CS512 - Assignment 2

Mohammadreza Asherloo
Department of Mechanical, Materials and Aerospace Engineering
Illinois Institute of Technology

November 10, 2020

Problem Statement:

Implementation of Convolutional Neural Network (CNN)

Solution:

The steps taken are as follows:

1. Take images for pre-processing (split, resize, normalization).
2. Build a base CNN with 2 convolution layers with max pooling and 2 fully connected layers.
3. Change different parameters one by one and plot the results for comparison.
4. Select the best model and save it's weights and architecture for later use.
5. Use the saved model for prediction.

Details:

We load the MNIST data first:

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Then we stack train images and test images for a different split:

```
x = np.concatenate((train_images, test_images))
y = np.concatenate((train_labels, test_labels))
```

Then split the images to train, validation and test parts (55000/10000/5000):

```
np.random.seed(2019)
train_size = 55076/70000
index = np.random.rand(len(x)) < train_size
```

```
train_images, test_images = x[index], x[~index]
train_labels, test_labels = y[index], y[~index]

index = np.random.rand(len(test_images)) < 0.6585
val_images, test_images = test_images[index], test_images[~index]
val_labels, test_labels = test_labels[index], test_labels[~index]
```

Then we change the labels from numbers to odd/even:

```
for i in range(len(train_labels)):
  if train_labels[i] % 2 == 0:
    train_labels[i] = 1
  else:
    train_labels[i] = 0

for i in range(len(test_labels)):
  if test_labels[i] % 2 == 0:
    test_labels[i] = 1
  else:
    test_labels[i] = 0

for i in range(len(val_labels)):
  if val_labels[i] % 2 == 0:
    val_labels[i] = 1
  else:
    val_labels[i] = 0
```

Then we need to reshape the input tensors:

```
train_images = train_images.reshape((55000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((5000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

val_images = val_images.reshape((10000, 28, 28, 1))
val_images = val_images.astype('float32') / 255
```

Then change the label type to categorical:

```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
val_labels = to_categorical(val_labels)
```

We build our base CNN, fit it and gather loss and accuracy:

```
keras.backend.clear_session()

net_2conv_2dense = models.Sequential([
```

```
        layers.Conv2D(5, kernel_size = (3, 3), activation = 'relu',
                      input_shape = (28, 28, 1),
                      kernel_initializer=keras.initializers.GlorotNormal()),
        layers.MaxPool2D((2, 2), strides = 2),
        layers.Conv2D(2, kernel_size = (3, 3), activation = 'relu',
                      kernel_initializer=keras.initializers.GlorotNormal()),
        layers.MaxPool2D((2, 2), strides = 2),
        layers.Dropout(0.2),
        layers.Flatten(),
        layers.Dense(5, activation= 'relu'),
        layers.Dense(2, activation= 'sigmoid')
])

net_2conv_2dense.summary()

net_2conv_2dense.compile(optimizer = "adam",
                         loss = 'binary_crossentropy',
                         metrics = ['accuracy'])

history = net_2conv_2dense.fit(train_images, train_labels,
              epochs = 10, verbose = 1,
              validation_data=(val_images, val_labels))
```

Then we plot the data:

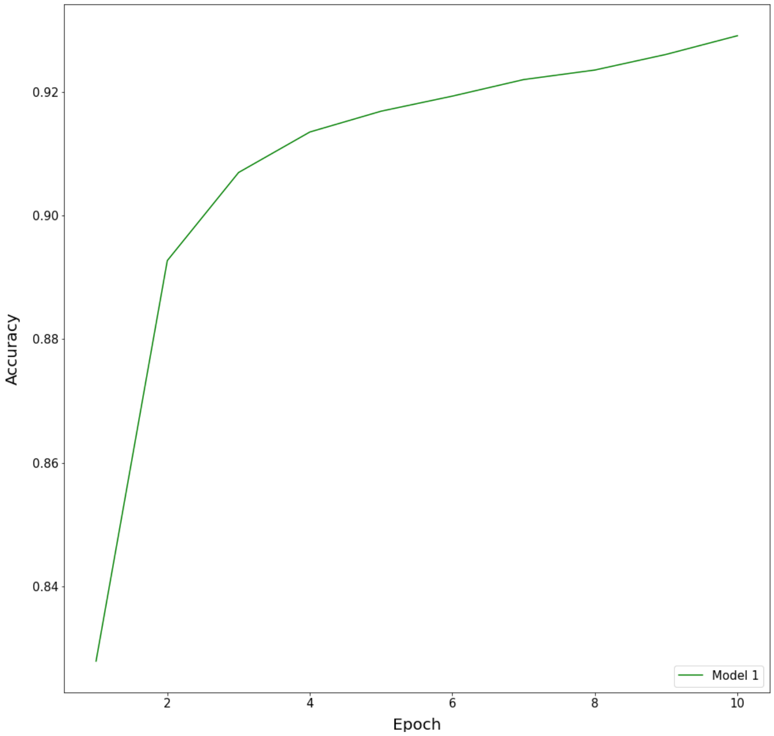
```
plt.rcParams['xtick.labelsize'] = 15
plt.rcParams['ytick.labelsize'] = 15
plt.figure(figsize=(16, 16))
loss = history.history['loss']
val_loss = history.history['val_loss']
accuracy = history.history['accuracy']
print("First model: 2 Conv-Maxpool + 2 Dense")
plt.plot(np.linspace(1, 10, 10), accuracy, 'g')
plt.xlabel("Epoch", fontsize = 20, labelpad = 10)
plt.ylabel("Accuracy", fontsize = 20, labelpad = 15)
plt.legend(['Model 1'], loc = 'lower right', prop = {'size': 15})
plt.show()
print("First model: Loss of final step: {},
      Accuracy of final step: {}".format(loss[-1], accuracy[-1]))
```

After this step we will change some parameters and repeat the fitting and plotting steps. Results are shown in the next section.

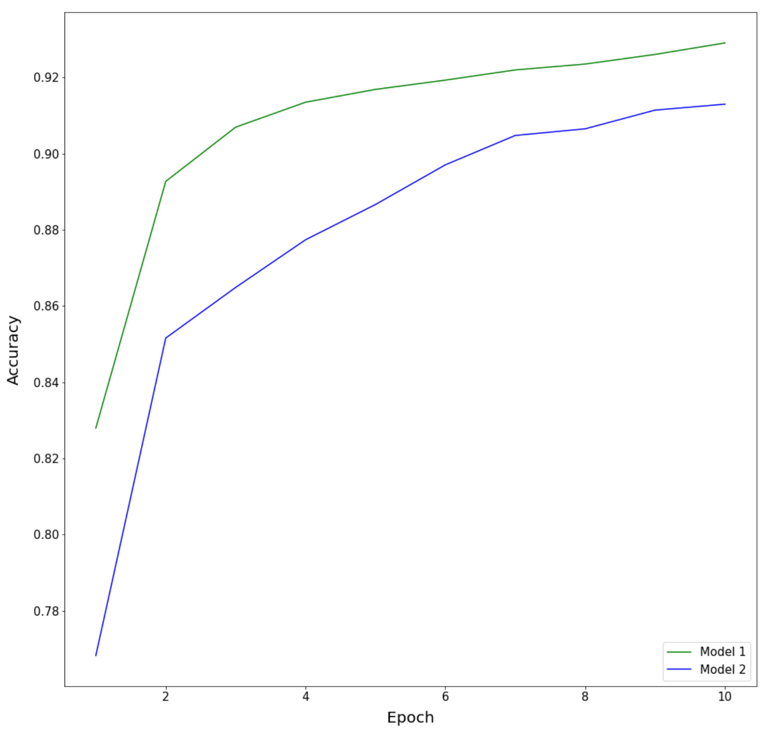
Results and discussion:

Here we will show the different CNN models step by step for comparison:

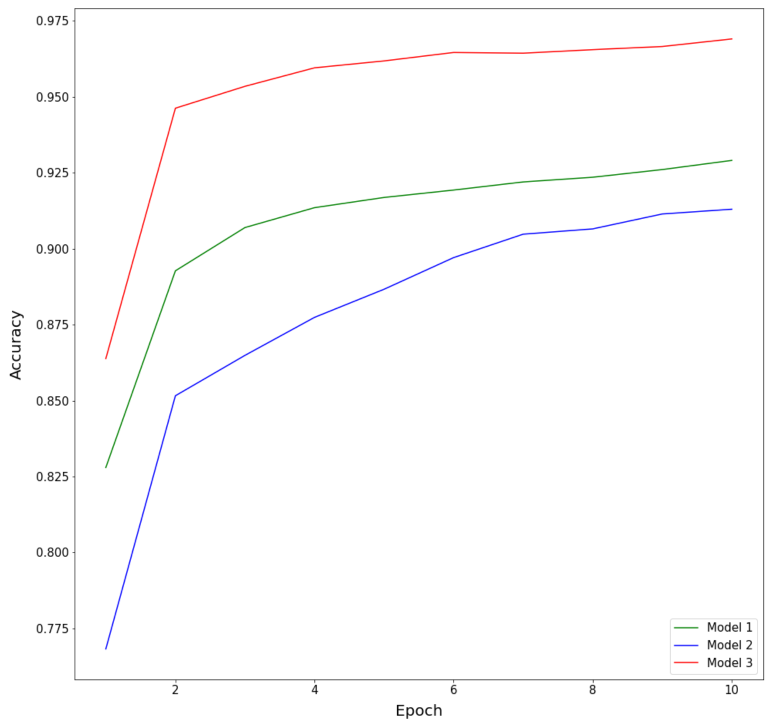
First model with 2 convolution layers (5 filters and 2 filters) and 2 fully connected layers (10 filters and filters):



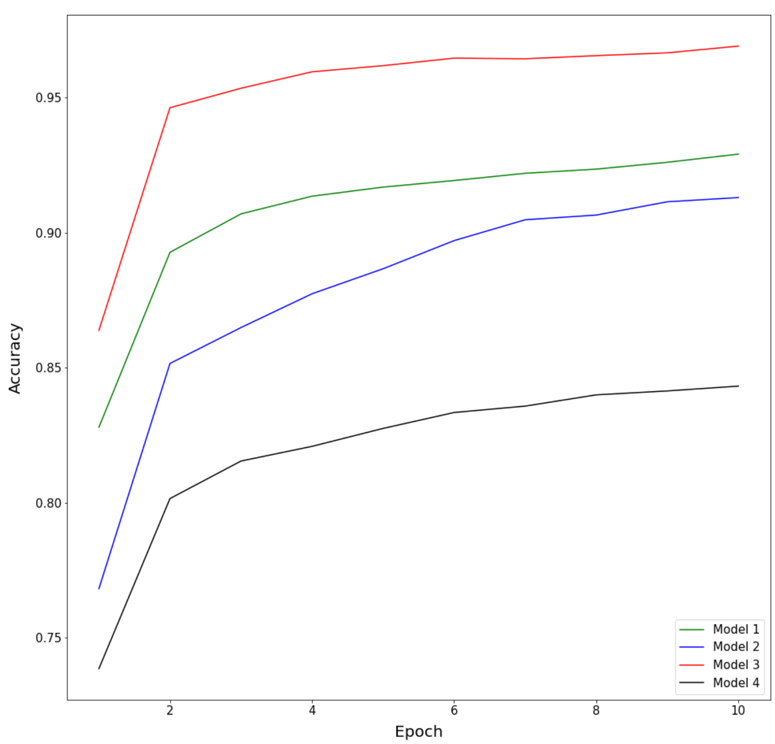
For the second CNN we added a new convolution layer without max pool and the model clearly is performing worse than the base model:



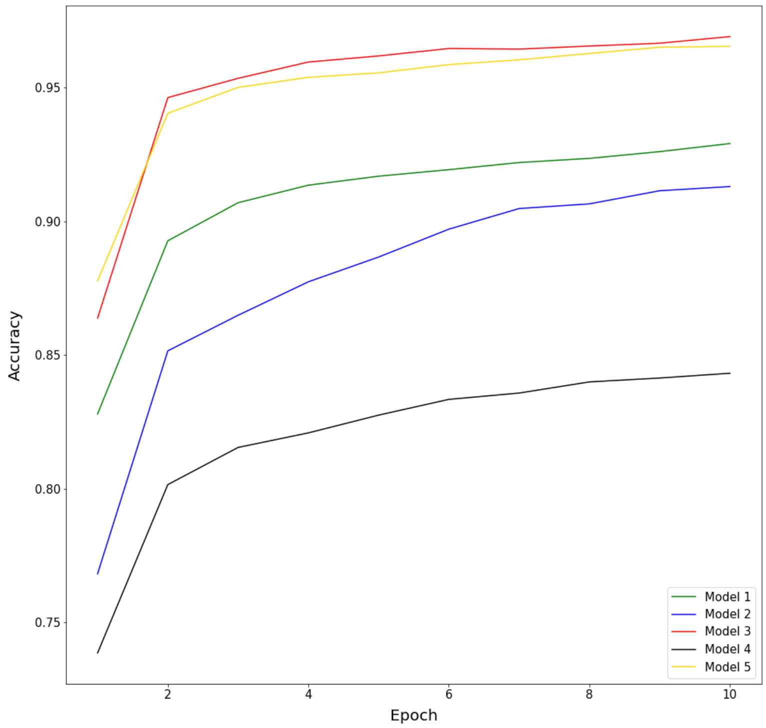
For the third CNN we added a new fully connected layer and the model is performing better than the base model:



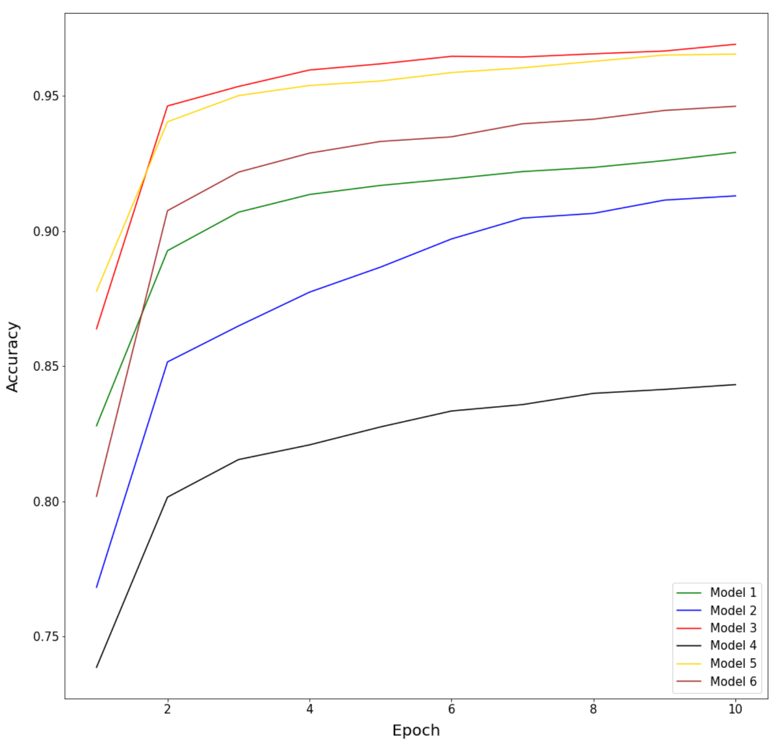
For the fourth CNN we changed the stride of convolution layer to 2 from 1 and the model is performing worse than the base model:



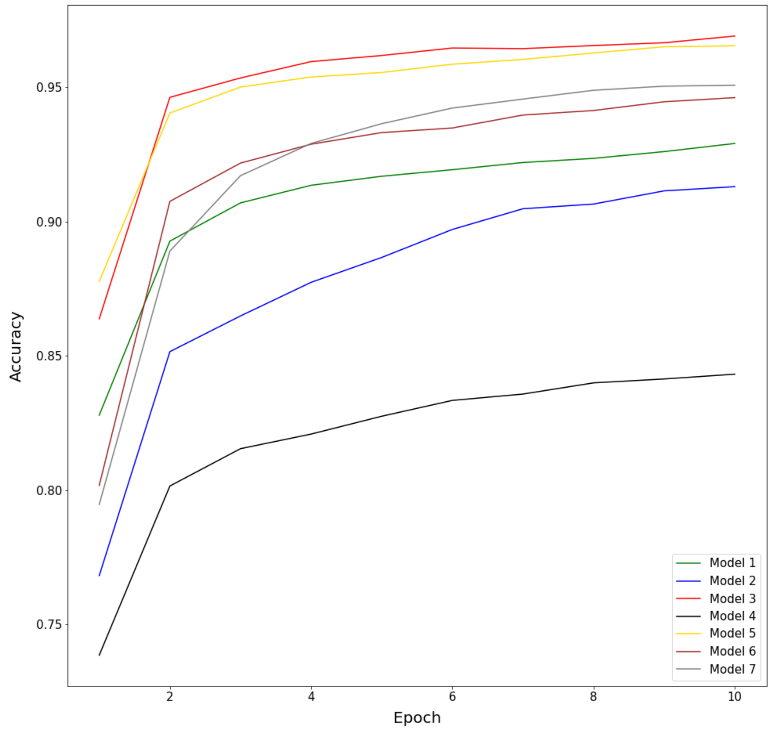
For the fifth CNN we changed the size of filters layer to 5 by 5 from 3 by 3 and the model is performing better than the base model:



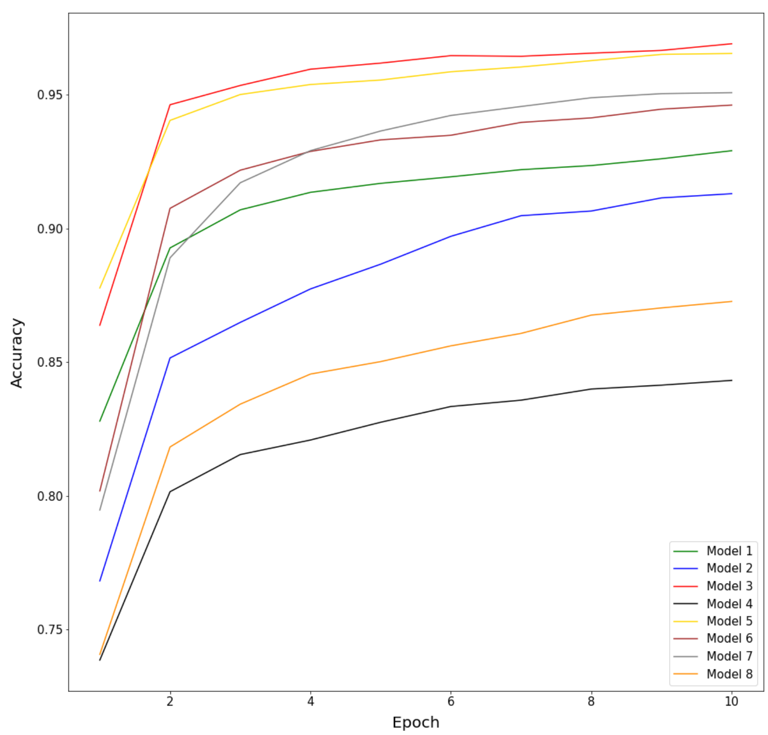
For the sixth CNN we changed the optimizer from Adam to RMSProp and the model is performing better than the base model:



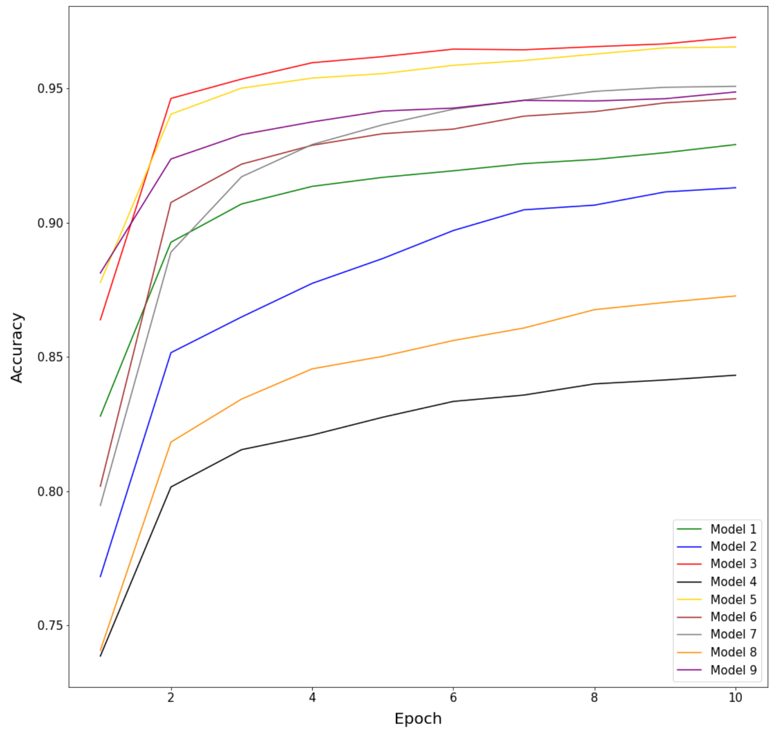
For the seventh CNN we changed the loss function to Mean Square Error from Binary Cross Entropy and the model is performing better than the base model:



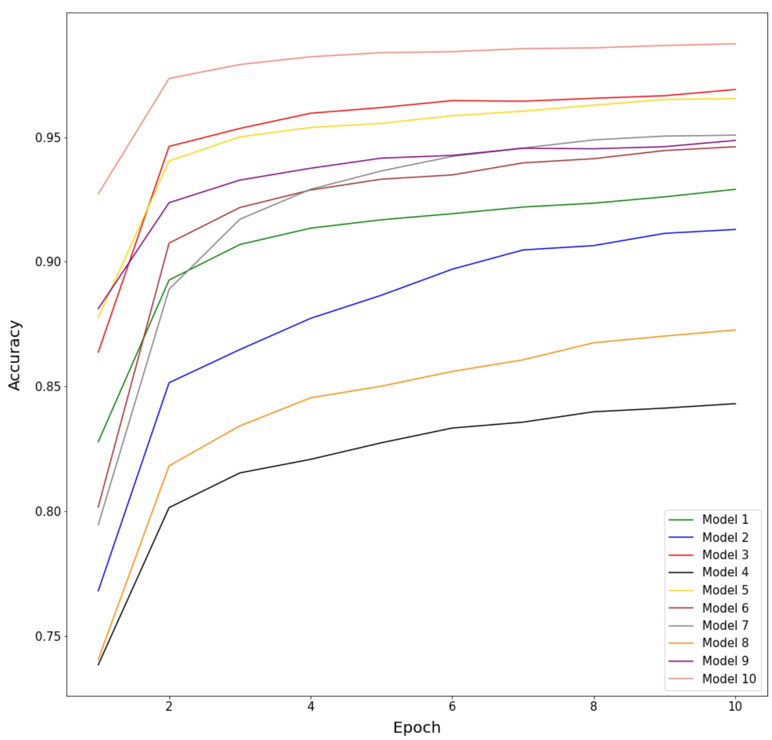
For the eighth CNN we changed the dropout rate to 0.5 from 0.2 and the model is performing worse than the base model:



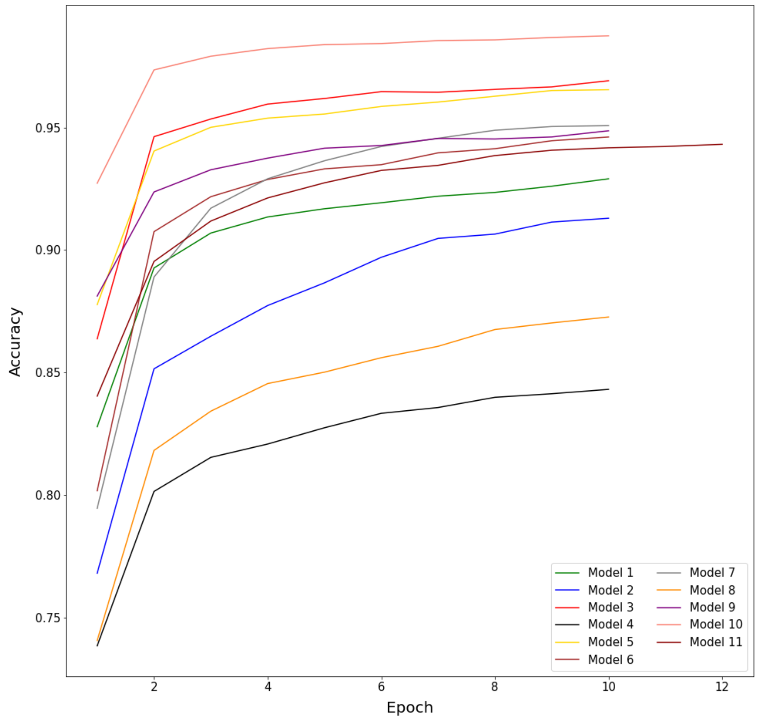
For the ninth CNN we changed the learning rate to 0.1 from 0.001 and the model is performing better than the base model:



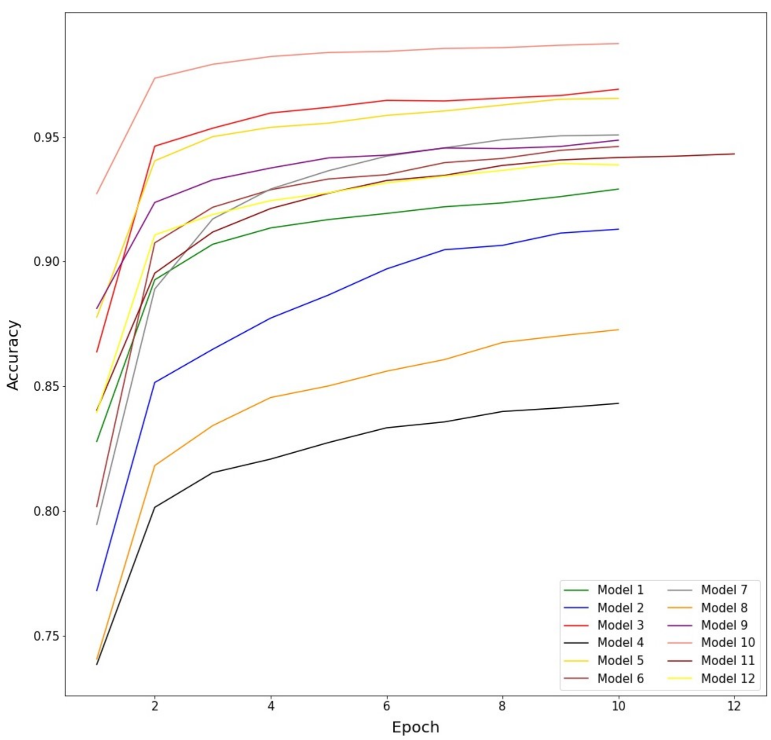
For the tenth CNN we changed the number of filters in the second convolution layer to 5 from 2 and the model is performing better than the base model (in fact this is the best model):



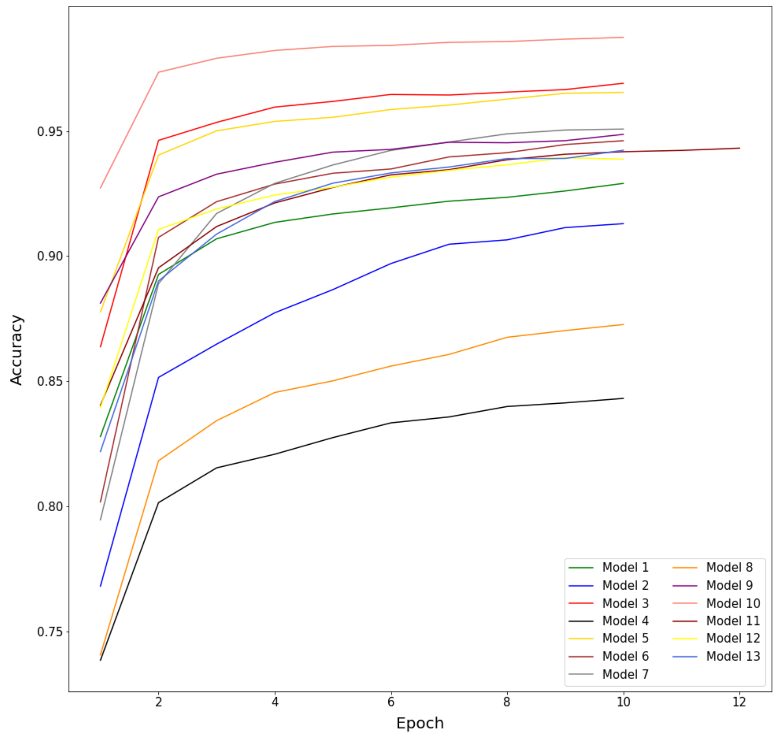
For the eleventh CNN we changed the number of epochs to 12 from 10 and the model is performing better than the base model:



For the twelfth CNN we added a batch normalization layer and the model is performing slightly better than the base model:



For the thirteenth CNN we changed the weight initializer to orthogonal from Glorot Normal and the model is performing worse than the base model:



This is the summary of models:
First model: 2 Conv-Maxpool + 2 Dense
Second model: 2 Conv-Maxpool + 1 Conv + 2 Dense
Third model: 2 Conv-Maxpool + 3 Dense
Fourth model: 2 Conv-Maxpool + Stride of 2 + 2 Dense
Fifth model: 2 Conv-Maxpool + Kernel of 5 by 5 + 2 Dense
Sixth model: 2 Conv-Maxpool + 2 Dense + RMSProp
Seventh model: 2 Conv-Maxpool + 2 Dense + MSE
Eighth model: 2 Conv-Maxpool + 2 Dense + Dropout 0.5
Ninth model: 2 Conv-Maxpool + 2 Dense + LR 0.01
Tenth model: 2 Conv-Maxpool (5 - 5) + 2 Dense
Eleventh model: 2 Conv-Maxpool + 2 Dense + Double Epoch
Twelfth model: 2 Conv-Maxpool + 2 Dense + Batch Normalization
Thirteenth model: 2 Conv-Maxpool + 2 Dense + Init

And this is the summary of loss and accuracy of models:
First model: Loss of final step: 0.18856, Accuracy of final step: 0.92905
Second model: Loss of final step: 0.22245, Accuracy of final step: 0.91298
Third model: Loss of final step: 0.08941, Accuracy of final step: 0.96903

Fourth model: Loss of final step: 0.38377, Accuracy of final step: 0.8432
Fifth model: Loss of final step: 0.09802, Accuracy of final step: 0.96539
Sixth model: Loss of final step: 0.14072, Accuracy of final step: 0.94610
Seventh model: Loss of final step: 0.03795, Accuracy of final step: 0.95074
Eighth model: Loss of final step: 0.29705, Accuracy of final step: 0.8727
Ninth model: Loss of final step: 0.14569, Accuracy of final step: 0.94863
Tenth model: Loss of final step: 0.0365, Accuracy of final step: 0.98736
Eleventh model: Loss of final step: 0.15341, Accuracy of final step: 0.94312
Twelfth model: Loss of final step: 0.16467, Accuracy of final step: 0.93874
Thirteenth model: Loss of final step: 0.15641, Accuracy of final step: 0.9423