Allison Roeser

Kaggle username: aroeser

Digit Recognizer: Neural Networks

Data Preparation, Exploration, Visualization:

The MNIST dataset contains 70,000 images of handwritten digits from 0 to 9. 42,000 of the images were provided a training that that included a label denoting which numerical digit each image represented. The test contained 28,000 unlabeled images. Figure 1 shows the first 5 observations of the train dataset. The first column is the label value and the other 784 columns represent the 784 pixels present on the the square image (28 pixels by 28 pixels). These 784 columns are binary to denote if the handwritten digit is contained in the pixel (value of 1) or if the pixel is bland (value of 0). Figure 2 & 3 show various images for the numbers 4 & 5. The digits are usually found centered in the image with blank space surrounding each digit. The amount of whitespace varies with each image.

Figure 4 displays the frequency of each digit in the training set. Digit 1 is most frequent with 4684 observations, and digit 5 is least frequent with 3,795 observations. The value counts for the digits are shown in Figure 5. Because each digit is roughly equally represented in the training set, frequency of occurrence should not have a strong impact on predicted value.

Implementation and Programming:

Full code is attached in the Appendix to show the specific implementation. The TensorFlow Keras package was leveraged to create four different Neural Network structures for digit identification. The Neural Networks differed in the number of layers and the number of nodes per layer. Time to fit each model was also measured in order to weight the tradeoff between improved accuracy from additional layers and nodes with the longer processing times generally required. Training, validation, and test sets were utilized to verify that the models generalized well. Confusion matrixes to were used to evaluate the classification accuracy of potential models.

Learning curve plots were used to illustrate the improvement in training and validation set accuracy as the number of epochs used in modeling increased. Pandas and Seaborn were leveraged for exploratory data analysis and visualization.

Review Research Design and Modeling Methods:

Four Neural Networks were created using the TensorFlow Keras API. The design of the experiment was to determine how differing the number of hidden layers and the number of nodes per layer impacted processing time and training / validation / testing accuracy. The model structures were: 2 hidden layers with 10 nodes per layer, 2 hidden layers with 40 nodes per layer, 6 hidden layers with 10 nodes per, and 6 hidden layers with 40 nodes per layer. Predictions from each model were submitted to Kaggle, and Kaggle provided a ranking score.

Each model was run for 10 epochs. Learning curve plots were created to illustrate how the training and validation accuracy improved with each epoch. Confusion matrixes were created for the training and validation sets to understand how well each model classified each of the ten digits.

Review Results, Evaluate Models:

The chart below displays the results of the four tests. All four neural networks were highly successful in correctly identifying the hand-written digit. Validation and test set accuracy scores

Model Number	Number of Hidden Layers	Nodes per Layer	Processing Time	Training Set Accuracy	Validation Set Accuracy	Test Set Accuracy
1	2	10	23.91	0.940	0.932	0.925
2	6	10	28.40	0.928	0.916	0.898
3	2	40	27.98	0.942	0.939	0.930
4	6	40	33.37	0.948	0.933	0.932

were very close to the training scores. These models generalized well and were not overfit to the training data. Increasing the number of hidden layers and the number of nodes per layers, both increased computation time. However, the computation time was under 40 seconds for all four modes. The most accurate model was Model 4 with six hidden layers and forty nodes per layer. Figures 6-9 display the learning curve plots for each model by epoch. The accuracy scores improved significantly thru the third epoch and then begin to plateau.

The confusion matrices for training and validation data for Model 4 are shown in Figures 10 & 11. 1,3, and 7 were the digits most accurately predicted. 5 was the digit most frequently predicted incorrectly. 5 was usually mistaken for a 3.

Exposition, Problem Description and Management Recommendations:

Increasing the number of layers and the number of nodes per layer, both produced more accurate results. The processing time does increase as model complexity increases; however, the time difference was short. A longer computation time is a valid tradeoff for improved accuracy. If the financial institution's primary goal is model accuracy, they should go with the most accurate model. For this study that was the model with the most layers (6) and the most nodes per layer (40). They may even want a model with even more layers / nodes for improved accuracy. The financial institution should quantify the accuracy score they are looking to achieve and understand how many layers and node combinations would be required to meet their goals.

Another item for them to consider is the size of their dataset. The processing of this dataset was relatively quick because there are fewer than 100,000 observations. If the financial institution's dataset is much larger, computation time will increase. An increase in computation time could change how they view the accuracy / processing time tradeoff.

Appendix

Figure 1: First five rows of the training dataset

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...
0	1	0	0	0	0	0	0	0	0	0	...
1	0	0	0	0	0	0	0	0	0	0	...
2	1	0	0	0	0	0	0	0	0	0	...
3	4	0	0	0	0	0	0	0	0	0	...
4	0	0	0	0	0	0	0	0	0	0	...

Figure 2: Four images of the handwritten digit 4

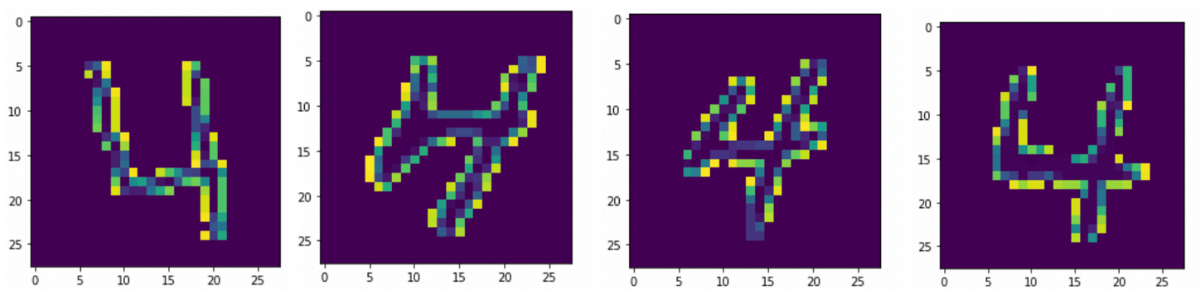


Figure 3: Four images of the handwritten digit 5

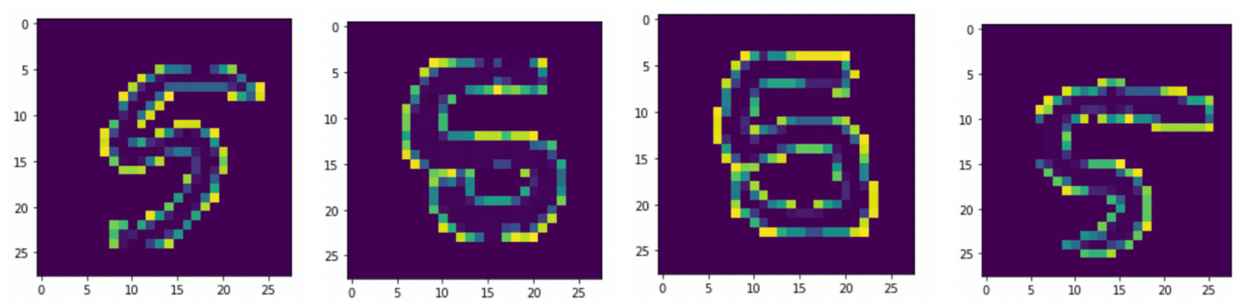


Figure 4: Frequency of each digit in the training set

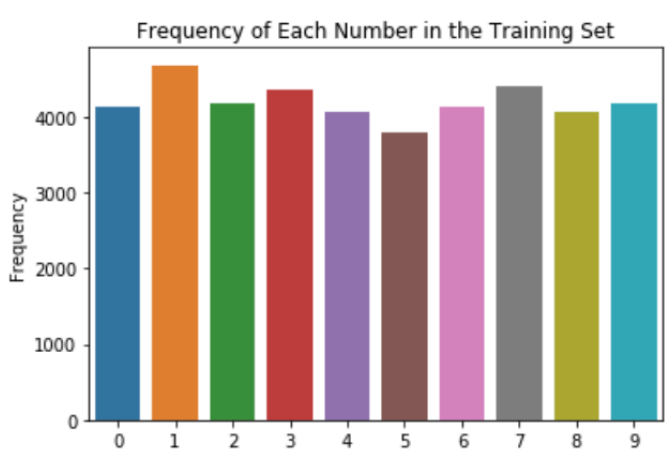


Figure 5. Value counts of each digit in the training set

1	4684
7	4401
3	4351
9	4188
2	4177
6	4137
0	4132
4	4072
8	4063
5	3795

Figure 6. Model 1 learning curve plot

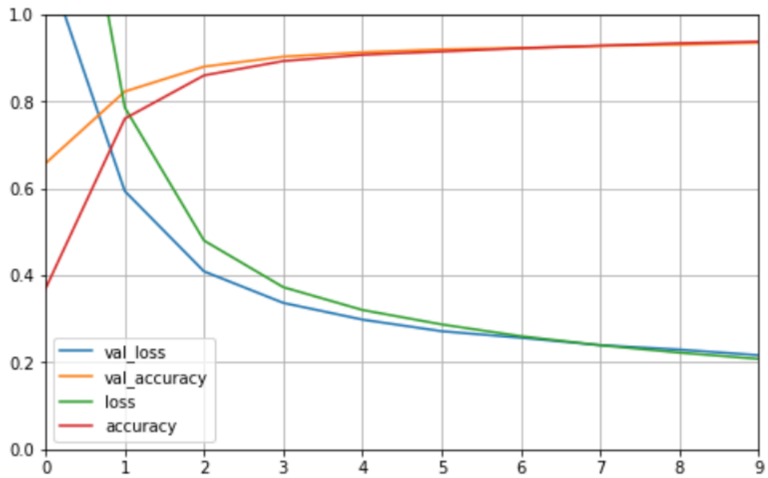


Figure 7. Model 2 learning curve plot

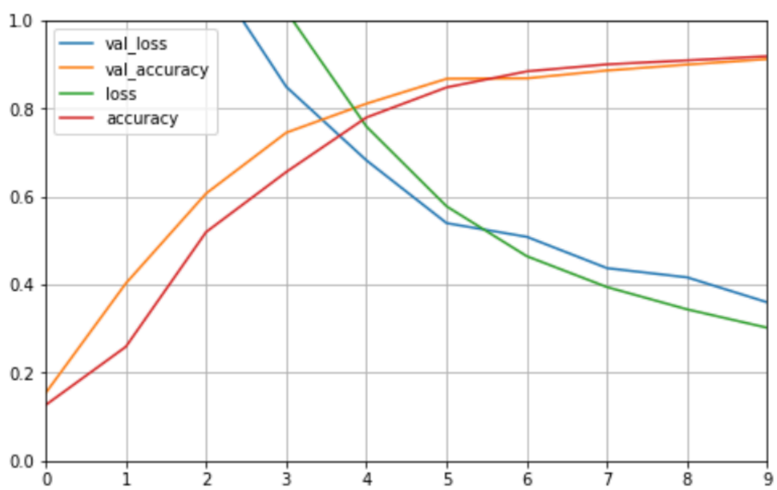


Figure 8. Model 3 learning curve plot

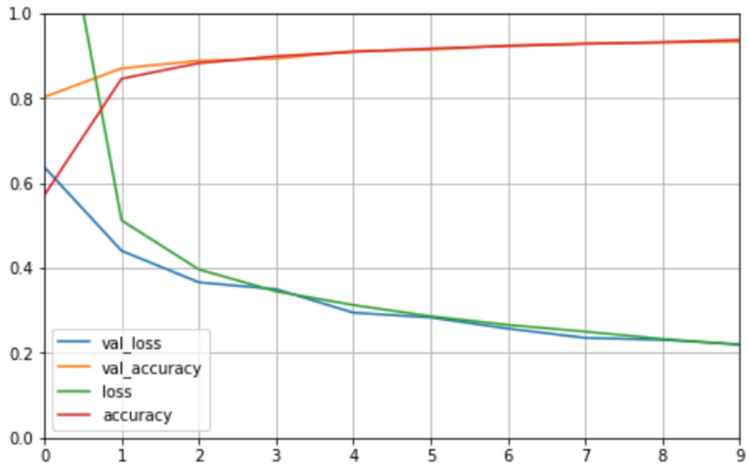


Figure 9. Model 4 learning curve plot

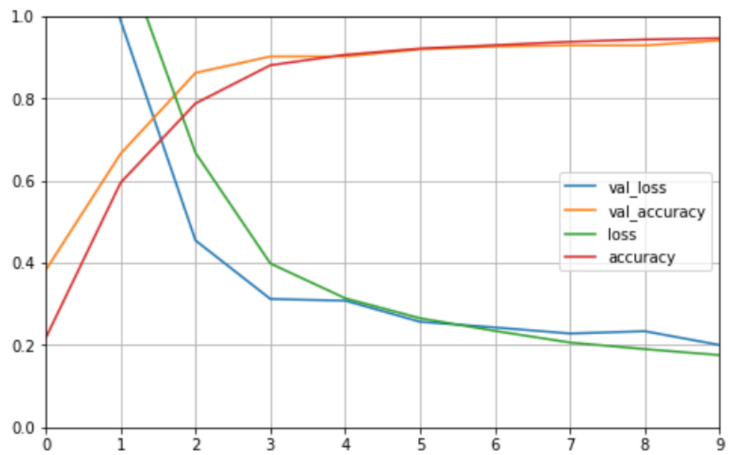


Figure 10. Model 4 Confusion Matrix (Training Set)

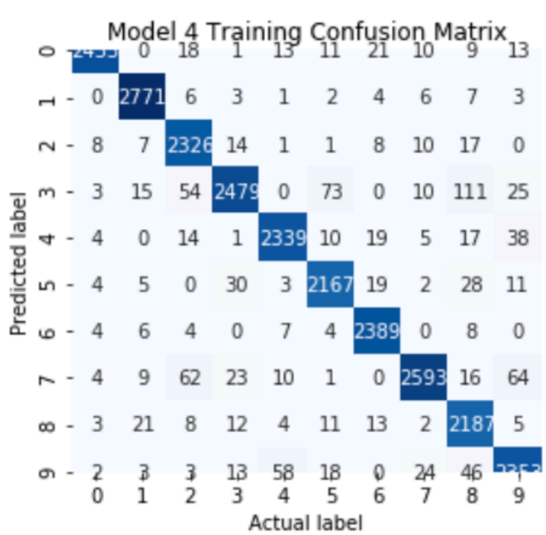
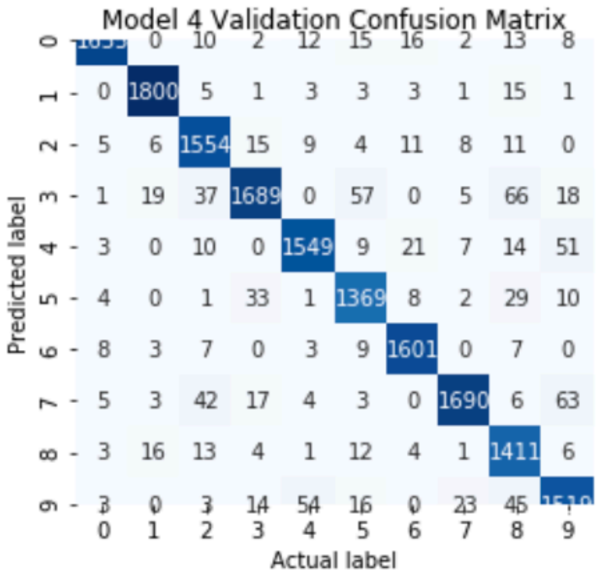


Figure 11. Model 4 Confusion Matrix (Validation Set)



In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score
from sklearn import metrics
import time
from sklearn.metrics import roc_auc_score, confusion_matrix, mean_squared_e
from sklearn.model_selection import GridSearchCV
import tensorflow as tf
from datetime import datetime
import os
import keras
import utils
```

```
Using TensorFlow backend.
```

In [2]:
```
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
```

In [3]:
```
dfX_train, dfX_valid, dfy_train, dfy_valid = train_test_split(train_df.drop
X_train=dfX_train.to_numpy()
X_valid=dfX_valid.to_numpy()
X_test = test_df.to_numpy()
y_train=dfy_train.to_numpy()
y_valid=dfy_valid.to_numpy()
```

In [4]:
```
#S4 Standard Functions
#Reset Graphs for Tensorboard
def reset_graph(seed=42):
    tf.reset_default_graph()
    tf.set_random_seed(seed)
    np.random.seed(seed)

#Save images to working directory
def save_fig(fig_id, tight_layout=True):
    path = os.path.join(work_dir, "images", chp_id, fig_id + ".png")
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format='png', dpi=300)

#Randomly Sort Batches
def shuffle_batch(X, y, batch_size):
    rnd_idx = np.random.permutation(len(X))
    n_batches = len(X) // batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_batch, y_batch = X[batch_idx], y[batch_idx]
        yield X_batch, y_batch

#S5 Load/Import data
mnist = tf.keras.datasets.mnist
mnist
#Segment in Train and Test
#(X_train, y_train),(X_test, y_test) = mnist.load_data()
X_train = X_train.astype(np.float32).reshape(-1, 28*28) / 255.0
X_valid = X_valid.astype(np.float32).reshape(-1, 28*28) / 255.0
X_test = X_test.astype(np.float32).reshape(-1, 28*28) / 255.0
y_train = y_train.astype(np.int32)
y_valid = y_valid.astype(np.int32)
#Scale Images
#X_train, X_test = X_train / 255.0, X_test / 255.0
print(X_train.shape)
print(X_valid.shape)
print(y_train.shape)
print(y_valid.shape)
```

```
(25200, 784)
(16800, 784)
(25200,)
(16800,)
```

In [ ]:

In [5]:
```
# Flatten the images
image_vector_size = 28*28
X_train = X_train.reshape(X_train.shape[0], image_vector_size)
X_valid = X_valid.reshape(X_valid.shape[0], image_vector_size)
```

In [6]:
```
X_valid.shape
```

Out[6]:
```
(16800, 784)
```

In [7]:
```
from keras.layers import Dense # Dense layers are "fully connected" layers
from keras.models import Sequential # Documentation: https://keras.io/model

image_size = 784 # 28*28
num_classes = 10 # ten unique digits

start_time = time.process_time()
model_1 = Sequential()

# The input layer requires the special input_shape parameter which should m
# the shape of our training data.
model_1.add(Dense(units=32, activation='relu', input_shape=(image_size,)))
model_1.add(Dense(units=num_classes, activation='relu'))
model_1.add(Dense(units=num_classes, activation='relu'))
model_1.add(Dense(units=num_classes, activation='softmax'))
model_1.summary()

model_1.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model_1.fit(x=X_train,y=y_train, epochs=10, validation_data = (X_

end_time = time.process_time()
model_1_time = round(end_time - start_time,2)
model_1_time
```

```
WARNING:tensorflow:From /Users/allisonroeser/opt/anaconda3/lib/python3.7/
site-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: c
alling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource
_variable_ops) with constraint is deprecated and will be removed in a fut
ure version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
_________________________________________________________________
dense_3 (Dense)              (None, 10)                110
_________________________________________________________________
dense_4 (Dense)              (None, 10)                110
=================================================================
Total params: 25,670
Trainable params: 25,670
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From /Users/allisonroeser/opt/anaconda3/lib/python3.7/
site-packages/keras/backend/tensorflow_backend.py:422: The name tf.global
_variables is deprecated. Please use tf.compat.v1.global_variables instea
d.

Train on 25200 samples, validate on 16800 samples
Epoch 1/10
```

```
25200/25200 [==============================] - 1s 40us/step - loss: 1.797
0 - accuracy: 0.3696 - val_loss: 1.1290 - val_accuracy: 0.6579
Epoch 2/10
25200/25200 [==============================] - 1s 39us/step - loss: 0.786
3 - accuracy: 0.7607 - val_loss: 0.5939 - val_accuracy: 0.8230
Epoch 3/10
25200/25200 [==============================] - 1s 40us/step - loss: 0.480
1 - accuracy: 0.8602 - val_loss: 0.4092 - val_accuracy: 0.8802
Epoch 4/10
25200/25200 [==============================] - 1s 37us/step - loss: 0.372
9 - accuracy: 0.8931 - val_loss: 0.3367 - val_accuracy: 0.9032
Epoch 5/10
25200/25200 [==============================] - 1s 37us/step - loss: 0.320
4 - accuracy: 0.9075 - val_loss: 0.2983 - val_accuracy: 0.9133
Epoch 6/10
25200/25200 [==============================] - 1s 42us/step - loss: 0.287
2 - accuracy: 0.9150 - val_loss: 0.2716 - val_accuracy: 0.9201
Epoch 7/10
25200/25200 [==============================] - 1s 40us/step - loss: 0.260
1 - accuracy: 0.9225 - val_loss: 0.2566 - val_accuracy: 0.9231
Epoch 8/10
25200/25200 [==============================] - 1s 43us/step - loss: 0.239
3 - accuracy: 0.9285 - val_loss: 0.2395 - val_accuracy: 0.9279
Epoch 9/10
25200/25200 [==============================] - 1s 39us/step - loss: 0.222
3 - accuracy: 0.9341 - val_loss: 0.2293 - val_accuracy: 0.9308
Epoch 10/10
25200/25200 [==============================] - 1s 41us/step - loss: 0.207
9 - accuracy: 0.9377 - val_loss: 0.2164 - val_accuracy: 0.9355
```

Out[7]: 23.75

In [8]:
```
#Score test dataset
scr=model_1.predict_classes(X_test)
#Conver array to Pandas dataframe with submission titles
pd_scr=pd.DataFrame(scr)
pd_scr.index.name = 'ImageId'
pd_scr.columns = ['label']
print(pd_scr)
#Export to Excel
pd_scr.to_excel("nn_1.xlsx")
```

```
         label
ImageId
0            2
1            0
2            9
3            9
4            3
...        ...
27995        9
27996        7
27997        3
27998        9
27999        2

[28000 rows x 1 columns]
```

In [9]:
```
model_1_train_acc = max(history.history["accuracy"])
model_1_train_acc
```

Out[9]:
```
0.9376587
```

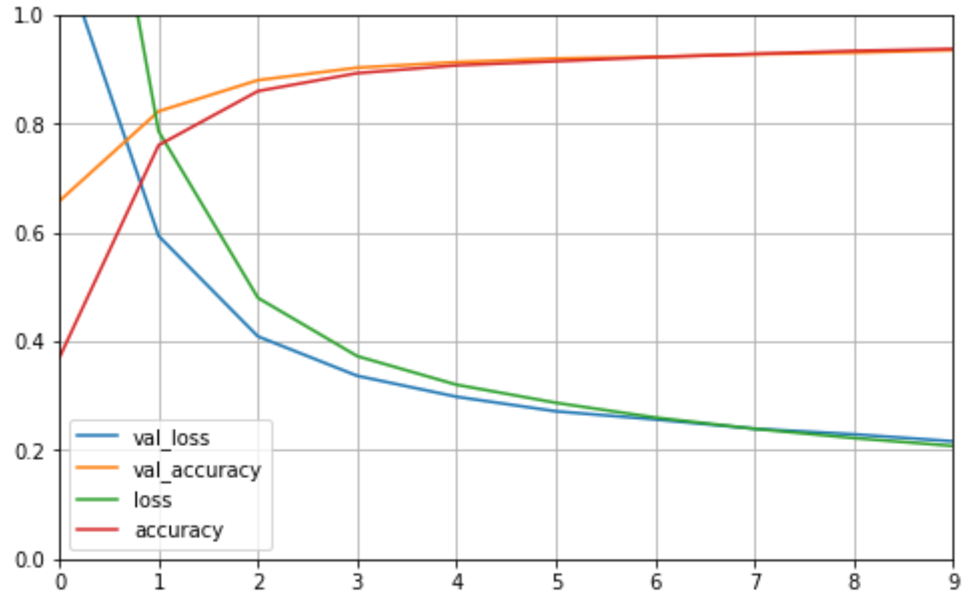
In [10]:
```
model_1_val_loss, model_1_val_acc = model_1.evaluate(X_valid, y_valid)
model_1_val_acc
```

```
16800/16800 [==============================] - 0s 12us/step
```

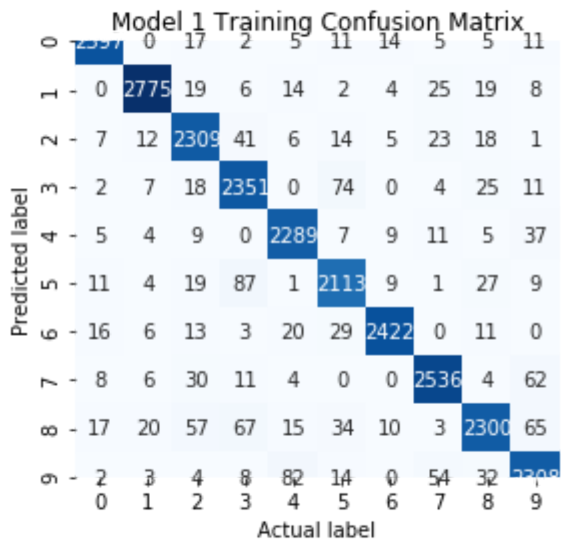
Out[10]:
```
0.935535728931427
```

In [11]:
```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
```

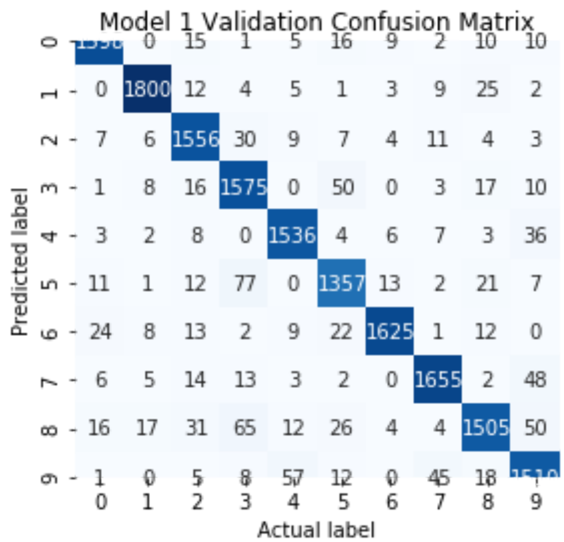
Out[11]: (0, 1)



In [49]:
```
#Plot Confusion Matrix DNN
cm_tst = confusion_matrix(y_train, model_1.predict_classes(X_train))
cm_tst_plt=sns.heatmap(cm_tst.T, square=True, annot=True, fmt='d', cbar=Fal
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 1 Training Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
fig3.savefig('TestCMHL2NPL300100.png',
        bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
        orientation='portrait', papertype=None, format=None,
        transparent=True, pad_inches=0.25)
```



In [50]:
```
#Plot Confusion Matrix DNN
cm_tst = confusion_matrix(y_valid, model_1.predict_classes(X_valid))
cm_tst_plt=sns.heatmap(cm_tst.T, square=True, annot=True, fmt='d', cbar=Fal
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 1 Validation Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
fig3.savefig('TestCMHL2NPL300100.png',
        bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
        orientation='portrait', papertype=None, format=None,
        transparent=True, pad_inches=0.25)
```



Model 2 (6 layers, 10 nodes per layer)

In [12]:
```
from keras.layers import Dense # Dense layers are "fully connected" layers
from keras.models import Sequential # Documentation: https://keras.io/model

image_size = 784 # 28*28
num_classes = 10 # ten unique digits

start_time = time.process_time()
model_2 = Sequential()

# The input layer requires the special input_shape parameter which should m
# the shape of our training data.
model_2.add(Dense(units=32, activation='relu', input_shape=(image_size,)))
model_2.add(Dense(units=num_classes, activation='relu'))
model_2.add(Dense(units=num_classes, activation='relu'))
model_2.add(Dense(units=num_classes, activation='relu'))
model_2.add(Dense(units=num_classes, activation='relu'))
model_2.add(Dense(units=num_classes, activation='relu'))
model_2.add(Dense(units=num_classes, activation='relu'))
model_2.add(Dense(units=num_classes, activation='softmax'))
model_2.summary()

model_2.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model_2.fit(x=X_train,y=y_train, epochs=10, validation_data = (X_

end_time = time.process_time()
model_2_time = round(end_time - start_time,2)
model_2_time
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_6 (Dense)              (None, 10)                330
_________________________________________________________________
dense_7 (Dense)              (None, 10)                110
_________________________________________________________________
dense_8 (Dense)              (None, 10)                110
_________________________________________________________________
dense_9 (Dense)              (None, 10)                110
_________________________________________________________________
dense_10 (Dense)             (None, 10)                110
_________________________________________________________________
dense_11 (Dense)             (None, 10)                110
_________________________________________________________________
dense_12 (Dense)             (None, 10)                110
=================================================================
Total params: 26,110
Trainable params: 26,110
Non-trainable params: 0
_________________________________________________________________
Train on 25200 samples, validate on 16800 samples
Epoch 1/10
```

```
25200/25200 [==============================] - 1s 56us/step - loss: 2.294
1 - accuracy: 0.1268 - val_loss: 2.2697 - val_accuracy: 0.1540
Epoch 2/10
25200/25200 [==============================] - 1s 45us/step - loss: 2.038
5 - accuracy: 0.2592 - val_loss: 1.6528 - val_accuracy: 0.4031
Epoch 3/10
25200/25200 [==============================] - 1s 46us/step - loss: 1.349
1 - accuracy: 0.5196 - val_loss: 1.1268 - val_accuracy: 0.6072
Epoch 4/10
25200/25200 [==============================] - 1s 42us/step - loss: 1.021
5 - accuracy: 0.6561 - val_loss: 0.8495 - val_accuracy: 0.7454
Epoch 5/10
25200/25200 [==============================] - 1s 44us/step - loss: 0.759
2 - accuracy: 0.7799 - val_loss: 0.6824 - val_accuracy: 0.8112
Epoch 6/10
25200/25200 [==============================] - 1s 42us/step - loss: 0.577
9 - accuracy: 0.8482 - val_loss: 0.5398 - val_accuracy: 0.8677
Epoch 7/10
25200/25200 [==============================] - 1s 42us/step - loss: 0.464
7 - accuracy: 0.8845 - val_loss: 0.5087 - val_accuracy: 0.8687
Epoch 8/10
25200/25200 [==============================] - 1s 42us/step - loss: 0.394
6 - accuracy: 0.9005 - val_loss: 0.4376 - val_accuracy: 0.8865
Epoch 9/10
25200/25200 [==============================] - 1s 42us/step - loss: 0.344
0 - accuracy: 0.9094 - val_loss: 0.4166 - val_accuracy: 0.9000
Epoch 10/10
25200/25200 [==============================] - 1s 42us/step - loss: 0.302
1 - accuracy: 0.9188 - val_loss: 0.3599 - val_accuracy: 0.9123
```

Out[12]: 27.41

In [13]:

```python
#Score test dataset
scr=model_2.predict_classes(X_test)
#Conver array to Pandas dataframe with submission titles
pd_scr=pd.DataFrame(scr)
pd_scr.index.name = 'ImageId'
pd_scr.columns = ['label']
print(pd_scr)
#Export to Excel
pd_scr.to_excel("nn_2.xlsx")
```

```
         label
ImageId
0            2
1            0
2            9
3            9
4            3
...        ...
27995        9
27996        7
27997        3
27998        9
27999        2

[28000 rows x 1 columns]
```

In [14]:
```
model_2_train_acc = max(history.history["accuracy"])
model_2_train_acc
```

Out[14]: 0.91884923

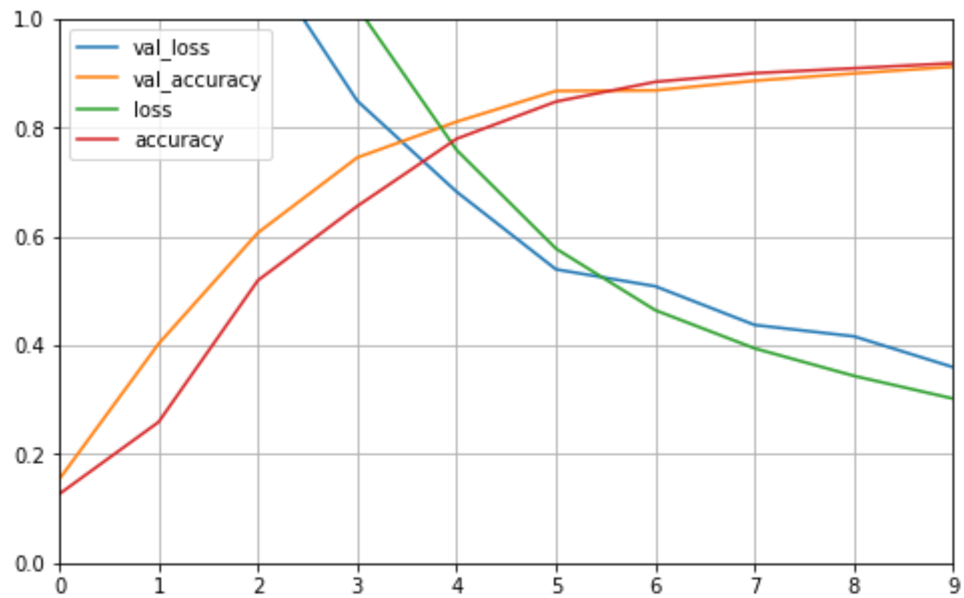
In [15]:
```
model_2_val_loss, model_2_val_acc = model_2.evaluate(X_valid, y_valid)
model_2_val_acc
```

```
16800/16800 [==============================] - 0s 14us/step
```

Out[15]: 0.9123214483261108

In [16]:
```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
```

Out[16]: (0, 1)



Model 3 (2 layers, 40 nodes per layer)

In [17]:
```
image_size = 784 # 28*28
num_classes = 40 # ten unique digits

start_time = time.process_time()
model_3 = Sequential()

# The input layer requires the special input_shape parameter which should m
# the shape of our training data.
model_3.add(Dense(units=32, activation='relu', input_shape=(image_size,)))
model_3.add(Dense(units=num_classes, activation='relu'))
model_3.add(Dense(units=num_classes, activation='relu'))
model_3.add(Dense(units=num_classes, activation='softmax'))
model_3.summary()

model_3.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model_3.fit(x=X_train,y=y_train, epochs=10, validation_data = (X_

end_time = time.process_time()
model_3_time = round(end_time - start_time,2)
model_3_time
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_13 (Dense)             (None, 32)                25120     
_________________________________________________________________
dense_14 (Dense)             (None, 40)                1320      
_________________________________________________________________
dense_15 (Dense)             (None, 40)                1640      
_________________________________________________________________
dense_16 (Dense)             (None, 40)                1640      
=================================================================
Total params: 29,720
Trainable params: 29,720
Non-trainable params: 0
_________________________________________________________________
Train on 25200 samples, validate on 16800 samples
Epoch 1/10
25200/25200 [==============================] - 1s 57us/step - loss: 1.499
9 - accuracy: 0.5717 - val_loss: 0.6380 - val_accuracy: 0.8032
Epoch 2/10
25200/25200 [==============================] - 1s 44us/step - loss: 0.512
4 - accuracy: 0.8463 - val_loss: 0.4408 - val_accuracy: 0.8706
Epoch 3/10
25200/25200 [==============================] - 1s 46us/step - loss: 0.396
8 - accuracy: 0.8828 - val_loss: 0.3664 - val_accuracy: 0.8886
Epoch 4/10
25200/25200 [==============================] - 1s 42us/step - loss: 0.345
1 - accuracy: 0.8994 - val_loss: 0.3502 - val_accuracy: 0.8935
Epoch 5/10
25200/25200 [==============================] - 1s 44us/step - loss: 0.312
8 - accuracy: 0.9093 - val_loss: 0.2948 - val_accuracy: 0.9115
Epoch 6/10
```

```
25200/25200 [==============================] - 1s 42us/step - loss: 0.286
4 - accuracy: 0.9173 - val_loss: 0.2835 - val_accuracy: 0.9148
Epoch 7/10
25200/25200 [==============================] - 1s 40us/step - loss: 0.266
0 - accuracy: 0.9235 - val_loss: 0.2573 - val_accuracy: 0.9242
Epoch 8/10
25200/25200 [==============================] - 1s 40us/step - loss: 0.250
2 - accuracy: 0.9284 - val_loss: 0.2355 - val_accuracy: 0.9292
Epoch 9/10
25200/25200 [==============================] - 1s 40us/step - loss: 0.232
9 - accuracy: 0.9321 - val_loss: 0.2308 - val_accuracy: 0.9318
Epoch 10/10
25200/25200 [==============================] - 1s 42us/step - loss: 0.219
3 - accuracy: 0.9378 - val_loss: 0.2203 - val_accuracy: 0.9338
```

Out[17]: 27.76

In [18]:

```
#Score test dataset
scr=model_3.predict_classes(X_test)
#Conver array to Pandas dataframe with submission titles
pd_scr=pd.DataFrame(scr)
pd_scr.index.name = 'ImageId'
pd_scr.columns = ['label']
print(pd_scr)
#Export to Excel
pd_scr.to_excel("nn_3.xlsx")
```

```
          label
ImageId
0             2
1             0
2             9
3             9
4             3
...         ...
27995         9
27996         7
27997         3
27998         9
27999         2

[28000 rows x 1 columns]
```

In [19]:

```
model_3_train_acc = max(history.history["accuracy"])
model_3_train_acc
```

Out[19]: 0.93777776

In [20]:

```
model_3_val_acc  = max(history.history["val_accuracy"])
model_3_val_acc
```

Out[20]: 0.9338095188140869

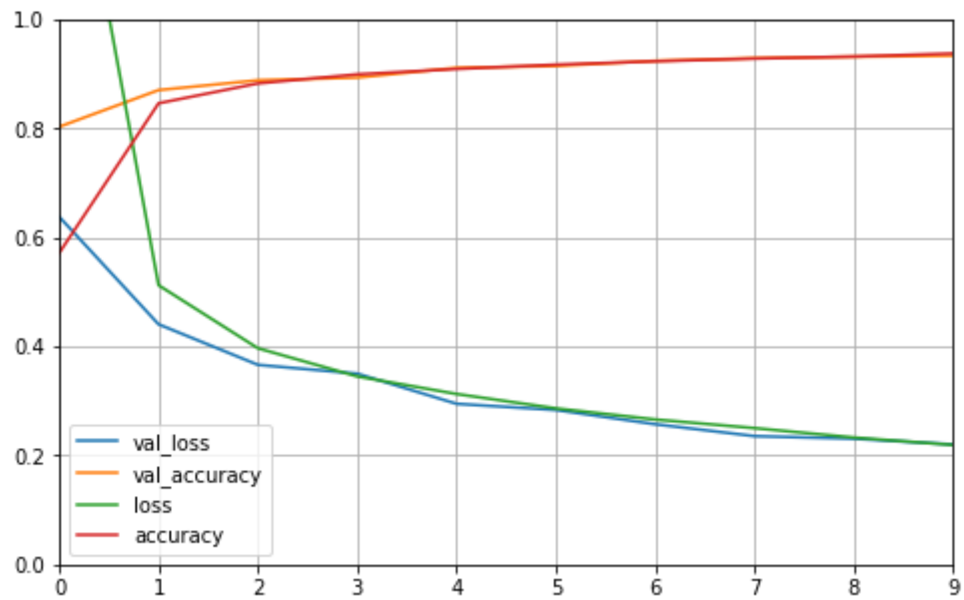
In [21]:
```
model_3_val_loss, model_3_val_acc = model_3.evaluate(X_valid, y_valid)
model_3_val_acc
```

```
16800/16800 [==============================] - 0s 12us/step
```

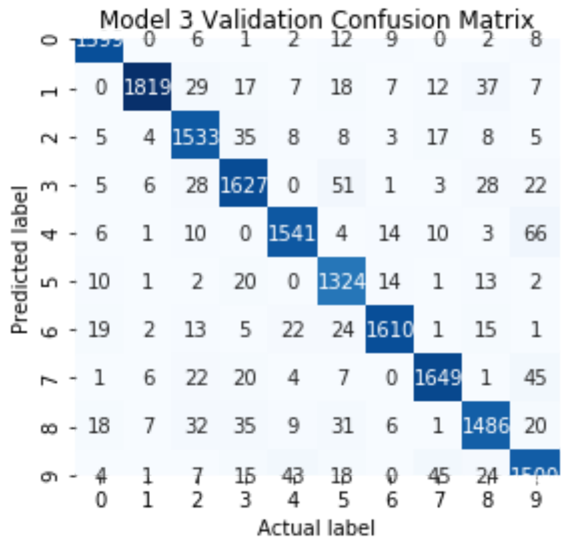
Out[21]:
```
0.9338095188140869
```

In [22]:
```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
```

Out[22]:
```
(0, 1)
```



In [23]:
```
#Plot Confusion Matrix DNN
cm_tst = confusion_matrix(y_valid, model_3.predict_classes(X_valid))
cm_tst_plt=sns.heatmap(cm_tst.T, square=True, annot=True, fmt='d', cbar=Fal
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 3 Validation Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
fig3.savefig('TestCMHL2NPL300100.png',
        bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
        orientation='portrait', papertype=None, format=None,
        transparent=True, pad_inches=0.25)
```



Model 4 (6 layers, 20 nodes per layer)

In [24]:
```
image_size = 784 # 28*28
num_classes = 40 # ten unique digits

start_time = time.process_time()
model_4 = Sequential()

# The input layer requires the special input_shape parameter which should m
# the shape of our training data.
model_4.add(Dense(units=32, activation='relu', input_shape=(image_size,)))
model_4.add(Dense(units=num_classes, activation='relu'))
model_4.add(Dense(units=num_classes, activation='relu'))
model_4.add(Dense(units=num_classes, activation='relu'))
model_4.add(Dense(units=num_classes, activation='relu'))
model_4.add(Dense(units=num_classes, activation='relu'))
model_4.add(Dense(units=num_classes, activation='relu'))
model_4.add(Dense(units=num_classes, activation='softmax'))
model_4.summary()

model_4.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model_4.fit(x=X_train,y=y_train, epochs=10, validation_data = (X_

end_time = time.process_time()
model_4_time = round(end_time - start_time,2)
model_4_time
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 32)                25120
_________________________________________________________________
dense_18 (Dense)             (None, 40)                1320
_________________________________________________________________
dense_19 (Dense)             (None, 40)                1640
_________________________________________________________________
dense_20 (Dense)             (None, 40)                1640
_________________________________________________________________
dense_21 (Dense)             (None, 40)                1640
_________________________________________________________________
dense_22 (Dense)             (None, 40)                1640
_________________________________________________________________
dense_23 (Dense)             (None, 40)                1640
_________________________________________________________________
dense_24 (Dense)             (None, 40)                1640
=================================================================
Total params: 36,280
Trainable params: 36,280
Non-trainable params: 0
_________________________________________________________________
Train on 25200 samples, validate on 16800 samples
Epoch 1/10
25200/25200 [==============================] - 2s 63us/step - loss: 2.357
6 - accuracy: 0.2172 - val_loss: 1.5857 - val_accuracy: 0.3826
Epoch 2/10
25200/25200 [==============================] - 1s 46us/step - loss: 1.172
```

```
9 - accuracy: 0.5952 - val_loss: 0.9896 - val_accuracy: 0.6659
Epoch 3/10
25200/25200 [==============================] - 1s 47us/step - loss: 0.667
9 - accuracy: 0.7882 - val_loss: 0.4547 - val_accuracy: 0.8624
Epoch 4/10
25200/25200 [==============================] - 1s 45us/step - loss: 0.398
6 - accuracy: 0.8811 - val_loss: 0.3123 - val_accuracy: 0.9024
Epoch 5/10
25200/25200 [==============================] - 1s 45us/step - loss: 0.313
9 - accuracy: 0.9065 - val_loss: 0.3082 - val_accuracy: 0.9024
Epoch 6/10
25200/25200 [==============================] - 1s 45us/step - loss: 0.265
4 - accuracy: 0.9217 - val_loss: 0.2567 - val_accuracy: 0.9199
Epoch 7/10
25200/25200 [==============================] - 1s 45us/step - loss: 0.234
5 - accuracy: 0.9299 - val_loss: 0.2430 - val_accuracy: 0.9267
Epoch 8/10
25200/25200 [==============================] - 1s 45us/step - loss: 0.206
3 - accuracy: 0.9382 - val_loss: 0.2283 - val_accuracy: 0.9294
Epoch 9/10
25200/25200 [==============================] - 1s 45us/step - loss: 0.190
3 - accuracy: 0.9436 - val_loss: 0.2339 - val_accuracy: 0.9292
Epoch 10/10
25200/25200 [==============================] - 1s 44us/step - loss: 0.175
7 - accuracy: 0.9466 - val_loss: 0.2002 - val_accuracy: 0.9415
```

Out[24]: 33.13

In [25]:

```
#Score test dataset
scr=model_4.predict_classes(X_test)
#Conver array to Pandas dataframe with submission titles
pd_scr=pd.DataFrame(scr)
pd_scr.index.name = 'ImageId'
pd_scr.columns = ['label']
print(pd_scr)
#Export to Excel
pd_scr.to_excel("nn_4.xlsx")
```

```
         label
ImageId
0            2
1            0
2            9
3            9
4            3
...        ...
27995        9
27996        7
27997        3
27998        9
27999        2

[28000 rows x 1 columns]
```

In [26]:
```
model_4_train_acc = max(history.history["accuracy"])
model_4_train_acc
```

Out[26]:
```
0.94662696
```

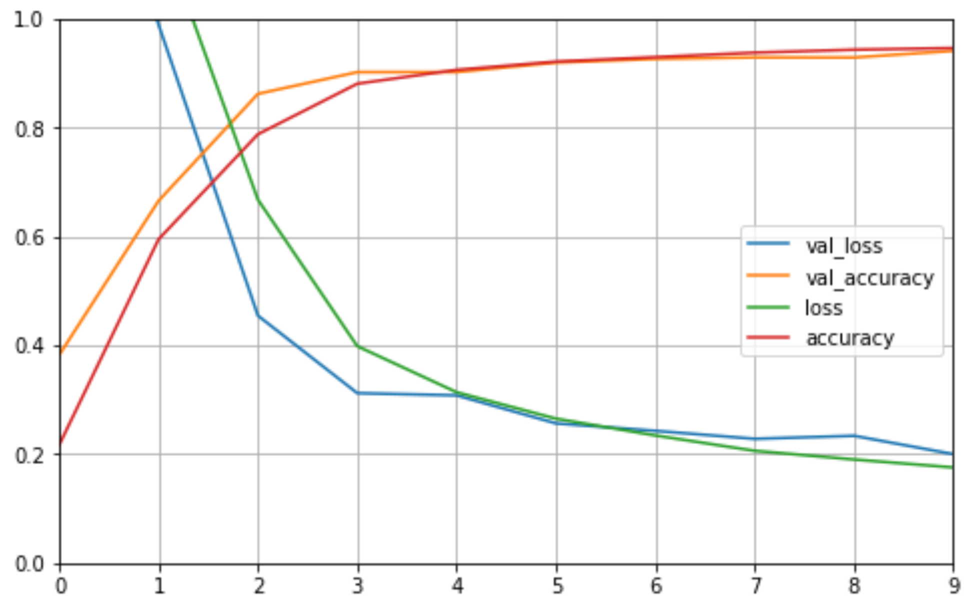
In [27]:
```
model_4_val_loss, model_4_val_acc = model_4.evaluate(X_valid, y_valid)
model_4_val_acc
```

```
16800/16800 [==============================] - 0s 15us/step
```

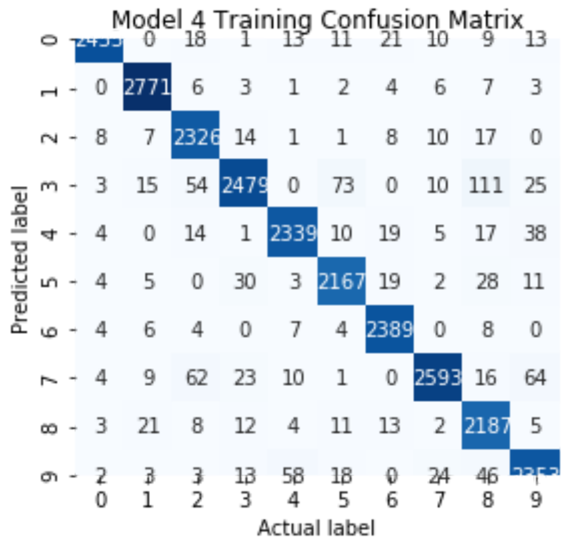
Out[27]:
```
0.9414880871772766
```

In [28]:
```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
```

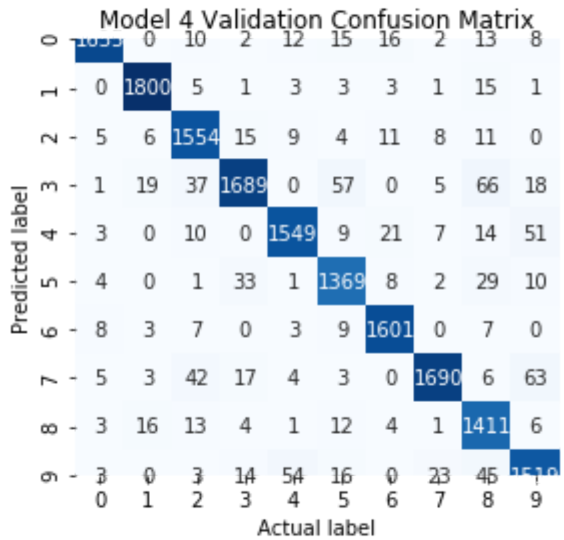
Out[28]:
```
(0, 1)
```



In [48]:
```
#Plot Confusion Matrix DNN
cm_tst = confusion_matrix(y_train, model_4.predict_classes(X_train))
cm_tst_plt=sns.heatmap(cm_tst.T, square=True, annot=True, fmt='d', cbar=Fal
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 4 Training Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
fig3.savefig('TestCMHL2NPL300100.png',
        bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
        orientation='portrait', papertype=None, format=None,
        transparent=True, pad_inches=0.25)
```



In [46]:
```
#Plot Confusion Matrix DNN
cm_tst = confusion_matrix(y_valid, model_4.predict_classes(X_valid))
cm_tst_plt=sns.heatmap(cm_tst.T, square=True, annot=True, fmt='d', cbar=Fal
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 4 Validation Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
fig3.savefig('TestCMHL2NPL300100.png',
        bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
        orientation='portrait', papertype=None, format=None,
        transparent=True, pad_inches=0.25)
```



In [29]:
```
# Kaggle scores
model_1_test_acc = 0.925
model_2_test_acc = 0.898
model_3_test_acc = 0.930
model_4_test_acc = 0.932
```

Summary DataFrame

In [30]:
```
data = [[1,2, 10,round(model_1_time,2), round(model_1_train_acc,3), round(m
        [2,6, 10,round(model_2_time,2), round(model_2_train_acc,3), round(m
        [3,2, 40,round(model_3_time,2), round(model_3_train_acc,3), round(m
        [4,6, 40,round(model_4_time,2), round(model_4_train_acc,3), round(m
df = pd.DataFrame(data, columns =
                   ['Model Number' ,'Number of Hidden Layers', 'Nodes per La
                    'Training Set Accuracy', "Validation Set Accuracy", "Tes
df
```

Out[30]:

	Model Number	Number of Hidden Layers	Nodes per Layer	Processing Time	Training Set Accuracy	Validation Set Accuracy	Test Set Accuracy
0	1	2	10	23.75	0.938	0.936	0.925
1	2	6	10	27.41	0.919	0.912	0.898
2	3	2	40	27.76	0.938	0.934	0.930
3	4	6	40	33.13	0.947	0.941	0.932

In [ ]:

In [31]:
```
history.params
```

Out[31]:
```
{'batch_size': 32,
 'epochs': 10,
 'steps': None,
 'samples': 25200,
 'verbose': 1,
 'do_validation': True,
 'metrics': ['loss', 'accuracy', 'val_loss', 'val_accuracy']}
```

In [32]:
```
print(history.epoch)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [33]:
```
model_1.evaluate(X_valid, y_valid)
```

```
16800/16800 [==============================] - 0s 16us/step
```

Out[33]:
```
[0.2163638901213805, 0.935535728931427]
```

In [44]:
```
X_new = X_test[:5]
y_proba = model_4.predict_classes(X_new)
y_proba.round(2)
```

Out[44]:
```
array([2, 0, 9, 9, 3])
```

In [45]:
```
y_proba.round(2)[0]
```

Out[45]:
```
2
```

In [ ]:

In [ ]:

In [5]:
```
# Code from Chris' TA session utlized and modified
```

EDA

In [229]:
```
train_df.shape
```

Out[229]:
```
(42000, 785)
```

In [230]:
```
test_df.shape
```

Out[230]:
```
(28000, 784)
```

In [231]:
```
train_df.head()
```

Out[231]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775
0	1	0	0	0	0	0	0	0	0	0	...	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0

5 rows × 785 columns

In [232]:
```
#Split training data
X_train, X_valid, y_train, y_valid = train_test_split(train_df.drop(['label
                                                      train_size=.6, random_s
# Check the shape of the trainig data set array
print('Shape of X_train_data:', X_train.shape)
print('Shape of y_train_data:', y_train.shape)
print('Shape of X_valid_data:', X_valid.shape)
print('Shape of y_valid_data:', y_valid.shape)
```

```
Shape of X_train_data: (25200, 784)
Shape of y_train_data: (25200,)
Shape of X_valid_data: (16800, 784)
Shape of y_valid_data: (16800,)
```

In [233]:
```
y_test = pd.DataFrame(data = test_df, columns = ["label"])
```

In [234]:
```
X_test = pd.DataFrame(test_df)
```

In [235]:
```
X_train.head()
```

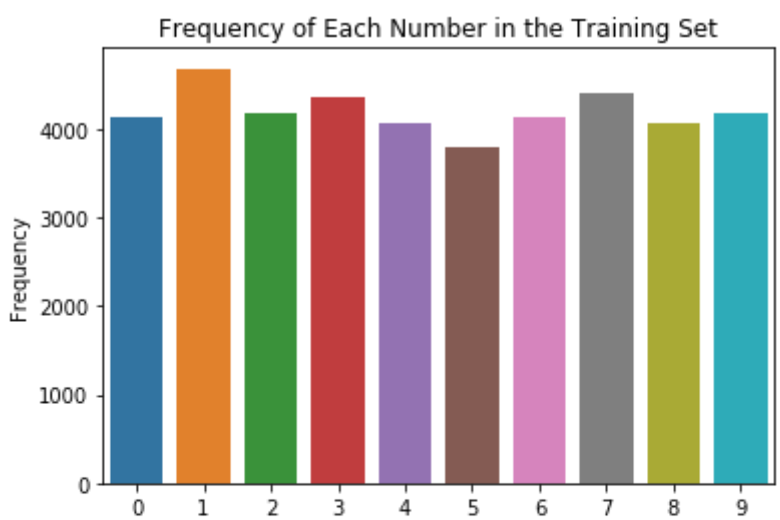
Out[235]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel'
25153	0	0	0	0	0	0	0	0	0	0	...	0	
14350	0	0	0	0	0	0	0	0	0	0	...	0	
24843	0	0	0	0	0	0	0	0	0	0	...	0	
6282	0	0	0	0	0	0	0	0	0	0	...	0	
41796	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 784 columns

In [236]:
```
lab_count = list(train_df.groupby('label')['label'].count())
index = list(train_df.groupby('label')['label'].count().index)

lab_plot = sns.barplot(y=lab_count,
                       x = index)
plt.ylabel("Frequency")
plt.title("Frequency of Each Number in the Training Set")
```

Out[236]:
```
Text(0.5, 1.0, 'Frequency of Each Number in the Training Set')
```



In [237]:
```
freq = train_df["label"].value_counts()
freq
```

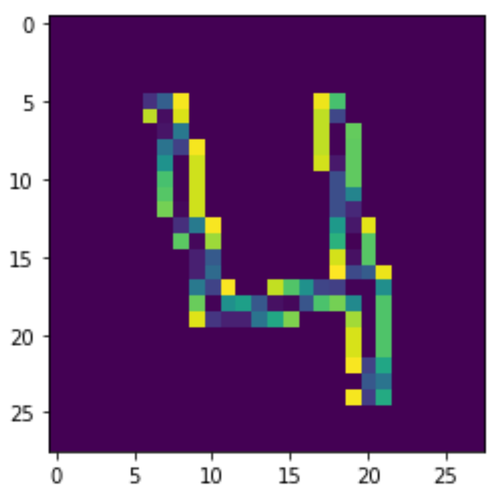
Out[237]:
```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```

In [238]:
```
train_4 = train_df[train_df['label'] == 4]
```

In [239]:
```
train_5 = train_df[train_df['label'] == 5]
```

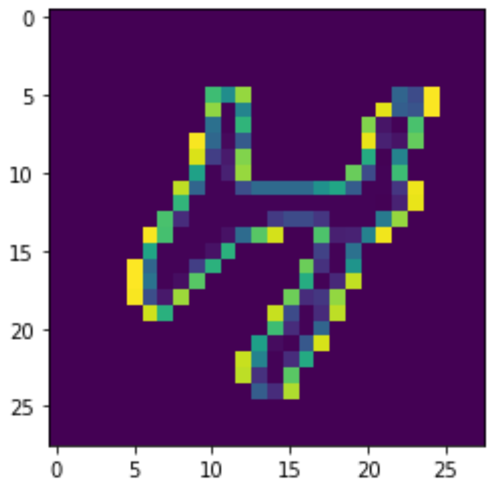
In [240]:
```
def gen_image(arr):
    two_d = (np.reshape(arr, (28,28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation = 'nearest')
    return plt

gen_image(train_4.iloc[0,1:].values)
```

Out[240]:
```
<module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/
python3.7/site-packages/matplotlib/pyplot.py'>
```



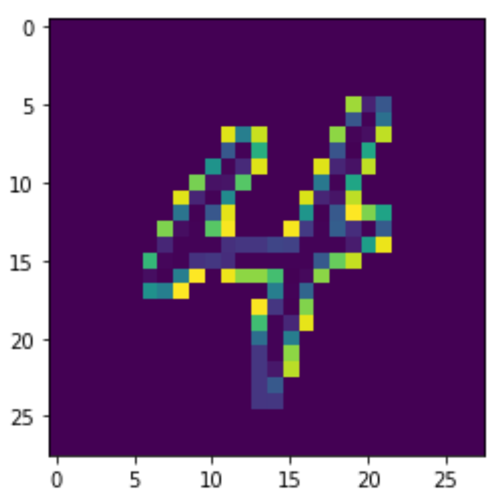
In [241]:
```
gen_image(train_4.iloc[1,1:].values)
```

Out[241]:
```
<module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>
```



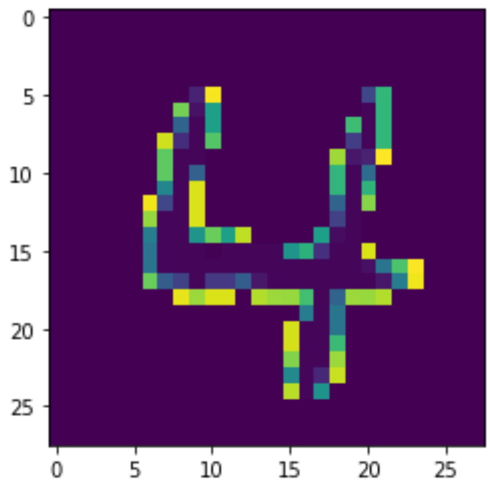
In [242]:
```
gen_image(train_4.iloc[2,1:].values)
```

Out[242]:
```
<module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>
```



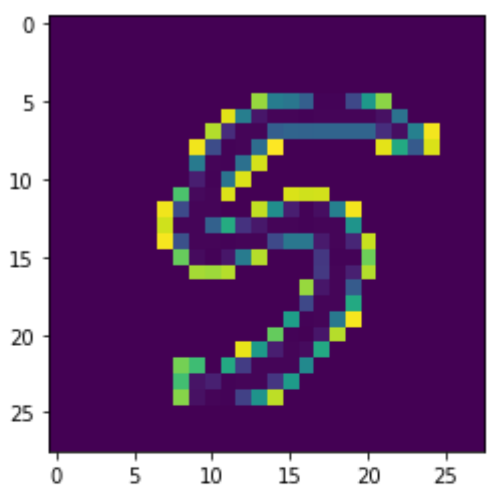
In [243]:
```
gen_image(train_4.iloc[3,1:].values)
```

Out[243]:
```
<module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>
```



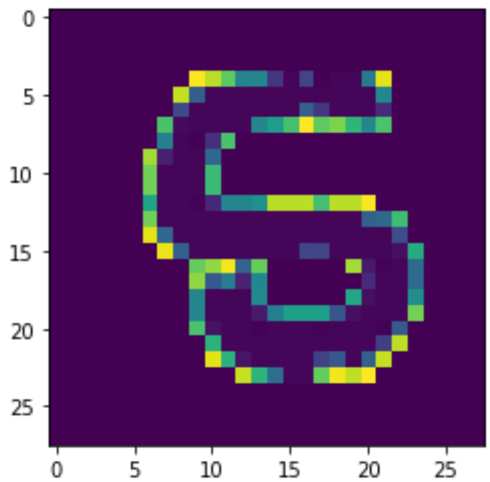
In [244]:
```
gen_image(train_5.iloc[0,1:].values)
```

Out[244]:
```
<module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>
```



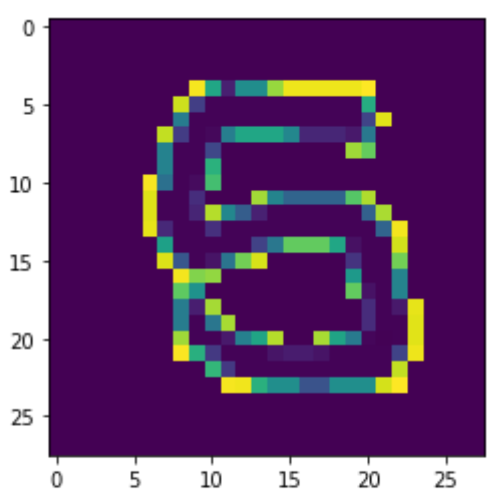
In [245]:
```
gen_image(train_5.iloc[1,1:].values)
```

Out[245]:
```
<module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>
```



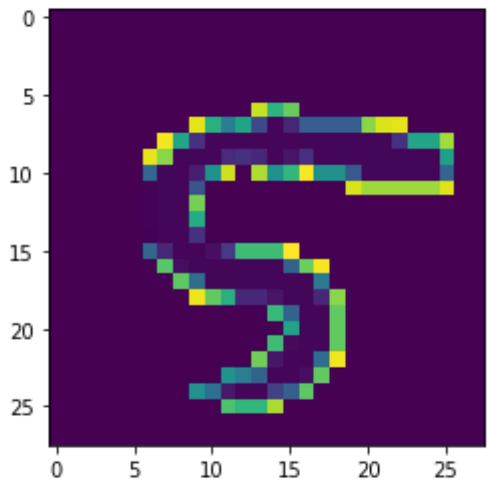
In [246]:
```
gen_image(train_5.iloc[2,1:].values)
```

Out[246]:
```
<module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>
```



In [247]:
```
gen_image(train_5.iloc[3,1:].values)
```

Out[247]:
```
<module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>
```



Random Forest Model