Introduction:
In this lab I experimented with Neural Network (NN)classifiers for handwritten digit optical recognition (a multi-class problem) using Keras with TensorFlow.
I created 3 groups of fully connected feed-forward (FNN) and 1 class of Convolutional Neural Net (CNN). All NNs use the softmax function for the output layer, which converts numbers (logits) into probabilities that sum up to one. Additionally, all NNs use 20% of training data as the validation set, 1-of-c output encoding (keras.utils.to_categorical) and early stopping to decide when to stop training to prevent overfitting (keras.callbacks.EarlyStopping). Also, I created with 20 different models for every FNN (60 in total) and 10 different models for the CNN and recorded the best model for the FNNs and the CNN including the overall classification accuracy, class accuracy, and confusion matrix for both training and testing data.

Feed forward fully connected neural nets and their hyper-parameters:
Feed forward neural nets consist on an input layer, one or more hidden layers and an output layer, which are connected by weights and do not form any cycles like recurrent neural nets. This means that all neurons between the input layer and the first hidden layer, the first and second hidden layer,…, the last hidden layer and the output layer are connected. However, this is not the case for convolutional neural nets that have a sparse connectivity between neurons due to the convolution operation. Both feed forward fully connected NN and convolutional NN use weights and biases and also an activation function in hidden layers. The activation function introduces non-linear properties into the NN and one of the most popular activation functions are: ReLU, Tanh, and Sigmoid. The ReLU (rectified linear units) is currently one of the most used activation functions for deep learning. When experimenting with neural nets, we first need to design the network structure (network topology, number of hidden units, activation functions) and then determine the weights for that structure. The network learns the weights through function optimization – the network find weights that minimize the error function (e.g.: mean-squared error, cross-entropy error function) using gradient descent. In particular, after designing the network structure, initializing the weights to small random numbers and present the network with training examples, the network does the following for each sampled training example. First, it makes a forward pass to compute the net activations and outputs, then it does a backward pass to compute errors and updates the weights based on that. One on the important hyper-parameters involved in learning the weights if learning rate, which controls how much the weights are adjusted. If the learning rate is too small, the gradient descent can be slow. However, if the learning rate is too large, the algorithm might miss the minimum and it may fail to converge. Another hyper-parameter that is paramount to finding the minimum is called momentum, which helps to prevent converging to local minima. A larger momentum increases the step size of the gradient descent algorithm and therefore, faster convergence. However, if both momentum and learning rate are too large, it minimum may be missed. It should be noted that there are various gradient descent optimizers, e.g.: stochastic gradient descent, batch gradient descent, Adam (adaptive moment estimation), Adamax,...
Another hyper-parameter is number of epochs, where one epoch represents a single presentation of all patterns in the training data – i.e.: the entire training set is passed through the network forward and back only once. Usually, the entire train set is not passed just once, but instead it is divided into batches. Batch size is a hyper-parameter that represents the total number of training example in one batch.

Convolutional neural nets and their hyper-parameters:
Convolutional NN have some hyper-parameters that fully connected feed-forward NN don't have due to the convolution operation that consists of three elements: input (usually image), feature detector (kernel/filter), and feature map that is outputted by applying the kernel to the input image. The size of the feature map depends on: depth (number of filters), stride (number of pixels we slide feature matrix over input matrix), and padding (we can pad the input matrix with zeros around the border). The higher number of filters, the more features are extracted and the better the CNN should recognize patterns. Also, a larger stride results in smaller feature maps. And padding the input matrix with zeros (wide convolution) controls the feature map size. In addition to convolution, CNNs can also use pooling. Spatial pooling reduces the dimension of each feature map, but retains the most important information. It helps to prevent over-fitting, because it

reduces the number of parameters and computations. The output of the convolution and pooling layers represents the high level features of the input image, which is then used as input into the last layer that is fully connected to make the classification decision.

Description of Feed forward fully connected neural nets designed:

All three FNNs utilize input scaling: the training and testing input data was normalized to have a mean of 0 and variance of 1 using Sklearn's StandardScaler. The first FNN (Task 1 (a)) uses sum-of-squares (mean-squared) error as the loss function and ReLU as the activation function for hidden layer(s). The second FNN (Task 1 (a)) uses cross-entropy as the loss function and ReLU as the activation function for hidden layer(s). The third FNN (Task 1 (b)) uses cross-entropy as the loss function and tanh as the activation function for hidden layer(s). For every FNN, I experimented 20 different models to experiment with different values of hyper-parameters:number of hidden layers, batch size, number of epochs, number of neurons in each hidden layer, output dimension, (activation and loss functions were given for each FNN), initial weights, initial bias, learning rate, optimizer, and momentum rates (only for SGD). I also measured the amount of time it takes to train each of the 20 models, and the loss and accuracy rates for both training and testing data.
The intuitive value for output dimension (the number of neurons in the output layer) is 10 since the goal is to classify handwritten digits into 10 categories (0, 1, …, 9). It is possible to try different output dimensions, but it may yield a model with lower accuracy(Nielsen, 2018). In addition to output dimension = 10, I also tried output dimension = 50 for the sake of experimentation. In the case of FNNs with mean square error loss function and ReLU activation, the training and testing accuracies of models with output dimension = 50 were lower on average than the ones with output dimension = 10. However, there was no significant difference in case of both FNNs with cross entropy loss functions.

Description of convolutional neural nets designed:

The CNN uses cross-entropy as the loss function and ReLU as the activation function in convolution layer(s). I created 10 different models with different values of hyper-parameters: number of convolution layers, batch size, number of epochs, number of filters, kernel size, strides, padding, (output dimension = 10, activation function and loss were given), initial weights, initial bias, learning rate, optimizer, momentum rates (only for SGD).

Summary of results:

In summary, fully connected feed forward neural nets generally performed better than convolutional neural nets regrading both accuracy (training and testing) and the amount of time spent on training.
The best accuracies were achieved by FNNs cross-entropy error with tahn activation function: 95% - almost 100% on testing and 89% - 97% on training. The lowest accuracies were measured on CNNs: 10% - 98% on training and 10% - 96% on testing – we can also see that CNN accuracies had the highest variance. Please see more detailed results discussion for Tasks 1 a,b and Task 2 below.

Detailed results:

Task 1 (a)
I. Fully connected feed forward neural net: Sum of squares error with ReLU activation function

Results Discussion:
The models with high accuracies are highlighted in yellow. There were 6 models with output dimension=10 that performed very well: 91% - 99% accuracy on training data and 90% - 95% accuracy on testing data. There were 4 models with output dimension=10 that performed quite badly: 15% - 49% accuracy on both training and testing data. Three of the four models that performed badly had three hidden layers, whereas the ones that performed well had only one or two hidden layers. Besides that, there don't seem to be any particular parameter values that the group of good or bad models have in common. They have various optimizers, batch sizes, number of epochs, learning rates,..
There were 4 models with output dimension=50 that performed very well: 91% - 98% testing accuracy and 88% - 96% training accuracy. All

four models had either one or two hidden layers, small learning rate, and used Adam or Adamax optimizer. There were 6 models with output dimension=50 that did not perform well: 1% - 40% accuracy on both training and testing data.
Overall, models with output dimension = 10, one or two hidden layers and Adam optimizer had the best performance.

Table I: Results of sample experiments with mean square error loss function and relu activation function

	layer	batch	epoch	unit	out_dim	activ	loss	weights	bias	learn_rate	optimizer	momentum	exe_time	loss_train	acc_train	loss_test	acc_test
0	1	10	10	100	10	relu	mean_squared_error	random_normal	zeros	0.1	SGD	0.8	1.89605	0.00518777	0.975474	0.00860868	0.945465
1	1	10	10	70	10	relu	mean_squared_error	random_uniform	zeros	0.001	Adam	nan	1.91806	0.00187313	0.992806	0.00660424	0.954368
2	1	5	15	100	10	relu	mean_squared_error	random_normal	zeros	0.02	SGD	0.9	2.46945	0.00645181	0.973512	0.009926	0.942682
3	2	10	50	70	10	relu	mean_squared_error	random_normal	zeros	0.01	Adam	nan	1.52554	0.0152487	0.919228	0.0194415	0.897607
4	2	20	20	100	10	relu	mean_squared_error	random_normal	zeros	0.0001	Adam	nan	2.11076	0.00933073	0.957489	0.0129419	0.93044
5	2	10	10	50	10	relu	mean_squared_error	random_normal	ones	0.03	Adam	nan	1.65223	0.0597538	0.489209	0.0605146	0.485253
6	2	10	10	100	10	relu	mean_squared_error	random_normal	zeros	0.001	Adamax	nan	1.61772	0.00590176	0.968934	0.0102878	0.933222
7	3	10	5	200	10	relu	mean_squared_error	random_normal	zeros	0.001	SGD	0.95	3.45522	0.0894496	0.214519	0.0894499	0.204786
8	3	15	10	40	10	relu	mean_squared_error	random_normal	zeros	0.015	SGD	0.7	2.17211	0.0899755	0.150425	0.0899795	0.148024
9	3	10	10	100	10	relu	mean_squared_error	random_normal	zeros	0.01	SGD	0.9	3.94287	0.0880727	0.223349	0.0881985	0.209238
10	1	10	10	100	50	relu	mean_squared_error	random_normal	zeros	0.1	SGD	0.8	4.27211	0.0145491	0.339111	0.0147937	0.330551
11	1	10	10	70	50	relu	mean_squared_error	random_uniform	zeros	0.001	Adam	nan	1.91805	0.000617028	0.984957	0.0014302	0.956038
12	1	5	15	100	50	relu	mean_squared_error	random_normal	zeros	0.02	SGD	0.9	6.68891	0.0177584	0.393721	0.0178942	0.397329
13	2	10	50	70	50	relu	mean_squared_error	random_normal	zeros	0.01	Adam	nan	2.68521	0.00340515	0.913342	0.0046608	0.880913
14	2	20	20	100	50	relu	mean_squared_error	random_normal	zeros	0.0001	Adam	nan	3.27309	0.0016609	0.961413	0.00239848	0.935448
15	2	10	10	50	50	relu	mean_squared_error	random_normal	ones	0.03	Adam	nan	3.04348	0.0194336	0.491825	0.0206996	0.457429
16	2	10	10	100	50	relu	mean_squared_error	random_normal	zeros	0.001	Adamax	nan	2.70812	0.0010694	0.970896	0.00187107	0.938787
17	3	10	5	200	50	relu	mean_squared_error	random_normal	zeros	0.001	SGD	0.95	3.55286	0.0196059	0.0107914	0.0196071	0.0139121
18	3	15	10	40	50	relu	mean_squared_error	random_normal	zeros	0.015	SGD	0.7	3.74498	0.0195954	0.0657292	0.0195954	0.0690039
19	3	10	10	100	50	relu	mean_squared_error	random_normal	zeros	0.01	SGD	0.9	5.3932	0.0195804	0.106279	0.0195803	0.101836

Table II: Best parameters configuration for mean square error loss function and relu activation function

	layer	batch	epoch	unit	out_dim	activ	loss	weights	bias	learn_rate	optimizer	momentum	exe_time	loss_train	acc_train	loss_test	acc_test
1	1	10	10	70	10	relu	mean_squared_error	random_uniform	zeros	0.001	Adam	nan	1.91806	0.00187313	0.992806	0.00660424	0.954368

Confusion matrix for train set

```
 [[294   0   0   0   1   0   1   0   0   0]
 [  0 306   1   0   0   0   0   1   2   2]
 [  0   0 312   0   0   0   0   0   2   1]
 [  0   1   0 304   0   2   0   1   1   3]
 [  0   0   0   0 303   0   3   0   0   2]
 [  0   0   0   0   0 312   0   0   1   3]
 [  0   0   0   0   1   0 304   0   1   0]
 [  0   0   0   0   0   0   0 299   0   1]
 [  0   4   0   0   0   0   1   1 289   0]
 [  1   2   0   2   1   1   0   0   2 289]]
```

Class Accuracies (training):

```
Accuracy of class 0 is =  0.9932432432432432
Accuracy of class 1 is =  0.9807692307692307
Accuracy of class 2 is =  0.9904761904761905
Accuracy of class 3 is =  0.9743589743589743
Accuracy of class 4 is =  0.9837662337662337
Accuracy of class 5 is =  0.9873417721518988
Accuracy of class 6 is =  0.9934640522875817
Accuracy of class 7 is =  0.9966666666666667
Accuracy of class 8 is =  0.9796610169491525
Accuracy of class 9 is =  0.9697986577181208
```

Confusion matrix for test set:

```
[[176   0   0   0   1   0   1   0   0   0]
 [  0 176   0   0   0   0   2   0   1   3]
 [  0   4 171   0   0   0   0   2   0   0]
 [  0   0   4 168   0   2   0   2   4   3]
 [  0   1   0   0 177   0   0   0   3   0]
 [  0   0   0   0   1 179   0   0   0   2]
 [  0   1   0   0   1   0 178   0   1   0]
 [  0   0   0   0   2   0   0 164   0  13]
 [  0   9   0   1   0   1   0   0 157   6]
 [  0   0   0   1   2   2   0   0   3 172]]
```

Class Accuracies (testing):

```
Accuracy of class 0 is =  0.9887640449438202
Accuracy of class 1 is =  0.967032967032967
Accuracy of class 2 is =  0.9661016949152542
Accuracy of class 3 is =  0.9180327868852459
Accuracy of class 4 is =  0.9779005524861878
Accuracy of class 5 is =  0.9835164835164835
Accuracy of class 6 is =  0.9834254143646409
Accuracy of class 7 is =  0.9162011173184358
Accuracy of class 8 is =  0.9022988505747126
Accuracy of class 9 is =  0.9555555555555556
```

II. Fully connected feed forward neural net: Cross-entropy error with ReLU activation function

Results Discussion:

On average, the models below that are using cross-entropy error as the loss function permed significantly better that the previous ones that used mean-squared-error even though the activation function was ReLU in both cases. The training accuracy on all models with cross-entropy error is between 65% and 99%, where only one model has a training accuracy below 91%. The testing accuracy is between 64% and 95%, which is very good. The hyper-parameter values differ amongst the models: the number of hidden layers (1, 2,or 3), batch size, number of epochs and even the different output dimension or learning rate did not have an adverse effect on accuracies. In conclusion, the cross-entropy error achieved much better accuracies than the mean-squared error used in the previous experiment.

Table I: Results of sample experiments with cross entropy loss function and relu activation function

	layer	batch	epoch	unit	out_dim	activ	loss	weights	bias	lr	optimizer	momentum	exe_time	loss_train	acc_train	loss_test	acc_test
0	1	10	10	100	10	relu	categorical_crossentropy	random_normal	zeros	0.1	SGD	0.8	1.65951	0.533285	0.936887	1.03852	0.882582
1	1	10	10	70	10	relu	categorical_crossentropy	random_uniform	zeros	0.001	Adam	nan	1.50363	0.0724068	0.983322	0.153317	0.952142
2	1	5	15	100	10	relu	categorical_crossentropy	random_normal	zeros	0.02	SGD	0.9	2.59151	0.0425254	0.985939	0.221713	0.945465
3	2	10	50	70	10	relu	categorical_crossentropy	random_normal	zeros	0.01	Adam	nan	1.60176	0.149298	0.963375	0.426296	0.90985
4	2	20	20	100	10	relu	categorical_crossentropy	random_normal	zeros	0.0001	Adam	nan	1.70781	0.292224	0.942773	0.359521	0.912632
5	2	10	10	50	10	relu	categorical_crossentropy	random_normal	ones	0.03	Adam	nan	3.09849	1.95496	0.653695	2.19293	0.638843
6	2	10	10	100	10	relu	categorical_crossentropy	random_normal	zeros	0.001	Adamax	nan	2.01656	0.123493	0.969588	0.216827	0.929327
7	3	10	5	200	10	relu	categorical_crossentropy	random_normal	zeros	0.001	SGD	0.95	2.20796	0.124224	0.966972	0.210477	0.940456
8	3	15	10	40	10	relu	categorical_crossentropy	random_normal	zeros	0.015	SGD	0.7	1.83918	0.172284	0.950948	0.273895	0.912632
9	3	10	10	100	10	relu	categorical_crossentropy	random_normal	zeros	0.01	SGD	0.9	2.04838	0.0423213	0.988228	0.167089	0.954925
10	1	10	10	100	50	relu	categorical_crossentropy	random_normal	zeros	0.1	SGD	0.8	2.63063	0.148026	0.972531	0.344686	0.947134
11	1	10	10	70	50	relu	categorical_crossentropy	random_uniform	zeros	0.001	Adam	nan	2.18886	0.0897943	0.980379	0.186826	0.943239
12	1	5	15	100	50	relu	categorical_crossentropy	random_normal	zeros	0.02	SGD	0.9	3.14188	0.21635	0.950294	0.357254	0.920979
13	2	10	50	70	50	relu	categorical_crossentropy	random_normal	zeros	0.01	Adam	nan	2.54855	0.0754748	0.980706	0.288425	0.933779
14	2	20	20	100	50	relu	categorical_crossentropy	random_normal	zeros	0.0001	Adam	nan	2.28506	0.478725	0.912688	0.534491	0.882582
15	2	10	10	50	50	relu	categorical_crossentropy	random_normal	ones	0.03	Adam	nan	2.96006	0.391848	0.907129	0.562372	0.892042

16	2	10	10	100	50	relu	categorical_crossentropy	random_normal	zeros	0.001	Adamax	nan	2.45767	0.144315	0.962394	0.222039	0.930996
17	3	10	5	200	50	relu	categorical_crossentropy	random_normal	zeros	0.001	SGD	0.95	2.9378	0.1154	0.975474	0.219691	0.937117
18	3	15	10	40	50	relu	categorical_crossentropy	random_normal	zeros	0.015	SGD	0.7	2.74134	0.23516	0.938849	0.421234	0.892042
19	3	10	10	100	50	relu	categorical_crossentropy	random_normal	zeros	0.01	SGD	0.9	2.84837	0.0541636	0.985612	0.260265	0.93823

Table II: Best parameters configuration for cross entropy loss function and relu activation function

	layer	batch	epoch	unit	out_dim	activ	loss	weights	bias	lr	optimizer	momentum	exe_time	loss_train	acc_train	loss_test	acc_test
1	1	10	10	70	10	relu	categorical_crossentropy	random_uniform	zeros	0.001	Adam	nan	1.50363	0.0724068	0.983322	0.153317	0.952142

Confusion matrix for train set

```
[[301   0   0   0   1   0   0   0   0   0]
 [  0 314   0   0   0   0   0   1   3   2]
 [  0   0 306   0   0   0   0   0   0   0]
 [  0   0   0 300   0   2   0   0   1   1]
 [  0   0   0   0 304   0   1   0   1   2]
 [  0   0   0   0   0 292   0   0   1   3]
 [  0   2   0   0   1   0 301   0   1   0]
 [  0   0   0   1   0   0   0 306   0   0]
 [  0  18   0   1   3   2   1   0 274   0]
 [  0   3   0   4   2   1   0   0   1 300]]
```

Class Accuracies (training):

```
Accuracy of class 0 is =  0.9966887417218543
Accuracy of class 1 is =  0.98125
Accuracy of class 2 is =  1.0
Accuracy of class 3 is =  0.9868421052631579
Accuracy of class 4 is =  0.987012987012987
Accuracy of class 5 is =  0.9864864864864865
Accuracy of class 6 is =  0.9868852459016394
Accuracy of class 7 is =  0.996742671009772
Accuracy of class 8 is =  0.9163879598662207
Accuracy of class 9 is =  0.9646302250803859
```

Confusion matrix for test set:

```
[[177   0   0   0   1   0   0   0   0   0]
 [  0 176   0   0   0   0   0   0   1   5]
 [  0   6 165   0   2   0   0   1   3   0]
 [  0   0   3 169   0   2   0   1   4   4]
 [  0   0   0   0 177   0   0   0   4   0]
 [  0   0   0   0   1 179   0   0   1   1]
 [  2   2   0   0   0   0 176   0   1   0]
 [  0   0   0   0   1   5   0 166   0   7]
 [  0  18   0   1   0   3   0   0 142  10]
 [  0   2   0   2   3   3   0   0   2 168]]
```

Class Accuracies (testing):

```
Accuracy of class 0 is =  0.9943820224719101
Accuracy of class 1 is =  0.967032967032967
Accuracy of class 2 is =  0.9322033898305084
Accuracy of class 3 is =  0.9234972677595629
Accuracy of class 4 is =  0.9779005524861878
Accuracy of class 5 is =  0.9835164835164835
Accuracy of class 6 is =  0.9723756906077348
Accuracy of class 7 is =  0.9273743016759777
Accuracy of class 8 is =  0.8160919540229885
Accuracy of class 9 is =  0.9333333333333333
```

Task 1 (b)

I. Fully connected feed forward neural net: Cross-entropy error with tahn activation function

Results Discussion:

FNNs with cross-entropy error function and than activation function had the highest training and testing accuracy compared to the FNNs above and the CNNs below. The testing accuracy on these FNN models is very high: between 95% and almost 100%, the training accuracy is between 89% and 97%. So, models with cross-entropy error function had higher training and testing accuracy than models using mean-squared error. Also, models with cross-entropy error function using tanh activation function had higher accuracies than models with cross-entropy error function using ReLU activation function. The amount of time if took to train the individual FNN models was about 1.4 to 5.4 seconds and the average time was very similar for all three groups of FNNs (mean-squared error with ReLU, cross-entropy with ReLU, cross-entropy with than).

Table I: Results of sample experiments with cross entropy loss function and tanh activation function

	layer	batch	epoch	unit	out_dim	activ	loss	weights	bias	lr	optimizer	momentum	exe_time	loss_train	acc_train	loss_test	acc_test
0	1	10	10	100	10	tanh	categorical_crossentropy	random_normal	zeros	0.1	SGD	0.8	1.5937	0.0249036	0.992806	0.159428	0.952699
1	1	10	10	70	10	tanh	categorical_crossentropy	random_uniform	zeros	0.001	Adam	nan	1.39602	0.0837966	0.982014	0.157969	0.952142
2	1	5	15	100	10	tanh	categorical_crossentropy	random_normal	zeros	0.02	SGD	0.9	2.92868	0.00749084	0.999346	0.139137	0.964385
3	2	10	50	70	10	tanh	categorical_crossentropy	random_normal	zeros	0.01	Adam	nan	1.65449	0.0729258	0.977763	0.2152	0.943795
4	2	20	20	100	10	tanh	categorical_crossentropy	random_normal	zeros	0.0001	Adam	nan	2.1065	0.235163	0.958797	0.30871	0.928214
5	2	10	10	50	10	tanh	categorical_crossentropy	random_normal	ones	0.03	Adam	nan	2.10132	0.13762	0.956508	0.333155	0.910963
6	2	10	10	100	10	tanh	categorical_crossentropy	random_normal	zeros	0.001	Adamax	nan	1.6933	0.132834	0.972531	0.205265	0.942682
7	3	10	5	200	10	tanh	categorical_crossentropy	random_normal	zeros	0.001	SGD	0.95	2.93571	0.0775141	0.983322	0.159889	0.956038
8	3	15	10	40	10	tanh	categorical_crossentropy	random_normal	zeros	0.015	SGD	0.7	1.70948	0.16497	0.962067	0.266817	0.925988
9	3	10	10	100	10	tanh	categorical_crossentropy	random_normal	zeros	0.01	SGD	0.9	3.28244	0.0103038	0.998692	0.127679	0.967168
10	1	10	10	100	50	tanh	categorical_crossentropy	random_normal	zeros	0.1	SGD	0.8	2.23604	0.0153539	0.99673	0.13736	0.961046
11	1	10	10	70	50	tanh	categorical_crossentropy	random_uniform	zeros	0.001	Adam	nan	2.01871	0.106441	0.975147	0.178667	0.94936
12	1	5	15	100	50	tanh	categorical_crossentropy	random_normal	zeros	0.02	SGD	0.9	3.96481	0.00664791	0.999346	0.128488	0.966611
13	2	10	50	70	50	tanh	categorical_crossentropy	random_normal	zeros	0.01	Adam	nan	3.38506	0.0585352	0.97809	0.264046	0.936004
14	2	20	20	100	50	tanh	categorical_crossentropy	random_normal	zeros	0.0001	Adam	nan	5.32772	0.108242	0.977763	0.173713	0.950473
15	2	10	10	50	50	tanh	categorical_crossentropy	random_normal	ones	0.03	Adam	nan	2.42355	0.182325	0.948005	0.417413	0.894825
16	2	10	10	100	50	tanh	categorical_crossentropy	random_normal	zeros	0.001	Adamax	nan	3.50414	0.0953986	0.977436	0.164406	0.952699
17	3	10	5	200	50	tanh	categorical_crossentropy	random_normal	zeros	0.001	SGD	0.95	2.74797	0.127029	0.967953	0.196256	0.941569
18	3	15	10	40	50	tanh	categorical_crossentropy	random_normal	zeros	0.015	SGD	0.7	2.2383	0.19391	0.960759	0.297562	0.920979
19	3	10	10	100	50	tanh	categorical_crossentropy	random_normal	zeros	0.01	SGD	0.9	2.47882	0.0637267	0.981033	0.171409	0.949917

Table II: Best parameters configuration for cross entropy loss function and tanh activation function

	layer	batch	epoch	unit	out_dim	activ	loss	weights	bias	lr	optimizer	momentum	exe_time	loss_train	acc_train	loss_test	acc_test
2	1	5	15	100	10	tanh	categorical_crossentropy	random_normal	zeros	0.02	SGD	0.9	2.92868	0.00749084	0.999346	0.139137	0.964385

Confusion matrix for train set

```
[[303   0   0   0   0   0   0   0   0   0]
 [  0 306   0   0   0   0   0   0   0   0]
 [  0   0 295   0   0   0   0   0   0   0]
 [  0   0   0 314   0   0   0   0   0   1]
 [  0   0   0   0 313   0   0   0   0   0]
 [  0   0   0   0   0 306   0   0   0   0]
 [  0   1   0   0   0   0 298   0   0   0]
 [  0   0   0   0   0   0   0 311   0   0]
 [  0   0   0   0   0   0   0   0 310   0]
 [  0   0   0   0   0   0   0   0   0 300]]
```

Class Accuracies (training):

```
Accuracy of class 0 is =  1.0
Accuracy of class 1 is =  1.0
Accuracy of class 2 is =  1.0
Accuracy of class 3 is =  0.9968253968253968
Accuracy of class 4 is =  1.0
```

```
Accuracy of class 5 is =  1.0
Accuracy of class 6 is =  0.9966555183946488
Accuracy of class 7 is =  1.0
Accuracy of class 8 is =  1.0
Accuracy of class 9 is =  1.0
```

Confusion matrix for test set:

```
 [[177   0   0   0   0   0   0   0   1   0]
 [  0 179   0   0   0   0   1   0   2   0]
 [  0   3 170   1   0   0   2   0   1   0]
 [  0   0   3 172   0   2   1   0   2   3]
 [  0   1   0   0 178   0   0   0   2   0]
 [  0   0   0   0   0 179   0   0   0   3]
 [  0   0   0   0   3   0 178   0   0   0]
 [  0   0   0   0   1   4   0 167   1   6]
 [  0   5   0   0   1   0   0   0 160   8]
 [  0   0   0   0   0   1   0   0   2 177]]
```

Class Accuracies (testing):

```
Accuracy of class 0 is =  0.9943820224719101
Accuracy of class 1 is =  0.9835164835164835
Accuracy of class 2 is =  0.96045197740113
Accuracy of class 3 is =  0.9398907103825137
Accuracy of class 4 is =  0.9834254143646409
Accuracy of class 5 is =  0.9835164835164835
Accuracy of class 6 is =  0.9834254143646409
Accuracy of class 7 is =  0.9329608938547486
Accuracy of class 8 is =  0.9195402298850575
Accuracy of class 9 is =  0.9833333333333333
```

Task 2

I. Convolutional neural net: Cross-entropy error with ReLU activation function

Results Discussion:

There were 6 models (highlighted in yellow) that performed well (test accuracy: 87% - 95%) and 4 models that performed badly (test accuracy: 10% - 52%). It seems like that it is the particular combination of hyper parameters that determines the overall performance of the model, the only common hyper-parameter value that all of the high performing models share is small learning rate, but besides that there don't seem to be any other common hyper-parameter values that would explain why these six models did much better than the other four. CNNs are often used for image recognition, so perhaps further hyper-parameter tuning would be needed to achieve better accuracies. The amount of time if took to train the individual FNN models was about 2.5 to 9.5 seconds – almost twice as long than the FNN models.

Table I: Results of sample experiments with cross entropy loss function and relu activation function

legend: L = number of hidden layers, B = batch size, E = number of epochs, F = filters, K = kernel ,S = strides, P = padding, OD = output dimension

	L	B	E	F	K	S	P	OD	activ	loss	weights	bias	lr	optim	momentum	exe_time	loss_train	acc_train	loss_test	acc_test
0	1	10	10	64	(5, 5)	(1, 1)	same	10	relu	categorical_crossentropy	random_normal	ones	0.1	SGD	0.8	2.27269	14.5316	0.0984303	14.5036	0.100167
1	1	10	10	64	(5, 5)	(1, 1)	valid	10	relu	categorical_crossentropy	random_normal	zeros	0.001	Adam	nan	3.85851	0.0546522	0.980379	0.184482	0.941569
2	1	5	15	64	(3, 3)	(1, 1)	valid	10	relu	categorical_crossentropy	random_normal	zeros	0.02	SGD	0.9	2.22411	1.33938	0.565402	1.48399	0.520312
3	2	10	50	30	(5, 5)	(1, 1)	same	10	relu	categorical_crossentropy	random_normal	zeros	0.01	Adam	nan	2.47133	0.197145	0.950294	0.388023	0.917084
4	2	20	20	10	(3, 3)	(2, 2)	same	10	relu	categorical_crossentropy	random_uniform	zeros	0.0001	Adam	nan	4.20051	0.339928	0.910072	0.437074	0.873678
5	2	10	10	64	(5, 5)	(1, 1)	same	10	relu	categorical_crossentropy	random_normal	ones	0.03	Adam	nan	6.38517	14.4736	0.102027	14.4767	0.101836
6	2	10	10	15	(5, 5)	(1, 1)	same	10	relu	categorical_crossentropy	random_normal	zeros	0.001	Adamax	nan	7.72169	0.0268182	0.995095	0.119625	0.959933
7	3	10	5	5	(3, 3)	(1, 1)	same	10	relu	categorical_crossentropy	random_normal	zeros	0.001	SGD	0.95	3.79129	0.0765093	0.975147	0.155155	0.948804
8	3	15	10	64	(5, 5)	(1, 1)	same	10	relu	categorical_crossentropy	random_normal	zeros	0.015	SGD	0.7	9.53167	14.5474	0.0974493	14.5215	0.099054
9	3	10	10	64	(3, 3)	(1, 1)	same	10	relu	categorical_crossentropy	random_normal	zeros	0.01	SGD	0.9	9.36257	0.135823	0.963702	0.249685	0.948804

Table II: Best parameters configuration for cross entropy loss function and relu activation function

The best performance was achieved by model2, model7, model8, and model10. Because the accuracies were very similar and slightly different every run, I arbitrarily selected one of them (model2) as the best one.

	L	B	E	F	K	S	P	OD	activ	loss	weights	bias	lr	optim	momentum	exe_time	loss_train	acc_train	loss_test	acc_test
1	1	10	10	64	(5, 5)	(1, 1)	valid	10	relu	categorical_crossentropy	random_normal	zeros	0.001	Adam	nan	3.85851	0.0546522	0.980379	0.184482	0.941569

Confusion matrix for train set

```
 [[298   0   0   0   0   0   0   0   0   0]
 [  0 274   0   0   0   0   0   0  34   4]
 [  0   0 298   0   0   0   0   0  10   0]
 [  0   0   0 312   0   0   0   0   0   0]
 [  0   0   0   0 319   0   0   0   0   0]
 [  0   0   0   0   0 293   0   0   0   0]
 [  0   0   0   0   0   0 293   0  10   0]
 [  0   0   0   0   0   0   0 302   0   0]
 [  0   0   0   0   0   0   0   0 310   0]
 [  0   0   0   0   0   0   0   0   2 299]]
```

Class Accuracies (training):

```
Accuracy of class 0 is =  1.0
Accuracy of class 1 is =  0.8782051282051282
Accuracy of class 2 is =  0.9675324675324676
Accuracy of class 3 is =  1.0
Accuracy of class 4 is =  1.0
Accuracy of class 5 is =  1.0
Accuracy of class 6 is =  0.966996699669967
Accuracy of class 7 is =  1.0
Accuracy of class 8 is =  1.0
Accuracy of class 9 is =  0.9933554817275747
```

Confusion matrix for test set:

```
 [[176   0   0   0   1   0   0   0   1   0]
 [  0 156   0   1   0   0   0   0  25   0]
 [  0   0 163   0   0   0   0   0  14   0]
 [  0   0   0 174   0   1   0   0   6   2]
 [  0   0   0   0 179   0   0   0   1   1]
 [  0   0   0   1   0 181   0   0   0   0]
 [  0   0   0   0   0   2 162   0  17   0]
 [  0   0   0   0   0   2   0 158  10   9]
 [  0   1   0   0   0   0   0   0 172   1]
 [  0   0   0   2   0   2   0   0   5 171]]
```

Class Accuracies (testing):

```
Accuracy of class 0 is =  0.9887640449438202
Accuracy of class 1 is =  0.8571428571428571
Accuracy of class 2 is =  0.9209039548022598
Accuracy of class 3 is =  0.9508196721311475
Accuracy of class 4 is =  0.988950276243094
Accuracy of class 5 is =  0.9945054945054945
Accuracy of class 6 is =  0.8950276243093923
Accuracy of class 7 is =  0.88268156424581
Accuracy of class 8 is =  0.9885057471264368
Accuracy of class 9 is =  0.95
```

References:

Nielsen (2018): Michael Nielsen, Neural Networks and Deep Learning