↪y_valid), callbacks=[early_stopping])
```

```
Epoch 1/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.3568 -
accuracy: 0.8903 - val_loss: 0.1092 - val_accuracy: 0.9672
Epoch 2/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.1121 -
accuracy: 0.9706 - val_loss: 0.0619 - val_accuracy: 0.9815
Epoch 3/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0759 -
accuracy: 0.9796 - val_loss: 0.0616 - val_accuracy: 0.9829
Epoch 4/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0630 -
accuracy: 0.9841 - val_loss: 0.0568 - val_accuracy: 0.9871
Epoch 5/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0546 -
accuracy: 0.9859 - val_loss: 0.0505 - val_accuracy: 0.9876
Epoch 6/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0430 -
accuracy: 0.9880 - val_loss: 0.0506 - val_accuracy: 0.9867
Epoch 7/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0393 -
accuracy: 0.9897 - val_loss: 0.0525 - val_accuracy: 0.9880
Epoch 8/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0356 -
accuracy: 0.9910 - val_loss: 0.0736 - val_accuracy: 0.9856
Epoch 9/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0298 -
```

```
accuracy: 0.9917 - val_loss: 0.0988 - val_accuracy: 0.9844
Epoch 10/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0274 -
accuracy: 0.9929 - val_loss: 0.0689 - val_accuracy: 0.9885
Epoch 11/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0281 -
accuracy: 0.9925 - val_loss: 0.0608 - val_accuracy: 0.9886
Epoch 12/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0248 -
accuracy: 0.9930 - val_loss: 0.0668 - val_accuracy: 0.9889
Epoch 13/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0244 -
accuracy: 0.9935 - val_loss: 0.0744 - val_accuracy: 0.9868
Epoch 14/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0193 -
accuracy: 0.9944 - val_loss: 0.0661 - val_accuracy: 0.9888
Epoch 15/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0224 -
accuracy: 0.9945 - val_loss: 0.0890 - val_accuracy: 0.9849
Epoch 16/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0191 -
accuracy: 0.9948 - val_loss: 0.0775 - val_accuracy: 0.9881
Epoch 17/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0213 -
accuracy: 0.9942 - val_loss: 0.1367 - val_accuracy: 0.9827
Epoch 18/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0172 -
accuracy: 0.9956 - val_loss: 0.0605 - val_accuracy: 0.9918
Epoch 19/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0170 -
accuracy: 0.9960 - val_loss: 0.0889 - val_accuracy: 0.9861
Epoch 20/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0147 -
accuracy: 0.9962 - val_loss: 0.0804 - val_accuracy: 0.9860
Epoch 21/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0141 -
accuracy: 0.9965 - val_loss: 0.0871 - val_accuracy: 0.9899
Epoch 22/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0183 -
accuracy: 0.9954 - val_loss: 0.0682 - val_accuracy: 0.9898
Epoch 23/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0135 -
accuracy: 0.9966 - val_loss: 0.1265 - val_accuracy: 0.9861
Epoch 24/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0172 -
accuracy: 0.9953 - val_loss: 0.0656 - val_accuracy: 0.9909
Epoch 25/50
1563/1563 [==============================] - 20s 13ms/step - loss: 0.0158 -
```

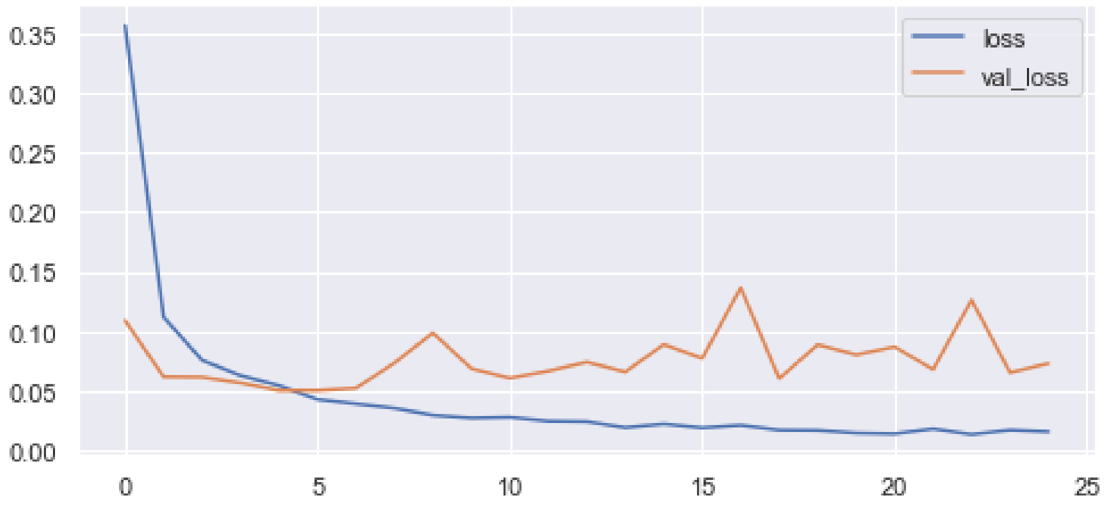
```
accuracy: 0.9962 - val_loss: 0.0732 - val_accuracy: 0.9892
```

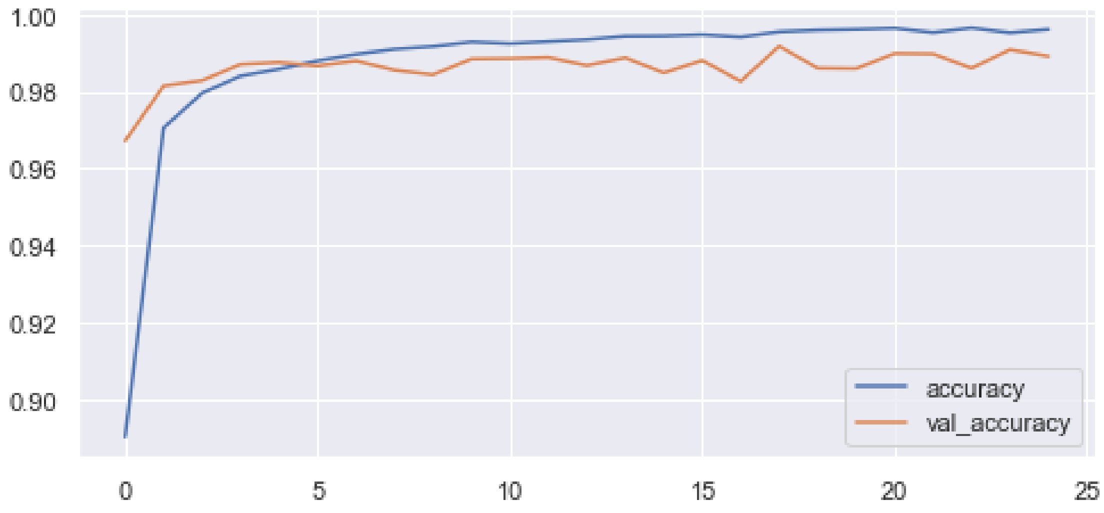
[81]:
```
# predict against the validation set and print the classification report

y_valid_pred2 = model2.predict(X_valid)

print("Classification report for validation set:")
print(classification_report(y_valid, y_valid_pred2.argmax(axis=1)))
```

```
Classification report for validation set:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       999
           1       1.00      0.99      0.99      1082
           2       0.99      0.99      0.99       968
           3       1.00      0.99      0.99      1034
           4       0.99      0.99      0.99      1025
           5       0.99      0.99      0.99       879
           6       0.99      1.00      0.99       967
           7       0.99      0.99      0.99      1017
           8       0.99      0.99      0.99      1039
           9       0.98      0.99      0.99       990

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

[82]:
```
# plot loss, val_loss on one axis and accuracy, val_accuracy on the other
fig, ax = plt.subplots(2, 1, figsize=(8, 8))
ax[0].plot(history2.history['loss'], label='loss')
ax[0].plot(history2.history['val_loss'], label='val_loss')
ax[0].legend()

ax[1].plot(history2.history['accuracy'], label='accuracy')
ax[1].plot(history2.history['val_accuracy'], label='val_accuracy')
ax[1].legend()
```

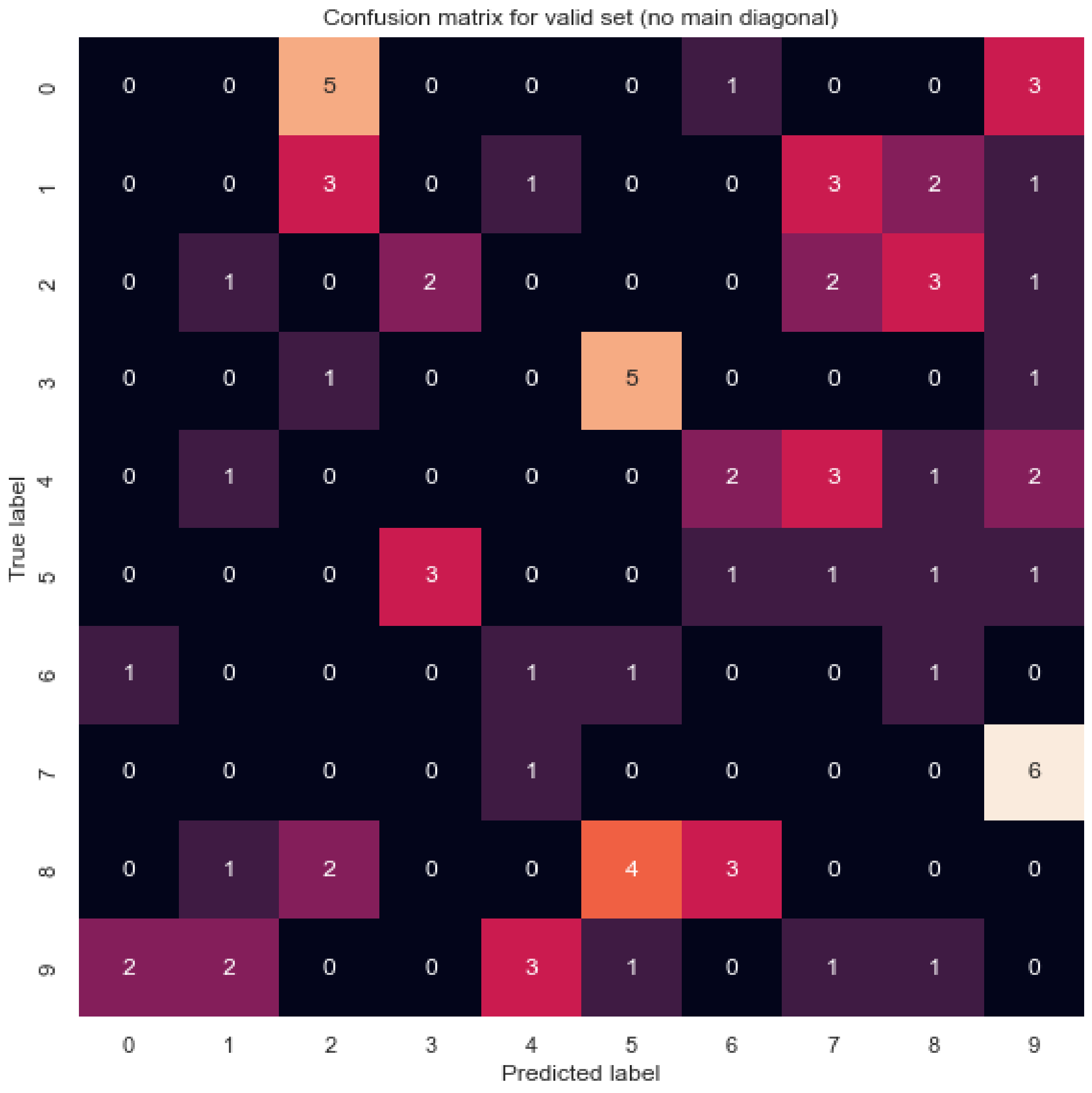
[82]:
```
<matplotlib.legend.Legend at 0x1a265bd60>
```





[83]:
```
# plot the validation confusion matrix

valid_confusion_matrix_no_diagonal_2 = confusion_matrix(y_valid, y_valid_pred2.
  ↪argmax(axis=1)) - np.diag(np.diag(confusion_matrix(y_valid, y_valid_pred2.
  ↪argmax(axis=1))))

fig, ax = plt.subplots(1, 1, figsize=(9, 9))

sns.heatmap(valid_confusion_matrix_no_diagonal_2, annot=True, ax=ax, fmt='d',␣
  ↪cbar=False)
ax.set_title('Confusion matrix for valid set (no main diagonal)')
ax.set_ylabel('True label')
ax.set_xlabel('Predicted label')
```

[83]: Text(0.5, 57.5, 'Predicted label')



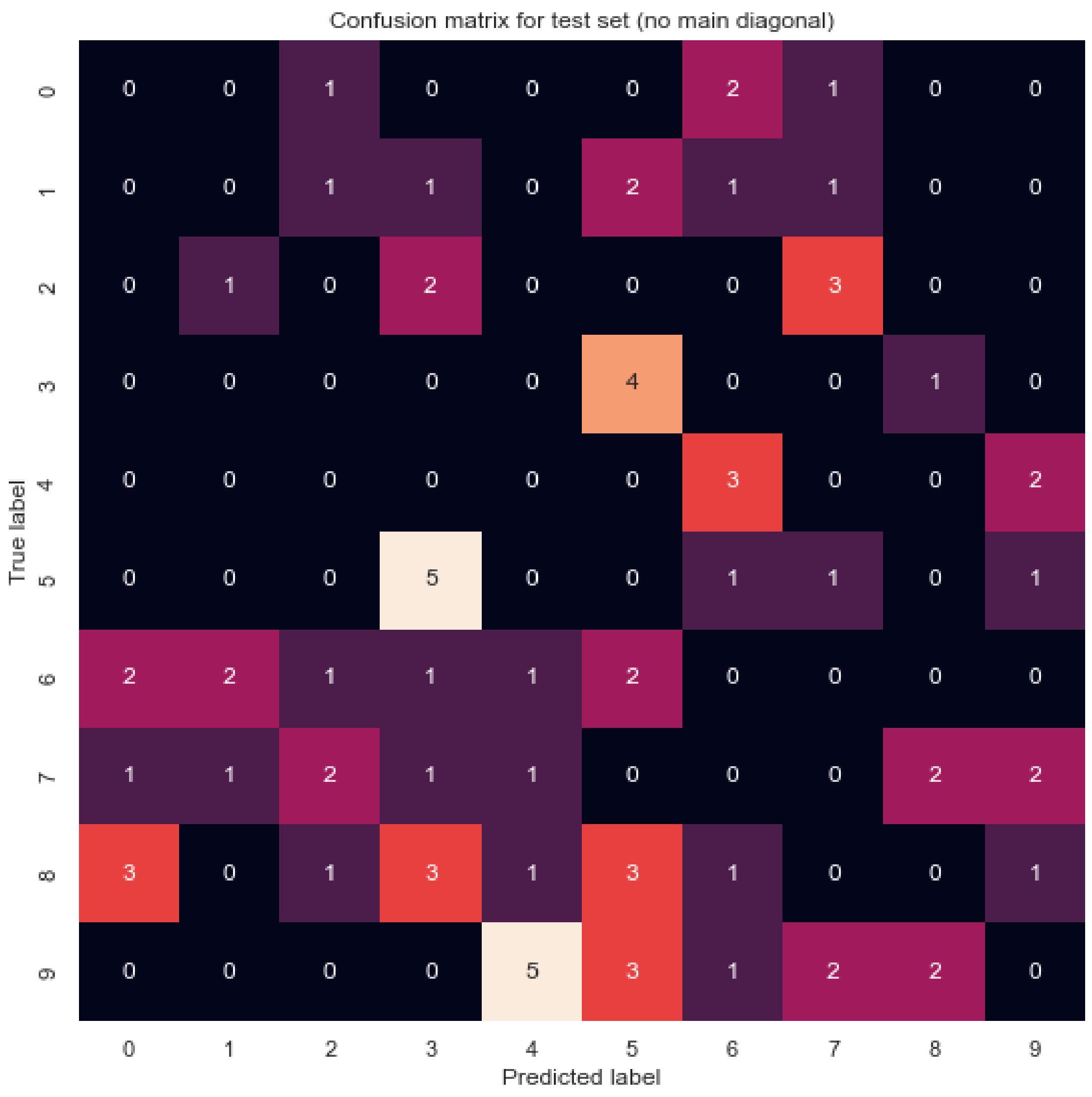
	0	1	2	3	4	5	6	7	8	9
0	0	0	5	0	0	0	1	0	0	3
1	0	0	3	0	1	0	0	3	2	1
2	0	1	0	2	0	0	0	2	3	1
3	0	0	1	0	0	5	0	0	0	1
4	0	1	0	0	0	0	2	3	1	2
5	0	0	0	3	0	0	1	1	1	1
6	1	0	0	0	1	1	0	0	1	0
7	0	0	0	0	1	0	0	0	0	6
8	0	1	2	0	0	4	3	0	0	0
9	2	2	0	0	3	1	0	1	1	0

[84]:
```python
# predict against the test set and print the classification report

y_test_pred2 = model2.predict(X_test)

print("Classification report for validation set:")
print(classification_report(y_test, y_test_pred2.argmax(axis=1)))
```

```
Classification report for validation set:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       980
           1       1.00      0.99      1.00      1135
           2       0.99      0.99      0.99      1032
           3       0.99      1.00      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.98      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

[85]:
```
# plot the validation confusion matrix

test_confusion_matrix_no_diagonal_2 = confusion_matrix(y_test, y_test_pred2.
 ↪argmax(axis=1)) - np.diag(np.diag(confusion_matrix(y_test, y_test_pred2.
 ↪argmax(axis=1))))

fig, ax = plt.subplots(1, 1, figsize=(9, 9))

sns.heatmap(test_confusion_matrix_no_diagonal_2, annot=True, ax=ax, fmt='d',␣
 ↪cbar=False)
ax.set_title('Confusion matrix for test set (no main diagonal)')
ax.set_ylabel('True label')
ax.set_xlabel('Predicted label')
```

[85]:
```
Text(0.5, 57.5, 'Predicted label')
```



The convolutional neural network architecture has better performance than the FFNN architecture. The test F1 score is consistently above 0.98 in the CNN, compared to above 0.95 in the FFNN. Therefore, while more complicated and longer to train, the CNN setup yields better performance than a simpler and faster FFNN.