Influence of Regularization

I. Methods

Dataset Information:

The dataset “Fashion MNIST” was used to conduct the experiments. It’s a dataset of Zalando’s article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Every example is a 28x28 grayscale image with the labels for 10 different classes of different clothing categories. The model was downloaded directly from the TensorFlow library but the detailed description of the dataset is available on GitHub (https://github.com/zalandoresearch/fashion-mnist).

The labels include the following items:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Dataset Splitting:

The model was already splitted into training and testing set in the ratio: 85/15. To separate the validation set from the training data, I used the validation_split function and executed it on the training data to hold another 15% from the 85% of the training data and the ratio of the split in the end is: 70/15/15 for all the experiments.

Neural Network Architecture:

The hyperparameters for the model were selected once and set for all the experiments including the top-performing final model.

The hyperparameters used are:

```
num_iter = 2000
opt = 'adam'
num_filters = 32
kernel_size=3
pool_size = (2,2)
strides= (2,2)
activation = 'relu'
padding = 'SAME'
loss = "categorical_crossentropy"
epochs = 10
batch_size = 128
```

Number of iterations chosen for all the models was 2000. The optimization techniques used for the model was Adam. The results of the models optimized with Adam techniques are generally better than other chosen algorithms. It's known also for faster computation and requires fewer parameters for tuning. In machine learning models, it is a well known technique and it's often set as a default optimizer. Number of filters was chosen to be 32. Since I dealt with the image classification problem, there are a lot of features to learn by the model. The bigger number of filters lets the model capture more detail from the first more general ones to the more detailed and subtle features like the actual textures of the clothes, compared to the general features of the picture like colors and edges in the first layers. The more complex pictures get, the larger combinations of patterns are there to capture. The size of the filters, so the kernel size, was set to 3 to be able to move through the filters with the width and height of 3. Every pooling layer used in the experiments had a size of 2x2 and all the layers were strided by 2x2 layers. It means that the layer uses a filter of 2x2 and slides it over the entire image. Every time it stops (by 2 'spots'), the filter makes the calculations of the matrix. The activation function used for all the layers (except the final layer where 'softmax' is used for the output calculations) was 'relu' function. I chose the non-linear function that is a pretty popular choice in ML architectures. One of the biggest advantages of using the ReLU function is that it does not activate all the

units in the layers at the same time so it's computationally more efficient. The neurons are activated only if the output is more than 0. This way the model also saves time. The padding technique was not used in the model architecture. Padding parameter is set to 'SAME' which means that the filter is applied to all the elements of the input. It results in more time to train the model but the image examples in inputs get fully covered by the filters in the stride 2x2. I chose the categorical cross-entropy function as the loss function because it's a pretty common choice for multi-classification problems. The success of this loss function for the multi-classification process lies in the fact that one example can be only considered to belong to one of the 10 categories labeled in the model and should assign the probability of 0 to all the other classes. Number of epochs was set to 10 which means that the model completes 10 passes through the training set. The batch size used for all the experiments was set to 128. It means that the number of samples processed before the model updates is 128. Choosing the batch size that is power of 2 helps GPU Amake the process smoother and faster because of the way memory is set up in GPUs in general.

In the experiment phase, there were 8 distinct models trained on the same training set (70%) and tested on the validation set extracted from the validation_split function available in Keras module. The models are set up in the following manner:

Model 1: no regularization / 3 conv layers
Model 2: no regularization / 1 conv layer
Model 3: L2 regularization / 3 conv layers
Model 4: L2 regularization / 1 conv layer
Model 5: Dropout / 3 conv layers
Model 6: Dropout / 1 conv layer
Model 7: Early Stopping / 3 conv layers
Model 8: Early Stopping / 1 conv layer

The purpose of the experiments was to check how the number of convolutional layers affect the performance of the model and how different regularization techniques do on different models. One thing I was very interested in was how the models with only one convolutional layer predict the results. I did not set up the high number of convolutional layers to capture if increasing the number of convolutional layers of only 2 will have an impact on the performance. The experiments with the regularizations were set up to first check if the model without any regularization can have a similar performance to the ones where regularization was used. The first 2 models were treated like a sort of control group to all the other experimental groups with regularization techniques.

Hardware Information:

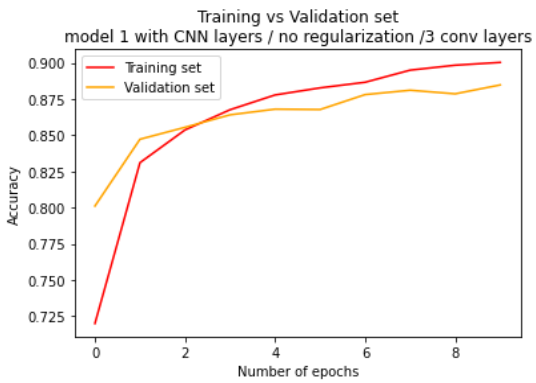
Processor: AMD Ryzen 5 2500U with Radeon Vega Mobile Gfx 2.00 GHz
Installed RAM: 8.00 GB (6.96 GB usable)
System Type: 64-bit operating system, x64-based processor

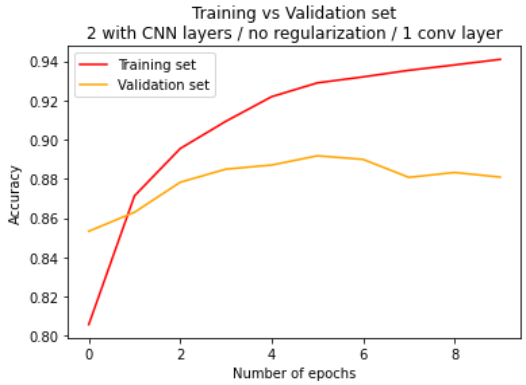
II. Results

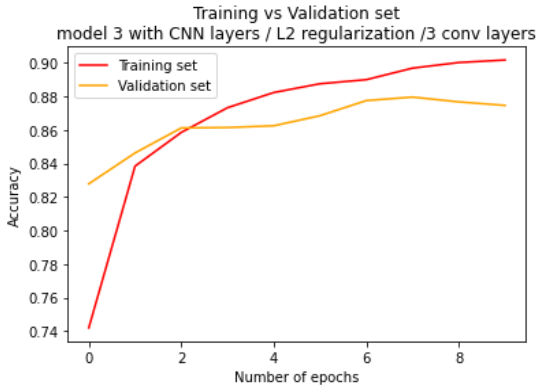
Results of Experiments:

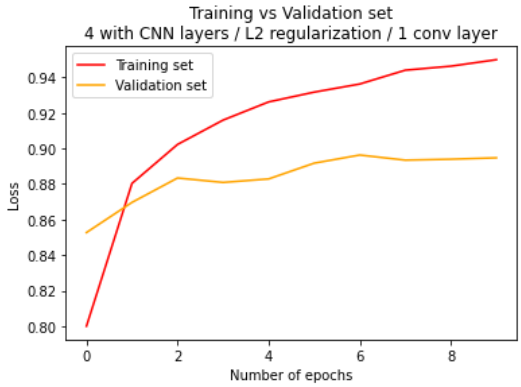
	# conv layers	Regularization	Train time (min)	Val accuracy	Test accuracy	Test loss
Model 1	3	-	5.03	0.885	-	-
Model 2	1	-	4.4	0.881	-	-
Model 3	3	L2	5.4	0.875	-	-
Model 4	1	L2	4.4	0.895	-	-
Model 5	3	Dropout	4.8	0.886	-	-
Model 6	1	Dropout	4.4	0.885	-	-
Model 7	3	Early Stopping	4.4	0.878	-	-
Model 8	1	Early Stopping	3.4	0.879	-	-
Top_model	1	L2	5.4	-	0.95	0.39

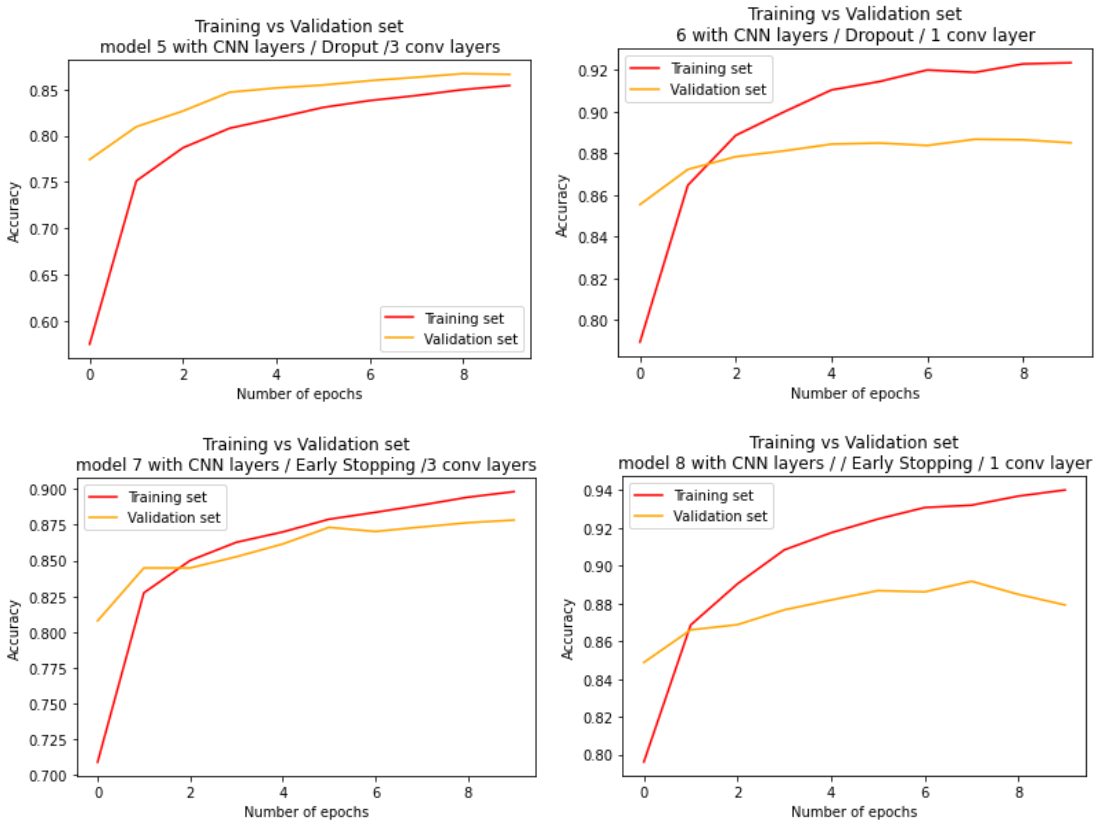
Learning Curves:











III. Analysis

Trends Observed:

One of the trends that was surprising is the fact that the number of convolutional layers does not consistently improve the performance of the models. In the case with no regularization, the model improves but only by 0.004 in the accuracy by adding 2 more convolutional layers. However, in the case on model 3 and 4 where L2 was used, the model with 1 convolutional layer did better than the one with 3 convolutional layers. The model 4 did the best overall on the validation test and is the top-performing model to be used later in the experiments. When comparing model 5 and 6 (dropout), the accuracy was only 0.001 percent higher by adding 2 more convolutional layers. Though, in the case of the model 7 and 8 (early stopping), adding 2 convolutional layers decreased the performance by 0.001. Perhaps, the difference between convolutional layers number of 3 and 1 is not enough to give the model a significant improvement (over 0.5%). I suspect that additional experiments with the bigger number of convolutional layers (5-10) would increase the model performance. The difference between model 4 and 5 is 0.02 which is the biggest difference when comparing the variations in the number of convolutional layers. Surprisingly, the model with 1 convolutional layer did better.

Another trend that can be observed from the results is that in all cases the validation accuracy was in the end smaller than the training accuracy, except one case, Model #5 with dropout and 3 convolutional layers did better on the validation set. This

happened only once in all the experiment and it might be caused by the fact that the dropout technique was used. The model at validation time is more robust because during the training process a part of the training set is dropped out. Additionally, dropout gives more generalization to the model but it can also be an imbalance of the data. One of the possible reasons why this model has a higher accuracy on the validation set is also that the dataset split was not done randomly. And hence the validation set may be completely different, it might have less noise or less variance from the training set. This may lead to the fact that it's actually easier to predict the accuracy. Additionally, if the validation set is small compared to the training set, the model will fit better the validation set rather than the training set. It refers back to the fact that the model used dropout if shrunk the number of examples in the data. Another conclusion from this observation is that dropout penalizes model variance by randomly freezing neurons in a layer during model training. Unfortunately, this might also mean that the dropout causes model underfitting and more data is needed to perform the experiment in the correct manner.

In the cases of models: 2, 3 and 8, the accuracy curves start going lower on the validation set at the end of the training. It might be the sign that the model starts overfitting the data. In model 1 there was no regularization technique used and I suspect this is the reason for the training curve to be smooth and performing well while the validation curve starts going down. The model is not robust enough because it also has only 1 convolutional layer. The fact that there was no regularization used, does not help to balance and normalize the data so it learns successfully the examples in the training data and it fails on the unseen examples from the validation set. Since the number of convolutional layers is low, the model doesn't have time and enough space to learn complex features and generalize well. Additionally, since the data might be noisy, it's learning the noise and starts predicting it well in the train set. Again, when predicting on never seen new examples, it fails.

Another trend I can observe from the learning curves is that the difference between the validation and training accuracy is smaller for all the models with 3 convolutional layers versus the models with 1 convolutional layer. This observation proves that more experiments with a higher number of convolutional layers may improve the general performance. Adding more layers helps extract more features and since I dealt with image classification, pictures require a robust number of features to be learnt to predict the image successfully. However, adding the layers to the architecture would increase the time needed for training and computational resources necessary to train the model. There is a trade off between the size of the architecture and the time and computational efficiency that would need to be considered.

The training time observed in the experiments was not very surprising. Model 1 took the longest ro train. This model did not use any regularization and used 3 convolutional layers. It confirms that the size of the model influences the training time, the bigger the model, the longer it trains. The lack of regularization techniques made it more difficult for the model to extract the features. Model 8 took the least amount of time. It used early stopping and only 1 convolutional layer. The use of early stopping can be the reason for a shorter training time since it stops the model at some point when performance starts decreasing. The overall trend observed from the training time is that

the bigger model size results in the longer training time. Every model configuration, with respect to the regularization usage, had a shorter training time when the architecture included 1 convolutional layer, instead of 3.

The top-performing model has an increase of the training time (1 minute increase) while training on the full training + validation sets and test on the 15% of the test set. The accuracy on the test set compared to the accuracy of the validation set was also significantly higher (0.895 vs 0.95 on validation and test set respectively). One of the reasons can be extending the number of training examples by adding the validation set to the training process in the top-performing model and testing on a small 15% only, compared to 85% of training and validation set. The model can generalize better when more data is fed to the model so it learns the features better. This fact could also help in balancing the data. By adding more images, the model learnt more diverse representations of different classes and generalized better. The validation set was used only to tune parameters.

Interpreting CNN Representations

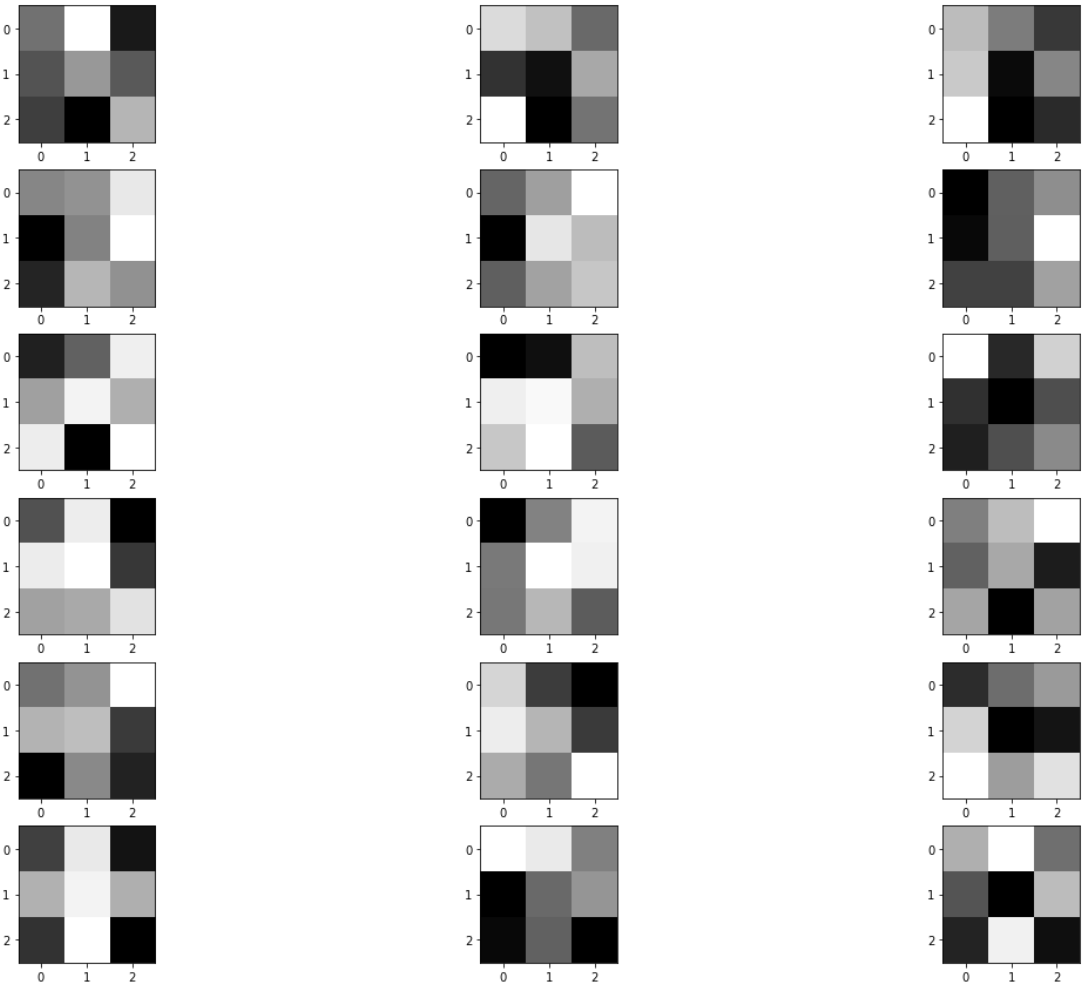
I. Methods

Model Architecture:

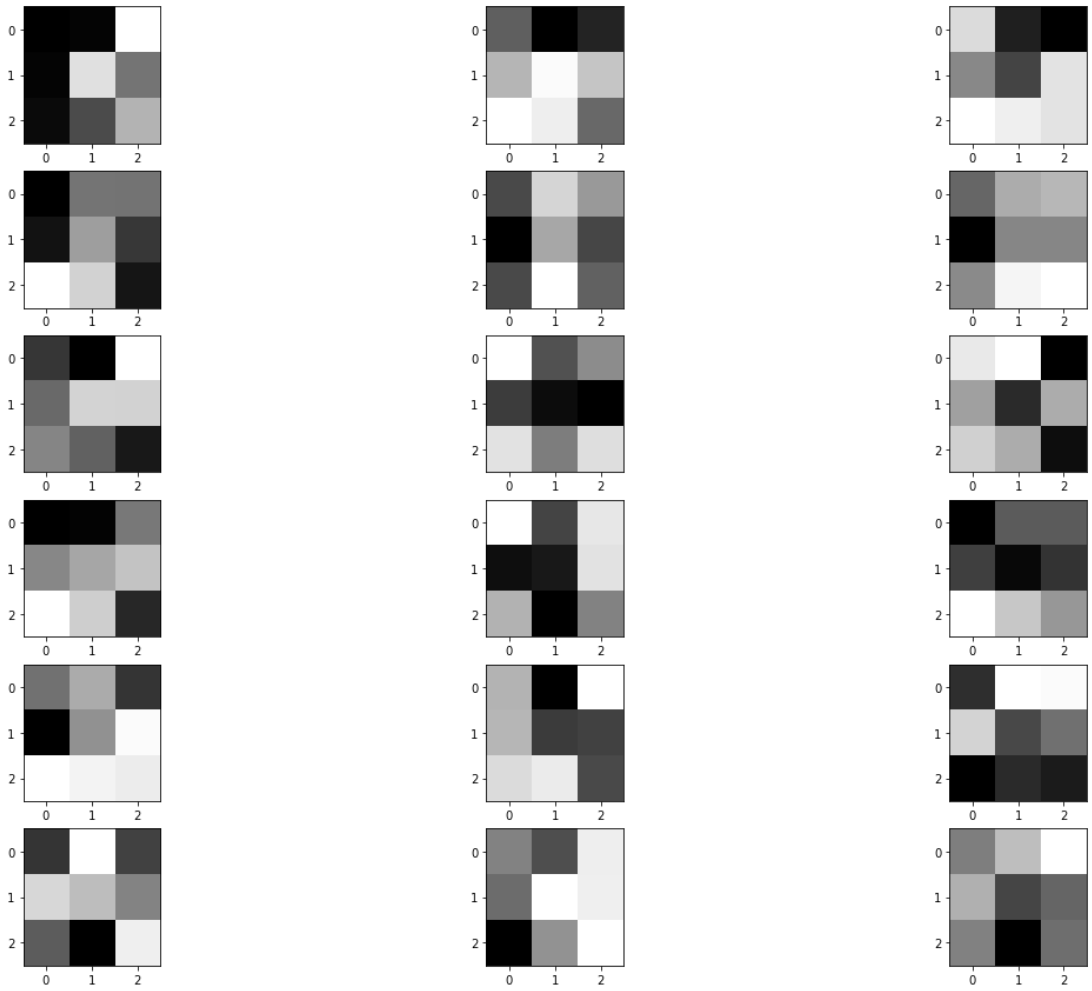
Since the best performing model from the previous 8 experiments had 1 convolutional layer, I built a new model with extra convolutional layers to be able to compare the results from multiple layers. The visualization model includes 4 convolutional layers and this is the only difference between the top-performing model from the previous part of the research. 3 more convolutional layers were added. All other hyperparameters selected and features of the model stayed the same.

II. Results

Visualization of 1st Convolutional Layer:



Visualization of 4th Convolutional Layer:



III. Analysis

Trends Observed:

First observed trend from the visualization of the convolutional layers is that the first channels of both layers are more concentrated at the edges of the image and as we're getting to the second and third channel, the point of focus is switched to the center of the image. This might mean that the channels subsequently over-first feature the pictures at the edges to detect the general trends of the image and then focus on more details that are mostly at the center of an image.

The second trend that can be observed is comparing the first and the fourth convolutional layer. It's clear that the dark squares on the fourth convolutional layer are more concentrated together. It means that the model in the further layers is looking for more condensed patterns. For example, 3 squares next to each other have a similar color on the greyscale. It means that this space on the picture is forming some kind of a

pattern, maybe a single line or it's a part of a bigger object. In the first layer of convolution the model recognizes these parts of one object only separately but does not put them together into bigger pictures.

The overall conclusion is that the model is not able to capture more complex features at the beginning of the training and each layer is responsible for some kind of different feature extraction. The deeper the model gets, the more complex features are recognized. This is why the visualizations show more condensed and more grouped together patterns on the further layers.

Influence of Regularization / Code Part I

In [ ]:
```
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import keras
from tensorflow.keras.utils import to_categorical
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
import time
from tensorflow.python.keras.layers import *
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from keras import regularizers
```

In [ ]:
```
# Enable GPU: "Runtime"-->"Change Runtime"-->"Hardware Accelerator"
#Check if GPU is enabled
tf.test.gpu_device_name()
```

Out[ ]:
```
''
```

In [ ]:
```
#2. Import dataset
data = tf.keras.datasets.fashion_mnist

(x_train, y_train), (x_test, y_test) = data.load_data()
#assert x_rem.shape == (60000, 28, 28)
assert x_test.shape == (10000, 28, 28)
#assert y_rem.shape == (60000,)
assert y_test.shape == (10000,)
#x_train, x_valid, y_train, y_valid = train_test_split(x_rem, y_rem, test_size=0.15)
assert x_train.shape == (60000, 28, 28)
#assert x_valid.shape == (9000, 28, 28)
assert y_train.shape == (60000,)
#assert y_valid.shape == (9000,)
```

In [ ]:
```
#Data pre-processing

# reshape data to fit the model
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_valid = x_valid.reshape(x_valid.shape[0], 28, 28, 1)

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
#y_valid = to_categorical(y_valid)

# Inspect what the one-hot encoding looks like for the first value
y_train[0]
```

Out[ ]:
```
array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.], dtype=float32)
```

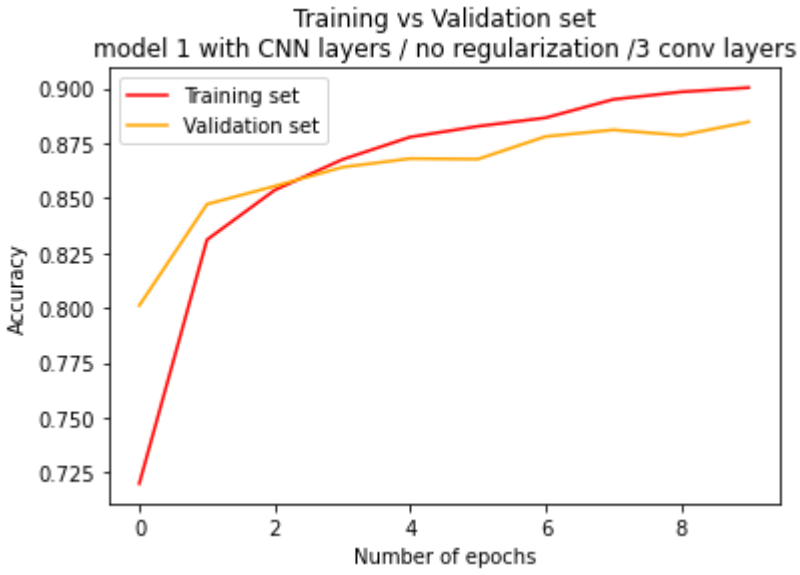
EXPERIMENTS

In [ ]:
```
from tensorflow.python.ops.gen_batch_ops import batch
# Set all the hyperparameters to the same for each model

num_iter = 2000
opt = 'adam'
num_filters = 32
kernel_size=3
pool_size = (2,2)
strides= (2,2)
activation = 'relu'
padding = 'SAME'
loss = "categorical_crossentropy"
epochs = 10
batch_size = 128
```

In [ ]:
```
# Create model 1 with CNN layers / no regularization /3 conv layers
title = 'model 1 with CNN layers / no regularization /3 conv layers'

model = Sequential()

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size= pool_size))

# Add a second conv layer with a stride of 2x2
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# Add a third conv layer with a stride of 2x2
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# Flatten the input
model.add(Flatten())

# Regular FC layer with output size 10 (for the 10 digits)
model.add(Dense(10, activation = "softmax"))

# Print the summary of the model to view the shape and number of parameters
model.summary()



# Specify the optimizer
model.compile(optimizer = opt, loss = loss, metrics = ['accuracy'])



# Time how fast the model train
start = time.time()

# Train using adam
model1 = model.fit(x_train, y_train, validation_split=0.15,batch_size = batch_size, ep
```

```
end = time.time()

num_mins = (end-start)/60
print("Total training time: "  + str(num_mins) + " minutes.")




# Evaluate the model
score = model.evaluate(x_test, y_test, verbose = 0)
print("Test loss: %.4f" % score[0])
print("Accuracy: %.2f" % (score[1] * 100.0))




# Plot accuracy (val vs test)
from matplotlib import pyplot as plt

plt.plot(model1.history["accuracy"], color = "red", label = "Training set")
plt.plot(model1.history["val_accuracy"], color = "orange", label = "Validation set")

plt.title("Training vs Validation set\n"+title)
plt.ylabel("Accuracy")
plt.xlabel("Number of epochs")
plt.legend()
plt.show()
```

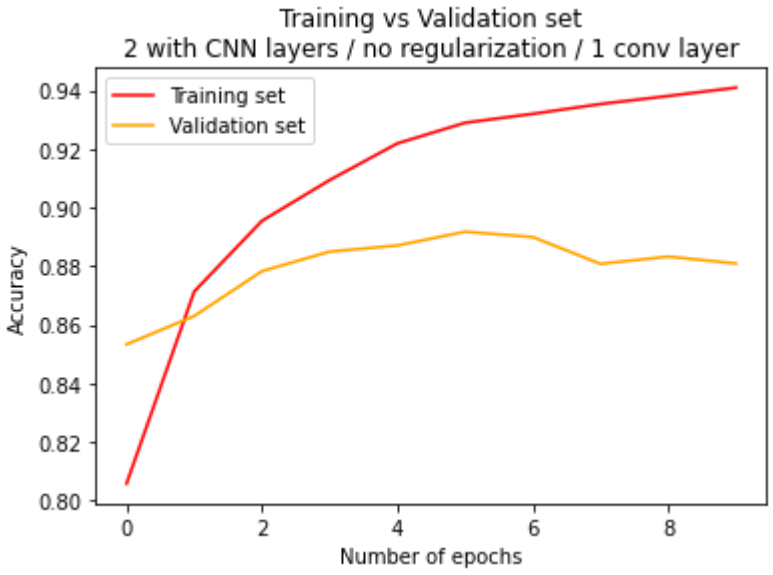
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 7, 7, 32)          9248
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 3, 3, 32)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 2, 2, 32)          9248
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                330
=================================================================
Total params: 19,146
Trainable params: 19,146
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
339/339 [==============================] - 26s 75ms/step - loss: 1.0358 - accuracy:
0.7199 - val_loss: 0.5741 - val_accuracy: 0.8010
Epoch 2/10
339/339 [==============================] - 34s 100ms/step - loss: 0.4844 - accuracy:
0.8310 - val_loss: 0.4556 - val_accuracy: 0.8472
Epoch 3/10
339/339 [==============================] - 37s 109ms/step - loss: 0.4129 - accuracy:
0.8536 - val_loss: 0.4143 - val_accuracy: 0.8554
Epoch 4/10
339/339 [==============================] - 44s 129ms/step - loss: 0.3715 - accuracy:
0.8676 - val_loss: 0.4050 - val_accuracy: 0.8641
Epoch 5/10
339/339 [==============================] - 26s 76ms/step - loss: 0.3441 - accuracy:
0.8778 - val_loss: 0.3880 - val_accuracy: 0.8680
Epoch 6/10
339/339 [==============================] - 25s 73ms/step - loss: 0.3240 - accuracy:
0.8827 - val_loss: 0.3825 - val_accuracy: 0.8677
Epoch 7/10
339/339 [==============================] - 25s 73ms/step - loss: 0.3116 - accuracy:
0.8866 - val_loss: 0.3555 - val_accuracy: 0.8780
Epoch 8/10
339/339 [==============================] - 26s 78ms/step - loss: 0.2907 - accuracy:
0.8949 - val_loss: 0.3540 - val_accuracy: 0.8810
Epoch 9/10
339/339 [==============================] - 27s 79ms/step - loss: 0.2796 - accuracy:
0.8983 - val_loss: 0.3509 - val_accuracy: 0.8786
Epoch 10/10
339/339 [==============================] - 33s 99ms/step - loss: 0.2696 - accuracy:
0.9003 - val_loss: 0.3479 - val_accuracy: 0.8847
Total training time: 5.033408737182617 minutes.
Test loss: 0.3669
Accuracy: 87.84
```



In [ ]:
```
# Create model 2 with CNN layers / no regularization / 1 conv layer

title = '2 with CNN layers / no regularization / 1 conv layer'

model = Sequential()

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# Flatten the input
model.add(Flatten())

# Regular FC layer with output size 10 (for the 10 digits)
model.add(Dense(10, activation = "softmax"))

# Print the summary of the model to view the shape and number of parameters
model.summary()



# Specify the optimizer
model.compile(optimizer = opt, loss = loss, metrics = ['accuracy'])



# Time how fast the model train
start = time.time()

# Train using adam
model1 = model.fit(x_train, y_train, validation_split=0.15,batch_size = batch_size, ep

end = time.time()

num_mins = (end-start)/60
print("Total training time: "  + str(num_mins) + " minutes.")
```

```
# Evaluate the model
#score = model.evaluate(x_test, y_test, verbose = 0)
#print("Test loss: %.4f" % score[0])
#print("Accuracy: %.2f" % (score[1] * 100.0))



# Plot loss function (val vs test)
from matplotlib import pyplot as plt

plt.plot(model1.history["accuracy"], color = "red", label = "Training set")
plt.plot(model1.history["val_accuracy"], color = "orange", label = "Validation set")

plt.title("Training vs Validation set\n"+title)
plt.ylabel("Accuracy")
plt.xlabel("Number of epochs")
plt.legend()
plt.show()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                62730
=================================================================
Total params: 63,050
Trainable params: 63,050
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
339/339 [==============================] - 21s 61ms/step - loss: 2.7215 - accuracy:
0.8058 - val_loss: 0.7411 - val_accuracy: 0.8533
Epoch 2/10
339/339 [==============================] - 25s 74ms/step - loss: 0.5239 - accuracy:
0.8714 - val_loss: 0.5791 - val_accuracy: 0.8630
Epoch 3/10
339/339 [==============================] - 21s 62ms/step - loss: 0.3330 - accuracy:
0.8955 - val_loss: 0.4412 - val_accuracy: 0.8783
Epoch 4/10
339/339 [==============================] - 24s 70ms/step - loss: 0.2638 - accuracy:
0.9094 - val_loss: 0.4021 - val_accuracy: 0.8850
Epoch 5/10
339/339 [==============================] - 24s 71ms/step - loss: 0.2213 - accuracy:
0.9219 - val_loss: 0.3911 - val_accuracy: 0.8871
Epoch 6/10
339/339 [==============================] - 29s 86ms/step - loss: 0.2028 - accuracy:
0.9289 - val_loss: 0.3992 - val_accuracy: 0.8918
Epoch 7/10
339/339 [==============================] - 20s 59ms/step - loss: 0.1877 - accuracy:
0.9319 - val_loss: 0.3853 - val_accuracy: 0.8899
Epoch 8/10
339/339 [==============================] - 20s 59ms/step - loss: 0.1763 - accuracy:
0.9353 - val_loss: 0.4459 - val_accuracy: 0.8808
Epoch 9/10
339/339 [==============================] - 20s 59ms/step - loss: 0.1715 - accuracy:
0.9380 - val_loss: 0.4318 - val_accuracy: 0.8833
Epoch 10/10
339/339 [==============================] - 20s 58ms/step - loss: 0.1638 - accuracy:
0.9409 - val_loss: 0.4402 - val_accuracy: 0.8809
Total training time: 4.377323408921559 minutes.
```

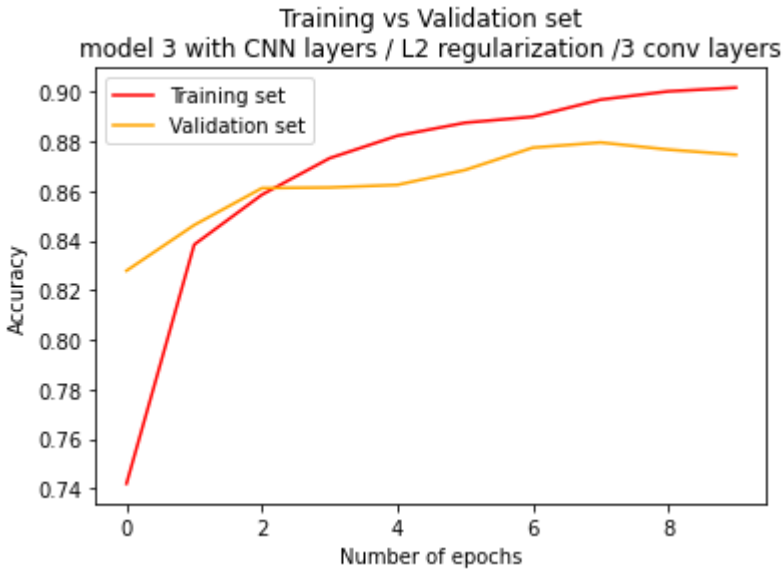


In [ ]:

```
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from keras import regularizers

# Create model 3 with CNN layers / L2 regularization /3 conv layers
title = 'model 3 with CNN layers / L2 regularization /3 conv layers'

model = Sequential()

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size= pool_size))

# Add a second conv layer with a stride of 2x2
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# Add a third conv layer with a stride of 2x2
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# Flatten the input
model.add(Flatten())

# Add L2 metod
model.add(Dense(100, activation = activation, kernel_regularizer=regularizers.l2(0.000

# Regular FC layer with output size 10 (for the 10 digits)
model.add(Dense(10, activation = "softmax"))

# Print the summary of the model to view the shape and number of parameters
model.summary()
```

```
# Specify the optimizer
model.compile(optimizer = opt, loss = loss, metrics = ['accuracy'])



# Time how fast the model train
start = time.time()

# Train using adam
model1 = model.fit(x_train, y_train, validation_split=0.15,batch_size = batch_size, ep

end = time.time()

num_mins = (end-start)/60
print("Total training time: "  + str(num_mins) + " minutes.")




# Evaluate the model
#score = model.evaluate(x_test, y_test, verbose = 0)
#print("Test loss: %.4f" % score[0])
#print("Accuracy: %.2f" % (score[1] * 100.0))




# Plot loss function (val vs test)
from matplotlib import pyplot as plt

plt.plot(model1.history["accuracy"], color = "red", label = "Training set")
plt.plot(model1.history["val_accuracy"], color = "orange", label = "Validation set")

plt.title("Training vs Validation set\n"+title)
plt.ylabel("Accuracy")
plt.xlabel("Number of epochs")
plt.legend()
plt.show()
```

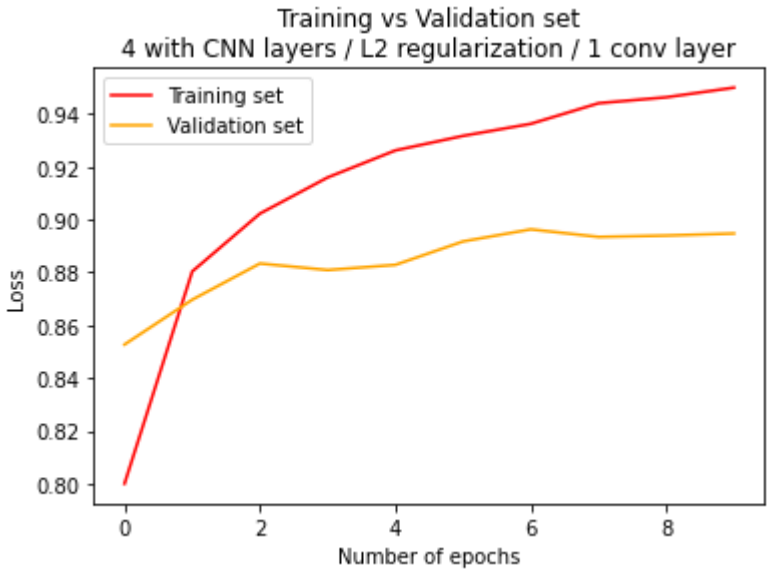
```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 7, 7, 32)          9248
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 3, 3, 32)          0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 2, 2, 32)          9248
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_6 (Dense)              (None, 100)               3300
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1010
=================================================================
Total params: 23,126
Trainable params: 23,126
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
339/339 [==============================] - 33s 95ms/step - loss: 0.7773 - accuracy:
0.7419 - val_loss: 0.4916 - val_accuracy: 0.8278
Epoch 2/10
339/339 [==============================] - 35s 103ms/step - loss: 0.4438 - accuracy:
0.8384 - val_loss: 0.4320 - val_accuracy: 0.8463
Epoch 3/10
339/339 [==============================] - 30s 90ms/step - loss: 0.3899 - accuracy:
0.8585 - val_loss: 0.4034 - val_accuracy: 0.8612
Epoch 4/10
339/339 [==============================] - 27s 79ms/step - loss: 0.3516 - accuracy:
0.8733 - val_loss: 0.4043 - val_accuracy: 0.8614
Epoch 5/10
339/339 [==============================] - 26s 78ms/step - loss: 0.3257 - accuracy:
0.8823 - val_loss: 0.3916 - val_accuracy: 0.8625
Epoch 6/10
339/339 [==============================] - 26s 75ms/step - loss: 0.3095 - accuracy:
0.8876 - val_loss: 0.3787 - val_accuracy: 0.8685
Epoch 7/10
339/339 [==============================] - 25s 75ms/step - loss: 0.2976 - accuracy:
0.8900 - val_loss: 0.3535 - val_accuracy: 0.8775
Epoch 8/10
339/339 [==============================] - 25s 75ms/step - loss: 0.2807 - accuracy:
0.8969 - val_loss: 0.3550 - val_accuracy: 0.8796
Epoch 9/10
339/339 [==============================] - 27s 80ms/step - loss: 0.2702 - accuracy:
0.9002 - val_loss: 0.3546 - val_accuracy: 0.8767
Epoch 10/10
339/339 [==============================] - 25s 74ms/step - loss: 0.2602 - accuracy:
0.9017 - val_loss: 0.3679 - val_accuracy: 0.8746
Total training time: 5.384840627511342 minutes.
```



In [ ]:
```
# Create model 4 with CNN layers / L2 regularization / 1 conv layer

title = '4 with CNN layers / L2 regularization / 1 conv layer'

model = Sequential()

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# Flatten the input
model.add(Flatten())

# Add L2 metod
model.add(Dense(100, activation = activation, kernel_regularizer=regularizers.l2(0.000

# Regular FC layer with output size 10 (for the 10 digits)
model.add(Dense(10, activation = "softmax"))

# Print the summary of the model to view the shape and number of parameters
model.summary()



# Specify the optimizer
model.compile(optimizer = opt, loss = loss, metrics = ['accuracy'])



# Time how fast the model train
start = time.time()

# Train using adam
model1 = model.fit(x_train, y_train, validation_split=0.15,batch_size = batch_size, ep

end = time.time()

num_mins = (end-start)/60
print("Total training time: "  + str(num_mins) + " minutes.")
```

```
# Evaluate the model
#score = model.evaluate(x_test, y_test, verbose = 0)
#print("Test loss: %.4f" % score[0])
#print("Accuracy: %.2f" % (score[1] * 100.0))



# Plot loss function (val vs test)
from matplotlib import pyplot as plt

plt.plot(model1.history["accuracy"], color = "red", label = "Training set")
plt.plot(model1.history["val_accuracy"], color = "orange", label = "Validation set")

plt.title("Training vs Validation set\n"+title)
plt.ylabel("Loss")
plt.xlabel("Number of epochs")
plt.legend()
plt.show()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
flatten_7 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_8 (Dense)              (None, 100)               627300
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1010
=================================================================
Total params: 628,630
Trainable params: 628,630
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
339/339 [==============================] - 25s 73ms/step - loss: 2.0214 - accuracy:
0.8000 - val_loss: 0.4934 - val_accuracy: 0.8527
Epoch 2/10
339/339 [==============================] - 24s 72ms/step - loss: 0.3656 - accuracy:
0.8804 - val_loss: 0.3947 - val_accuracy: 0.8697
Epoch 3/10
339/339 [==============================] - 24s 72ms/step - loss: 0.2847 - accuracy:
0.9023 - val_loss: 0.3697 - val_accuracy: 0.8834
Epoch 4/10
339/339 [==============================] - 24s 72ms/step - loss: 0.2411 - accuracy:
0.9160 - val_loss: 0.3753 - val_accuracy: 0.8809
Epoch 5/10
339/339 [==============================] - 24s 71ms/step - loss: 0.2086 - accuracy:
0.9263 - val_loss: 0.3771 - val_accuracy: 0.8829
Epoch 6/10
339/339 [==============================] - 26s 75ms/step - loss: 0.1914 - accuracy:
0.9318 - val_loss: 0.3487 - val_accuracy: 0.8918
Epoch 7/10
339/339 [==============================] - 24s 72ms/step - loss: 0.1746 - accuracy:
0.9363 - val_loss: 0.3496 - val_accuracy: 0.8963
Epoch 8/10
339/339 [==============================] - 25s 73ms/step - loss: 0.1578 - accuracy:
0.9441 - val_loss: 0.3585 - val_accuracy: 0.8935
Epoch 9/10
339/339 [==============================] - 25s 73ms/step - loss: 0.1454 - accuracy:
0.9464 - val_loss: 0.3798 - val_accuracy: 0.8940
Epoch 10/10
339/339 [==============================] - 25s 72ms/step - loss: 0.1360 - accuracy:
0.9500 - val_loss: 0.4102 - val_accuracy: 0.8948
Total training time: 4.374269696076711 minutes.
```

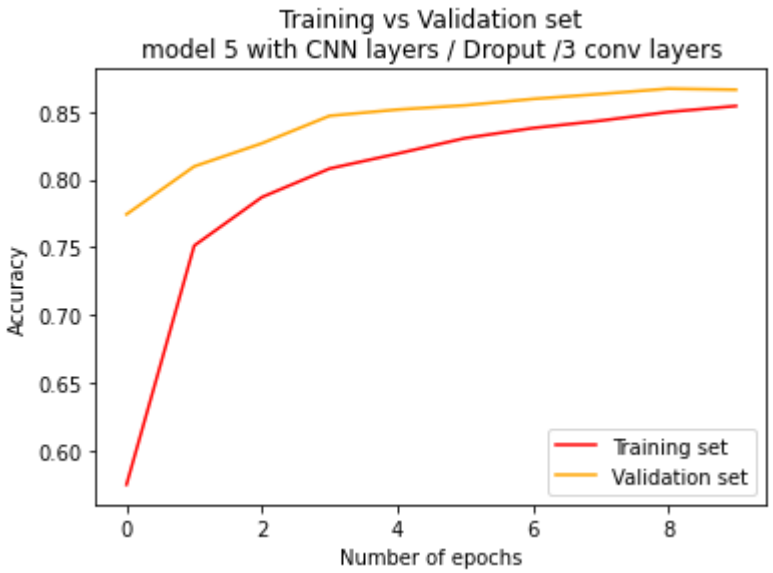


In [ ]:
```
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from tensorflow.keras.layers import Dropout

# Create model 5 with CNN layers / Dropout /3 conv layers
title = 'model 5 with CNN layers / Droput /3 conv layers'

model = Sequential()

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size= pool_size))

#Dropout
model.add(Dropout(0.1))

# Add a second conv layer with a stride of 2x2
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

#Dropout
model.add(Dropout(0.1))

# Add a third conv layer with a stride of 2x2
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

#Dropout
model.add(Dropout(0.1))

# Flatten the input
model.add(Flatten())

# Regular FC layer with output size 10 (for the 10 digits)
model.add(Dense(10, activation = "softmax"))
```

```
# Print the summary of the model to view the shape and number of parameters
model.summary()



# Specify the optimizer
model.compile(optimizer = opt, loss = loss, metrics = ['accuracy'])



# Time how fast the model train
start = time.time()

# Train using adam
model1 = model.fit(x_train, y_train, validation_split=0.15,batch_size = batch_size, ep

end = time.time()

num_mins = (end-start)/60
print("Total training time: "  + str(num_mins) + " minutes.")




# Evaluate the model
#score = model.evaluate(x_test, y_test, verbose = 0)
#print("Test loss: %.4f" % score[0])
#print("Accuracy: %.2f" % (score[1] * 100.0))




# Plot loss function (val vs test)
from matplotlib import pyplot as plt

plt.plot(model1.history["accuracy"], color = "red", label = "Training set")
plt.plot(model1.history["val_accuracy"], color = "orange", label = "Validation set")

plt.title("Training vs Validation set\n"+title)
plt.ylabel("Accuracy")
plt.xlabel("Number of epochs")
plt.legend()
plt.show()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
module_wrapper (ModuleWrappe (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 7, 7, 32)          9248
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 3, 3, 32)          0
_________________________________________________________________
module_wrapper_1 (ModuleWrap (None, 3, 3, 32)          0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 2, 2, 32)          9248
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
module_wrapper_2 (ModuleWrap (None, 1, 1, 32)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_10 (Dense)             (None, 10)                330
=================================================================
Total params: 19,146
Trainable params: 19,146
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
339/339 [==============================] - 29s 85ms/step - loss: 1.7987 - accuracy:
0.5753 - val_loss: 0.6391 - val_accuracy: 0.7744
Epoch 2/10
339/339 [==============================] - 30s 88ms/step - loss: 0.6951 - accuracy:
0.7513 - val_loss: 0.5282 - val_accuracy: 0.8097
Epoch 3/10
339/339 [==============================] - 29s 84ms/step - loss: 0.5919 - accuracy:
0.7870 - val_loss: 0.4809 - val_accuracy: 0.8267
Epoch 4/10
339/339 [==============================] - 29s 84ms/step - loss: 0.5269 - accuracy:
0.8081 - val_loss: 0.4348 - val_accuracy: 0.8469
Epoch 5/10
339/339 [==============================] - 29s 85ms/step - loss: 0.4933 - accuracy:
0.8191 - val_loss: 0.4129 - val_accuracy: 0.8515
Epoch 6/10
339/339 [==============================] - 29s 84ms/step - loss: 0.4634 - accuracy:
0.8305 - val_loss: 0.4007 - val_accuracy: 0.8546
Epoch 7/10
339/339 [==============================] - 29s 85ms/step - loss: 0.4410 - accuracy:
0.8379 - val_loss: 0.3862 - val_accuracy: 0.8593
Epoch 8/10
339/339 [==============================] - 30s 88ms/step - loss: 0.4201 - accuracy:
0.8433 - val_loss: 0.3763 - val_accuracy: 0.8630
Epoch 9/10
339/339 [==============================] - 29s 84ms/step - loss: 0.4048 - accuracy:
0.8497 - val_loss: 0.3620 - val_accuracy: 0.8669
Epoch 10/10
339/339 [==============================] - 29s 84ms/step - loss: 0.3923 - accuracy:
```

```
0.8541 - val_loss: 0.3609 - val_accuracy: 0.8661
Total training time: 4.822744429111481 minutes.
```

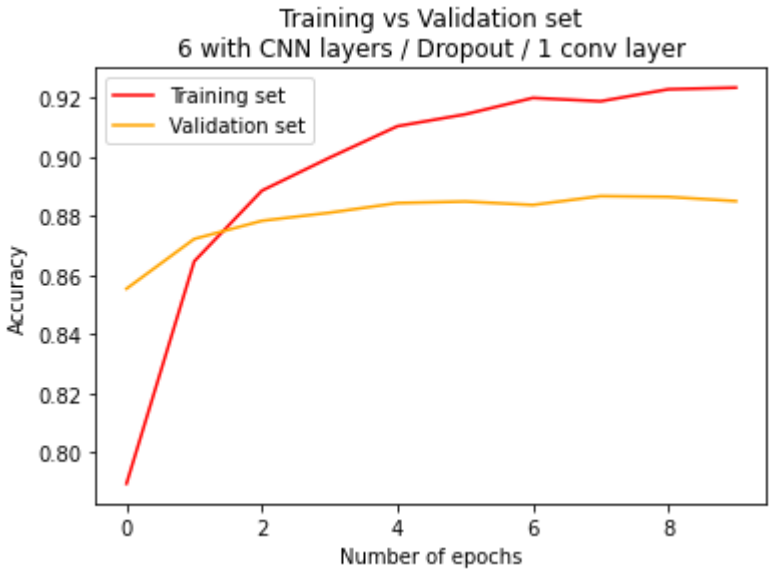


In [ ]:

```
# Create model 6 with CNN layers / Dropout / 1 conv layer

title = '6 with CNN layers / Dropout / 1 conv layer'

model = Sequential()

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padd

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

#Dropout
model.add(Dropout(0.1))

# Flatten the input
model.add(Flatten())

# Regular FC layer with output size 10 (for the 10 digits)
model.add(Dense(10, activation = "softmax"))

# Print the summary of the model to view the shape and number of parameters
model.summary()



# Specify the optimizer
model.compile(optimizer = opt, loss = loss, metrics = ['accuracy'])



# Time how fast the model train
start = time.time()

# Train using adam
model1 = model.fit(x_train, y_train, validation_split=0.15,batch_size = batch_size, ep

end = time.time()
```

```
num_mins = (end-start)/60
print("Total training time: "  + str(num_mins) + " minutes.")




# Evaluate the model
#score = model.evaluate(x_test, y_test, verbose = 0)
#print("Test loss: %.4f" % score[0])
#print("Accuracy: %.2f" % (score[1] * 100.0))




# Plot loss function (val vs test)
from matplotlib import pyplot as plt

plt.plot(model1.history["accuracy"], color = "red", label = "Training set")
plt.plot(model1.history["val_accuracy"], color = "orange", label = "Validation set")

plt.title("Training vs Validation set\n"+title)
plt.ylabel("Accuracy")
plt.xlabel("Number of epochs")
plt.legend()
plt.show()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
module_wrapper_3 (ModuleWrap (None, 14, 14, 32)        0
_________________________________________________________________
flatten_9 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                62730
=================================================================
Total params: 63,050
Trainable params: 63,050
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
339/339 [==============================] - 24s 69ms/step - loss: 3.4073 - accuracy:
0.7894 - val_loss: 0.6530 - val_accuracy: 0.8554
Epoch 2/10
339/339 [==============================] - 23s 68ms/step - loss: 0.4798 - accuracy:
0.8647 - val_loss: 0.4044 - val_accuracy: 0.8723
Epoch 3/10
339/339 [==============================] - 23s 68ms/step - loss: 0.3209 - accuracy:
0.8887 - val_loss: 0.3799 - val_accuracy: 0.8784
Epoch 4/10
339/339 [==============================] - 23s 68ms/step - loss: 0.2754 - accuracy:
0.8998 - val_loss: 0.3644 - val_accuracy: 0.8812
Epoch 5/10
339/339 [==============================] - 24s 72ms/step - loss: 0.2489 - accuracy:
0.9105 - val_loss: 0.3534 - val_accuracy: 0.8844
Epoch 6/10
339/339 [==============================] - 23s 68ms/step - loss: 0.2334 - accuracy:
0.9145 - val_loss: 0.3548 - val_accuracy: 0.8850
Epoch 7/10
339/339 [==============================] - 23s 68ms/step - loss: 0.2204 - accuracy:
0.9200 - val_loss: 0.3645 - val_accuracy: 0.8838
Epoch 8/10
339/339 [==============================] - 23s 68ms/step - loss: 0.2188 - accuracy:
0.9189 - val_loss: 0.3634 - val_accuracy: 0.8868
Epoch 9/10
339/339 [==============================] - 23s 68ms/step - loss: 0.2083 - accuracy:
0.9230 - val_loss: 0.3694 - val_accuracy: 0.8865
Epoch 10/10
339/339 [==============================] - 23s 68ms/step - loss: 0.2033 - accuracy:
0.9235 - val_loss: 0.3752 - val_accuracy: 0.8851
Total training time: 4.373109606901805 minutes.
```



In [ ]:
```
# Create model 7 with CNN layers / Early Stopping /3 conv layers

from tensorflow.python.keras.callbacks import EarlyStopping, ModelCheckpoint

title = 'model 7 with CNN layers / Early Stopping /3 conv layers'

model = Sequential()

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size= pool_size))

# Add a second conv layer with a stride of 2x2
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# Add a third conv layer with a stride of 2x2
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# Flatten the input
model.add(Flatten())

# Regular FC layer with output size 10 (for the 10 digits)
model.add(Dense(10, activation = "softmax"))

# Print the summary of the model to view the shape and number of parameters
model.summary()


#Early Stopping

es = EarlyStopping(monitor = "val_loss", mode = "min", verbose = 1, restore_best_weigh
mc = ModelCheckpoint("best_model_tutorial", monitor = "val_loss", save_best_only = Tru
```

```
# Specify the optimizer
model.compile(optimizer = opt, loss = loss, metrics = ['accuracy'])



# Time how fast the model train
start = time.time()

# Train using adam
model1 = model.fit(x_train, y_train, validation_split=0.15,batch_size = batch_size, ep

end = time.time()

num_mins = (end-start)/60
print("Total training time: "  + str(num_mins) + " minutes.")




# Evaluate the model
#score = model.evaluate(x_test, y_test, verbose = 0)
#print("Test loss: %.4f" % score[0])
#print("Accuracy: %.2f" % (score[1] * 100.0))




# Plot loss function (val vs test)
from matplotlib import pyplot as plt

plt.plot(model1.history["accuracy"], color = "red", label = "Training set")
plt.plot(model1.history["val_accuracy"], color = "orange", label = "Validation set")

plt.title("Training vs Validation set\n"+title)
plt.ylabel("Accuracy")
plt.xlabel("Number of epochs")
plt.legend()
plt.show()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 32)          9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 2, 2, 32)          9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 10)                330
=================================================================
Total params: 19,146
Trainable params: 19,146
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
339/339 [==============================] - 27s 77ms/step - loss: 1.3704 - accuracy:
0.7090 - val_loss: 0.5477 - val_accuracy: 0.8078
Epoch 2/10
339/339 [==============================] - 26s 75ms/step - loss: 0.4892 - accuracy:
0.8274 - val_loss: 0.4477 - val_accuracy: 0.8448
Epoch 3/10
339/339 [==============================] - 25s 73ms/step - loss: 0.4236 - accuracy:
0.8501 - val_loss: 0.4377 - val_accuracy: 0.8448
Epoch 4/10
339/339 [==============================] - 25s 73ms/step - loss: 0.3851 - accuracy:
0.8628 - val_loss: 0.4094 - val_accuracy: 0.8527
Epoch 5/10
339/339 [==============================] - 25s 73ms/step - loss: 0.3625 - accuracy:
0.8700 - val_loss: 0.3941 - val_accuracy: 0.8616
Epoch 6/10
339/339 [==============================] - 25s 72ms/step - loss: 0.3376 - accuracy:
0.8789 - val_loss: 0.3653 - val_accuracy: 0.8732
Epoch 7/10
339/339 [==============================] - 24s 72ms/step - loss: 0.3236 - accuracy:
0.8837 - val_loss: 0.3699 - val_accuracy: 0.8703
Epoch 8/10
339/339 [==============================] - 24s 72ms/step - loss: 0.3045 - accuracy:
0.8888 - val_loss: 0.3665 - val_accuracy: 0.8735
Epoch 9/10
339/339 [==============================] - 24s 72ms/step - loss: 0.2911 - accuracy:
0.8943 - val_loss: 0.3514 - val_accuracy: 0.8765
Epoch 10/10
339/339 [==============================] - 24s 72ms/step - loss: 0.2775 - accuracy:
0.8982 - val_loss: 0.3600 - val_accuracy: 0.8783
Total training time: 4.378224221865336 minutes.
```

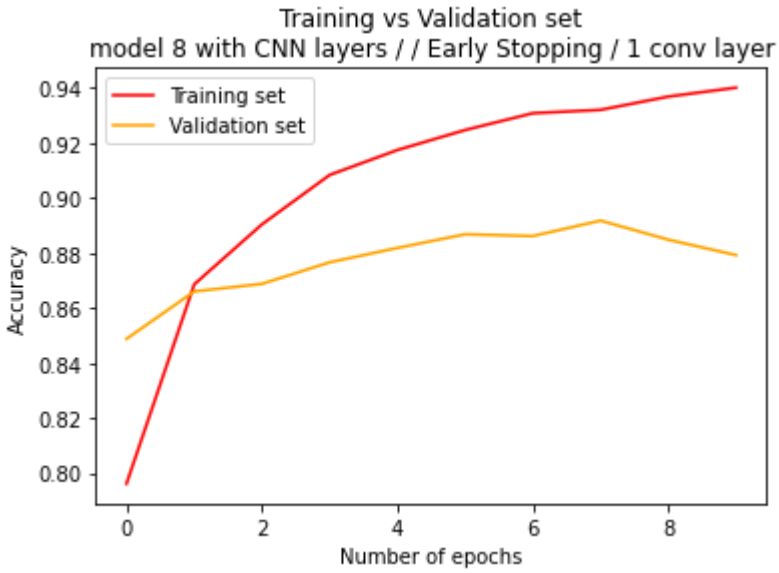
In [ ]:
```
# Create model 8 with CNN layers / / Early Stopping / 1 conv layer

title = 'model 8 with CNN layers / / Early Stopping / 1 conv layer'

model = Sequential()

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# Flatten the input
model.add(Flatten())

# Regular FC layer with output size 10 (for the 10 digits)
model.add(Dense(10, activation = "softmax"))

# Print the summary of the model to view the shape and number of parameters
model.summary()


#Early Stopping

es = EarlyStopping(monitor = "val_loss", mode = "min", verbose = 1, restore_best_weigh
mc = ModelCheckpoint("best_model_tutorial", monitor = "val_loss", save_best_only = Tru



# Specify the optimizer
model.compile(optimizer = opt, loss = loss, metrics = ['accuracy'])



# Time how fast the model train
start = time.time()

# Train using adam
model1 = model.fit(x_train, y_train, validation_split=0.15,batch_size = batch_size, ep

end = time.time()
```

```
num_mins = (end-start)/60
print("Total training time: "  + str(num_mins) + " minutes.")
```

```
# Evaluate the model
#score = model.evaluate(x_test, y_test, verbose = 0)
#print("Test loss: %.4f" % score[0])
#print("Accuracy: %.2f" % (score[1] * 100.0))
```

```
# Plot loss function (val vs test)
from matplotlib import pyplot as plt

plt.plot(model1.history["accuracy"], color = "red", label = "Training set")
plt.plot(model1.history["val_accuracy"], color = "orange", label = "Validation set")

plt.title("Training vs Validation set\n"+title)
plt.ylabel("Accuracy")
plt.xlabel("Number of epochs")
plt.legend()
plt.show()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                62730
=================================================================
Total params: 63,050
Trainable params: 63,050
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
339/339 [==============================] - 20s 58ms/step - loss: 4.0163 - accuracy:
0.7961 - val_loss: 0.9317 - val_accuracy: 0.8488
Epoch 2/10
339/339 [==============================] - 19s 57ms/step - loss: 0.5709 - accuracy:
0.8686 - val_loss: 0.5362 - val_accuracy: 0.8660
Epoch 3/10
339/339 [==============================] - 19s 57ms/step - loss: 0.3604 - accuracy:
0.8904 - val_loss: 0.4838 - val_accuracy: 0.8688
Epoch 4/10
339/339 [==============================] - 19s 57ms/step - loss: 0.2782 - accuracy:
0.9084 - val_loss: 0.4454 - val_accuracy: 0.8766
Epoch 5/10
339/339 [==============================] - 19s 57ms/step - loss: 0.2397 - accuracy:
0.9174 - val_loss: 0.4185 - val_accuracy: 0.8818
Epoch 6/10
339/339 [==============================] - 19s 57ms/step - loss: 0.2139 - accuracy:
0.9247 - val_loss: 0.4197 - val_accuracy: 0.8868
Epoch 7/10
339/339 [==============================] - 19s 56ms/step - loss: 0.1941 - accuracy:
0.9308 - val_loss: 0.4116 - val_accuracy: 0.8861
Epoch 8/10
339/339 [==============================] - 19s 57ms/step - loss: 0.1895 - accuracy:
0.9320 - val_loss: 0.4002 - val_accuracy: 0.8918
Epoch 9/10
339/339 [==============================] - 19s 57ms/step - loss: 0.1761 - accuracy:
0.9369 - val_loss: 0.4249 - val_accuracy: 0.8848
Epoch 10/10
339/339 [==============================] - 19s 56ms/step - loss: 0.1656 - accuracy:
0.9401 - val_loss: 0.4503 - val_accuracy: 0.8792
Total training time: 3.372301439444224 minutes.
```

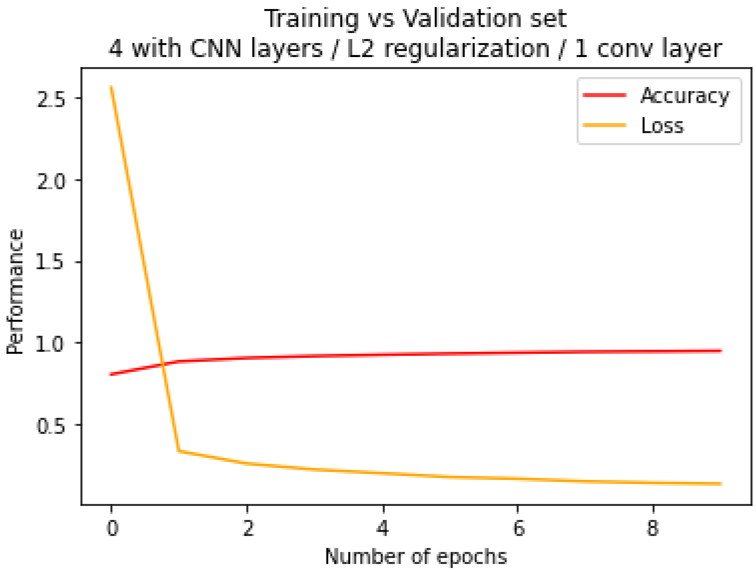


TOP-PERFORMING MODEL

In [ ]:
```
# Training on the whole train dataset on the top performing model from the experiments

# Create model 4 with CNN layers / L2 regularization / 1 conv layer

title = '4 with CNN layers / L2 regularization / 1 conv layer'

model = Sequential()

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padc

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# Flatten the input
model.add(Flatten())

# Add L2 metod
model.add(Dense(100, activation = activation, kernel_regularizer=regularizers.l2(0.000

# Regular FC layer with output size 10 (for the 10 digits)
model.add(Dense(10, activation = "softmax"))

# Print the summary of the model to view the shape and number of parameters
model.summary()



# Specify the optimizer
model.compile(optimizer = opt, loss = loss, metrics = ['accuracy'])



# Time how fast the model train
start = time.time()
```

```
# Train using adam
top_model = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs, verb

end = time.time()

num_mins = (end-start)/60
print("Total training time: "  + str(num_mins) + " minutes.")



# Evaluate the model
score = model.evaluate(x_test, y_test, verbose = 0)
print("Test loss: %.4f" % score[0])
print("Test accuracy: %.2f" % (score[1] * 100.0))



# Plot loss function (val vs test)
from matplotlib import pyplot as plt

plt.plot(top_model.history["accuracy"], color = "red", label = "Accuracy")
plt.plot(top_model.history["loss"], color = "orange", label = "Loss")

plt.title("Training vs Validation set\n"+title)
plt.ylabel("Performance")
plt.xlabel("Number of epochs")
plt.legend()
plt.show()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_4 (Dense)              (None, 100)               627300
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1010
=================================================================
Total params: 628,630
Trainable params: 628,630
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
399/399 [==============================] - 28s 70ms/step - loss: 2.5558 - accuracy:
0.8069
Epoch 2/10
399/399 [==============================] - 26s 65ms/step - loss: 0.3379 - accuracy:
0.8857
Epoch 3/10
399/399 [==============================] - 26s 64ms/step - loss: 0.2621 - accuracy:
0.9069
Epoch 4/10
399/399 [==============================] - 28s 69ms/step - loss: 0.2254 - accuracy:
0.9186
Epoch 5/10
399/399 [==============================] - 26s 65ms/step - loss: 0.2028 - accuracy:
0.9266
Epoch 6/10
399/399 [==============================] - 26s 65ms/step - loss: 0.1795 - accuracy:
0.9334
Epoch 7/10
399/399 [==============================] - 26s 65ms/step - loss: 0.1691 - accuracy:
0.9393
Epoch 8/10
399/399 [==============================] - 26s 65ms/step - loss: 0.1529 - accuracy:
0.9445
Epoch 9/10
399/399 [==============================] - 26s 65ms/step - loss: 0.1440 - accuracy:
0.9473
Epoch 10/10
399/399 [==============================] - 26s 66ms/step - loss: 0.1384 - accuracy:
0.9504
Total training time: 5.373602437973022 minutes.
Test loss: 0.3863
Test accuracy: 89.24
```



Interpreting CNN Representations / Code Part II

In [ ]:

```
# Training on the whole train dataset on the top performing model from the experiments
# Creating a new model with the bse of best oerformning model
# Create new model with CNN layers / L2 regularization / 1 conv layer

title = 'model with CNN layers / L2 regularization / 1 conv layer'

model = Sequential()

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padd

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padd

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padd

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# add a conv layer with "same" zero padding
model.add(Conv2D(num_filters, kernel_size = kernel_size, activation = activation, padd

# Add a pooling layer
model.add(MaxPooling2D(pool_size = pool_size))

# Flatten the input
model.add(Flatten())
```

```
# Add L2 metod
model.add(Dense(100, activation = activation, kernel_regularizer=regularizers.l2(0.000

# Regular FC layer with output size 10 (for the 10 digits)
model.add(Dense(10, activation = "softmax"))

# Print the summary of the model to view the shape and number of parameters
model.summary()



# Specify the optimizer
model.compile(optimizer = opt, loss = loss, metrics = ['accuracy'])



# Time how fast the model train
start = time.time()

# Train using adam
history = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs, verbos

end = time.time()

num_mins = (end-start)/60
print("Total training time: "  + str(num_mins) + " minutes.")
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 7, 7, 32)          9248
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 3, 3, 32)          0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 3, 3, 32)          9248
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_12 (Dense)             (None, 100)               3300
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1010
=================================================================
Total params: 32,374
Trainable params: 32,374
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
399/399 [==============================] - 49s 122ms/step - loss: 0.6694 - accuracy:
0.7729
Epoch 2/10
399/399 [==============================] - 48s 120ms/step - loss: 0.3855 - accuracy:
0.8615
Epoch 3/10
399/399 [==============================] - 48s 120ms/step - loss: 0.3275 - accuracy:
0.8812
Epoch 4/10
399/399 [==============================] - 49s 122ms/step - loss: 0.2959 - accuracy:
0.8922
Epoch 5/10
399/399 [==============================] - 48s 120ms/step - loss: 0.2733 - accuracy:
0.9011
Epoch 6/10
399/399 [==============================] - 48s 120ms/step - loss: 0.2584 - accuracy:
0.9070
Epoch 7/10
399/399 [==============================] - 48s 120ms/step - loss: 0.2394 - accuracy:
0.9123
Epoch 8/10
399/399 [==============================] - 49s 122ms/step - loss: 0.2296 - accuracy:
0.9157
Epoch 9/10
399/399 [==============================] - 48s 120ms/step - loss: 0.2166 - accuracy:
0.9209
Epoch 10/10
399/399 [==============================] - 48s 120ms/step - loss: 0.2104 - accuracy:
```

```
0.9212
Total training time: 8.37537084420522 minutes.
```

Out[ ]:
```
'from matplotlib import pyplot as plt\n\nplt.plot(top_model_visual.history["accurac
y"], color = "red", label = "Accuracy")\nplt.plot(top_model_visual.history["loss"], c
olor = "orange", label = "Loss")\n\nplt.title("Training vs Validation set\n"+title)\n
plt.ylabel("Performance")\nplt.xlabel("Number of epochs")\nplt.legend()\nplt.show()'
```

In [ ]:
```
# Printing model summary once again

model.summary()
```

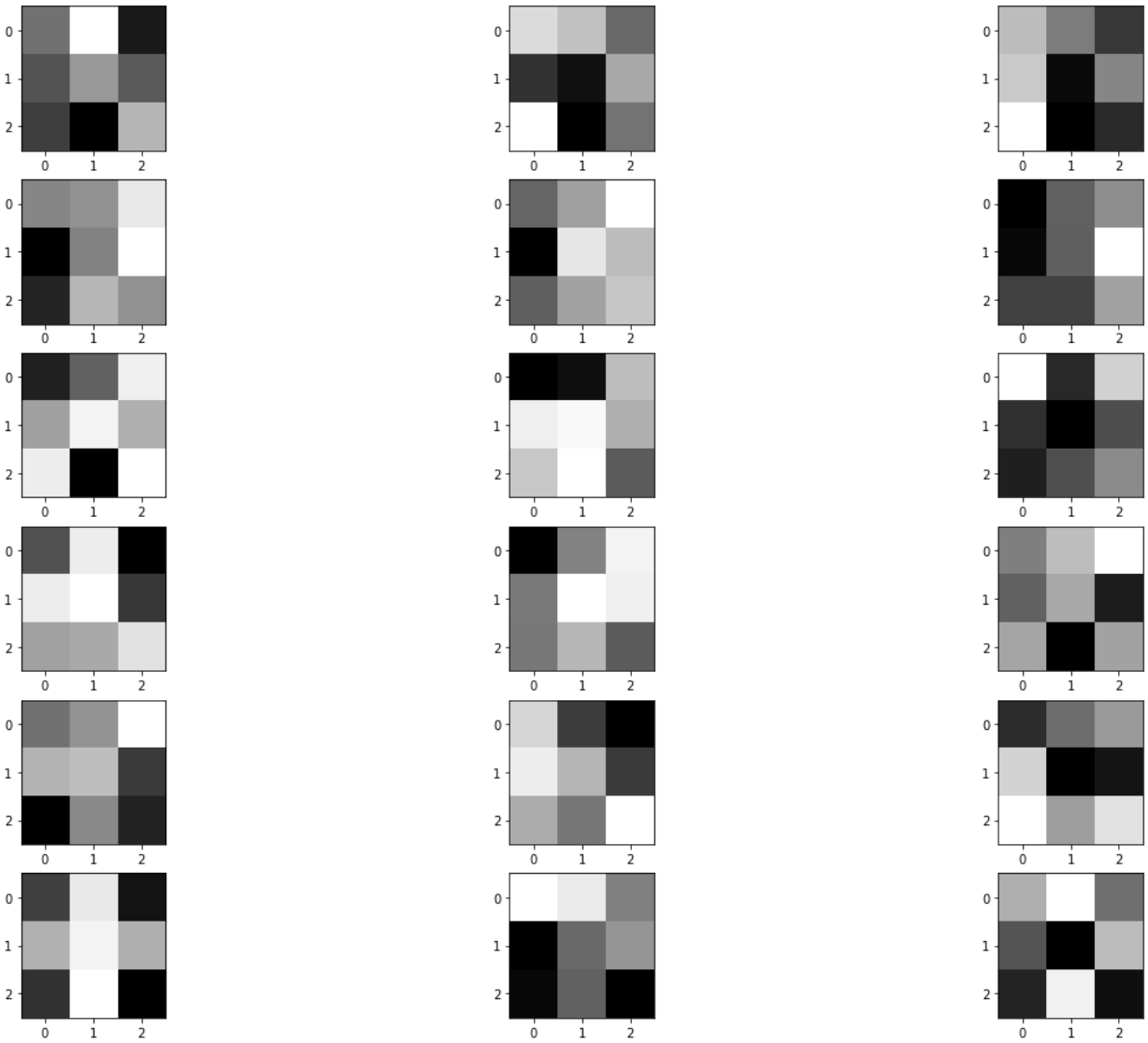
```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 7, 7, 32)          9248
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 3, 3, 32)          0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 3, 3, 32)          9248
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_12 (Dense)             (None, 100)               3300
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1010
=================================================================
Total params: 32,374
Trainable params: 32,374
Non-trainable params: 0
_________________________________________________________________
```

Visualizing Convolutional Layers

In [ ]:
```
# summarize filter shapes
for layer in model.layers:
        # check for convolutional layer
        if 'conv' not in layer.name:
                continue
        # get filter weights
        filters, biases = layer.get_weights()
        print(layer.name, filters.shape)


 # retrieve weights from the first conv layer
filters , bias = model.layers[2].get_weights()


# normalize filter values to 0-1 so we can visualize them
```

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


n_filters =6
ix=1
fig = plt.figure(figsize=(20,15))
for i in range(n_filters):
    # get the filters
    f = filters[:,:,:,i]
    for j in range(3):
        # subplot for 6 filters and 3 channels
        plt.subplot(n_filters,3,ix)
        plt.imshow(f[:,:,j] ,cmap='gray')
        ix+=1
#plot the filters
plt.show()
```

```
conv2d_16 (3, 3, 1, 32)
conv2d_17 (3, 3, 32, 32)
conv2d_18 (3, 3, 32, 32)
conv2d_19 (3, 3, 32, 32)
```



In [ ]:
```
# summarize filter shapes
for layer in model.layers:
```

```
        # check for convolutional layer
        if 'conv' not in layer.name:
                continue
        # get filter weights
        filters, biases = layer.get_weights()
        print(layer.name, filters.shape)


 # retrieve weights from the fourth conv layer
filters , bias = model.layers[6].get_weights()


# normalize filter values to 0-1 so we can visualize them
f_min, f_max = filters.min(), filters.max()
filters = (filters - f_min) / (f_max - f_min)


n_filters =6
ix=1
fig = plt.figure(figsize=(20,15))
for i in range(n_filters):
    # get the filters
    f = filters[:,:,:,i]
    for j in range(3):
        # subplot for 6 filters and 3 channels
        plt.subplot(n_filters,3,ix)
        plt.imshow(f[:,:,j] ,cmap='gray')
        ix+=1
#plot the filters
plt.show()
```

```
conv2d_16 (3, 3, 1, 32)
conv2d_17 (3, 3, 32, 32)
conv2d_18 (3, 3, 32, 32)
conv2d_19 (3, 3, 32, 32)
```