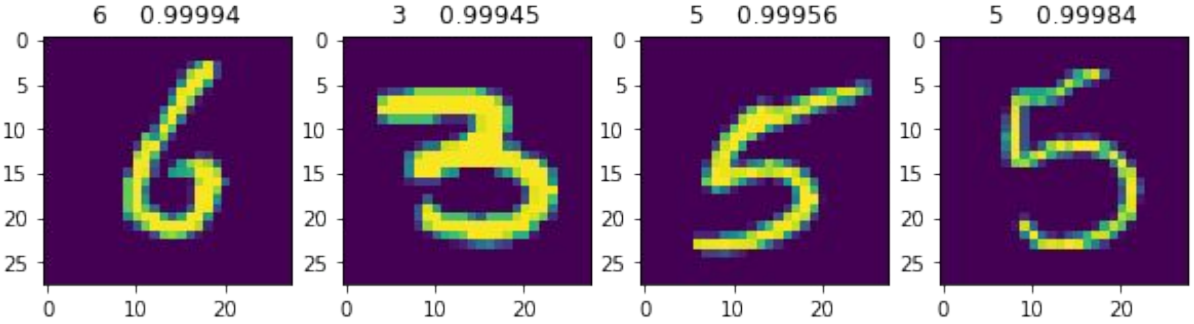
MNIST - FFNN vs CNN



CNN (best approach)

Accuracy: Training: 0.989, Validation: 0.986, Test: 0.987
Epochs: 22/60 (7-9s), Parameter: 21,304

Input Layer

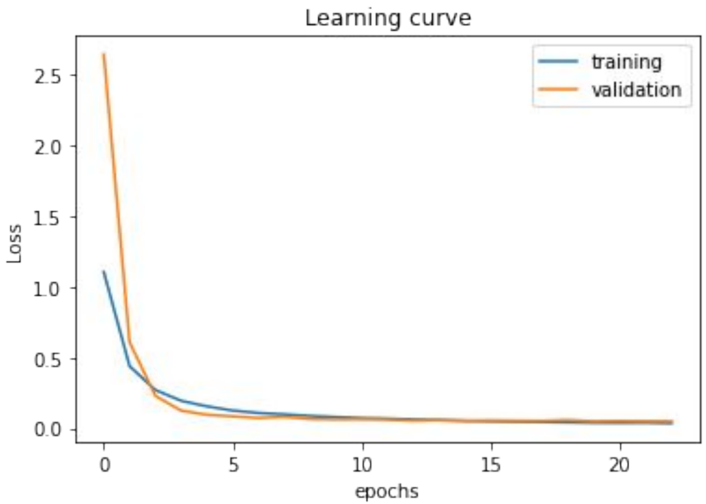
1. Conv2D(32, kernel=(3,3), strides=(2,2), input_shape=(28,28,1))
   ReLU, padding='same')
2. MaxPooling2D(pool=(2,2), strides=(2,2), padding='same')

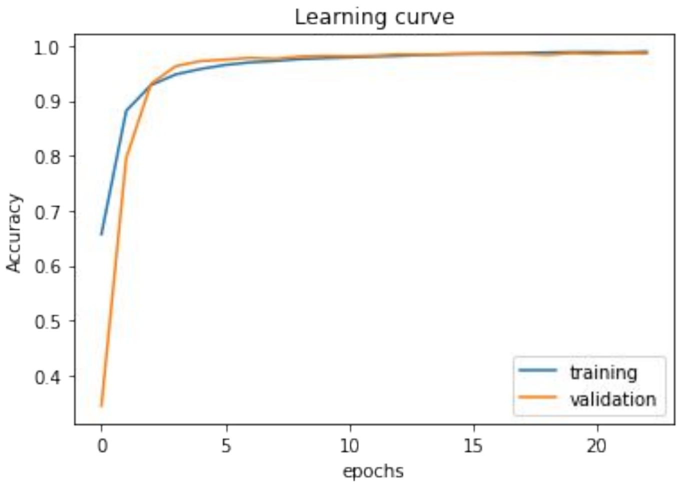
Hidden Layers

3. Conv2D(32, kernel=(3,3), strides=(2,2))
   ReLU, padding='same')
4. Conv2D(32, kernel=(3,3), strides=(2,2))
   ReLU, padding='same')
   BatchNormalisation()
5. MaxPooling2D(pool=(2,2), strides=(2,2), padding='same')
6. Flatten(), Dropout(0.3)
7. Dense(50, ReLU)
   BatchNormalisation()

Output Layer

8. Dense(10), Softmax
   Batch_size=1000, *optimizer=adam, EarlyStopping(patience=3)*





CNN (less kernels)

Accuracy: Training: 0.942, Validation: 0.972
Epochs: 30/60 (4-5s)

Input Layer

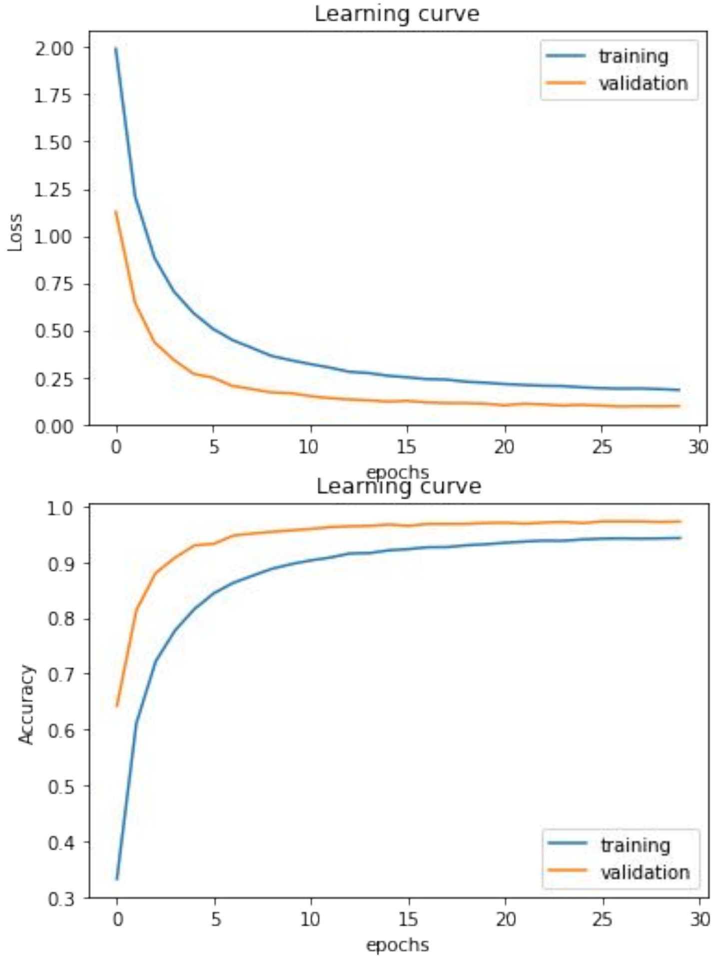
1. Conv2D(**16**, kernel=(3,3), strides=(2,2), input_shape=(28,28,1))
   ReLU, padding='same')
2. MaxPooling2D(pool=(2,2), strides=(2,2), padding='same')

Hidden Layers

3. Conv2D(**16**, kernel=(3,3), strides=(2,2))
   ReLU, padding='same')
4. Conv2D(**16**, kernel=(3,3), strides=(2,2))
   ReLU, padding='same')
5. MaxPooling2D(pool=(2,2), strides=(2,2), padding='same')
6. Flatten(), Dropout(0.3)
7. Dense(50, ReLU)
   BatchNormalisation()

Output Layer

8. Dense(10), Softmax
   Batch_size=1000, *optimizer=adam, EarlyStopping(patience=3)*



CNN (more kernels)

Accuracy: Training: 0.992, Validation: 0.987
Epochs: 16/60 **(12-13s)**

Input Layer

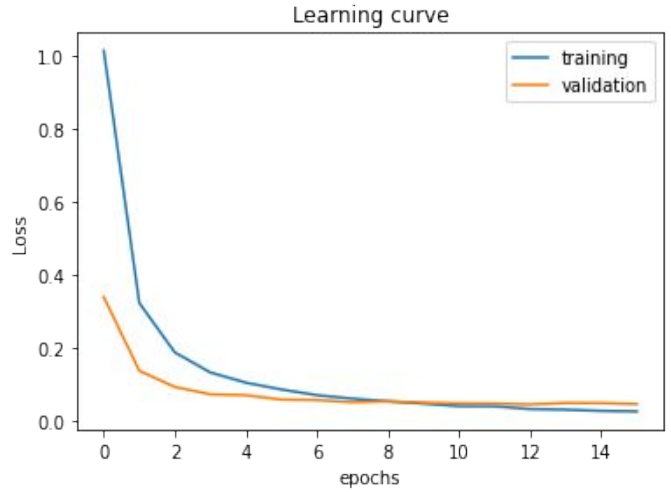
1. Conv2D(**64**, kernel=(3,3), strides=(2,2), input_shape=(28,28,1))
   ReLU, padding='same')
2. MaxPooling2D(pool=(2,2), strides=(2,2), padding='same')

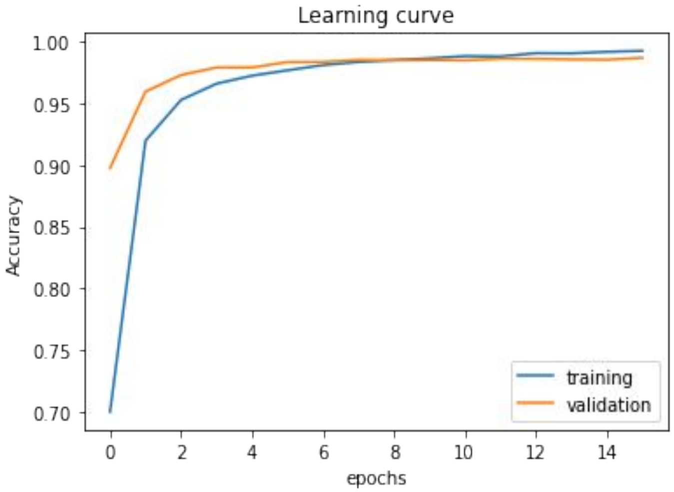
Hidden Layers

3. Conv2D(**64**, kernel=(3,3), strides=(2,2))
   ReLU, padding='same')
4. Conv2D(**64**, kernel=(3,3), strides=(2,2))
   ReLU, padding='same')
5. MaxPooling2D(pool=(2,2), strides=(2,2), padding='same')
6. Flatten(), Dropout(0.3)
7. Dense(50, ReLU)
   BatchNormalisation()

Output Layer

8. Dense(10), Softmax
   Batch_size=1000, *optimizer=adam, EarlyStopping(patience=3)*





CNN (no dropout)

Accuracy: Training: 0.981, Validation: 0.997
Epochs: 21/60 **(6-7s)**

Input Layer

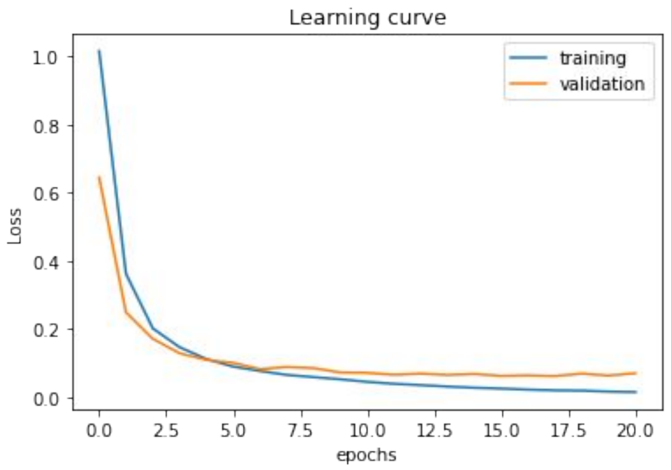
1. Conv2D(**32**, kernel=(3,3), strides=(2,2), input_shape=(28,28,1))
   ReLU, padding='same')
2. MaxPooling2D(pool=(2,2), strides=(2,2), padding='same')

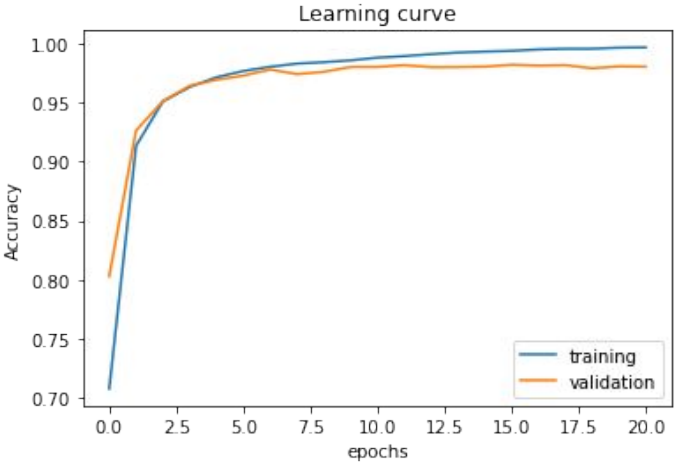
Hidden Layers

3. Conv2D(**32**, kernel=(3,3), strides=(2,2))
   ReLU, padding='same')
4. Conv2D(**32**, kernel=(3,3), strides=(2,2))
   ReLU, padding='same')
5. MaxPooling2D(pool=(2,2), strides=(2,2), padding='same')
6. Flatten(), **no Dropout**
7. Dense(50, ReLU)
   BatchNormalisation()

Output Layer

8. Dense(10), Softmax
   Batch_size=1000, *optimizer=adam, EarlyStopping(patience=3)*





CNN (BatchNorm-all)

Accuracy: Training: 1.0 , Validation: 0.983
Epochs: 24/60 **(7s)**

Input Layer

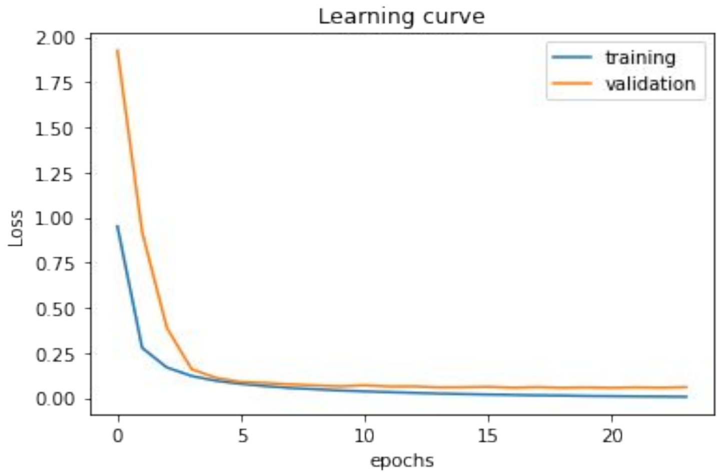
1. Conv2D(**32**, kernel=(3,3), strides=(2,2), input_shape=(28,28,1)) ReLU, padding='same')
2. MaxPooling2D(pool=(2,2), strides=(2,2), padding='same') **BatchNormalisation()**

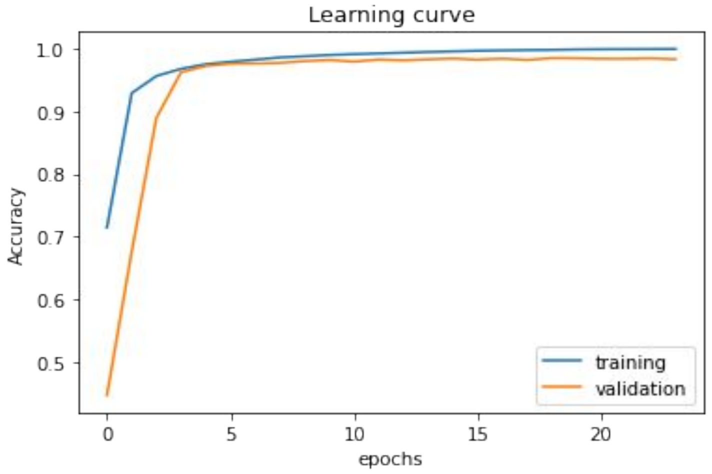
Hidden Layers

3. Conv2D(**32**, kernel=(3,3), strides=(2,2)) ReLU, padding='same')
4. Conv2D(**32**, kernel=(3,3), strides=(2,2)) ReLU, padding='same')
5. MaxPooling2D(pool=(2,2), strides=(2,2), padding='same') **BatchNormalisation()**
6. Flatten(), **no Dropout**
7. Dense(50, ReLU) BatchNormalisation()

Output Layer

8. Dense(10), Softmax Batch_size=1000, *optimizer=adam, EarlyStopping(patience=3)*





CNN (pooling size increased)

Accuracy: Training: 0.98 , Validation: 0.983
Epochs: 20/60 **(7s)**

Input Layer

1. Conv2D(**32**, kernel=(3,3), strides=(2,2), input_shape=(28,28,1)) ReLU, padding='same')
2. MaxPooling2D(pool=(**3,3**), strides=(2,2), padding='same')

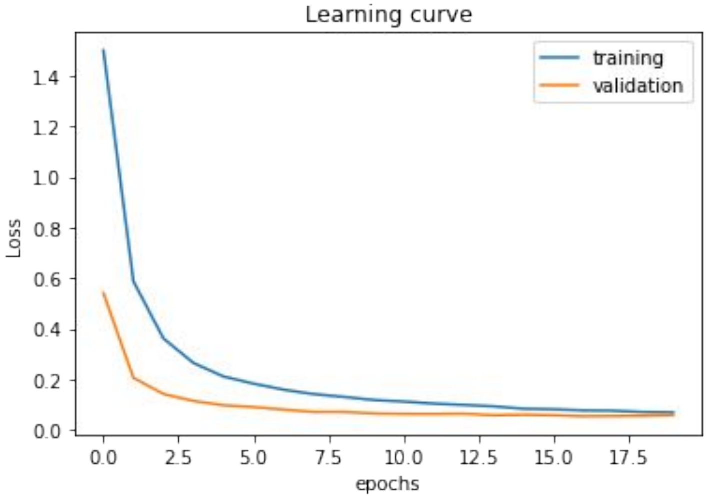
Hidden Layers

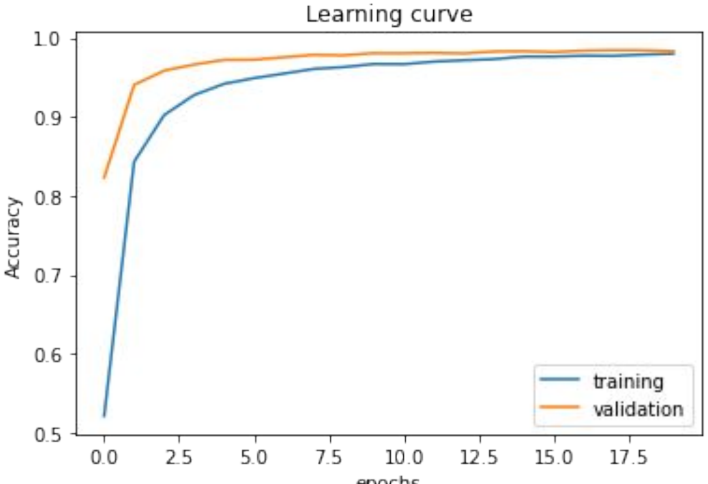
3. Conv2D(**32**, kernel=(3,3), strides=(2,2)) ReLU, padding='same')
4. Conv2D(**32**, kernel=(3,3), strides=(2,2)) ReLU, padding='same')
5. MaxPooling2D(pool=(**3,3**), strides=(2,2), padding='same')

6. Flatten(), Dropout(0.3)
7. Dense(50, ReLU) BatchNormalisation()

Output Layer

8. Dense(10), Softmax Batch_size=1000, *optimizer=adam, EarlyStopping(patience=3)*





CNN

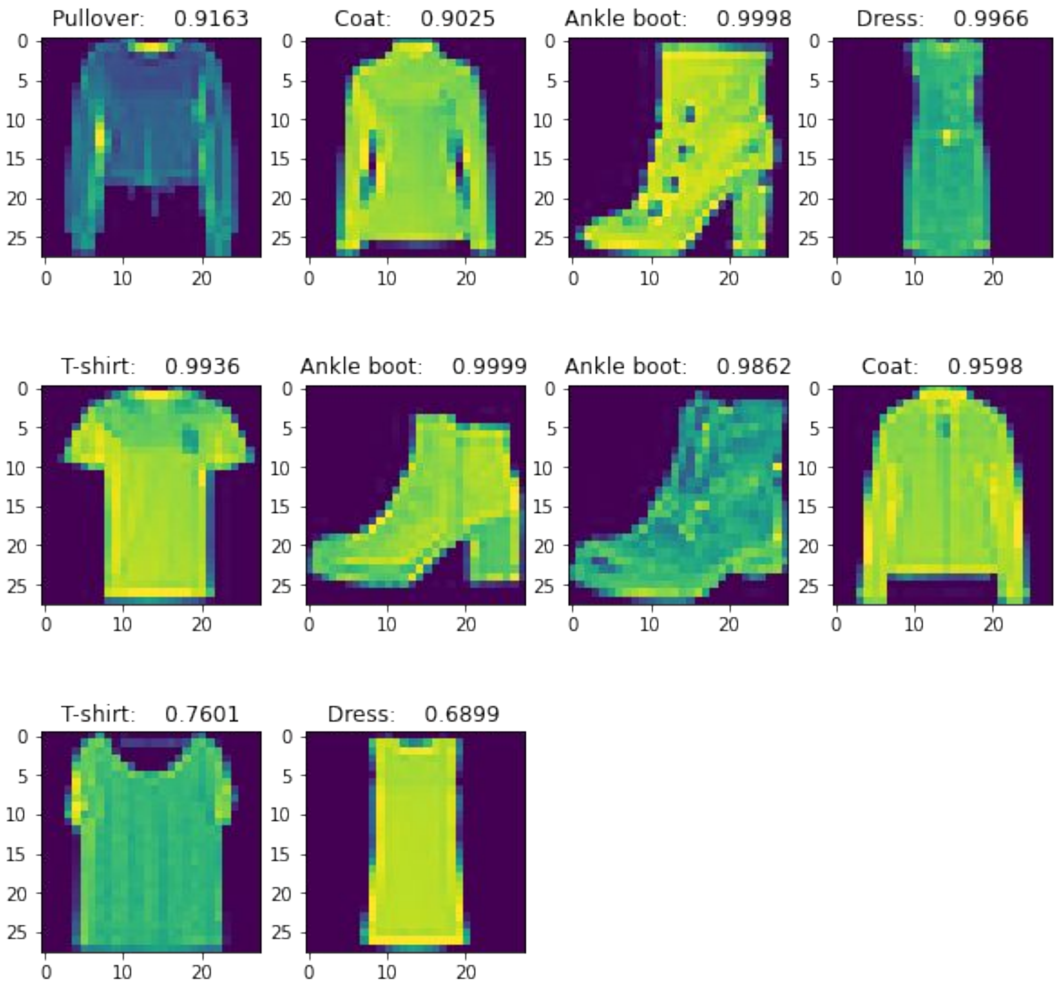
More kernels -> more accuracy, more prone to overfitting, resource heavier, faster learning

More BatchNormalisation -> more accuracy, more prone to overfitting, faster learning

Dropout -> better generalisation, too much Dropout slows down learning and reduces accuracy

Pooling size -> better generalisation and faster learning with smaller pooling size (but impact was small)

Fashion NIST
(test acc: 0.90)



FFNN (best approach)

Accuracy: Training: 0.969, Validation: 0.969
Epochs: 23/100

Input Layers

1. Dense(50, input_shape=(784,))
   Dropout(0.2), ELU, BatchNormalisation

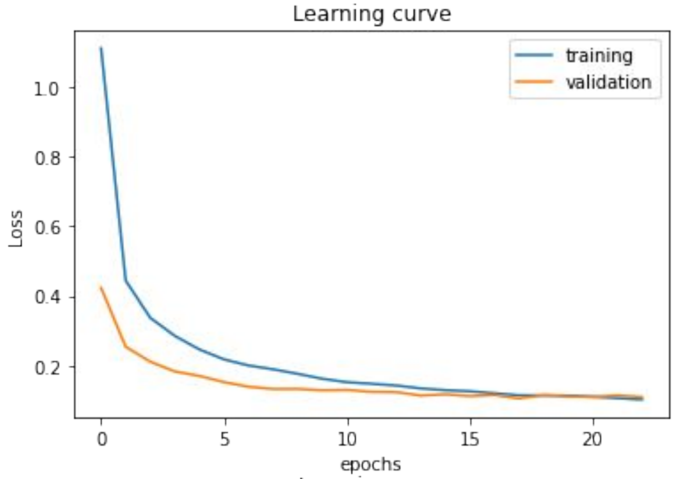
Hidden Layers

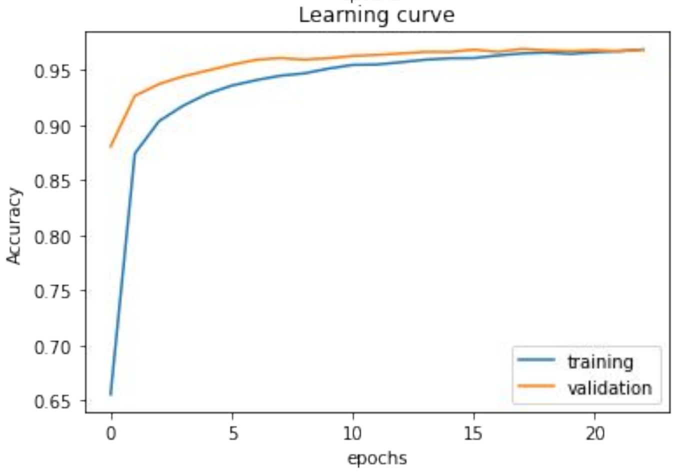
2. Dense(50)
   No Dropout, ELU, BatchNormalisation
3. Dense(50)
   No Dropout, ELU, BatchNormalisation
4. Dense(50)
   No Dropout, ELU, BatchNormalisation
5. Dense(50)
   Dropout(0.3), ELU, BatchNormalisation

Output Layer

6. Dense(10), Softmax

batch_size=1000, *optimizer=adam, EarlyStopping(patience=5)*





FFNN (too much normalisation)

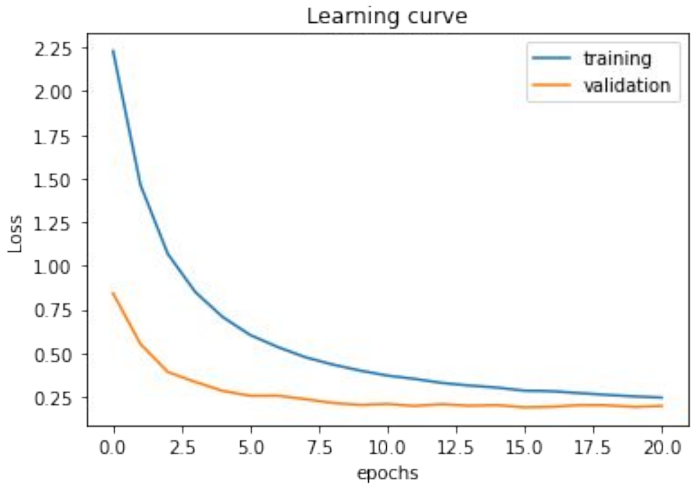
Accuracy: Training: 0.969, Validation: 0.969
Epochs: 21/100

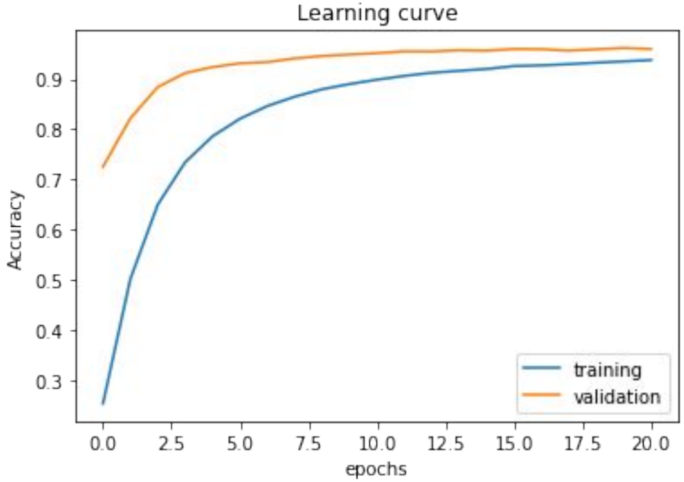
Input Layer
1. Dense(50, input_shape=(784,))
   Dropout(0.2), ELU, BatchNormalisation

Hidden Layer
2. Dense(50)
   Dropout (0.5), ELU, BatchNormalisation
3. Dense(50)
   Dropout (0.5), ELU, BatchNormalisation
4. Dense(50)
   Dropout (0.5), ELU, BatchNormalisation
5. Dense(50)
   Dropout(0.3), ELU, BatchNormalisation

Output Layer
6. Dense(10), Softmax

batch_size=1000, *optimizer=adam, EarlyStopping(patience=5)*





FFNN (overfitting)

Accuracy: Training: 0.994, Validation: 0.966
Epochs: 15/100

Input Layer

1. Dense(50, input_shape=(784,))
   ELU, BatchNormalisation

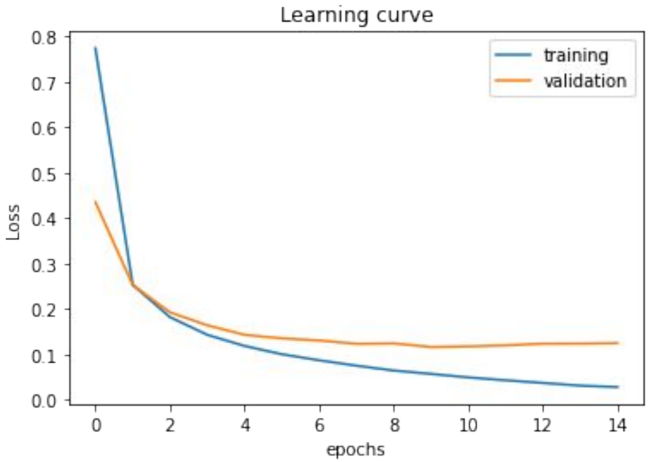
Hidden Layer

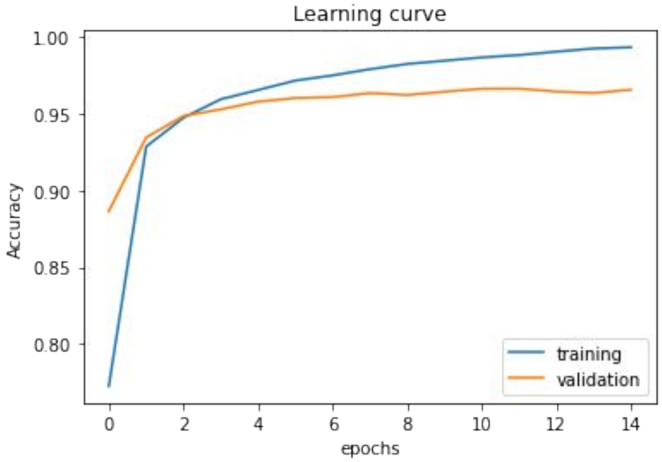
2. Dense(50)
   ELU, BatchNormalisation()
3. Dense(50)
   ELU, BatchNormalisation()
4. Dense(50)
   ELU, BatchNormalisation()
5. Dense(50)
   ELU, BatchNormalisation()

Output Layer

6. Dense(10), Softmax

batch_size=1000, *optimizer=adam, EarlyStopping(patience=5)*





FFNN (overfitting)

Accuracy: Training: 0.987, Validation: 0.954
Epochs: 33/100

Input Layer

1. Dense(50, input_shape=(784,))
   ELU

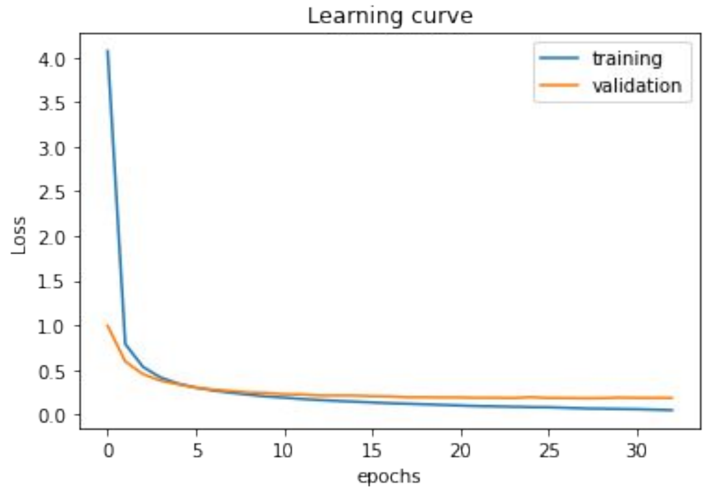
Hidden Layers

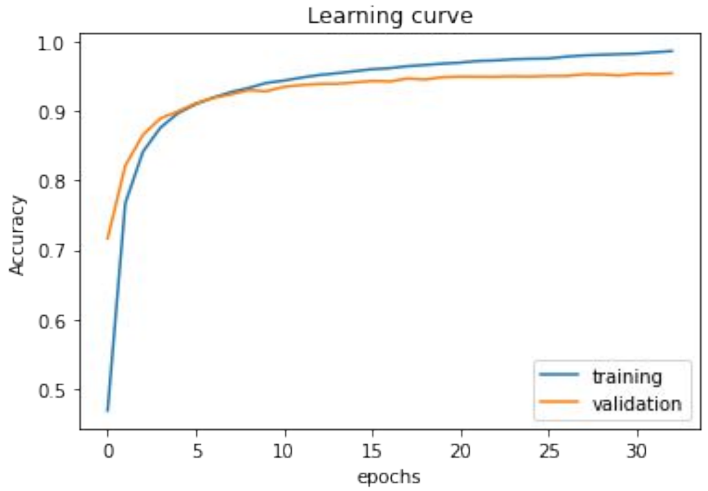
2. Dense(50)
   ELU
3. Dense(50)
   ELU
4. Dense(50)
   ELU
5. Dense(50)
   ELU

Output Layer

6. Dense(10), Softmax

Batch_size=1000, *optimizer=adam, EarlyStopping(patience=5)*





FFNN

- More layers and neurons -> more accuracy, more overfitting, slower
- More BatchNormalisation -> increase accuracy, more overfitting
- Dropout -> increase in validation accuracy on the cost of trainings accuracy
    - To much dropout slow learning and bad trainings accuracy