Reid Glaze
Dr Gurari
CSCI 5922
27 Sept 2022

Lab Assignment 2

1. Influence of Regularization

Methods

The dataset I used was called fashion_mnist, which can be loaded from keras.datasets. It includes 70,000 images in between the training and test sets. I only used the training set to perform my own splits, which includes 60,000 images. Each image is a 28 by 28 grayscale image. The associated labels are made up of 10 classes, each representing a clothing item shown in figure 1.. After using test_train_split, I got a training dataset with 42,000 samples and validation and test datasets with 9,000 samples each.

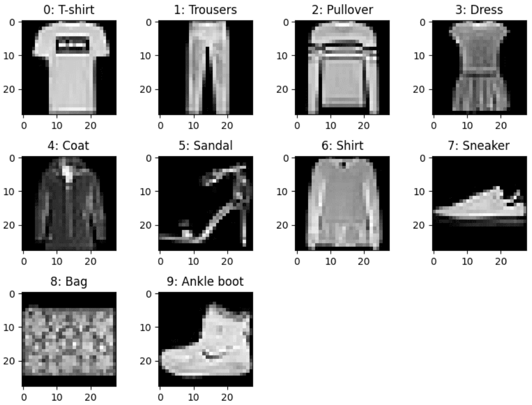


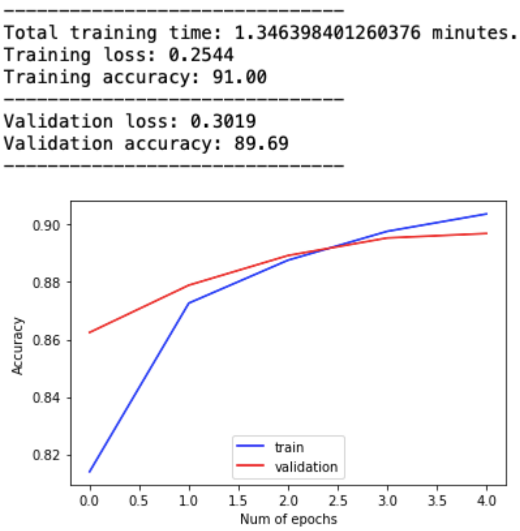
Figure 1 - Dataset Classes

I used keras in tensorflow to build 8 neural networks. 4 of these models included one convolutional layer and the other 4 included two convolutional layers. For each set of 4, I created one network without using regularization, one network using L2 kernel regularization, one network using a dropout function, and one network using batch normalization.

For these models, there were several hyperparameters that I kept constant. I fit each model using the same training data. In each model, I used mini batch gradient descent with 5 epochs and a batch size of 100. For each convolutional layer, I used a filter size of 16, the relu activation function, and a kernel size of 3. After each convolutional layer, I used a pooling layer with a pool size of 2,2. At the end of each model, I used two dense layers. The first dense layer uses the relu activation function and has a size of 80. The second dense layer uses the softmax activation function and has a size of 10. Each time I used L2 regularization, I used a value of 0.01. This was used on convolutional layers but not on the dense layers. Each time I used dropout, I chose a value of 0.1. Dropout was used after each convolutional layer and after the first dense layer. When Batch Normalization was used, it was done after each convolutional layer and after the first dense layer. Analysis was performed using the validation data.

Results

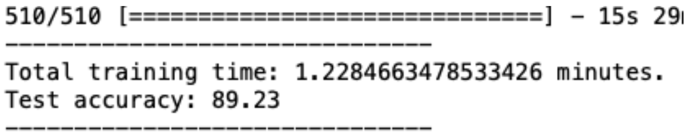
Of the 8 models I trained. The model that had one convolutional layer and used batch normalization returned the highest validation accuracy. This value was 86.69, shown in the Figure 2. The training and validation accuracies are also shown in the plot over each of the 5 epochs. The training time was 1.35 minutes.



```
------------------------------
Total training time: 1.346398401260376 minutes.
Training loss: 0.2544
Training accuracy: 91.00
------------------------------
Validation loss: 0.3019
Validation accuracy: 89.69
------------------------------
```

Figure 2 - Optimal Model

After creating each of the 8 neural networks and analyzing the optimal approach, I applied the model on the test data. Instead of using the train data to train the model, I used the training and validation data together. This resulted in the train / test data split being 85 / 15. I got a final accuracy score of 89.23, shown in figure 3. The training time was 1.23 minutes.



```
510/510 [==============================] - 15s 29
--------------------------------
Total training time: 1.2284663478533426 minutes.
Test accuracy: 89.23
--------------------------------
```

Figure 3 - Final Accuracy

Reid Glaze
Dr Gurari
CSCI 5922
27 Sept 2022

Analysis

One trend observed is that one convolutional layer generally resulted in a higher validation accuracy than two convolutional layers did despite having shorter training times. When no regularization was used, the respective accuracies were 85.41 and 81.24. With L2 regularization, 80.11 and 82.82. With dropout, they were 87.72 and 71.12. Finally, with batch normalization, they were 89.69 and 85.98. My friend found the opposite to be true. Additionally, online sources say that a higher number of convolutional layers usually results in a higher accuracy. I am assuming that I am getting these results because I am using a smaller filter size than my friend. I am using a filter size of 16, while he is using a filter size of 64. With a small filter size, I may be limited in the number of abstractions I can make, and the accuracy may be more likely to decrease for this reason.

Another trend observed is that batch normalization consistently resulted in a higher validation accuracy than the other regularization methods. L2 regularization did not seem to work very well. This method forces the weights towards 0 but does not make them 0. I used a value of 0.01 and it is possible that this value penalized the weights too much. When comparing results from the different number of convolutional layers, both had similar validation accuracies. Over-regularization could have resulted in underfitting in each of these models. When using dropout, I had a good accuracy with one layer but poor accuracy with two layers. This could be explained by dropout having a larger effect a network with a smaller number of neurons. Batch normalization, which reduces the distribution of input variables, was shown to be most effective. It was especially effective with one layer. It is likely that this regularization technique worked the best because it did not have the problems that the other two regularization techniques had. It is also likely that it was more accurate than no regularization at all because the model containing no regularization had overfitting.

2. Interpreting CNN Representation

Methods

For part b, I visualized the kernel for two different convolutional layers. The assignment asks us to use our model that results in the highest validation accuracy. However, my optimal model only has one convolutional layer. For this reason, I have used a different model to visualize and understand multiple convolutional layers. I will choose the model where I have two convolutional layers and use batch normalization for regularization.

To visualize the convolutional layers, I first selected the convolutional layer and used the get_weights() function. From get weights, I extracted the filters and the bias. Next, I printed the shape of each of the filters. The shapes can be shown in each of the images. I created a subplot with the number of columns as the number of inputs and the number of rows as the number of filters in that convolutional layer. I plotted each of the 3 by 3 kernels using a grayscale colormap.

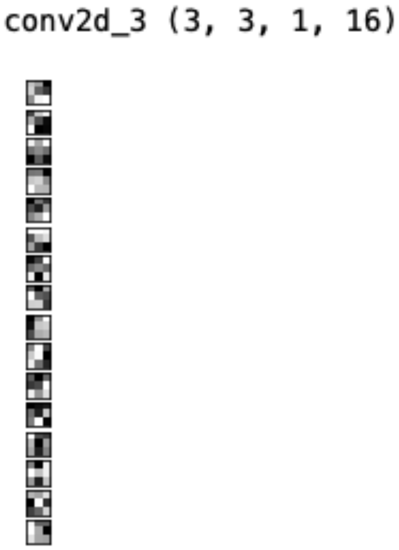


Figure 4 -1st convolutional layer

Results

For the first convolutional layer, I got 16 different kernels. For the second convolutional layer, I got 256 or 16 * 16 of these kernels. The kernels for the first layer are shown in figure 4. The kernels for the second layer are shown in figure 5.

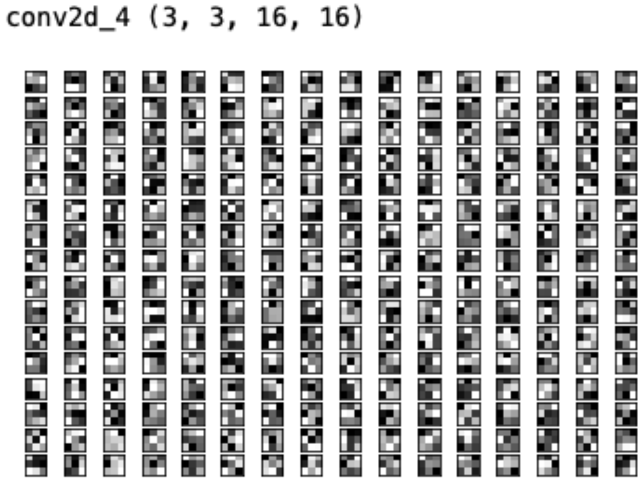


Figure 5 - 2nd convolutional layer

Analysis

The main difference that can be noted when comparing the first convolutional layer and the second one is that the second layer has way more kernels. The first convolutional layer has a single column with 16 rows. This is because there is only one input, and the filter size is 16. The second convolutional layer has an input size of 16 neurons. When a convolutional layer with a filter size of 16 is applied to this input, we get a 16 by 16 array of 3 by 3 kernels.

One thing that can be inferred from these visualizations is that the abstractions made in the first convolutional layer are important and that it makes sense to have a higher number of filters in the first layer. In the first layer, 16 abstractions are made. Each of these outputs are assigned a weight. In the second layer, there are 256 abstractions and each of these are assigned a weight. The model is simplified if only 16 abstractions are made in the first layer. Because of this, the abstractions are going to be more generalized. As we go further into the network, these abstractions become more specialized. The accuracy does not benefit from this specialization when the initial layer is too generalized. This would explain why my friend, who used a filter size of 64 was able to benefit more from this specialization in the 2nd layer.

Lab Assignment 2

September 27, 2022

[2]:

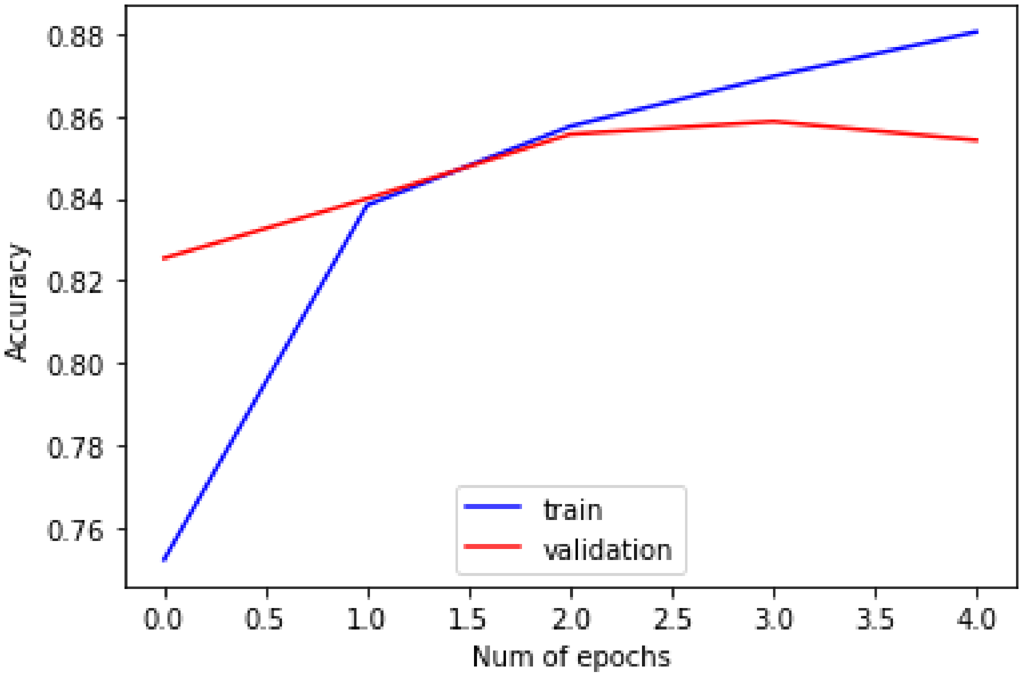
```
#Reid Glaze
#Dr Gurari
#27 Sept 2022
#Lab Assignment 2

from tensorflow.keras.utils import to_categorical
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import BatchNormalization

#import dataset - will only use the train portion to split into train, test,␣
 ↪and validation
from keras.datasets import fashion_mnist
from keras import regularizers
import time
import numpy as np
from matplotlib import pyplot as plt
(X, y), (X_test, y_test) = fashion_mnist.load_data()

#train, test, validation split 70-15-15
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=123)
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5,␣
 ↪random_state=123)


#Data pre-processing
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
X_val = X_val.reshape(X_val.shape[0], 28, 28, 1)

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_val = to_categorical(y_val)
```

[110]:

```
#1) one convolutional layer, no regularization



model = Sequential()
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', input_shape =␣
 ↪(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(80, activation = 'relu'))
model.add(Dense(10, activation = "softmax"))
model.summary()

model.compile(loss = "categorical_crossentropy", optimizer = "SGD", metrics =␣
 ↪["accuracy"])

scores = []
histories = []
start = time.time()
history = model.fit(X_train, y_train, epochs = 5, batch_size = 100,␣
 ↪validation_data= (X_val, y_val))
end = time.time()
num_mins = (end-start)/60
print("-------------------------------")
print("Total training time: "  + str(num_mins) + " minutes.")
score = model.evaluate(X_train, y_train, verbose = 0)
print("Training loss: %.4f" % score[0])
print("Training accuracy: %.2f" % (score[1] * 100.0))
print("-------------------------------")
score = model.evaluate(X_val, y_val, verbose = 0)
print("Validation loss: %.4f" % score[0])
print("Validation accuracy: %.2f" % (score[1] * 100.0))
print("-------------------------------")
scores.append(score[1])
histories.append(history)
plt.plot(histories[0].history["accuracy"], color = "blue", label = "train")
plt.plot(histories[0].history["val_accuracy"], color = "red", label =␣
 ↪"validation")
plt.ylabel("Accuracy")
plt.xlabel("Num of epochs")
leg = plt.legend(loc='lower center')
plt.show()
```

```
Model: "sequential_99"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_151 (Conv2D)          (None, 26, 26, 16)        160
```

```
_________________________________________________________________
max_pooling2d_151 (MaxPoolin (None, 13, 13, 16)        0
_________________________________________________________________
flatten_99 (Flatten)         (None, 2704)              0
_________________________________________________________________
dense_198 (Dense)            (None, 80)                216400
_________________________________________________________________
dense_199 (Dense)            (None, 10)                810
=================================================================
Total params: 217,370
Trainable params: 217,370
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
420/420 [==============================] - 13s 30ms/step - loss: 1.2219 -
accuracy: 0.7522 - val_loss: 0.4927 - val_accuracy: 0.8254
Epoch 2/5
420/420 [==============================] - 12s 29ms/step - loss: 0.4497 -
accuracy: 0.8383 - val_loss: 0.4527 - val_accuracy: 0.8399
Epoch 3/5
420/420 [==============================] - 11s 26ms/step - loss: 0.3923 -
accuracy: 0.8575 - val_loss: 0.4150 - val_accuracy: 0.8556
Epoch 4/5
420/420 [==============================] - 10s 23ms/step - loss: 0.3530 -
accuracy: 0.8696 - val_loss: 0.4037 - val_accuracy: 0.8587
Epoch 5/5
420/420 [==============================] - 9s 23ms/step - loss: 0.3245 -
accuracy: 0.8804 - val_loss: 0.4190 - val_accuracy: 0.8541
-------------------------------
Total training time: 0.9325737158457438 minutes.
Training loss: 0.3123
Training accuracy: 88.37
-------------------------------
Validation loss: 0.4190
Validation accuracy: 85.41
-------------------------------
```



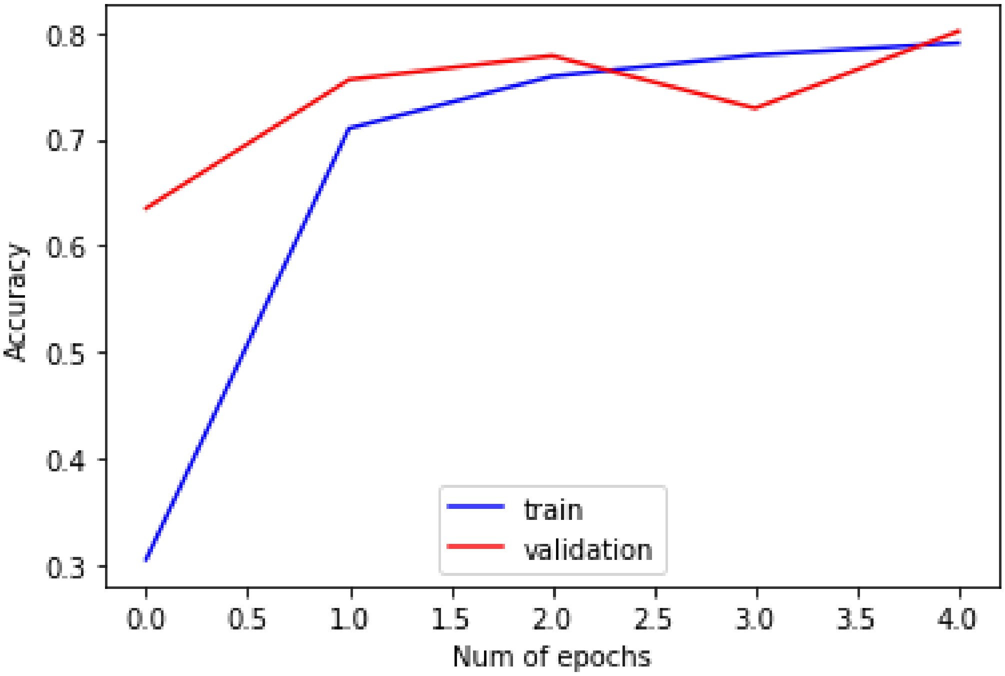
[87]:

```
#2) one convolutional layer, use L2 kernel regularization

model = Sequential()
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', input_shape =␣
 ↪(28,28,1), kernel_regularizer = regularizers.l2(0.01)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(80, activation = 'relu'))
model.add(Dense(10, activation = "softmax"))
model.summary()

model.compile(loss = "categorical_crossentropy", optimizer = "SGD", metrics =␣
 ↪["accuracy"])

scores = []
histories = []
start = time.time()
history = model.fit(X_train, y_train, epochs = 5, batch_size = 100,␣
 ↪validation_data= (X_val, y_val))
end = time.time()
num_mins = (end-start)/60
print("--------------------------------")
print("Total training time: "  + str(num_mins) + " minutes.")
score = model.evaluate(X_train, y_train, verbose = 0)
```

```
print("Training loss: %.4f" % score[0])
print("Training accuracy: %.2f" % (score[1] * 100.0))
print("-------------------------------")
score = model.evaluate(X_val, y_val, verbose = 0)
print("Validation loss: %.4f" % score[0])
print("Validation accuracy: %.2f" % (score[1] * 100.0))
print("-------------------------------")
scores.append(score[1])
histories.append(history)
plt.plot(histories[0].history["accuracy"], color = "blue", label = "train")
plt.plot(histories[0].history["val_accuracy"], color = "red", label =␣
 ↪"validation")
plt.ylabel("Accuracy")
plt.xlabel("Num of epochs")
leg = plt.legend(loc='lower center')
plt.show()
```

```
Model: "sequential_84"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_132 (Conv2D)          (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_132 (MaxPoolin (None, 13, 13, 16)        0
_________________________________________________________________
flatten_84 (Flatten)         (None, 2704)              0
_________________________________________________________________
dense_168 (Dense)            (None, 80)                216400
_________________________________________________________________
dense_169 (Dense)            (None, 10)                810
=================================================================
Total params: 217,370
Trainable params: 217,370
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
420/420 [==============================] - 11s 26ms/step - loss: 7.7318 -
accuracy: 0.3049 - val_loss: 4.8455 - val_accuracy: 0.6350
Epoch 2/5
420/420 [==============================] - 9s 20ms/step - loss: 4.2121 -
accuracy: 0.7100 - val_loss: 3.7600 - val_accuracy: 0.7559
Epoch 3/5
420/420 [==============================] - 9s 21ms/step - loss: 3.4967 -
accuracy: 0.7590 - val_loss: 3.2318 - val_accuracy: 0.7782
Epoch 4/5
420/420 [==============================] - 10s 23ms/step - loss: 3.0022 -
accuracy: 0.7789 - val_loss: 2.9048 - val_accuracy: 0.7290
Epoch 5/5
```

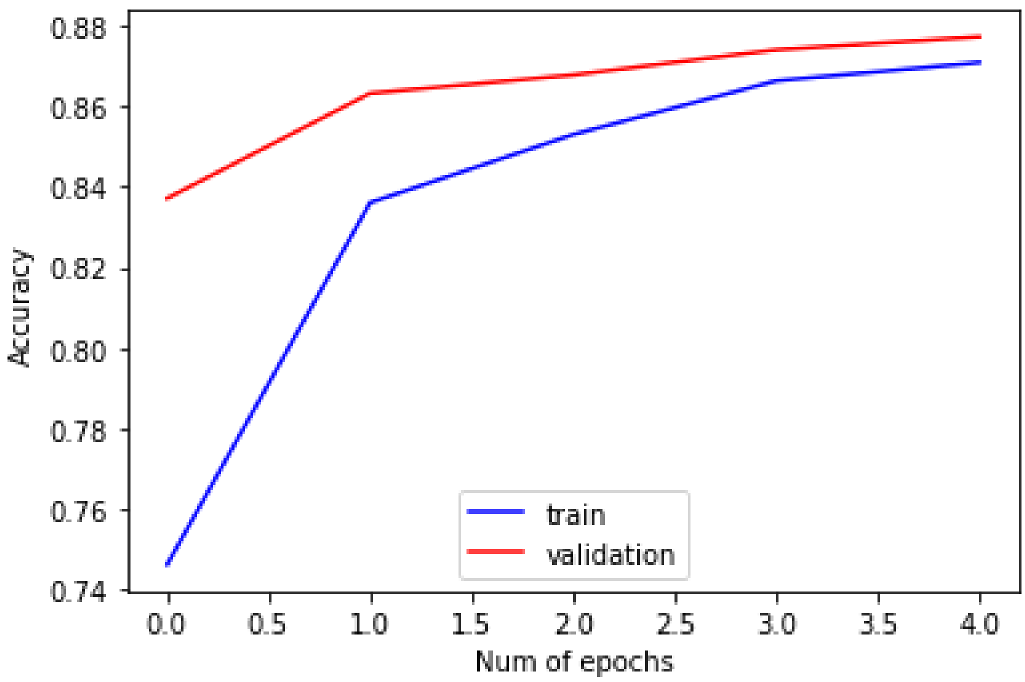
```
420/420 [==============================] - 9s 21ms/step - loss: 2.6023 -
accuracy: 0.7902 - val_loss: 2.4195 - val_accuracy: 0.8011
-------------------------------
Total training time: 0.7899572650591532 minutes.
Training loss: 2.4131
Training accuracy: 80.23
-------------------------------
Validation loss: 2.4195
Validation accuracy: 80.11
-------------------------------
```



[89]:
```
#3) one convolutional layer, use Droupout function

model = Sequential()
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', input_shape =␣
 ↪(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.1))
model.add(Flatten())
model.add(Dense(80, activation = 'relu'))
model.add(Dropout(0.1))
model.add(Dense(10, activation = "softmax"))
model.summary()
model.compile(loss = "categorical_crossentropy", optimizer = "SGD", metrics =␣
 ↪["accuracy"])

scores = []
histories = []
start = time.time()
history = model.fit(X_train, y_train, epochs = 5, batch_size = 100,␣
 ↪validation_data= (X_val, y_val))
end = time.time()
num_mins = (end-start)/60
print("--------------------------------")
print("Total training time: "  + str(num_mins) + " minutes.")
score = model.evaluate(X_train, y_train, verbose = 0)
print("Training loss: %.4f" % score[0])
print("Training accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
score = model.evaluate(X_val, y_val, verbose = 0)
print("Validation loss: %.4f" % score[0])
print("Validation accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
scores.append(score[1])
histories.append(history)
plt.plot(histories[0].history["accuracy"], color = "blue", label = "train")
plt.plot(histories[0].history["val_accuracy"], color = "red", label =␣
 ↪"validation")
plt.ylabel("Accuracy")
plt.xlabel("Num of epochs")
leg = plt.legend(loc='lower center')
plt.show()
```

```
Model: "sequential_86"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_134 (Conv2D)          (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_134 (MaxPoolin (None, 13, 13, 16)        0
_________________________________________________________________
module_wrapper_39 (ModuleWra (None, 13, 13, 16)        0
_________________________________________________________________
flatten_86 (Flatten)         (None, 2704)              0
_________________________________________________________________
dense_172 (Dense)            (None, 80)                216400
_________________________________________________________________
module_wrapper_40 (ModuleWra (None, 80)                0
_________________________________________________________________
dense_173 (Dense)            (None, 10)                810
```

```
=================================================================
Total params: 217,370
Trainable params: 217,370
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
420/420 [==============================] - 12s 27ms/step - loss: 0.9949 -
accuracy: 0.7462 - val_loss: 0.4719 - val_accuracy: 0.8371
Epoch 2/5
420/420 [==============================] - 10s 24ms/step - loss: 0.4566 -
accuracy: 0.8360 - val_loss: 0.3961 - val_accuracy: 0.8632
Epoch 3/5
420/420 [==============================] - 10s 25ms/step - loss: 0.4088 -
accuracy: 0.8530 - val_loss: 0.3861 - val_accuracy: 0.8678
Epoch 4/5
420/420 [==============================] - 10s 24ms/step - loss: 0.3749 -
accuracy: 0.8663 - val_loss: 0.3584 - val_accuracy: 0.8740
Epoch 5/5
420/420 [==============================] - 11s 25ms/step - loss: 0.3537 -
accuracy: 0.8708 - val_loss: 0.3507 - val_accuracy: 0.8772
-------------------------------
Total training time: 0.8832791845003763 minutes.
Training loss: 0.2892
Training accuracy: 89.72
-------------------------------
Validation loss: 0.3507
Validation accuracy: 87.72
-------------------------------
```

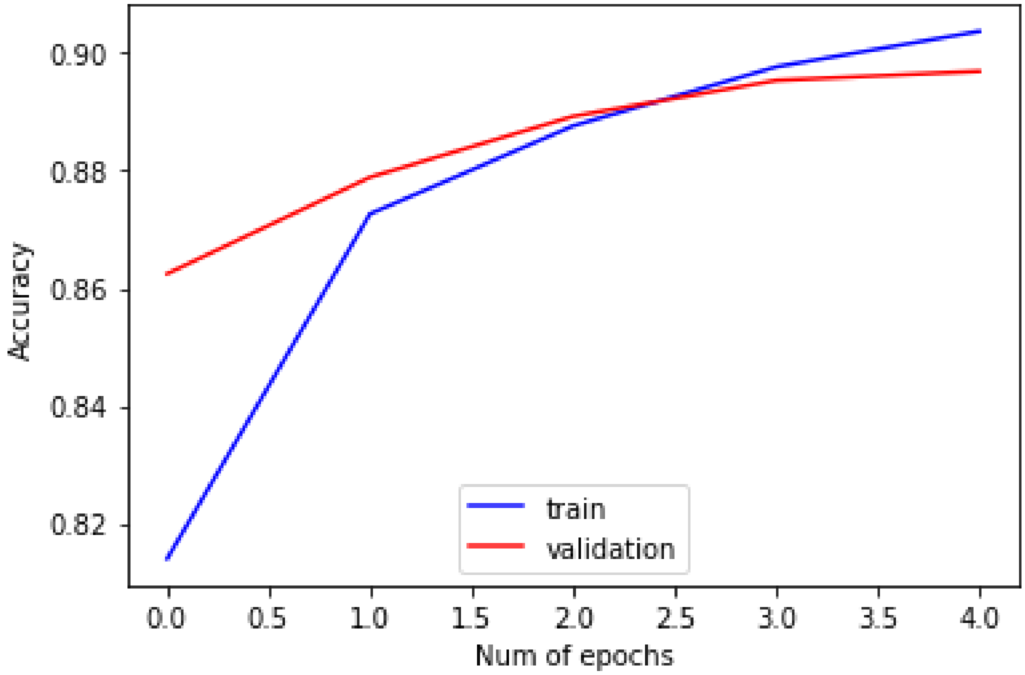


[95]:

```
#4) one convolutional layer, batch normalization

model = Sequential()
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', input_shape =␣
 ↪(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(80, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dense(10, activation = "softmax"))
model.summary()

model.compile(loss = "categorical_crossentropy", optimizer = "SGD", metrics =␣
 ↪["accuracy"])

scores = []
histories = []
start = time.time()
history = model.fit(X_train, y_train, epochs = 5, batch_size = 100,␣
 ↪validation_data= (X_val, y_val))
end = time.time()
num_mins = (end-start)/60
print("-------------------------------")
print("Total training time: "  + str(num_mins) + " minutes.")
score = model.evaluate(X_train, y_train, verbose = 0)
print("Training loss: %.4f" % score[0])
print("Training accuracy: %.2f" % (score[1] * 100.0))
print("-------------------------------")
score = model.evaluate(X_val, y_val, verbose = 0)
print("Validation loss: %.4f" % score[0])
print("Validation accuracy: %.2f" % (score[1] * 100.0))
print("-------------------------------")
score = model.evaluate(X_val, y_val, verbose = 0)
scores.append(score[1])
histories.append(history)
plt.plot(histories[0].history["accuracy"], color = "blue", label = "train")
plt.plot(histories[0].history["val_accuracy"], color = "red", label =␣
 ↪"validation")
plt.ylabel("Accuracy")
plt.xlabel("Num of epochs")
leg = plt.legend(loc='lower center')
plt.show()
```

```
Model: "sequential_92"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_144 (Conv2D)          (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_144 (MaxPoolin (None, 13, 13, 16)        0
_________________________________________________________________
module_wrapper_49 (ModuleWra (None, 13, 13, 16)        64
_________________________________________________________________
flatten_92 (Flatten)         (None, 2704)              0
_________________________________________________________________
dense_184 (Dense)            (None, 80)                216400
_________________________________________________________________
module_wrapper_50 (ModuleWra (None, 80)                320
_________________________________________________________________
dense_185 (Dense)            (None, 10)                810
=================================================================
Total params: 217,754
Trainable params: 217,562
Non-trainable params: 192
_________________________________________________________________
Epoch 1/5
420/420 [==============================] - 19s 43ms/step - loss: 0.5430 -
accuracy: 0.8140 - val_loss: 0.4056 - val_accuracy: 0.8624
Epoch 2/5
420/420 [==============================] - 17s 40ms/step - loss: 0.3714 -
accuracy: 0.8726 - val_loss: 0.3607 - val_accuracy: 0.8789
```

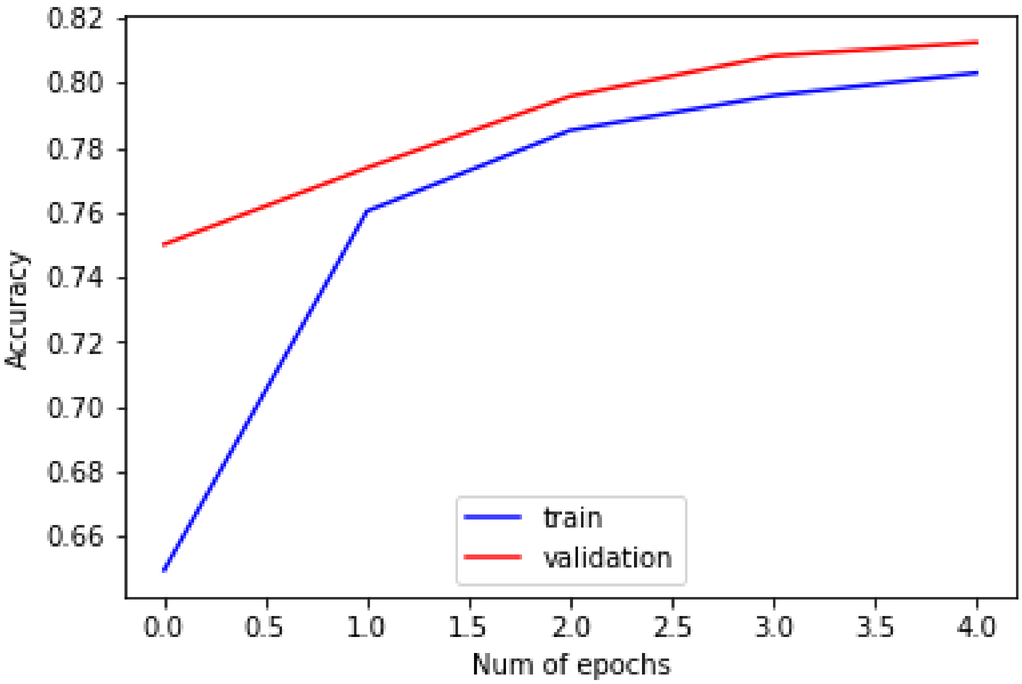
```
Epoch 3/5
420/420 [==============================] - 15s 37ms/step - loss: 0.3251 -
accuracy: 0.8876 - val_loss: 0.3247 - val_accuracy: 0.8892
Epoch 4/5
420/420 [==============================] - 15s 35ms/step - loss: 0.2949 -
accuracy: 0.8976 - val_loss: 0.3173 - val_accuracy: 0.8953
Epoch 5/5
420/420 [==============================] - 14s 34ms/step - loss: 0.2737 -
accuracy: 0.9037 - val_loss: 0.3019 - val_accuracy: 0.8969
------------------------------
Total training time: 1.346398401260376 minutes.
Training loss: 0.2544
Training accuracy: 91.00
------------------------------
Validation loss: 0.3019
Validation accuracy: 89.69
------------------------------
```



[91]:
```
#5) two convolutional layers, no regularization


model = Sequential()
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', input_shape =␣
 ↪(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(16, kernel_size = 3, activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(80, activation = 'relu'))
model.add(Dense(10, activation = "softmax"))
model.summary()

model.compile(loss = "categorical_crossentropy", optimizer = "SGD", metrics =␣
 ↪["accuracy"])

scores = []
histories = []
start = time.time()
history = model.fit(X_train, y_train, epochs = 5, batch_size = 100,␣
 ↪validation_data= (X_val, y_val))
end = time.time()
num_mins = (end-start)/60
print("--------------------------------")
print("Total training time: "  + str(num_mins) + " minutes.")
score = model.evaluate(X_train, y_train, verbose = 0)
print("Training loss: %.4f" % score[0])
print("Training accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
score = model.evaluate(X_val, y_val, verbose = 0)
print("Validation loss: %.4f" % score[0])
print("Validation accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
score = model.evaluate(X_val, y_val, verbose = 0)
scores.append(score[1])
histories.append(history)
plt.plot(histories[0].history["accuracy"], color = "blue", label = "train")
plt.plot(histories[0].history["val_accuracy"], color = "red", label =␣
 ↪"validation")
plt.ylabel("Accuracy")
plt.xlabel("Num of epochs")
leg = plt.legend(loc='lower center')
plt.show()
```

```
Model: "sequential_88"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_136 (Conv2D)          (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_136 (MaxPoolin (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_137 (Conv2D)          (None, 11, 11, 16)        2320
```

```
_________________________________________________________________
max_pooling2d_137 (MaxPoolin (None, 5, 5, 16)          0
_________________________________________________________________
flatten_88 (Flatten)         (None, 400)               0
_________________________________________________________________
dense_176 (Dense)            (None, 80)                32080
_________________________________________________________________
dense_177 (Dense)            (None, 10)                810
=================================================================
Total params: 35,370
Trainable params: 35,370
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
420/420 [==============================] - 13s 31ms/step - loss: 3.6413 -
accuracy: 0.6495 - val_loss: 0.7028 - val_accuracy: 0.7501
Epoch 2/5
420/420 [==============================] - 12s 29ms/step - loss: 0.6501 -
accuracy: 0.7604 - val_loss: 0.6127 - val_accuracy: 0.7738
Epoch 3/5
420/420 [==============================] - 13s 31ms/step - loss: 0.5825 -
accuracy: 0.7854 - val_loss: 0.5633 - val_accuracy: 0.7959
Epoch 4/5
420/420 [==============================] - 15s 36ms/step - loss: 0.5478 -
accuracy: 0.7961 - val_loss: 0.5239 - val_accuracy: 0.8083
Epoch 5/5
420/420 [==============================] - 13s 30ms/step - loss: 0.5261 -
accuracy: 0.8030 - val_loss: 0.5141 - val_accuracy: 0.8124
-------------------------------
Total training time: 1.1152189135551454 minutes.
Training loss: 0.5029
Training accuracy: 81.42
-------------------------------
Validation loss: 0.5141
Validation accuracy: 81.24
-------------------------------
```



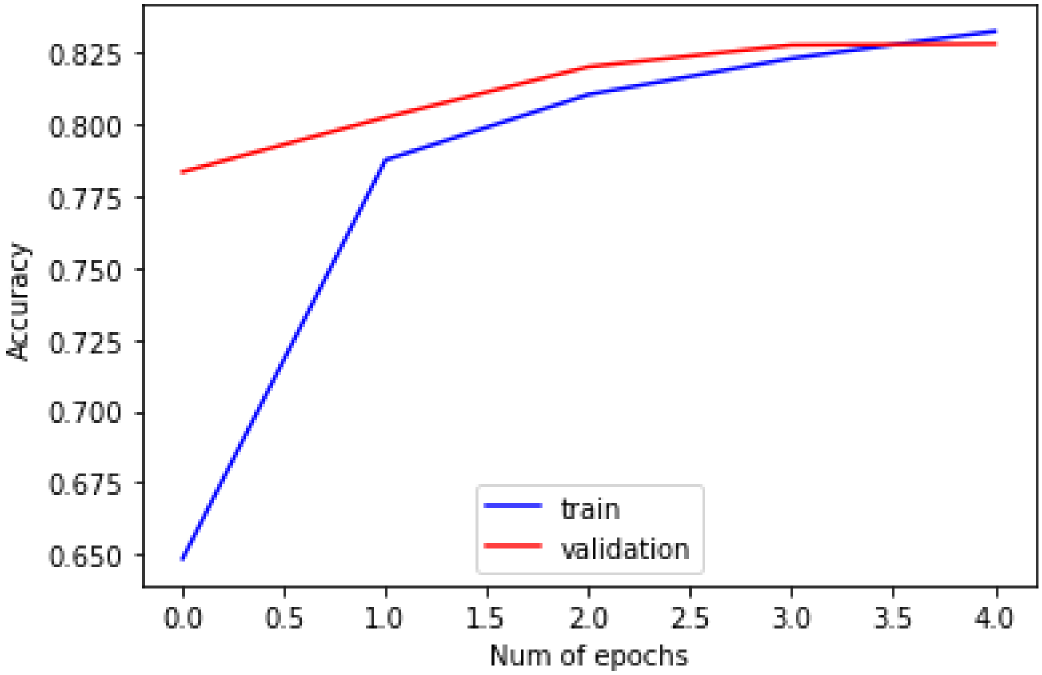
[92]:

```
#6) two convolutional layers using kernel regularization

model = Sequential()
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', input_shape =␣
 ↪(28,28,1), kernel_regularizer = regularizers.l2(0.01)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', kernel_regularizer =␣
 ↪regularizers.l2(0.01)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(80, activation = 'relu'))
model.add(Dense(10, activation = "softmax"))
model.summary()

model.compile(loss = "categorical_crossentropy", optimizer = "SGD", metrics =␣
 ↪["accuracy"])

scores = []
histories = []
start = time.time()
history = model.fit(X_train, y_train, epochs = 5, batch_size = 100,␣
 ↪validation_data= (X_val, y_val))
end = time.time()
num_mins = (end-start)/60
```

```
print("--------------------------------")
print("Total training time: "  + str(num_mins) + " minutes.")
score = model.evaluate(X_train, y_train, verbose = 0)
print("Training loss: %.4f" % score[0])
print("Training accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
score = model.evaluate(X_val, y_val, verbose = 0)
print("Validation loss: %.4f" % score[0])
print("Validation accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
scores.append(score[1])
histories.append(history)
plt.plot(histories[0].history["accuracy"], color = "blue", label = "train")
plt.plot(histories[0].history["val_accuracy"], color = "red", label =␣
 ↪"validation")
plt.ylabel("Accuracy")
plt.xlabel("Num of epochs")
leg = plt.legend(loc='lower center')
plt.show()
```

```
Model: "sequential_89"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_138 (Conv2D)          (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_138 (MaxPoolin (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_139 (Conv2D)          (None, 11, 11, 16)        2320
_________________________________________________________________
max_pooling2d_139 (MaxPoolin (None, 5, 5, 16)          0
_________________________________________________________________
flatten_89 (Flatten)         (None, 400)               0
_________________________________________________________________
dense_178 (Dense)            (None, 80)                32080
_________________________________________________________________
dense_179 (Dense)            (None, 10)                810
=================================================================
Total params: 35,370
Trainable params: 35,370
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
420/420 [==============================] - 16s 35ms/step - loss: 1.6282 -
accuracy: 0.6485 - val_loss: 0.9356 - val_accuracy: 0.7836
Epoch 2/5
420/420 [==============================] - 14s 34ms/step - loss: 0.8677 -
accuracy: 0.7877 - val_loss: 0.8074 - val_accuracy: 0.8026
```

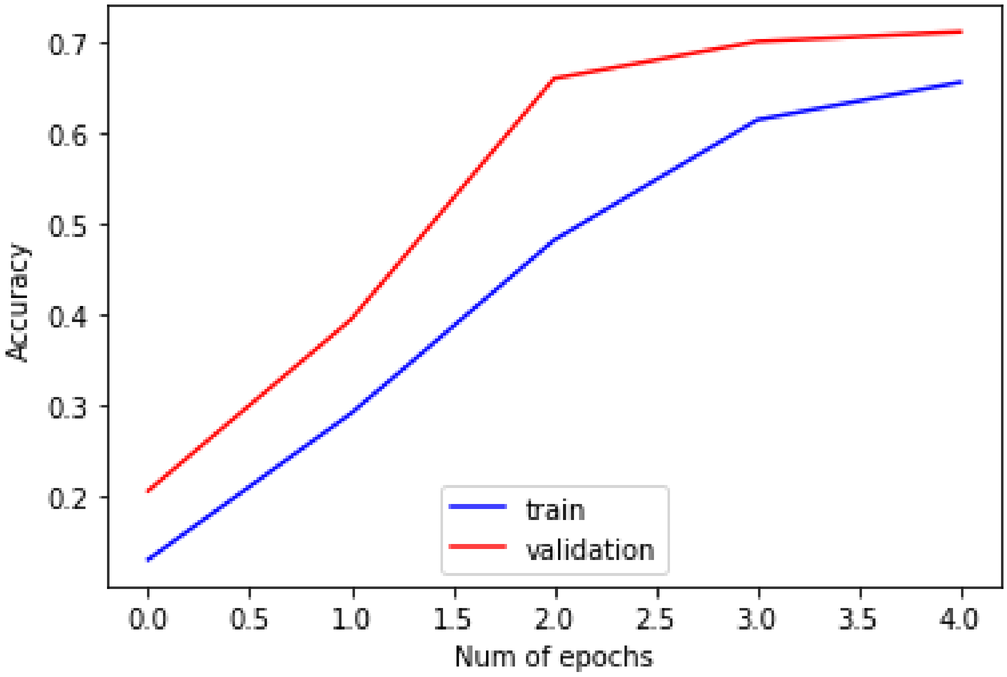
```
Epoch 3/5
420/420 [==============================] - 15s 35ms/step - loss: 0.7567 -
accuracy: 0.8105 - val_loss: 0.7263 - val_accuracy: 0.8202
Epoch 4/5
420/420 [==============================] - 16s 39ms/step - loss: 0.6794 -
accuracy: 0.8232 - val_loss: 0.6668 - val_accuracy: 0.8278
Epoch 5/5
420/420 [==============================] - 16s 37ms/step - loss: 0.6236 -
accuracy: 0.8326 - val_loss: 0.6397 - val_accuracy: 0.8282
------------------------------
Total training time: 1.2868525544802347 minutes.
Training loss: 0.6040
Training accuracy: 83.72
------------------------------
Validation loss: 0.6397
Validation accuracy: 82.82
------------------------------
```



[93]:

```
#7) two convolutional layers using dropout

model = Sequential()
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', input_shape =␣
 ↪(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.1))
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', input_shape =␣
 ↪(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.1))
model.add(Flatten())
model.add(Dense(80, activation = 'relu'))
model.add(Dropout(0.1))
model.add(Dense(10, activation = "softmax"))
model.summary()

model.compile(loss = "categorical_crossentropy", optimizer = "SGD", metrics =␣
 ↪["accuracy"])

scores = []
histories = []
start = time.time()
history = model.fit(X_train, y_train, epochs = 5, batch_size = 100,␣
 ↪validation_data= (X_val, y_val))
end = time.time()
num_mins = (end-start)/60
print("--------------------------------")
print("Total training time: "  + str(num_mins) + " minutes.")
score = model.evaluate(X_train, y_train, verbose = 0)
print("Training loss: %.4f" % score[0])
print("Training accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
score = model.evaluate(X_val, y_val, verbose = 0)
print("Validation loss: %.4f" % score[0])
print("Validation accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
scores.append(score[1])
histories.append(history)
plt.plot(histories[0].history["accuracy"], color = "blue", label = "train")
plt.plot(histories[0].history["val_accuracy"], color = "red", label =␣
 ↪"validation")
plt.ylabel("Accuracy")
plt.xlabel("Num of epochs")
leg = plt.legend(loc='lower center')
plt.show()
```

```
Model: "sequential_90"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_140 (Conv2D)          (None, 26, 26, 16)        160

_________________________________________________________________
max_pooling2d_140 (MaxPoolin (None, 13, 13, 16)        0
_________________________________________________________________
module_wrapper_43 (ModuleWra (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_141 (Conv2D)          (None, 11, 11, 16)        2320
_________________________________________________________________
max_pooling2d_141 (MaxPoolin (None, 5, 5, 16)          0
_________________________________________________________________
module_wrapper_44 (ModuleWra (None, 5, 5, 16)          0
_________________________________________________________________
flatten_90 (Flatten)         (None, 400)               0
_________________________________________________________________
dense_180 (Dense)            (None, 80)                32080
_________________________________________________________________
module_wrapper_45 (ModuleWra (None, 80)                0
_________________________________________________________________
dense_181 (Dense)            (None, 10)                810
=================================================================
Total params: 35,370
Trainable params: 35,370
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
420/420 [==============================] - 18s 41ms/step - loss: 3.7297 -
accuracy: 0.1301 - val_loss: 2.2019 - val_accuracy: 0.2059
Epoch 2/5
420/420 [==============================] - 16s 38ms/step - loss: 2.1061 -
accuracy: 0.2911 - val_loss: 1.8523 - val_accuracy: 0.3944
Epoch 3/5
420/420 [==============================] - 16s 38ms/step - loss: 1.4171 -
accuracy: 0.4820 - val_loss: 0.9859 - val_accuracy: 0.6600
Epoch 4/5
420/420 [==============================] - 16s 38ms/step - loss: 0.9912 -
accuracy: 0.6143 - val_loss: 0.8114 - val_accuracy: 0.7006
Epoch 5/5
420/420 [==============================] - 17s 40ms/step - loss: 0.8815 -
accuracy: 0.6559 - val_loss: 0.7465 - val_accuracy: 0.7112
-------------------------------
Total training time: 1.3820756196975708 minutes.
Training loss: 0.7578
Training accuracy: 70.34
-------------------------------
Validation loss: 0.7465
Validation accuracy: 71.12
-------------------------------
```



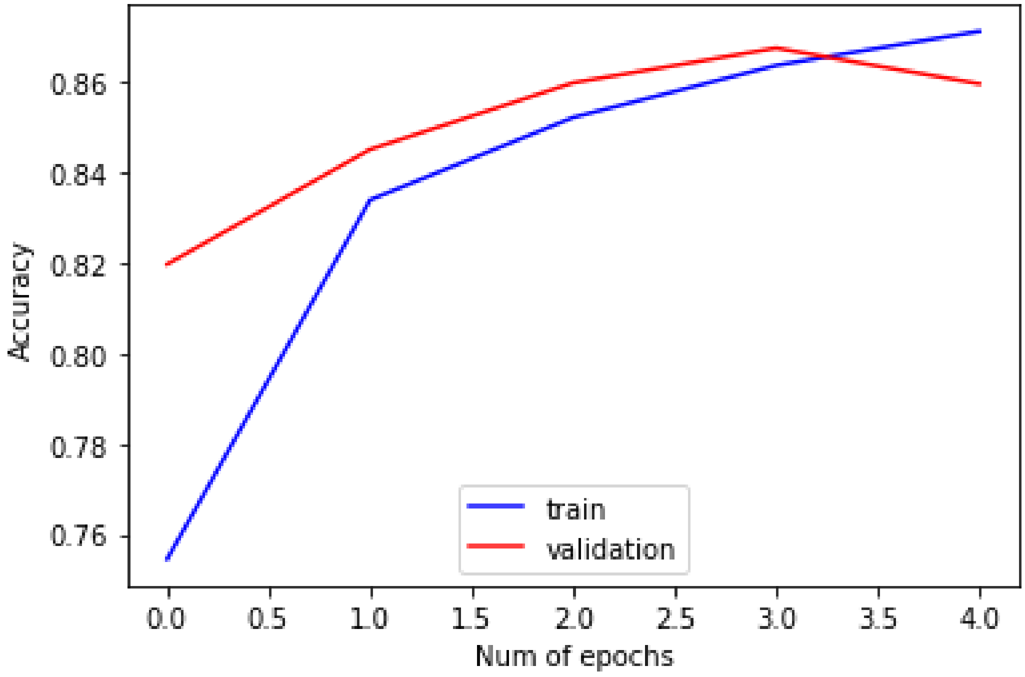
[94]:

```
#8) two convolutional layer, batch normalization

model = Sequential()
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', input_shape =␣
 ↪(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(16, kernel_size = 3, activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(80, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dense(10, activation = "softmax"))
model.summary()

model.compile(loss = "categorical_crossentropy", optimizer = "SGD", metrics =␣
 ↪["accuracy"])

scores = []
histories = []
start = time.time()
history = model.fit(X_train, y_train, epochs = 5, batch_size = 100,␣
 ↪validation_data= (X_val, y_val))
```

```
end = time.time()
num_mins = (end-start)/60
print("--------------------------------")
print("Total training time: "  + str(num_mins) + " minutes.")
score = model.evaluate(X_train, y_train, verbose = 0)
print("Training loss: %.4f" % score[0])
print("Training accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
score = model.evaluate(X_val, y_val, verbose = 0)
print("Validation loss: %.4f" % score[0])
print("Validation accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
scores.append(score[1])
histories.append(history)
plt.plot(histories[0].history["accuracy"], color = "blue", label = "train")
plt.plot(histories[0].history["val_accuracy"], color = "red", label =␣
 ↪"validation")
plt.ylabel("Accuracy")
plt.xlabel("Num of epochs")
leg = plt.legend(loc='lower center')
plt.show()
```

```
Model: "sequential_91"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_142 (Conv2D)          (None, 26, 26, 16)        160

_________________________________________________________________
max_pooling2d_142 (MaxPoolin (None, 13, 13, 16)        0

_________________________________________________________________
module_wrapper_46 (ModuleWra (None, 13, 13, 16)        64

_________________________________________________________________
conv2d_143 (Conv2D)          (None, 11, 11, 16)        2320

_________________________________________________________________
max_pooling2d_143 (MaxPoolin (None, 5, 5, 16)          0

_________________________________________________________________
module_wrapper_47 (ModuleWra (None, 5, 5, 16)          64

_________________________________________________________________
flatten_91 (Flatten)         (None, 400)               0

_________________________________________________________________
dense_182 (Dense)            (None, 80)                32080

_________________________________________________________________
module_wrapper_48 (ModuleWra (None, 80)                320

_________________________________________________________________
dense_183 (Dense)            (None, 10)                810
=================================================================
Total params: 35,818
Trainable params: 35,594
```

```
Non-trainable params: 224
_________________________________________________________________
Epoch 1/5
420/420 [==============================] - 29s 66ms/step - loss: 0.7374 -
accuracy: 0.7545 - val_loss: 0.5230 - val_accuracy: 0.8198
Epoch 2/5
420/420 [==============================] - 25s 60ms/step - loss: 0.4755 -
accuracy: 0.8340 - val_loss: 0.4464 - val_accuracy: 0.8452
Epoch 3/5
420/420 [==============================] - 24s 57ms/step - loss: 0.4170 -
accuracy: 0.8523 - val_loss: 0.4090 - val_accuracy: 0.8600
Epoch 4/5
420/420 [==============================] - 25s 59ms/step - loss: 0.3846 -
accuracy: 0.8639 - val_loss: 0.3899 - val_accuracy: 0.8677
Epoch 5/5
420/420 [==============================] - 24s 57ms/step - loss: 0.3630 -
accuracy: 0.8714 - val_loss: 0.3979 - val_accuracy: 0.8598
-------------------------------
Total training time: 2.116146699587504 minutes.
Training loss: 0.3704
Training accuracy: 86.47
-------------------------------
Validation loss: 0.3979
Validation accuracy: 85.98
-------------------------------
```



[10]:

```
#Train best model with train & validation data and measure accuracy using test␣
 ↪data.
#combine train and validation data
train_data = list(X_train)
train_labels = list(y_train)
val_data = list(X_val)
val_labels = list(y_val)
all_data = train_data+val_data
all_labels = train_labels+val_labels
new_X_train = np.array(all_data)
new_y_train = np.array(all_labels)

model = Sequential()
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', input_shape =␣
 ↪(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(80, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dense(10, activation = "softmax"))
model.summary()

model.compile(loss = "categorical_crossentropy", optimizer = "SGD", metrics =␣
 ↪["accuracy"])

scores = []
histories = []
start = time.time()
history = model.fit(new_X_train, new_y_train, epochs = 5, batch_size = 100)
end = time.time()
num_mins = (end-start)/60
print("--------------------------------")
print("Total training time: "  + str(num_mins) + " minutes.")
score = model.evaluate(X_test, y_test, verbose = 0)
print("Test accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 16)        0
_________________________________________________________________
module_wrapper_4 (ModuleWrap (None, 13, 13, 16)        64
_________________________________________________________________
flatten_2 (Flatten)          (None, 2704)              0

_________________________________________________________________
dense_4 (Dense)              (None, 80)                216400

_________________________________________________________________
module_wrapper_5 (ModuleWrap (None, 80)                320

_________________________________________________________________
dense_5 (Dense)              (None, 10)                810
=================================================================
Total params: 217,754
Trainable params: 217,562
Non-trainable params: 192

_________________________________________________________________
Epoch 1/5
510/510 [==============================] - 16s 28ms/step - loss: 0.5134 -
accuracy: 0.8281
Epoch 2/5
510/510 [==============================] - 14s 27ms/step - loss: 0.3517 -
accuracy: 0.8775
Epoch 3/5
510/510 [==============================] - 15s 29ms/step - loss: 0.3106 -
accuracy: 0.8928
Epoch 4/5
510/510 [==============================] - 14s 28ms/step - loss: 0.2831 -
accuracy: 0.9001
Epoch 5/5
510/510 [==============================] - 15s 29ms/step - loss: 0.2652 -
accuracy: 0.9071
-------------------------------
Total training time: 1.2284663478533426 minutes.
Test accuracy: 89.23
-------------------------------
```

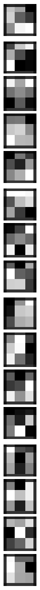
[14]:

```
import tensorflow as tf
import glob
import numpy as np
import pickle
import os

from matplotlib import pyplot as plt
from PIL import Image
from urllib import request
from io import BytesIO

from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from keras.applications.vgg16 import VGG16
```

[61]:

```python
layer = model.layers[0]
filters, bias = layer.get_weights()
print(layer.name, filters.shape)

f_min, f_max = filters.min(), filters.max()
filters = (filters-f_min) / (f_max - f_min)

n_filters = 16; ix = 1
for i in range(n_filters):
    f = filters[:, :, :, i]


    ax = plt.subplot(n_filters, 1, ix)
    ax.set_xticks([])
    ax.set_yticks([])
    plt.imshow(f[:, :], cmap = "gray")
    ix +=1

plt.show()
```

```
conv2d_3 (3, 3, 1, 16)
```



[48]:

```python
model = Sequential()
model.add(Conv2D(16, kernel_size = 3, activation = 'relu', input_shape =␣
 ↪(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(16, kernel_size = 3, activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(80, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dense(10, activation = "softmax"))
model.summary()

model.compile(loss = "categorical_crossentropy", optimizer = "SGD", metrics =␣
 ↪["accuracy"])

scores = []
histories = []
start = time.time()
history = model.fit(new_X_train, new_y_train, epochs = 5, batch_size = 100)
end = time.time()
num_mins = (end-start)/60
print("--------------------------------")
print("Total training time: "  + str(num_mins) + " minutes.")
score = model.evaluate(X_test, y_test, verbose = 0)
print("Test accuracy: %.2f" % (score[1] * 100.0))
print("--------------------------------")
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 16)        0
_________________________________________________________________
module_wrapper_6 (ModuleWrap (None, 13, 13, 16)        64
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 16)        2320
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
module_wrapper_7 (ModuleWrap (None, 5, 5, 16)          64
_________________________________________________________________
flatten_3 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_6 (Dense)              (None, 80)                32080
_________________________________________________________________
module_wrapper_8 (ModuleWrap (None, 80)                320
_________________________________________________________________
dense_7 (Dense)              (None, 10)                810
=================================================================
Total params: 35,818
Trainable params: 35,594
Non-trainable params: 224
_________________________________________________________________
Epoch 1/5
510/510 [==============================] - 27s 51ms/step - loss: 0.6815 -
accuracy: 0.7699
Epoch 2/5
510/510 [==============================] - 22s 43ms/step - loss: 0.4425 -
accuracy: 0.8439
Epoch 3/5
510/510 [==============================] - 20s 40ms/step - loss: 0.3895 -
accuracy: 0.8616
Epoch 4/5
510/510 [==============================] - 20s 40ms/step - loss: 0.3602 -
accuracy: 0.8713
Epoch 5/5
510/510 [==============================] - 20s 40ms/step - loss: 0.3402 -
accuracy: 0.8795
-------------------------------
Total training time: 1.8317898988723755 minutes.
Test accuracy: 86.64
-------------------------------
```

[72]:

```
layer = model.layers[0]
filters, bias = layer.get_weights()
print(layer.name, filters.shape)


n_filters = 16; ix = 1
for i in range(n_filters):
    f = filters[:, :, :, i]


    ax = plt.subplot(n_filters, 1, ix)
    ax.set_xticks([])
    ax.set_yticks([])
    plt.imshow(f[:, :], cmap = "gray")
    ix +=1

plt.show()

layer = model.layers[0]
filters, bias = layer.get_weights()


layer = model.layers[3]
```

```
filters, bias = layer.get_weights()
print(layer.name, filters.shape)

n_filters = 16; ix = 1
for i in range(n_filters):
    f = filters[:, :, :, i]

  #plot each channel
    for j in range(16):
            ax = plt.subplot(n_filters, 16, ix)
            ax.set_xticks([])
            ax.set_yticks([])
            plt.imshow(f[:, :, j], cmap = "gray")
            ix +=1

plt.show()
```

```
conv2d_3 (3, 3, 1, 16)
```

```
conv2d_4 (3, 3, 16, 16)
```

[ ]: