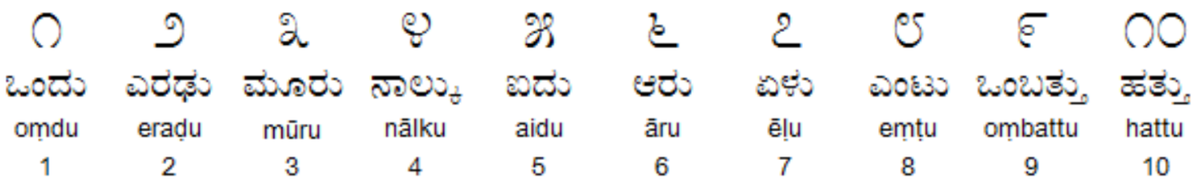
Kannada MNIST Kaggle Competition

Bored of MNIST?

The goal of this competition is to provide a simple extension to the classic MNIST competition we're all familiar with. Instead of using Arabic numerals, it uses a recently-released dataset of Kannada digits.

Kannada is a language spoken predominantly by people of Karnataka in southwestern India. The language has roughly 45 million native speakers and is written using the Kannada script.



This competition uses the same format as the MNIST competition in terms of how the data is structured, but it's different in that it is a synchronous re-run Kernels competition. You write your code in a Kaggle Notebook, and when you submit the results, your code is scored on both the public test set, as well as a private (unseen) test set.

Requirements

1) Conduct your analysis using a cross-validation design.
2) Conduct / refine EDA.
3) Conduct Design of Experiments to evaluate the performance of various neural networks by changing the layers and nodes. Tested neural network structures should be explored within a benchmark experiment, a 2x2 completely crossed design. An example of a completely crossed designed with {2, 5} layers and {10,20} nodes follows:

Layers	Nodes	Time	Training Accuracy	Testing Accuracy
2	10	63.61	0.935	0.927
2	20	115.25	0.967	0.952
5	10	74.28	0.944	0.933
5	20	75.1	0.964	0.952

4) Due to the time required to fit each neural network, we will observe only one trial for each

cell in the design.

5) You will build your models on csv and submit your forecasts for test.csv to Kaggle.com, providing your name and user ID for each experimental trial.

6) Evaluate goodness of fit metrics on the training and validation sets.

7) Provide a multi-class confusion matrix.

8) Discuss how your models performed.

Data Loading and Preparation

In [1]:

```
# import modules
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
import re
import numpy as np
from datetime import datetime
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix, accuracy
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RepeatedStratifiedKFold

import tensorflow as tf
from tensorflow.python import keras
# from tensorflow.keras import models
# from tensorflow.keras import layers
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import LeakyReLU, Dense, Dropout, Flatten, Conv2
from tensorflow.keras.callbacks import ReduceLROnPlateau, LearningRateSchedul
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential

# Figures inline and set visualization style
%matplotlib inline
sns.set()

# setting seed to control for randomness
np.random.seed(42)
```

In [2]:

```
# import train data
train = pd.read_csv("train.csv")
# train = pd.read_csv("../input/Kannada-MNIST/train.csv")
train.head()
```

Out[2]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixe
0	0	0	0	0	0	0	0	0	0	0	...	0	
1	1	0	0	0	0	0	0	0	0	0	...	0	
2	2	0	0	0	0	0	0	0	0	0	...	0	

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixe
3	3	0	0	0	0	0	0	0	0	0	...	0	
4	4	0	0	0	0	0	0	0	0	0	...	0	

Since there's no way to submit to Kaggle (the competition is closed), I'll use the csv file called `Dig-MNIST.csv` to evaluate our model. We'll make this our new test2 set.

In [3]:
```
# import real test data
test = pd.read_csv("test.csv")
# test = pd.read_csv("../input/Kannada-MNIST/test.csv")
test.head()
```

Out[3]:

	id	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel77
0	0	0	0	0	0	0	0	0	0	0	...	0	
1	1	0	0	0	0	0	0	0	0	0	...	0	
2	2	0	0	0	0	0	0	0	0	0	...	0	
3	3	0	0	0	0	0	0	0	0	0	...	0	
4	4	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 785 columns

In [4]:
```
# import dig data that we will use for evaluation
dig = pd.read_csv("Dig-MNIST.csv")
# dig = pd.read_csv("../input/Kannada-MNIST/Dig-MNIST.csv")
dig.head()
```

Out[4]:

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

5 rows × 785 columns

In [5]:
```
print(train.shape, test.shape, dig.shape)
```

```
(60000, 785) (5000, 785) (10240, 785)
```

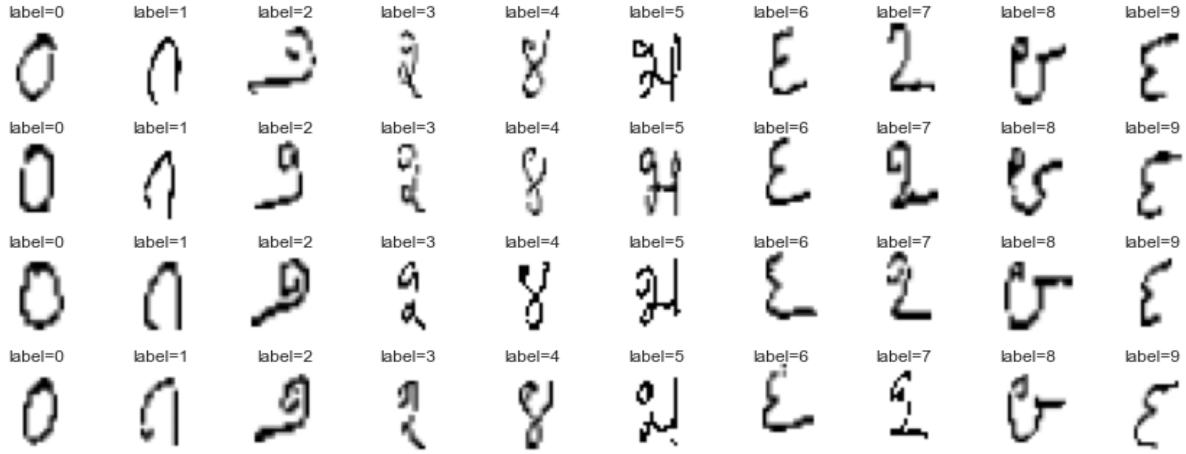
I'll split the data for ease of interpretation: X_train and y_train. X_train is the dataframe containing the features, from pixel0 to pixel783. y_train is a NumPy array containing the labels from the training set. We'll do the same with the dig test set.

In [6]:
```
# splitting train into X_train, y_train
X_train = train.drop(['label'], axis=1)
y_train = np.array(train.label)

# splitting test into X_dig, y_dig
X_dig = dig.drop(['label'], axis=1)
y_dig = np.array(dig.label)

# removing 'id' column from test set
test_id = test.id
test.drop(['id'], axis=1, inplace=True)

# getting y_data to be a concatenation of y_train and y_test
y_data = np.append(y_train, y_dig)
assert len(y_data) == (len(y_train) + len(y_dig)), "wrong size for y_data"
```

Light EDA

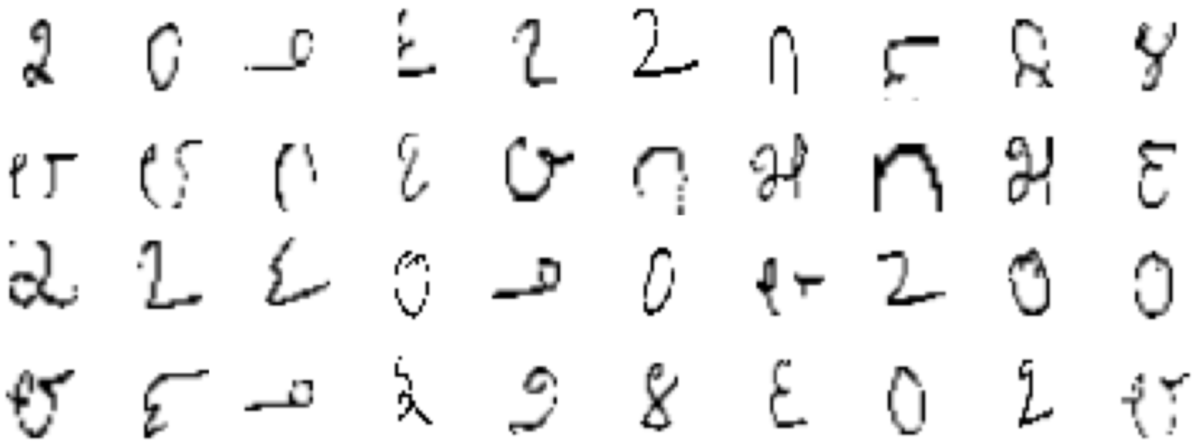
Train Data Visualization

In [7]:
```
plt.figure(figsize=(15,6))
for i in range(40):
    plt.subplot(4, 10, i+1)
    plt.imshow(X_train.values[i].reshape((28,28)),cmap=plt.cm.binary)
    plt.title("label=%d" % y_train[i],y=0.9)
    plt.axis('off')
plt.subplots_adjust(wspace=0.3, hspace=-0.1)
plt.show()
```



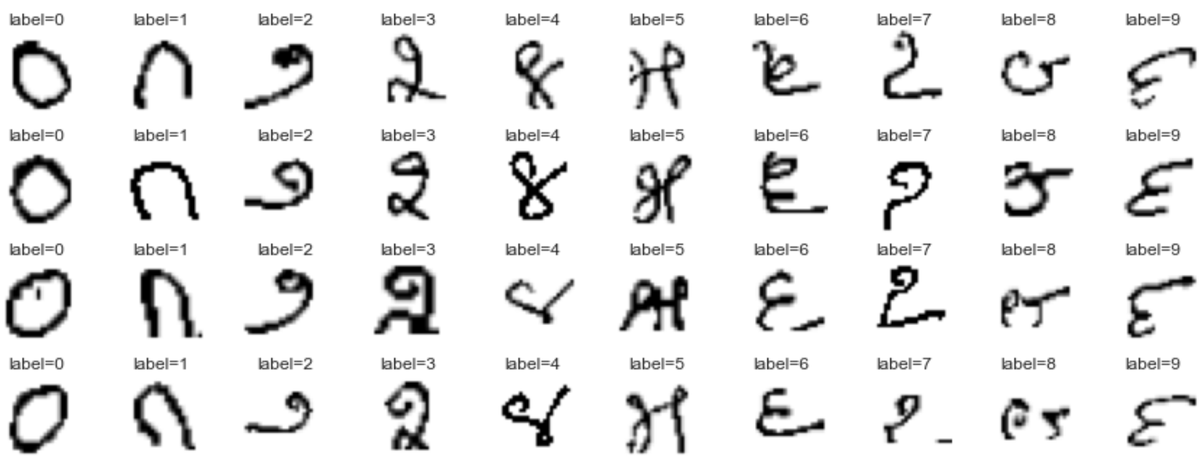
Test Data Visualization

In [8]:
```
plt.figure(figsize=(15,6))
for i in range(40):
    plt.subplot(4, 10, i+1)
    plt.imshow(test.values[i].reshape((28,28)),cmap=plt.cm.binary)
    plt.axis('off')
plt.subplots_adjust(wspace=0.3, hspace=-0.1)
plt.show()
```



Dig Data Visualization

In [9]:
```
plt.figure(figsize=(15,6))
for i in range(40):
    plt.subplot(4, 10, i+1)
    plt.imshow(X_dig.values[i].reshape((28,28)),cmap=plt.cm.binary)
    plt.title("label=%d" % y_dig[i],y=0.9)
    plt.axis('off')
plt.subplots_adjust(wspace=0.3, hspace=-0.1)
plt.show()
```



Prepare the Data for Modeling

Apart from the Dig-MNIST data set, I want to have a separate validation set from the training data. I'll allocate 20% of the training data to be a validation set. Before doing so, we'll normalize the data. The maximum value is 255, so we need to divide all data points by this number to normalize. Furthermore, we need to categorize the `label` column in our train set rather than making them numerical.

In [10]:
```
max(X_train.iloc[1, :])
```

Out[10]: 255

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_si
print(f"Training set shape: {X_train.shape}, {y_train.shape}")
print(f"Validation set shape: {X_test.shape}, {y_test.shape}")
```

```
Training set shape: (48000, 784), (48000,)
Validation set shape: (12000, 784), (12000,)
```

In [12]:
```
# reshape flattened data into 3D tensor & standardize the values in the datas
# n_x = 28
# X_train.values.reshape((-1, n_x, n_x, 1)) / 255.0
X_train = X_train / 255.0
X_test = X_test / 255.0     # similarly for dev set
test = test / 255.0   # similarly for test set
X_dig = X_dig / 255.0     # similarly for dig set

# one-hot encode the labels in y_train, y_test, y_dig
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_dig = to_categorical(y_dig)
```

In [13]:
```
print(X_train.shape, X_test.shape, test.shape, y_test.shape)
```

```
(48000, 784) (12000, 784) (5000, 784) (12000, 10)
```

Model #1: Neural Network w/ 2 Layers and 10 Nodes

Let's first clarify the meaning of "layers." As an example, an MLP (multi-layer perceptron) that has an input layer, two hidden layers, and one output layer is a 2-layer MLP. This means that we do not count the input or output layers as a layer.

To further clarify, we want 10 nodes in **EACH** layer, and not 10 nodes total.

Our notation from here on out will be as follows: as an example, a network with two variables in the input layer, one hidden layer with eight nodes, and an output layer with one node would be described using the notation: 2/8/1.

For our data set, the notation is: 784/10/10/10

Here, we will be using TensorFlow's `Sequential()` model. A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

In [14]:
```
X_train.shape
```

Out[14]:
```
(48000, 784)
```

In [15]:
```
# taking this from textbook
def build_model(n_hidden=2, n_nodes=10):
    model = Sequential()
    model.add(Dense(n_nodes, activation="relu", input_shape=(28*28, )))
    for layer in range(n_hidden-1):
        # no need to add Flatten layer since data is already flat
        model.add(Dense(n_nodes, activation="relu"))
    model.add(Dense(10, activation="softmax"))
    return model
```

In [16]:
```
# model1 = build_model()
# model1.summary()
```

In [17]:
```
# model1.compile(optimizer="sgd", loss="categorical_crossentropy", metrics=["
```

In [18]:
```
# # start timer
# start = datetime.now()

# history = model1.fit(X_train, y_train, epochs=10, validation_data=(X_test,

# # stopping timer
# end = datetime.now()

# # printing the time it took to fit the model
# print(f"It took {end-start} to fit the 784/10/10/10 model")
```

Plotting Learning Curves

In [19]:
```
# pd.DataFrame(history.history).plot(figsize=(8, 5))
# plt.grid(True)
# plt.gca().set_ylim(0, 1) # set the vertical range from [0-1]
# plt.show()
```

In [20]:
```
# model1.evaluate(X_train, y_train)
```

In [21]:
```
# model1.evaluate(X_test, y_test)
```

In [22]:
```
# model1.evaluate(X_dig, y_dig)
```

Model 1 Train Set Evaluation

In [23]:
```
# # Predict the values from the train set
# y_train_pred = model1.predict(X_train)

# # Convert predictions classes to one hot vectors
# y_train_pred = np.argmax(y_train_pred, axis = 1)

# # Convert train observations to one hot vectors
# y_train_classes = np.argmax(y_train, axis = 1)

# print(classification_report(y_train_classes, y_train_pred))
# print(f"Accuracy: {round(accuracy_score(y_train_classes, y_train_pred), 3)}
# sns.heatmap(confusion_matrix(y_train_classes, y_train_pred), annot=True);
```

Model 1 Test Set Evaluation

In [24]:
```
# # Predict the values from the validation set
# y_test_pred = model1.predict(X_test)

# # Convert predictions classes to one hot vectors
# y_test_pred = np.argmax(y_test_pred, axis = 1)

# # Convert validation observations to one hot vectors
# y_test_classes = np.argmax(y_test, axis = 1)

# print(classification_report(y_test_classes, y_test_pred))
# print(f"Accuracy: {round(accuracy_score(y_test_classes, y_test_pred), 3)}")
# sns.heatmap(confusion_matrix(y_test_classes, y_test_pred), annot=True);
```

Model 1 Dig Set Evaluation

In [25]:
```
# # Predict the values from the validation set
# y_dig_pred = model1.predict(X_dig)

# # Convert predictions classes to one hot vectors
# y_dig_pred = np.argmax(y_dig_pred, axis = 1)

# # Convert train observations to one hot vectors
# y_dig_classes = np.argmax(y_dig, axis = 1)

# print(classification_report(y_dig_classes, y_dig_pred))
# print(f"Accuracy: {round(accuracy_score(y_dig_classes, y_dig_pred), 3)}")
# sns.heatmap(confusion_matrix(y_dig_classes, y_dig_pred), annot=True);
```

Insights:

- Time: 11.7 seconds
- Training Accuracy: 0.9653
- Testing Accuracy: 0.9625
- Accuracy increases as we progress through epochs
- Train and test accuracies are almost the same
- Loss is minimal
- This model already performs well!

Model #2: Neural Network w/ 2 Layers and 20 Nodes

Repeat what I did above, but with 20 nodes instead.

In [26]:
```
# model2 = build_model(n_hidden=2, n_nodes=20)
# model2.summary()
```

In [27]:
```
# model2.compile(optimizer="sgd", loss="categorical_crossentropy", metrics=["

# # start timer
# start = datetime.now()

# history = model2.fit(X_train, y_train, epochs=10, validation_data=(X_test,

# # stopping timer
# end = datetime.now()

# # printing the time it took to fit the model
# print(f"It took {end-start} to fit the 784/20/20/10 model")
```

Plotting Learning Curves

In [28]:
```
# pd.DataFrame(history.history).plot(figsize=(8, 5))
# plt.grid(True)
# plt.gca().set_ylim(0, 1) # set the vertical range from [0-1]
# plt.show()
```

In [29]:
```
# # model evaluation on train set
# model2.evaluate(X_train, y_train)
```

In [30]:
```
# # model eval on validation set
# model2.evaluate(X_test, y_test)
```

In [31]:
```
# # model eval on dig set
# model2.evaluate(X_dig, y_dig)
```

Model 2 Train Set Evaluation

In [32]:
```
# # Predict the values from the train set
# y_train_pred = model2.predict(X_train)

# # Convert predictions classes to one hot vectors
# y_train_pred = np.argmax(y_train_pred, axis = 1)

# # Convert train observations to one hot vectors
# y_train_classes = np.argmax(y_train, axis = 1)

# print(classification_report(y_train_classes, y_train_pred))
# print(f"Accuracy: {round(accuracy_score(y_train_classes, y_train_pred), 3)}
# sns.heatmap(confusion_matrix(y_train_classes, y_train_pred), annot=True);
```

Model 2 Test Set Evaluation

In [33]:
```
# # Predict the values from the validation set
# y_test_pred = model2.predict(X_test)

# # Convert predictions classes to one hot vectors
# y_test_pred = np.argmax(y_test_pred, axis = 1)

# # Convert validation observations to one hot vectors
# y_test_classes = np.argmax(y_test, axis = 1)

# print(classification_report(y_test_classes, y_test_pred))
# print(f"Accuracy: {round(accuracy_score(y_test_classes, y_test_pred), 3)}")
# sns.heatmap(confusion_matrix(y_test_classes, y_test_pred), annot=True);
```

Model 2 Dig Set Evaluation

In [34]:
```
# # Predict the values from the validation set
# y_dig_pred = model2.predict(X_dig)

# # Convert predictions classes to one hot vectors
# y_dig_pred = np.argmax(y_dig_pred, axis = 1)

# # Convert train observations to one hot vectors
# y_dig_classes = np.argmax(y_dig, axis = 1)

# print(classification_report(y_dig_classes, y_dig_pred))
# print(f"Accuracy: {round(accuracy_score(y_dig_classes, y_dig_pred), 3)}")
# sns.heatmap(confusion_matrix(y_dig_classes, y_dig_pred), annot=True);
```

Insights:

- Time: 12.4 seconds
  - Only a slight increase from the 784/10/10/10 model
  - For only a second delay, I would choose this model
- Training Accuracy: 0.9706
- Testing Accuracy: 0.9662
- Model performs slightly worse on Dig set
- Accuracy starts at a higher value since the first epoch compared to the 784/10/10/10 model
- Loss is minimal
- This model performs better overall compared to the 784/10/10/10 model

Model #3: Neural Network w/ 5 Layers and 10 Nodes

Similar to Model #1, except this model will have 5 layers.

In [35]:
```
model3 = build_model(n_hidden=5, n_nodes=10)
model3.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10)                7850
```

```
_________________________________________________________________
dense_1 (Dense)              (None, 10)                110
_________________________________________________________________
dense_2 (Dense)              (None, 10)                110
_________________________________________________________________
dense_3 (Dense)              (None, 10)                110
_________________________________________________________________
dense_4 (Dense)              (None, 10)                110
_________________________________________________________________
dense_5 (Dense)              (None, 10)                110
=================================================================
Total params: 8,400
Trainable params: 8,400
Non-trainable params: 0
```

In [36]:

```
model3.compile(optimizer="sgd", loss="categorical_crossentropy", metrics=["ac
```

In [37]:

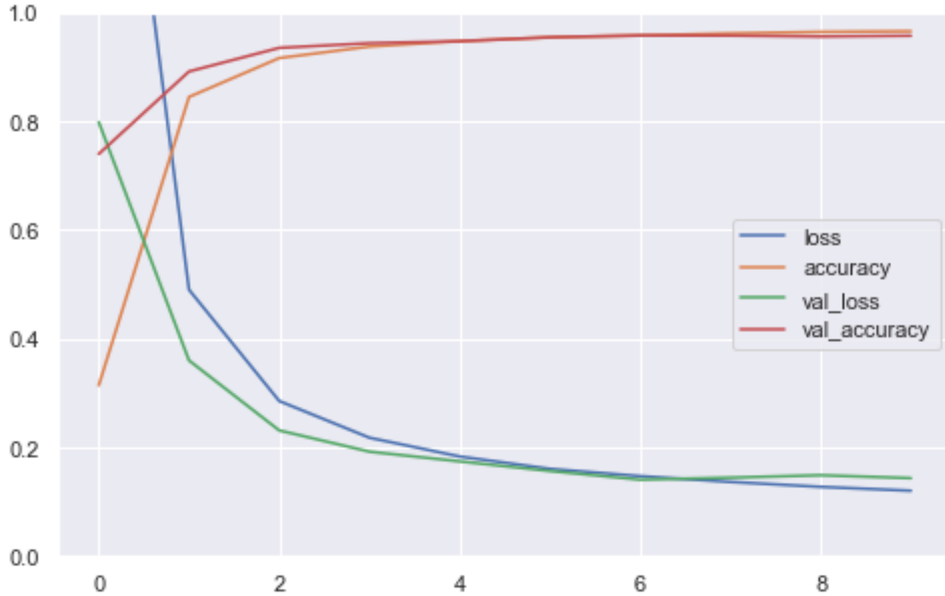
```
# start timer
start = datetime.now()

history = model3.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_

# stopping timer
end = datetime.now()

# printing the time it took to fit the model
print(f"It took {end-start} to fit the 5 Layer, 10 Nodes model")
```

```
Epoch 1/10
1500/1500 [==============================] - 2s 1ms/step - loss: 1.7809 - acc
uracy: 0.3138 - val_loss: 0.7976 - val_accuracy: 0.7391
Epoch 2/10
1500/1500 [==============================] - 1s 744us/step - loss: 0.4896 - a
ccuracy: 0.8439 - val_loss: 0.3596 - val_accuracy: 0.8907
Epoch 3/10
1500/1500 [==============================] - 1s 740us/step - loss: 0.2850 - a
ccuracy: 0.9154 - val_loss: 0.2309 - val_accuracy: 0.9341
Epoch 4/10
1500/1500 [==============================] - 1s 784us/step - loss: 0.2175 - a
ccuracy: 0.9365 - val_loss: 0.1919 - val_accuracy: 0.9427
Epoch 5/10
1500/1500 [==============================] - 1s 817us/step - loss: 0.1830 - a
ccuracy: 0.9465 - val_loss: 0.1740 - val_accuracy: 0.9463
Epoch 6/10
1500/1500 [==============================] - 1s 729us/step - loss: 0.1602 - a
ccuracy: 0.9534 - val_loss: 0.1563 - val_accuracy: 0.9536
Epoch 7/10
1500/1500 [==============================] - 1s 922us/step - loss: 0.1469 - a
ccuracy: 0.9570 - val_loss: 0.1404 - val_accuracy: 0.9569
Epoch 8/10
1500/1500 [==============================] - 1s 814us/step - loss: 0.1363 - a
ccuracy: 0.9607 - val_loss: 0.1441 - val_accuracy: 0.9567
Epoch 9/10
1500/1500 [==============================] - 1s 891us/step - loss: 0.1272 - a
ccuracy: 0.9634 - val_loss: 0.1486 - val_accuracy: 0.9549
Epoch 10/10
1500/1500 [==============================] - 1s 963us/step - loss: 0.1201 - a
ccuracy: 0.9647 - val_loss: 0.1432 - val_accuracy: 0.9564
It took 0:00:13.809096 to fit the 5 Layer, 10 Nodes model
```

Plotting Learning Curves

In [38]:

```
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1) # set the vertical range from [0-1]
plt.show()
```



In [39]:
```
# model evaluation on train set
model3.evaluate(X_train, y_train)
```

```
1500/1500 [==============================] - 1s 823us/step - loss: 0.1151 - a
ccuracy: 0.9662
```

Out[39]:
```
[0.11505535989999771, 0.9661874771118164]
```

In [40]:
```
# model eval on validation set
model3.evaluate(X_test, y_test)
```

```
375/375 [==============================] - 0s 639us/step - loss: 0.1432 - acc
uracy: 0.9564
```

Out[40]:
```
[0.14321769773960114, 0.956416666507721]
```

In [41]:
```
# model eval on dig set
model3.evaluate(X_dig, y_dig)
```

```
320/320 [==============================] - 0s 610us/step - loss: 2.5748 - acc
uracy: 0.5971
```

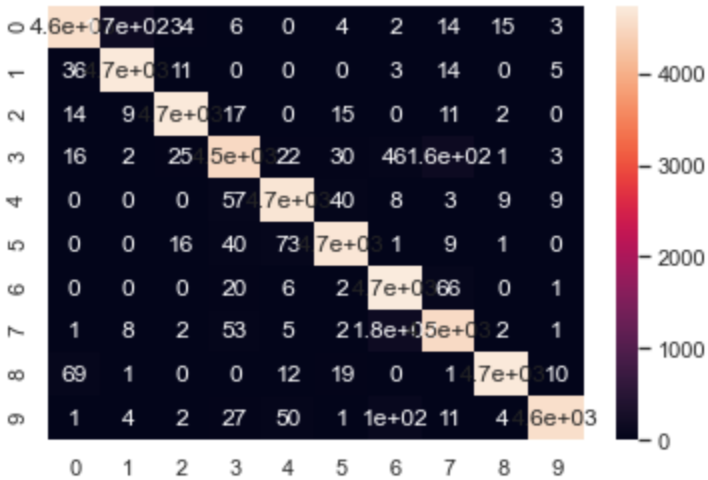
Out[41]:
```
[2.5748422145843506, 0.5970703363418579]
```

Model 3 Train Set Evaluation

In [42]:
```
# Predict the values from the train set
y_train_pred = model3.predict(X_train)

# Convert predictions classes to one hot vectors
y_train_pred = np.argmax(y_train_pred, axis = 1)

# Convert train observations to one hot vectors
y_train_classes = np.argmax(y_train, axis = 1)

print(classification_report(y_train_classes, y_train_pred))
print(f"Accuracy: {round(accuracy_score(y_train_classes, y_train_pred), 3)}")
sns.heatmap(confusion_matrix(y_train_classes, y_train_pred), annot=True);
```

```
              precision    recall  f1-score   support
```

```
           0       0.97      0.95      0.96      4823
           1       0.96      0.99      0.97      4782
           2       0.98      0.99      0.98      4776
           3       0.95      0.94      0.94      4816
           4       0.97      0.97      0.97      4779
           5       0.98      0.97      0.97      4812
           6       0.93      0.98      0.96      4831
           7       0.94      0.95      0.94      4781
           8       0.99      0.98      0.98      4814
           9       0.99      0.96      0.97      4786

    accuracy                           0.97     48000
   macro avg       0.97      0.97      0.97     48000
weighted avg       0.97      0.97      0.97     48000

Accuracy: 0.966
```

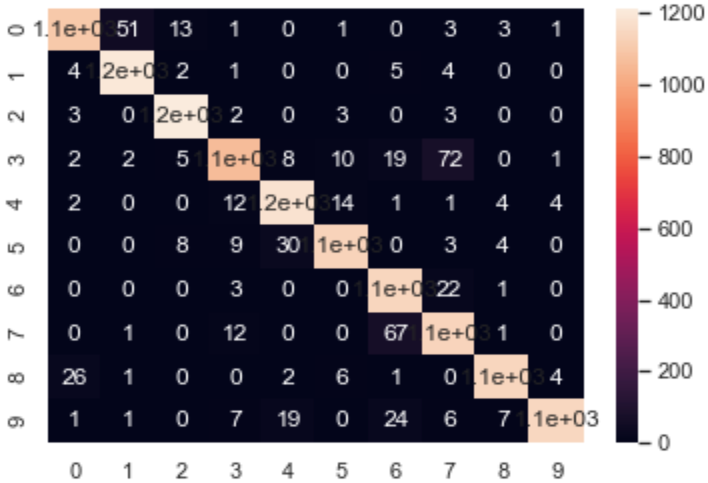


Model 3 Test Set Evaluation

In [43]:
```
# Predict the values from the validation set
y_test_pred = model3.predict(X_test)

# Convert predictions classes to one hot vectors
y_test_pred = np.argmax(y_test_pred, axis = 1)

# Convert validation observations to one hot vectors
y_test_classes = np.argmax(y_test, axis = 1)

print(classification_report(y_test_classes, y_test_pred))
print(f"Accuracy: {round(accuracy_score(y_test_classes, y_test_pred), 3)}")
sns.heatmap(confusion_matrix(y_test_classes, y_test_pred), annot=True);
```

```
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      1177
           1       0.96      0.99      0.97      1218
           2       0.98      0.99      0.98      1224
           3       0.96      0.90      0.93      1184
           4       0.95      0.97      0.96      1221
           5       0.97      0.95      0.96      1188
           6       0.91      0.98      0.94      1169
           7       0.91      0.93      0.92      1219
           8       0.98      0.97      0.97      1186
           9       0.99      0.95      0.97      1214
```

```
    accuracy                           0.96     12000
   macro avg       0.96      0.96      0.96     12000
weighted avg       0.96      0.96      0.96     12000

Accuracy: 0.956
```



Model 3 Dig Set Evaluation

In [44]:

```python
# Predict the values from the validation set
y_dig_pred = model3.predict(X_dig)

# Convert predictions classes to one hot vectors
y_dig_pred = np.argmax(y_dig_pred, axis = 1)

# Convert train observations to one hot vectors
y_dig_classes = np.argmax(y_dig, axis = 1)

print(classification_report(y_dig_classes, y_dig_pred))
print(f"Accuracy: {round(accuracy_score(y_dig_classes, y_dig_pred), 3)}")
sns.heatmap(confusion_matrix(y_dig_classes, y_dig_pred), annot=True);
```

```
              precision    recall  f1-score   support

           0       0.60      0.55      0.58      1024
           1       0.79      0.49      0.61      1024
           2       0.58      0.76      0.66      1024
           3       0.59      0.25      0.35      1024
           4       0.61      0.67      0.64      1024
           5       0.48      0.84      0.61      1024
           6       0.57      0.58      0.57      1024
           7       0.70      0.51      0.59      1024
           8       0.55      0.69      0.61      1024
           9       0.70      0.63      0.67      1024

    accuracy                           0.60     10240
   macro avg       0.62      0.60      0.59     10240
weighted avg       0.62      0.60      0.59     10240

Accuracy: 0.597
```

Insights:

- Time: 13.4 seconds
- Training Accuracy: 0.9700
- Testing Accuracy: 0.9613
- Loss is minimal, but we see that the validation set loss is slightly higher than the training set loss
- This model performs worse on the Dig set compared to Model 1 and Model 2
- This model performs well overall; we have very high accuracy

Model #4: Neural Network w/ 5 Layers and 20 Nodes

Similar to Model #2, except this model will have 5 layers.

In [45]:
```
model4 = build_model(n_hidden=5, n_nodes=20)
model4.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_6 (Dense)              (None, 20)                15700     
_________________________________________________________________
dense_7 (Dense)              (None, 20)                420       
_________________________________________________________________
dense_8 (Dense)              (None, 20)                420       
_________________________________________________________________
dense_9 (Dense)              (None, 20)                420       
_________________________________________________________________
dense_10 (Dense)             (None, 20)                420       
_________________________________________________________________
dense_11 (Dense)             (None, 10)                210       
=================================================================
Total params: 17,590
Trainable params: 17,590
Non-trainable params: 0
_________________________________________________________________
```

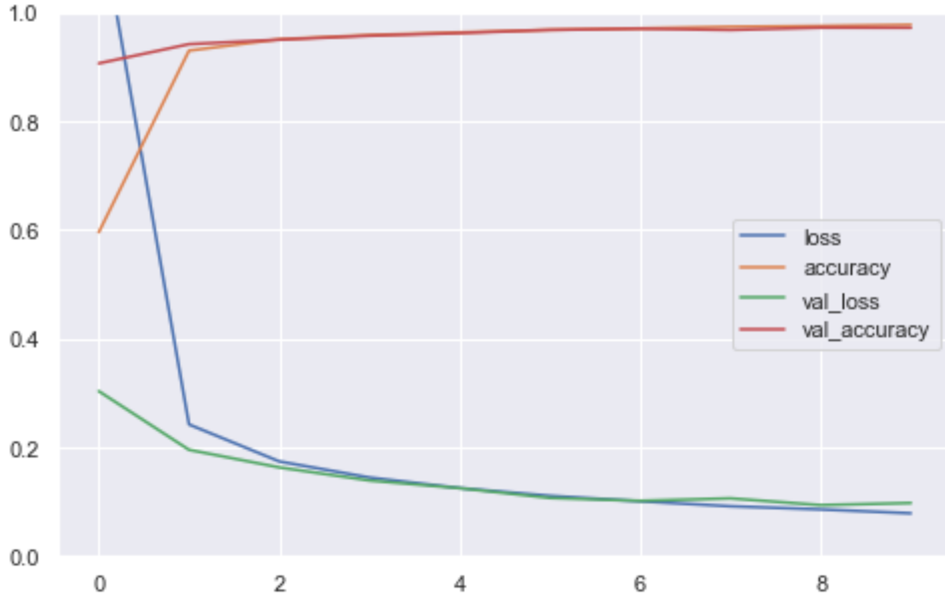
In [46]:
```
model4.compile(optimizer="sgd", loss="categorical_crossentropy", metrics=["ac
```

In [47]:
```
# start timer
start = datetime.now()

history = model4.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_

# stopping timer
end = datetime.now()

# printing the time it took to fit the model
print(f"It took {end-start} to fit the 5 Layer, 20 Nodes model")
```

```
Epoch 1/10
1500/1500 [==============================] - 3s 1ms/step - loss: 1.1836 - acc
uracy: 0.5952 - val_loss: 0.3031 - val_accuracy: 0.9056
Epoch 2/10
1500/1500 [==============================] - 2s 1ms/step - loss: 0.2419 - acc
uracy: 0.9290 - val_loss: 0.1953 - val_accuracy: 0.9409
Epoch 3/10
1500/1500 [==============================] - 1s 943us/step - loss: 0.1740 - a
ccuracy: 0.9499 - val_loss: 0.1629 - val_accuracy: 0.9491
Epoch 4/10
1500/1500 [==============================] - 1s 941us/step - loss: 0.1443 - a
ccuracy: 0.9577 - val_loss: 0.1389 - val_accuracy: 0.9564
Epoch 5/10
1500/1500 [==============================] - 1s 846us/step - loss: 0.1250 - a
ccuracy: 0.9625 - val_loss: 0.1247 - val_accuracy: 0.9613
Epoch 6/10
1500/1500 [==============================] - 2s 1ms/step - loss: 0.1107 - acc
uracy: 0.9672 - val_loss: 0.1067 - val_accuracy: 0.9672
Epoch 7/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.1007 - acc
uracy: 0.9698 - val_loss: 0.1015 - val_accuracy: 0.9693
Epoch 8/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0916 - acc
uracy: 0.9729 - val_loss: 0.1061 - val_accuracy: 0.9670
Epoch 9/10
1500/1500 [==============================] - 2s 1ms/step - loss: 0.0857 - acc
uracy: 0.9743 - val_loss: 0.0936 - val_accuracy: 0.9712
Epoch 10/10
1500/1500 [==============================] - 1s 863us/step - loss: 0.0786 - a
ccuracy: 0.9765 - val_loss: 0.0975 - val_accuracy: 0.9711
It took 0:00:19.485091 to fit the 5 Layer, 20 Nodes model
```

Plotting Learning Curves

In [48]:
```
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1) # set the vertical range from [0-1]
plt.show()
```



In [49]:
```
# model evaluation on train set
model4.evaluate(X_train, y_train)
```

```
1500/1500 [==============================] - 1s 940us/step - loss: 0.0723 - a
ccuracy: 0.9779
```

Out[49]: [0.07230456173419952, 0.9778958559036255]

In [50]:
```
# model eval on validation set
model4.evaluate(X_test, y_test)
```

```
375/375 [==============================] - 0s 719us/step - loss: 0.0975 - acc
uracy: 0.9711
```

Out[50]: [0.09752164036035538, 0.9710833430290222]

In [51]:
```
# model eval on dig set
model4.evaluate(X_dig, y_dig)
```

```
320/320 [==============================] - 0s 536us/step - loss: 2.3908 - acc
uracy: 0.6082
```

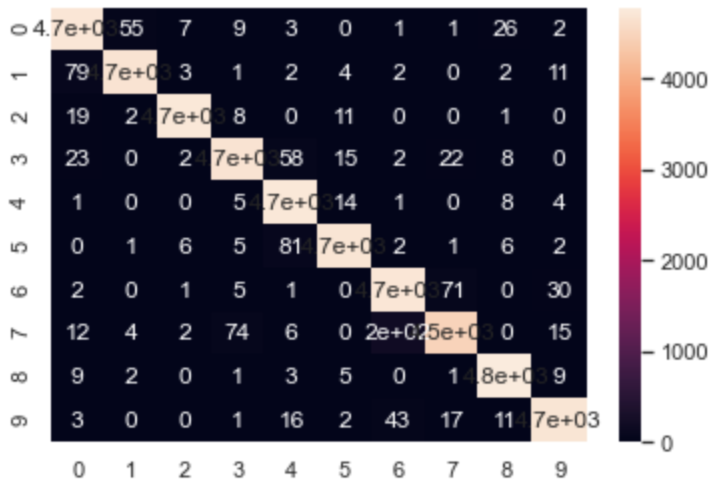
Out[51]: [2.3908329010009766, 0.608203113079071]

Model 4 Train Set Evaluation

In [52]:
```
# Predict the values from the train set
y_train_pred = model4.predict(X_train)

# Convert predictions classes to one hot vectors
y_train_pred = np.argmax(y_train_pred, axis = 1)

# Convert train observations to one hot vectors
y_train_classes = np.argmax(y_train, axis = 1)

print(classification_report(y_train_classes, y_train_pred))
print(f"Accuracy: {round(accuracy_score(y_train_classes, y_train_pred), 3)}")
sns.heatmap(confusion_matrix(y_train_classes, y_train_pred), annot=True);
```

```
              precision    recall  f1-score   support
```

```
           0       0.97      0.98      0.97      4823
           1       0.99      0.98      0.98      4782
           2       1.00      0.99      0.99      4776
           3       0.98      0.97      0.98      4816
           4       0.97      0.99      0.98      4779
           5       0.99      0.98      0.98      4812
           6       0.95      0.98      0.96      4831
           7       0.98      0.93      0.95      4781
           8       0.99      0.99      0.99      4814
           9       0.98      0.98      0.98      4786

    accuracy                           0.98     48000
   macro avg       0.98      0.98      0.98     48000
weighted avg       0.98      0.98      0.98     48000

Accuracy: 0.978
```



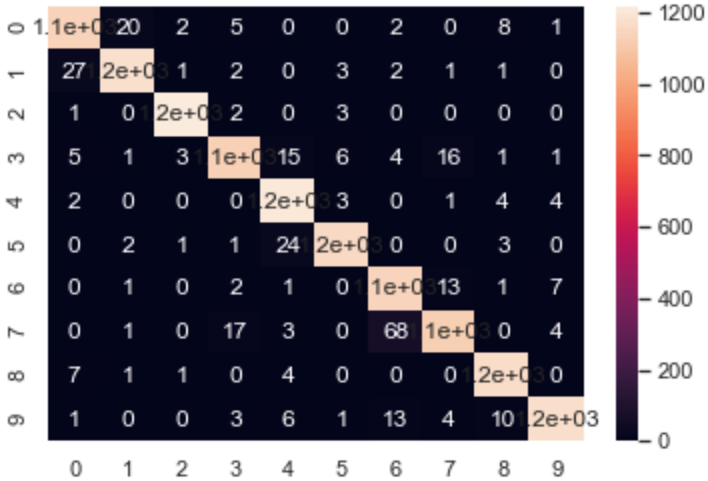
Model 4 Test Set Evaluation

In [53]:

```python
# Predict the values from the validation set
y_test_pred = model4.predict(X_test)

# Convert predictions classes to one hot vectors
y_test_pred = np.argmax(y_test_pred, axis = 1)

# Convert validation observations to one hot vectors
y_test_classes = np.argmax(y_test, axis = 1)

print(classification_report(y_test_classes, y_test_pred))
print(f"Accuracy: {round(accuracy_score(y_test_classes, y_test_pred), 3)}")
sns.heatmap(confusion_matrix(y_test_classes, y_test_pred), annot=True);
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      1177
           1       0.98      0.97      0.97      1218
           2       0.99      1.00      0.99      1224
           3       0.97      0.96      0.96      1184
           4       0.96      0.99      0.97      1221
           5       0.99      0.97      0.98      1188
           6       0.93      0.98      0.95      1169
           7       0.97      0.92      0.95      1219
           8       0.98      0.99      0.98      1186
           9       0.99      0.97      0.98      1214
```

```
    accuracy                           0.97     12000
   macro avg       0.97      0.97      0.97     12000
weighted avg       0.97      0.97      0.97     12000

Accuracy: 0.971
```



Model 4 Dig Set Evaluation

In [54]:
```
# Predict the values from the validation set
y_dig_pred = model4.predict(X_dig)

# Convert predictions classes to one hot vectors
y_dig_pred = np.argmax(y_dig_pred, axis = 1)

# Convert train observations to one hot vectors
y_dig_classes = np.argmax(y_dig, axis = 1)

print(classification_report(y_dig_classes, y_dig_pred))
print(f"Accuracy: {round(accuracy_score(y_dig_classes, y_dig_pred), 3)}")
sns.heatmap(confusion_matrix(y_dig_classes, y_dig_pred), annot=True);
```

```
              precision    recall  f1-score   support

           0       0.68      0.47      0.56      1024
           1       0.69      0.54      0.61      1024
           2       0.61      0.87      0.72      1024
           3       0.73      0.29      0.41      1024
           4       0.79      0.61      0.69      1024
           5       0.45      0.84      0.59      1024
           6       0.55      0.61      0.58      1024
           7       0.83      0.40      0.54      1024
           8       0.53      0.82      0.64      1024
           9       0.67      0.63      0.65      1024

    accuracy                           0.61     10240
   macro avg       0.65      0.61      0.60     10240
weighted avg       0.65      0.61      0.60     10240

Accuracy: 0.608
```

Insights:

- Time: 13.8 seconds
  - Only very slightly higher than Model 3
- Training Accuracy: 0.9768
  - Higher than Model 3
- Testing Accuracy: 0.9665
  - Higher than Model 3
- Loss is minimal AND lower than Model 3
- For only a .4 seconds increase in time to fit/train the model, this model is well worth using for its high accuracy

Model 5: My Attempt at a CNN

Looking through Kaggle notebooks and discussions, it seems like CNNs work really well for this data set. I will attempt to build one and see how this performs on the Kaggle dataset. The CNN requires that the data be shaped differently, so I will reimport the data sets and reshape them.

In [55]:

```
# splitting train into X_train, y_train
X_train = train.drop(['label'], axis=1)
y_train = np.array(train.label)

# splitting test into X_dig, y_dig
X_dig = dig.drop(['label'], axis=1)
y_dig = np.array(dig.label)

# reimporting test set
test = pd.read_csv("test.csv")
# test = pd.read_csv("../input/Kannada-MNIST/test.csv")
test.drop(['id'], axis=1, inplace=True)

# getting y_data to be a concatenation of y_train and y_test
y_data = np.append(y_train, y_dig)
assert len(y_data) == (len(y_train) + len(y_dig)), "wrong size for y_data"

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_si
print(f"Training set shape: {X_train.shape}, {y_train.shape}")
print(f"Validation set shape: {X_test.shape}, {y_test.shape}")
```

```
Training set shape: (48000, 784), (48000,)
```

```
Validation set shape: (12000, 784) (12000,)
```

In [56]:

```
# reshape flattened data into 3D tensor & standardize the values in the datas
n_x = 28
X_train = X_train.values.reshape((-1, n_x, n_x, 1)) / 255.0
X_test = X_test.values.reshape((-1, n_x, n_x, 1)) / 255.0     # similarly for
test = test.values.reshape((-1, n_x, n_x, 1)) / 255.0   # similarly for test
X_dig = X_dig.values.reshape((-1, n_x, n_x, 1)) / 255.0     # similarly for d
print(X_train.shape, X_test.shape, test.shape, X_dig.shape)

# one-hot encode the labels in y_train, y_test, y_dig
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_dig = to_categorical(y_dig)
```

```
(48000, 28, 28, 1) (12000, 28, 28, 1) (5000, 28, 28, 1) (10240, 28, 28, 1)
```

In [57]:

```
# data_augment = ImageDataGenerator(rotation_range=10, zoom_range=0.1,
#                                   width_shift_range=0.1, height_shift_range=

# model5 = Sequential()
# model5.add(Conv2D(32, kernel_size=3, padding='same', activation='relu', inp
# model5.add(Conv2D(32, kernel_size=3, padding='same', activation='relu'))
# model5.add(BatchNormalization(momentum=0.15))
# model5.add(MaxPool2D(pool_size=(2,2)))
# model5.add(Conv2D(32, kernel_size=5, padding='same', activation='relu'))
# model5.add(Dropout(0.4))
# model5.add(Conv2D(64, kernel_size=3, padding='same', activation='relu'))
# model5.add(Conv2D(64, kernel_size=3, padding='same', activation='relu'))
# model5.add(BatchNormalization(momentum=0.15))
# model5.add(MaxPool2D(pool_size=(2,2)))
# model5.add(Conv2D(64, kernel_size=5, padding='same', activation='relu'))
# model5.add(Dropout(0.4))
# model5.add(Conv2D(128, kernel_size=3, padding='same', activation='relu'))
# model5.add(Conv2D(128, kernel_size=3, padding='same', activation='relu'))
# model5.add(BatchNormalization(momentum=0.15))
# model5.add(MaxPool2D(pool_size=(2,2)))
# model5.add(Conv2D(128, kernel_size=5, padding='same', activation='relu'))
# model5.add(Dropout(0.4))
# model5.add(Flatten())
# model5.add(Dense(128, activation='relu'))
# model5.add(Dropout(0.4))
# model5.add(Dense(64, activation='relu'))
# model5.add(Dropout(0.4))
# model5.add(Dense(10, activation='softmax'))
# model5.summary()
```

In [58]:
```
model5 = Sequential()
model5.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same', activat
model5.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same', activat
model5.add(MaxPool2D(pool_size=(2,2)))
model5.add(Dropout(0.25))
model5.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same', activat
model5.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same', activat
model5.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model5.add(Dropout(0.25))
model5.add(Flatten())
model5.add(Dense(256, activation = "relu"))
model5.add(Dropout(0.5))
model5.add(Dense(10, activation = "softmax"))
model5.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        832       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 32)        25632     
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496     
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        36928     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0         
_________________________________________________________________
dense_12 (Dense)             (None, 256)               803072    
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 10)                2570      
=================================================================
Total params: 887,530
Trainable params: 887,530
Non-trainable params: 0
_________________________________________________________________
```

In [59]:
```
# Define the optimizer
optimizer = RMSprop(learning_rate=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
```

In [60]:
```
model5.compile(optimizer=optimizer, loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [61]:
```
# Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc', patience=3,
                                            verbose=1, factor=0.5, min_lr=0.0
```

In [62]:
```
gen = ImageDataGenerator()
batches = gen.flow(X_train, y_train, batch_size=64)
val_batches=gen.flow(X_test, y_test, batch_size=64)
```

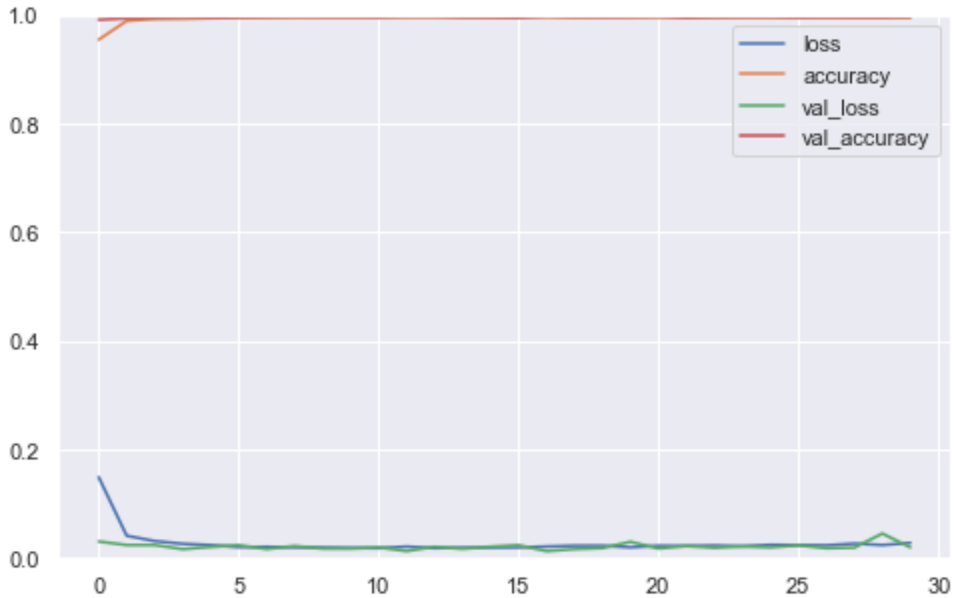
In [63]:
```
# Train and validate the model
epochs = 30
batch_size = 86

# start timer
start = datetime.now()

# fit model
# steps_per_epoch=X_train.shape[0]//batch_size
# batch_size = batch_size
history = model5.fit(X_train, y_train,
                     batch_size=batch_size, epochs=epochs,
                     validation_data = (X_test, y_test), verbose = 2)


# stopping timer
end = datetime.now()

# printing the time it took to fit the model
print(f"It took {end-start} to fit the CNN model")
```

```
Epoch 1/30
559/559 - 114s - loss: 0.1488 - accuracy: 0.9537 - val_loss: 0.0306 - val_acc
uracy: 0.9906
Epoch 2/30
559/559 - 103s - loss: 0.0411 - accuracy: 0.9883 - val_loss: 0.0240 - val_acc
uracy: 0.9929
Epoch 3/30
559/559 - 91s - loss: 0.0313 - accuracy: 0.9916 - val_loss: 0.0240 - val_accu
racy: 0.9940
Epoch 4/30
559/559 - 90s - loss: 0.0263 - accuracy: 0.9921 - val_loss: 0.0165 - val_accu
racy: 0.9951
Epoch 5/30
559/559 - 90s - loss: 0.0239 - accuracy: 0.9933 - val_loss: 0.0205 - val_accu
racy: 0.9939
Epoch 6/30
559/559 - 90s - loss: 0.0202 - accuracy: 0.9941 - val_loss: 0.0238 - val_accu
racy: 0.9946
Epoch 7/30
559/559 - 90s - loss: 0.0206 - accuracy: 0.9941 - val_loss: 0.0167 - val_accu
racy: 0.9952
Epoch 8/30
559/559 - 89s - loss: 0.0198 - accuracy: 0.9944 - val_loss: 0.0223 - val_accu
racy: 0.9958
Epoch 9/30
559/559 - 90s - loss: 0.0194 - accuracy: 0.9948 - val_loss: 0.0174 - val_accu
racy: 0.9953
Epoch 10/30
559/559 - 91s - loss: 0.0187 - accuracy: 0.9948 - val_loss: 0.0174 - val_accu
racy: 0.9953
Epoch 11/30
559/559 - 91s - loss: 0.0182 - accuracy: 0.9951 - val_loss: 0.0196 - val_accu
racy: 0.9949
```

```
Epoch 12/30
559/559 - 91s - loss: 0.0212 - accuracy: 0.9946 - val_loss: 0.0129 - val_accu
racy: 0.9963
Epoch 13/30
559/559 - 91s - loss: 0.0183 - accuracy: 0.9952 - val_loss: 0.0208 - val_accu
racy: 0.9958
Epoch 14/30
559/559 - 91s - loss: 0.0192 - accuracy: 0.9951 - val_loss: 0.0170 - val_accu
racy: 0.9947
Epoch 15/30
559/559 - 92s - loss: 0.0195 - accuracy: 0.9948 - val_loss: 0.0210 - val_accu
racy: 0.9946
Epoch 16/30
559/559 - 92s - loss: 0.0197 - accuracy: 0.9951 - val_loss: 0.0238 - val_accu
racy: 0.9940
Epoch 17/30
559/559 - 92s - loss: 0.0213 - accuracy: 0.9948 - val_loss: 0.0128 - val_accu
racy: 0.9967
Epoch 18/30
559/559 - 93s - loss: 0.0227 - accuracy: 0.9941 - val_loss: 0.0166 - val_accu
racy: 0.9958
Epoch 19/30
559/559 - 94s - loss: 0.0228 - accuracy: 0.9946 - val_loss: 0.0184 - val_accu
racy: 0.9954
Epoch 20/30
559/559 - 94s - loss: 0.0198 - accuracy: 0.9950 - val_loss: 0.0300 - val_accu
racy: 0.9952
Epoch 21/30
559/559 - 93s - loss: 0.0227 - accuracy: 0.9948 - val_loss: 0.0176 - val_accu
racy: 0.9962
Epoch 22/30
559/559 - 92s - loss: 0.0222 - accuracy: 0.9947 - val_loss: 0.0222 - val_accu
racy: 0.9939
Epoch 23/30
559/559 - 93s - loss: 0.0231 - accuracy: 0.9949 - val_loss: 0.0190 - val_accu
racy: 0.9948
Epoch 24/30
559/559 - 93s - loss: 0.0217 - accuracy: 0.9950 - val_loss: 0.0214 - val_accu
racy: 0.9951
Epoch 25/30
559/559 - 93s - loss: 0.0244 - accuracy: 0.9944 - val_loss: 0.0197 - val_accu
racy: 0.9955
Epoch 26/30
559/559 - 93s - loss: 0.0239 - accuracy: 0.9947 - val_loss: 0.0236 - val_accu
racy: 0.9945
Epoch 27/30
559/559 - 94s - loss: 0.0235 - accuracy: 0.9947 - val_loss: 0.0181 - val_accu
racy: 0.9948
Epoch 28/30
559/559 - 93s - loss: 0.0267 - accuracy: 0.9940 - val_loss: 0.0193 - val_accu
racy: 0.9949
Epoch 29/30
559/559 - 93s - loss: 0.0241 - accuracy: 0.9949 - val_loss: 0.0451 - val_accu
racy: 0.9945
Epoch 30/30
559/559 - 93s - loss: 0.0278 - accuracy: 0.9937 - val_loss: 0.0197 - val_accu
racy: 0.9958
```

Plotting Learning Curves

In [64]:
```
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1) # set the vertical range from [0-1]
plt.show()
```



In [65]:
```
# model evaluation on train set
model5.evaluate(X_train, y_train)
```

```
1500/1500 [==============================] - 25s 16ms/step - loss: 0.0045 - a
ccuracy: 0.9986
```

Out[65]: `[0.004507437814027071, 0.9986041784286499]`

In [66]:
```
# model eval on validation set
model5.evaluate(X_test, y_test)
```

```
375/375 [==============================] - 6s 17ms/step - loss: 0.0197 - accu
racy: 0.9958
```

Out[66]: `[0.019693169742822647, 0.9958333373069763]`

In [67]:
```
# model eval on dig set
model5.evaluate(X_dig, y_dig)
```

```
320/320 [==============================] - 5s 17ms/step - loss: 2.3581 - accu
racy: 0.8168
```

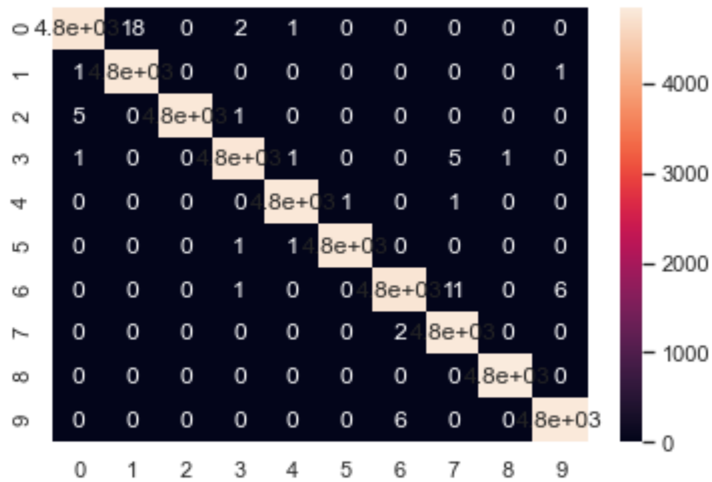
Out[67]: `[2.3580920696258545, 0.8167968988418579]`

Model 5 Train Set Evaluation

In [68]:

```
# Predict the values from the train set
y_train_pred = model5.predict(X_train)

# Convert predictions classes to one hot vectors
y_train_pred = np.argmax(y_train_pred, axis = 1)

# Convert train observations to one hot vectors
y_train_classes = np.argmax(y_train, axis = 1)

print(classification_report(y_train_classes, y_train_pred))
print(f"Accuracy: {round(accuracy_score(y_train_classes, y_train_pred), 3)}")
sns.heatmap(confusion_matrix(y_train_classes, y_train_pred), annot=True);
```

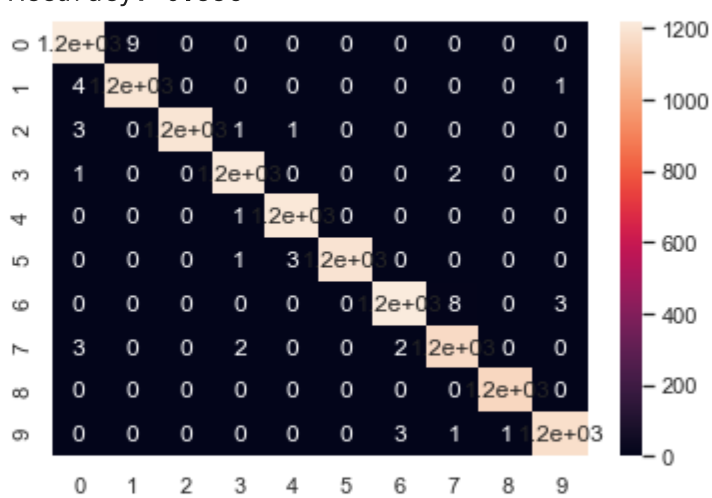
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4771
           1       1.00      1.00      1.00      4798
           2       1.00      1.00      1.00      4797
           3       1.00      1.00      1.00      4786
           4       1.00      1.00      1.00      4793
           5       1.00      1.00      1.00      4801
           6       1.00      1.00      1.00      4770
           7       1.00      1.00      1.00      4826
           8       1.00      1.00      1.00      4843
           9       1.00      1.00      1.00      4815

    accuracy                           1.00     48000
   macro avg       1.00      1.00      1.00     48000
weighted avg       1.00      1.00      1.00     48000

Accuracy: 0.999
```



Model 5 Test Set Evaluation

In [69]:
```
# Predict the values from the validation set
y_test_pred = model5.predict(X_test)

# Convert predictions classes to one hot vectors
y_test_pred = np.argmax(y_test_pred, axis = 1)

# Convert validation observations to one hot vectors
y_test_classes = np.argmax(y_test, axis = 1)

print(classification_report(y_test_classes, y_test_pred))
print(f"Accuracy: {round(accuracy_score(y_test_classes, y_test_pred), 3)}")
sns.heatmap(confusion_matrix(y_test_classes, y_test_pred), annot=True);
```

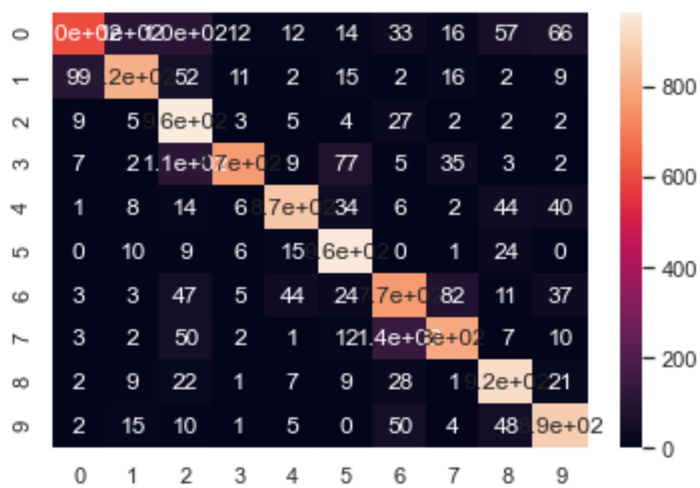
```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1229
           1       0.99      1.00      0.99      1202
           2       1.00      1.00      1.00      1203
           3       1.00      1.00      1.00      1214
           4       1.00      1.00      1.00      1207
           5       1.00      1.00      1.00      1199
           6       1.00      0.99      0.99      1230
           7       0.99      0.99      0.99      1174
           8       1.00      1.00      1.00      1157
           9       1.00      1.00      1.00      1185

    accuracy                           1.00     12000
   macro avg       1.00      1.00      1.00     12000
weighted avg       1.00      1.00      1.00     12000

Accuracy: 0.996
```



Model 5 Dig Set Evaluation

In [70]:
```python
# Predict the values from the validation set
y_dig_pred = model5.predict(X_dig)

# Convert predictions classes to one hot vectors
y_dig_pred = np.argmax(y_dig_pred, axis = 1)

# Convert train observations to one hot vectors
y_dig_classes = np.argmax(y_dig, axis = 1)

print(classification_report(y_dig_classes, y_dig_pred))
print(f"Accuracy: {round(accuracy_score(y_dig_classes, y_dig_pred), 3)}")
sns.heatmap(confusion_matrix(y_dig_classes, y_dig_pred), annot=True);
```

```
              precision    recall  f1-score   support

           0       0.83      0.59      0.69      1024
           1       0.84      0.80      0.82      1024
           2       0.70      0.94      0.80      1024
           3       0.94      0.75      0.84      1024
           4       0.90      0.85      0.87      1024
           5       0.84      0.94      0.88      1024
           6       0.73      0.75      0.74      1024
           7       0.83      0.78      0.81      1024
           8       0.82      0.90      0.86      1024
           9       0.83      0.87      0.85      1024

    accuracy                           0.82     10240
   macro avg       0.82      0.82      0.82     10240
weighted avg       0.82      0.82      0.82     10240

Accuracy: 0.817
```



Insights:

- Time: 46 minutes and 20 seconds
  - This is the slowest model
- Training Accuracy: 0.999
  - Highest of all models
- Testing Accuracy: 0.996
  - Hightest of all models
- Dig Accuracy: 0.817

- Highest of all models

Final Task: Predicting on the Test Set for Kaggle Submission

Uncomment as needed to submit to Kaggle

Model Test Set Prediction

Change the model accordingly when submitting to Kaggle.

In [71]:
```
# # Predict the values from the test set
# test_pred = model1.predict(test)

# # Convert predictions classes to one hot vectors
# test_pred = np.argmax(test_pred, axis = 1)

# # Show test_pred
# test_pred
```

In [72]:
```
# submission = pd.DataFrame(data={"id": test_id, "label": test_pred})
# submission.to_csv('submission.csv', index=False)
```

Conclusion

From Models 1-4, my best performing model is Model 4 with a public score of 0.89720, which is not as high as my random forest model from last week that scored 0.92420.

In [ ]:
```

```