Model Performance And Fit - 2

One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Objective	Complete
Implement a custom neural network for different batch sizes	
Evaluate the neural network for different number of epochs	

Goal for this section

- We will again work with the `credit_card_data` dataset
- We will build a model and then compare the loss and accuracy of the validation data for different levels of epochs and batch sizes

Wide and deep neural networks

- The architecture of a neural network is defined by 2 parameters: **width** and **depth**
  - Depth is measured by the number of layers in the neural network, excluding the input layer
  - Width is measured by the maximal number of nodes present in a layer of a neural network
- In general, as we increase the width and the depth in a model, the accuracy increases, but so does the complexity
- Choosing the depth and width of the layers in a neural network is a **trial-and-error process** highly dependent on the data we have in hand, but having a single hidden layer is always a good start

- **Note:** Remember that if we try to increase the width and depth of a neural network beyond a certain point, the model might overfit!

Define and compile a sequential model

- Let's create a convenience function to define and compile the model with an input layer, two hidden layers, and an output layer

```
def create_model(lr = .0001):
  # Let's set the seed so that we can reproduce the results.
  tf.random.set_seed(1)
  opt = Adam(learning_rate = lr) # <- set optimizer

  model = Sequential([
          Dense(32, activation='relu', input_dim = 30),#<- set input and 1st hidden layer
          Dense(32, activation='relu'),                #<- set 2nd hidden layer
          Dense(1, activation='sigmoid')               #<- set output layer

  ])

  model.compile(optimizer = opt,              #<- set optimizer
                loss='binary_crossentropy',   #<- set loss function to binary_crossentropy
                metrics=['accuracy'])         #<- set performance metric
  return model
```

Fit the model

- We will now fit the model with **different batch sizes** and compare how the validation loss and the accuracy look by comparing their results
- We will keep the number of `epochs` the same for every experiment

Default batch size

- The default batch size while we fit the model is `32`, and hence we need not specify the parameter explicitly

```
model = create_model()
bt_default = model.fit(X_train_scaled, y_train,                  #<- train data and labels
                       epochs = 25,                              #<- number of epochs
                       validation_data = (X_val_scaled, y_val))  #<- pass val data
```

```
Epoch 1/25
657/657 [==============================] - 1s 994us/step - loss: 0.4889 - accuracy: 0.7938 -
val_loss: 0.4522 - val_accuracy: 0.8196
Epoch 2/25
657/657 [==============================] - 1s 840us/step - loss: 0.4496 - accuracy: 0.8161 -
val_loss: 0.4504 - val_accuracy: 0.8162
...
Epoch 24/25
657/657 [==============================] - 1s 858us/step - loss: 0.4372 - accuracy: 0.8169 -
val_loss: 0.4337 - val_accuracy: 0.8202
Epoch 25/25
657/657 [==============================] - 1s 810us/step - loss: 0.4294 - accuracy: 0.8236 -
val_loss: 0.4355 - val_accuracy: 0.8216
```

Small batch size

- Let's set the batch size to `8`

```
model = create_model()
bt_small = model.fit(X_train_scaled, y_train,                    #<- train data and labels
                     epochs = 25,                                  #<- number of epochs
                     batch_size= 8,                               #<- set batch_size
                     validation_data = (X_val_scaled, y_val))  #<- pass val data
```

```
Epoch 1/25
2625/2625 [==============================] - 2s 789us/step - loss: 0.4854 - accuracy: 0.7975
- val_loss: 0.4472 - val_accuracy: 0.8213
Epoch 2/25
2625/2625 [==============================] - 2s 732us/step - loss: 0.4517 - accuracy: 0.8153
- val_loss: 0.4482 - val_accuracy: 0.8171
...
Epoch 24/25
2625/2625 [==============================] - 2s 702us/step - loss: 0.4394 - accuracy: 0.8166
- val_loss: 0.4344 - val_accuracy: 0.8236
Epoch 25/25
2625/2625 [==============================] - 2s 702us/step - loss: 0.4318 - accuracy: 0.8223
- val_loss: 0.4348 - val_accuracy: 0.8216
```

Large batch size

- Now, let’s set the batch size as `512`

```
model = create_model()
bt_large = model.fit(X_train_scaled, y_train,    #<- train data + labels
                     epochs = 25,                #<- number of epochs
                     batch_size= 512,            #<- set batch_size
                     validation_data = (X_val_scaled, y_val))  #<- val data + labels
```

```
Epoch 1/25
42/42 [==============================] - 0s 4ms/step - loss: 0.5435 - accuracy: 0.7636 -
val_loss: 0.4715 - val_accuracy: 0.8071
Epoch 2/25
42/42 [==============================] - 0s 2ms/step - loss: 0.4683 - accuracy: 0.8081 -
val_loss: 0.4567 - val_accuracy: 0.8158
...
Epoch 24/25
42/42 [==============================] - 0s 2ms/step - loss: 0.4367 - accuracy: 0.8205 -
val_loss: 0.4324 - val_accuracy: 0.8249
Epoch 25/25
42/42 [==============================] - 0s 2ms/step - loss: 0.4284 - accuracy: 0.8240 -
val_loss: 0.4298 - val_accuracy: 0.8249
```

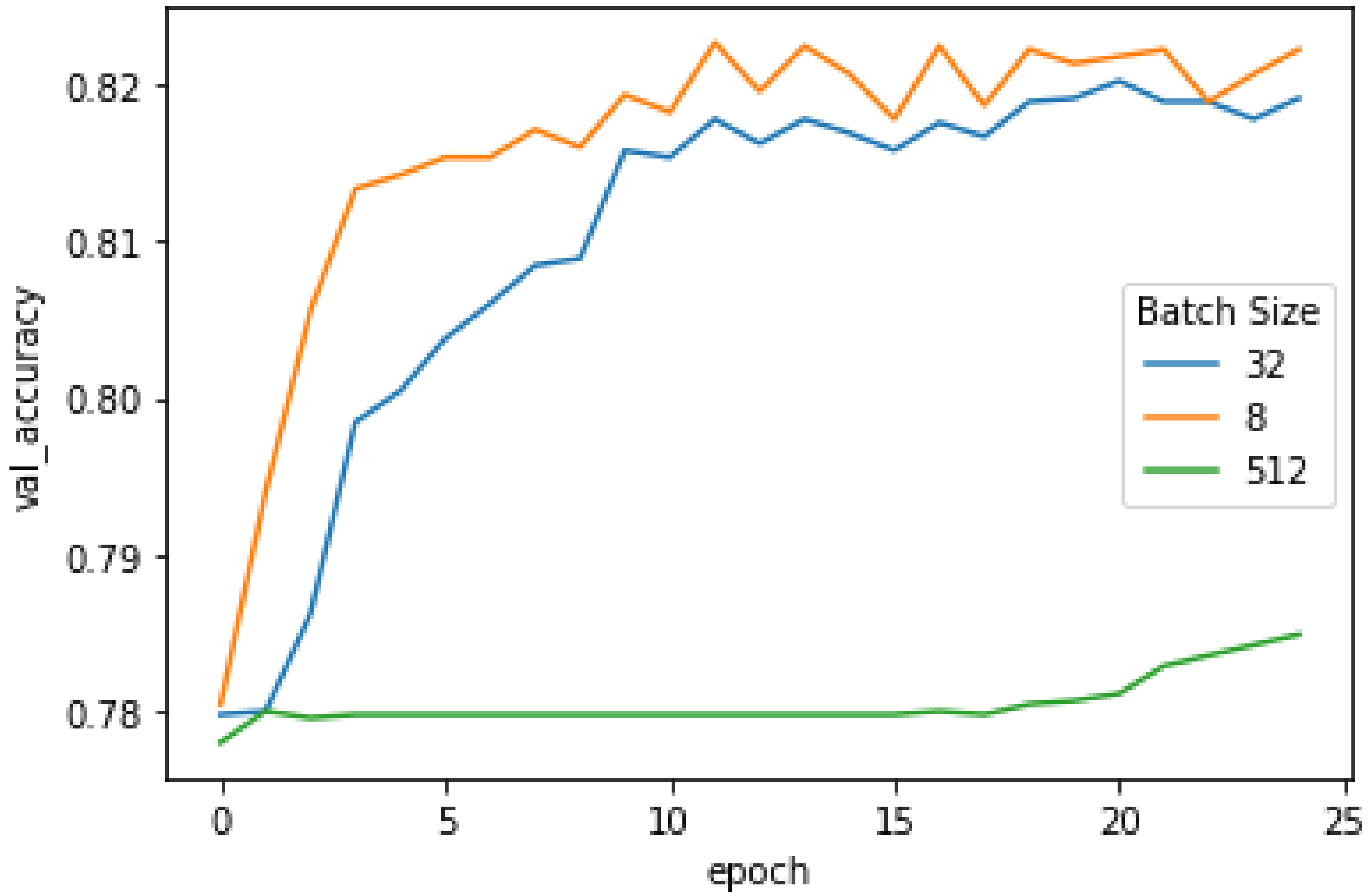
Visualize results for various batch sizes

- Let’s create a dataframe with the loss and accuracy for training and validation data along with their epoch and batch size

```
batch_sizes = []

for exp, result in zip([bt_default, bt_small, bt_large], ["32", "8", "512"]):

  df = pd.DataFrame.from_dict(exp.history)
  df['epoch'] = df.index.values
  df['Batch Size'] = result

  batch_sizes.append(df)

df_summary = pd.concat(batch_sizes)
df_summary['Batch Size'] = df_summary['Batch Size'].astype('str')
```

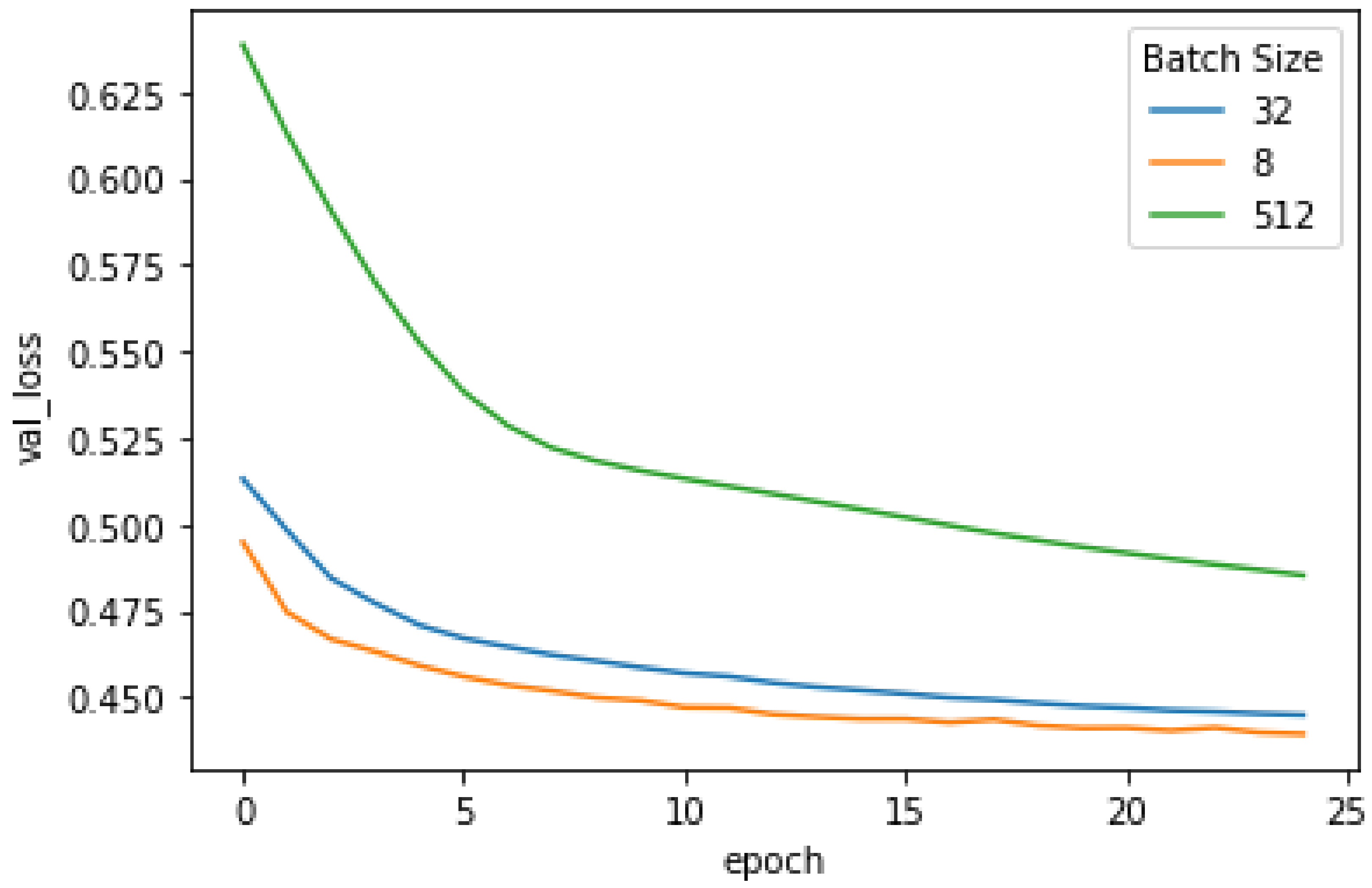
Visualize results for various batch sizes (cont'd)

```
sns.lineplot(x='epoch', y='val_accuracy',
             hue='Batch Size', data=df_summary)
```



Visualize results for various batch sizes (cont'd)

```
sns.lineplot(x='epoch', y='val_loss',
             hue='Batch Size', data=df_summary)
```



- We obtain the best results when the batch size is set to the default **32**, and for small batch size is set to **8**

Module completion checklist

Objective	Complete
Implement a custom neural network for different batch sizes	✔
Evaluate the neural network for different number of epochs	

Using a different number of epochs

- Now, we will train the model using different number of `epochs` and compare our results
- We will keep the batch size and the learning rate at their default values in this analysis

Higher number of epochs

- We initially run the model by setting the number of epochs as `150` with the default `batch_size`

```
ep_high = create_model().fit(X_train_scaled, y_train,
                             epochs = 150,
                             validation_data=(X_val_scaled, y_val))
```

```
Epoch 1/150
657/657 [==============================] - 1s 1ms/step - loss: 0.4889 - accuracy: 0.7938 -
val_loss: 0.4522 - val_accuracy: 0.8196
Epoch 2/150
657/657 [==============================] - 1s 919us/step - loss: 0.4496 - accuracy: 0.8161 -
val_loss: 0.4504 - val_accuracy: 0.8162
...
Epoch 149/150
657/657 [==============================] - 1s 788us/step - loss: 0.4204 - accuracy: 0.8253 -
val_loss: 0.4492 - val_accuracy: 0.8222
Epoch 150/150
657/657 [==============================] - 1s 800us/step - loss: 0.4210 - accuracy: 0.8242 -
val_loss: 0.4761 - val_accuracy: 0.8227
```

Medium number of epochs

- Let's set the epochs parameter to `100` and fit the model

```
ep_medium = create_model().fit(X_train_scaled, y_train,
                               epochs = 100,
                               validation_data=(X_val_scaled, y_val))
```

```
Epoch 1/100
657/657 [==============================] - 1s 990us/step - loss: 0.4889 - accuracy: 0.7938 -
val_loss: 0.4522 - val_accuracy: 0.8196
Epoch 2/100
657/657 [==============================] - 1s 841us/step - loss: 0.4496 - accuracy: 0.8161 -
val_loss: 0.4504 - val_accuracy: 0.8162
...
Epoch 99/100
657/657 [==============================] - 1s 802us/step - loss: 0.4281 - accuracy: 0.8200 -
val_loss: 0.4500 - val_accuracy: 0.8209
Epoch 100/100
657/657 [==============================] - 1s 799us/step - loss: 0.4286 - accuracy: 0.8217 -
val_loss: 0.4537 - val_accuracy: 0.8220
```

Lower number of epochs

- We set the number of epochs as `25` and fit the model

```
ep_low = create_model().fit(X_train_scaled, y_train,
                            epochs = 25,
                            validation_data = (X_val_scaled, y_val))
```

```
Epoch 1/25
657/657 [==============================] - 1s 1ms/step - loss: 0.4889 - accuracy: 0.7938 -
val_loss: 0.4522 - val_accuracy: 0.8196
Epoch 2/25
657/657 [==============================] - 1s 884us/step - loss: 0.4496 - accuracy: 0.8161 -
val_loss: 0.4504 - val_accuracy: 0.8162
...
Epoch 24/25
657/657 [==============================] - 1s 917us/step - loss: 0.4372 - accuracy: 0.8169 -
val_loss: 0.4337 - val_accuracy: 0.8202
Epoch 25/25
657/657 [==============================] - 1s 878us/step - loss: 0.4294 - accuracy: 0.8236 -
val_loss: 0.4355 - val_accuracy: 0.8216
```

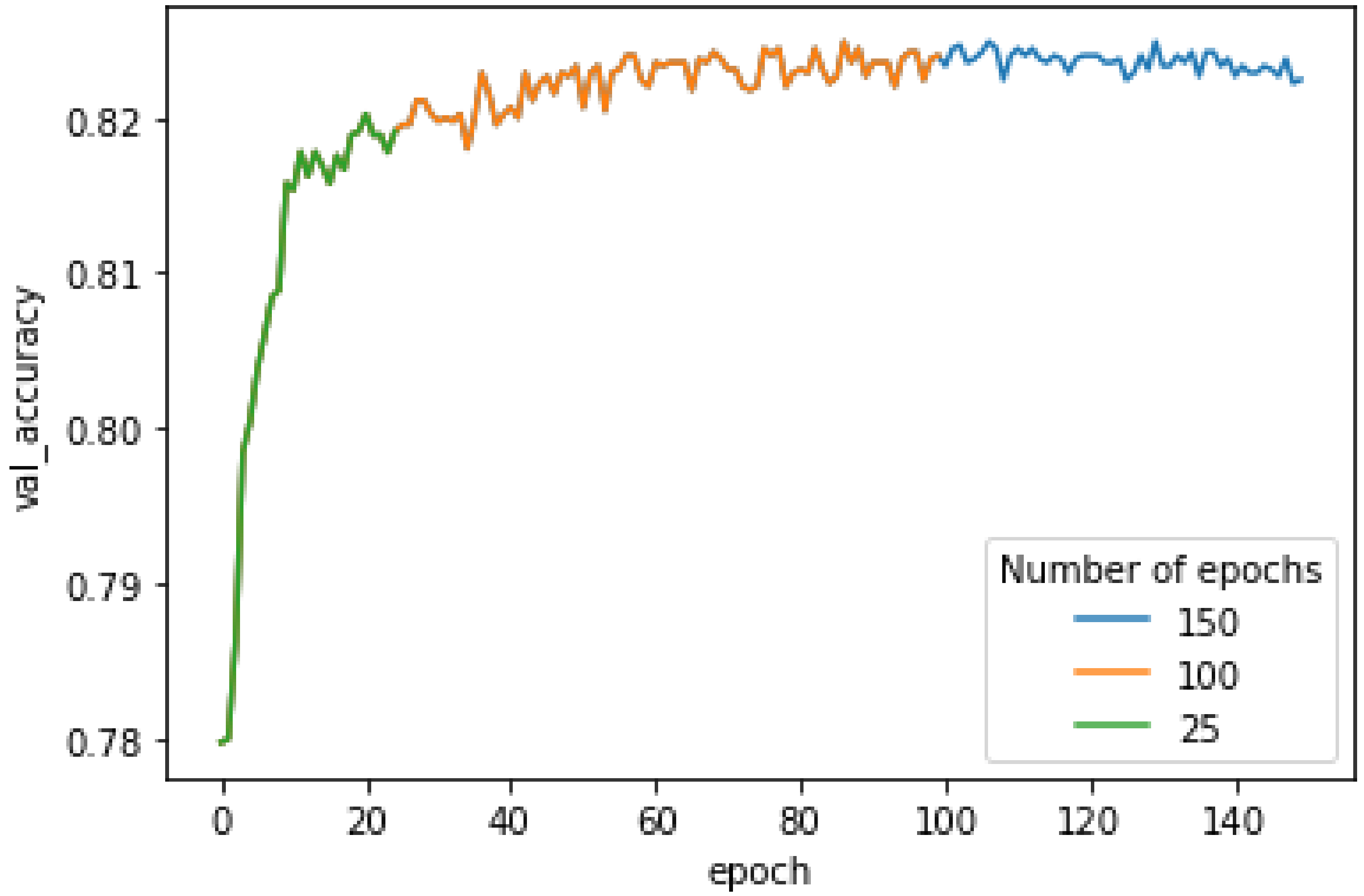
Visualize results for epoch sizes

- Let's create a dataframe with the loss and accuracy for training and validation data along with their corresponding epoch and the number of epochs they have been trained on

```
epoch_sizes = []

for exp, result in zip([ep_high, ep_medium, ep_low], ["150", "100", "25"]):

  df = pd.DataFrame.from_dict(exp.history)
  df['epoch'] = df.index.values
  df['Number of epochs'] = result

  epoch_sizes.append(df)

df_epochs = pd.concat(epoch_sizes)
df_epochs['Number of epochs'] = df_epochs['Number of epochs'].astype('str')
```

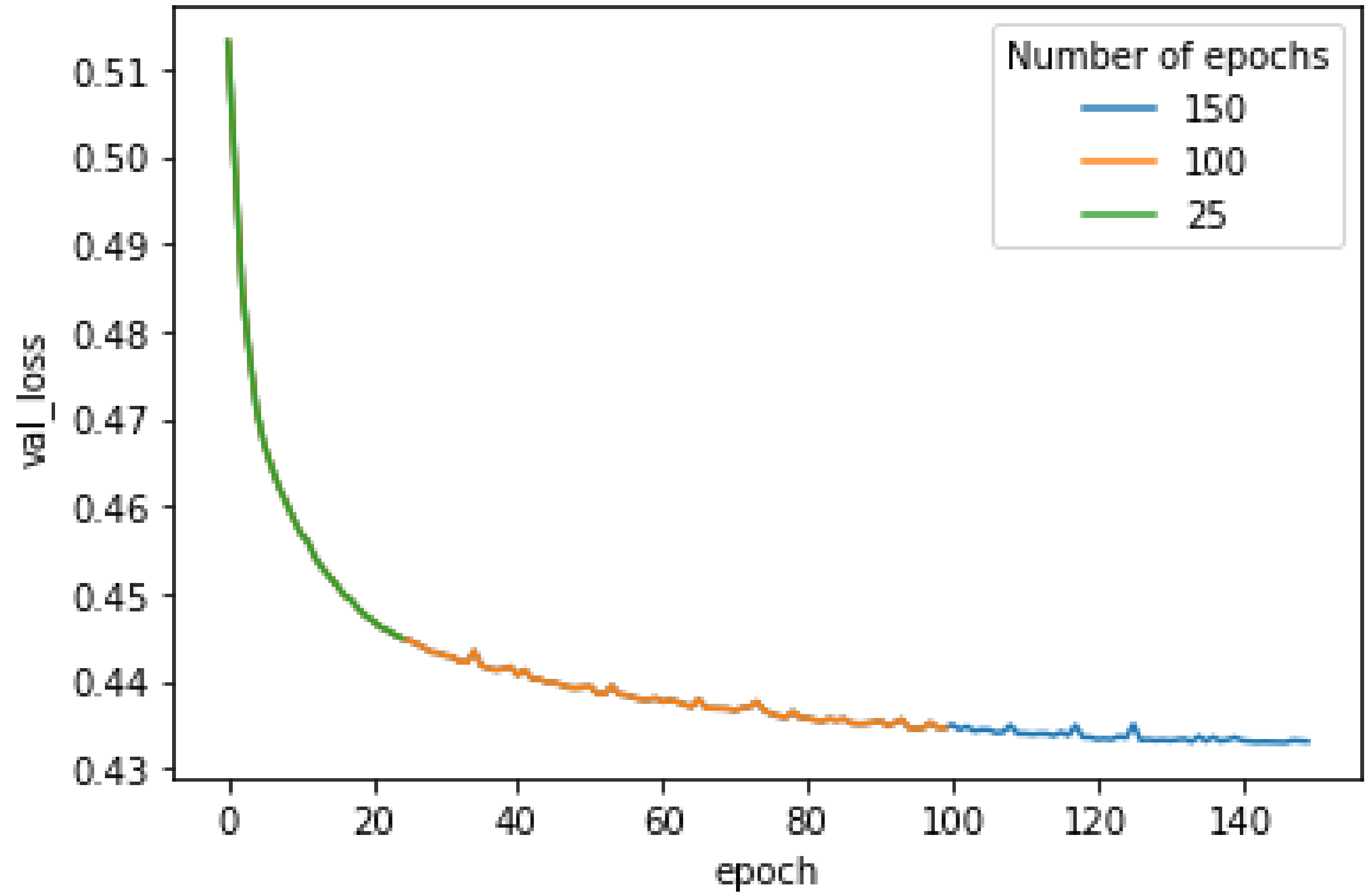
Visualize results for epoch sizes (cont'd)

```
sns.lineplot(x='epoch', y='val_accuracy', hue='Number of epochs', data=df_epochs)
```



Visualize results for epoch sizes (cont'd)

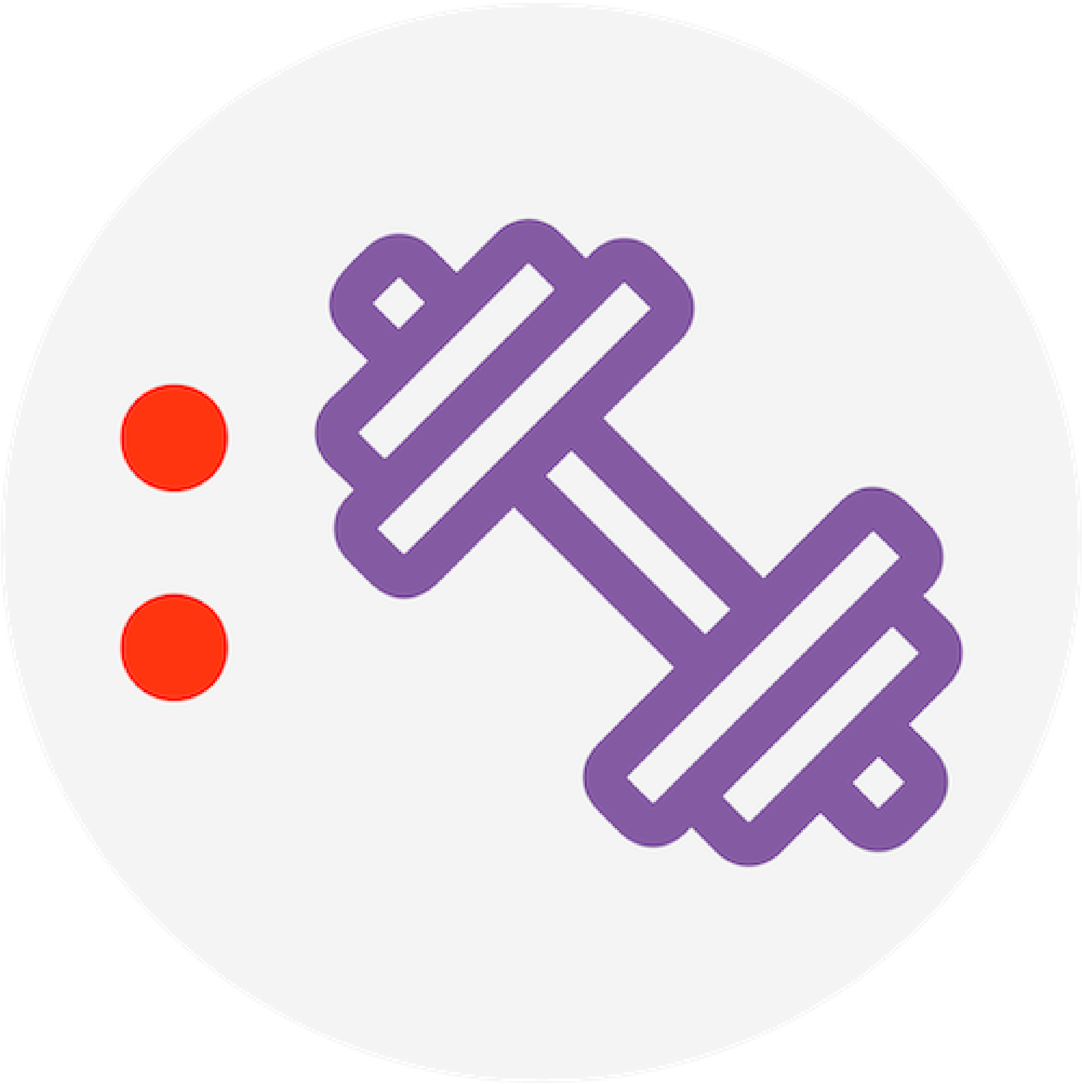
```
sns.lineplot(x='epoch', y='val_loss', hue='Number of epochs', data=df_epochs)
```



- The validation loss and accuracy obtained when the epochs are set to 150 are close to the values obtained when the number of epochs are set to 100

Knowledge check

Exercise



You are now ready to try tasks 8-12 in the Exercise for this topic

Module completion checklist

Objective	Complete
Implement a custom neural network for different batch sizes	✔
Evaluate the neural network for different number of epochs	✔

Congratulations on completing this module!