In [27]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import keras
import keras.backend as K
from keras.optimizers import Adam
from keras.models import Sequential
from keras.utils import Sequence
from keras.layers import *

# open csv file, please change this if you're opening a file from your end
df = pd.read_csv(r"C:\Users\GLUTEUSMAXIMUS&OPPAI\Desktop\352_project\sudoku.csv")
rating_df = pd.read_csv(r"C:\Users\GLUTEUSMAXIMUS&OPPAI\Desktop\352_project\sudoku-3m
# taking a quick glance at the data
df.head()
```

Out[27]:

	quizzes	solutions
0	004300209005009001070060043006002087190007400…	864371259325849761971265843436192587198657432…
1	040100050107003960520008000000000017000906800…	346179258187523964529648371965832417472916835…
2	600120384008459072000006005000264030070080006…	695127384138459672724836915851264739273981546…
3	497200000100400005000016098620300040300900000…	497258316186439725253716498629381547375964182…
4	005910308009403060027500100030000201000820007…	465912378189473562327568149738645291954821637…

In [28]:
```
# initializing the data generator class of keras
class DataGenerator(Sequence):
    def __init__(self, df,batch_size = 16,subset = "train",shuffle = False):
        super().__init__()
        self.df = df
        self.batch_size = batch_size
        self.shuffle = shuffle
        self.subset = subset
        self.data_path = "\sudoku.csv"
        self.on_epoch_end()

    def __len__(self):
        return int(np.floor(len(self.df)/self.batch_size))

    def on_epoch_end(self):
        self.indexes = np.arange(len(self.df))
        if self.shuffle==True:
            np.random.shuffle(self.indexes)

    def __getitem__(self,index):
        X = np.empty((self.batch_size, 9,9,1))
        y = np.empty((self.batch_size,81,1))
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]
        for i,f in enumerate(self.df['quizzes'].iloc[indexes]):
            self.info[index*self.batch_size+i]=f
            X[i,] = (np.array(list(map(int,list(f)))).reshape((9,9,1))/9)-0.5
        if self.subset == 'train':
            for i,f in enumerate(self.df['solutions'].iloc[indexes]):
                self.info[index*self.batch_size+i]=f
                y[i,] = np.array(list(map(int,list(f)))).reshape((81,1)) - 1
        if self.subset == 'train': return X, y
        else: return X
```

In [29]:
```
#building the sequential model
model = Sequential()
model.add(Conv2D(64, kernel_size=(3,3), activation='relu', padding='same', input_shape
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, kernel_size=(1,1), activation='relu', padding='same'))
model.add(Flatten())
model.add(Dense(81*9))
model.add(Reshape((-1, 9)))
model.add(Activation('softmax'))
adam = keras.optimizers.Adam(lr=.001)
model.compile(loss='sparse_categorical_crossentropy', optimizer=adam, metrics=['accura
```

In [30]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 9, 9, 64)          640
_________________________________________________________________
batch_normalization_4 (Batch (None, 9, 9, 64)          256
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 9, 9, 64)          36928
_________________________________________________________________
batch_normalization_5 (Batch (None, 9, 9, 64)          256
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 9, 9, 128)         8320
_________________________________________________________________
flatten_2 (Flatten)          (None, 10368)             0
_________________________________________________________________
dense_2 (Dense)              (None, 729)               7559001
_________________________________________________________________
reshape_2 (Reshape)          (None, 81, 9)             0
_________________________________________________________________
activation_2 (Activation)    (None, 81, 9)             0
=================================================================
Total params: 7,605,401
Trainable params: 7,605,145
Non-trainable params: 256
_________________________________________________________________
```

In [31]:
```
train_idx = int(len(df)*0.95)
df = df.sample(frac=1).reset_index(drop=True)
training_generator = DataGenerator(df.iloc[:train_idx], subset = "train", batch_size=
validation_generator = DataGenerator(df.iloc[train_idx:], subset = "train",  batch_si
```

In [32]:
```
from keras.callbacks import Callback, ModelCheckpoint, ReduceLROnPlateau
filepath1="weights-improvement-{epoch:02d}-{val_accuracy:.2f}.hdf5"
filepath2 = "best_weights.hdf5"
checkpoint1 = ModelCheckpoint(filepath1, monitor='val_accuracy', verbose=1, save_best_
checkpoint2 = ModelCheckpoint(filepath2, monitor='val_accuracy', verbose=1, save_best_

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    patience=3,
    verbose=1,
    min_lr=1e-6
)
callbacks_list = [checkpoint1,checkpoint2,reduce_lr]
```

In [21]:
```
history = model.fit(training_generator, validation_data = validation_generator, epochs
```

```
Epoch 1/5
1484/1484 [==============================] - 426s 287ms/step - loss: 0.3713 - accurac
y: 0.1110 - val_loss: 0.3688 - val_accuracy: 0.1132

Epoch 00001: val_accuracy improved from 0.10992 to 0.11315, saving model to weights-im
provement-01-0.11.hdf5

Epoch 00001: val_accuracy improved from 0.10992 to 0.11315, saving model to best_weigh
ts.hdf5
Epoch 2/5
1484/1484 [==============================] - 429s 289ms/step - loss: 0.3570 - accurac
y: 0.1111 - val_loss: 0.3624 - val_accuracy: 0.1066

Epoch 00002: val_accuracy did not improve from 0.11315

Epoch 00002: val_accuracy did not improve from 0.11315
Epoch 3/5
1484/1484 [==============================] - 434s 292ms/step - loss: 0.3472 - accurac
y: 0.1112 - val_loss: 0.3595 - val_accuracy: 0.1109

Epoch 00003: val_accuracy did not improve from 0.11315

Epoch 00003: val_accuracy did not improve from 0.11315
Epoch 4/5
1484/1484 [==============================] - 423s 285ms/step - loss: 0.3394 - accurac
y: 0.1112 - val_loss: 0.3588 - val_accuracy: 0.1090

Epoch 00004: val_accuracy did not improve from 0.11315

Epoch 00004: val_accuracy did not improve from 0.11315
Epoch 5/5
1484/1484 [==============================] - 432s 291ms/step - loss: 0.3327 - accurac
y: 0.1112 - val_loss: 0.3588 - val_accuracy: 0.1142

Epoch 00005: val_accuracy improved from 0.11315 to 0.11417, saving model to weights-im
provement-05-0.11.hdf5

Epoch 00005: val_accuracy improved from 0.11315 to 0.11417, saving model to best_weigh
ts.hdf5
```

In [33]:
```
# saving the best weights
model.load_weights('best_weights.hdf5')
```

In [49]:
```
# using the model to solve, reference the CNN guide for this section of the reformatt
def solve_sudoku(game):
    game = np.array(game).reshape((9,9,1))
    game = (game/9)-0.5
    while(1):
        out = model.predict(game.reshape((1,9,9,1)))
        out = out.squeeze()
        pred = np.argmax(out, axis=1).reshape((9,9))+1
        prob = np.around(np.max(out, axis=1).reshape((9,9)), 2)
        game = ((game+.5)*9).reshape((9,9))
        mask = (game==0)
        if(mask.sum()==0):
            break
        prob_new = prob*mask
        ind = np.argmax(prob_new)
        x, y = (ind//9),(ind%9)
        val = pred[x][y]
        game[x][y] = val
        game = (game/9)-0.5
    return pred
    return game
```

In [50]:
```
# sample board input
game = [[0, 0, 0, 2, 4, 8, 7, 1, 9],
        [7, 0, 0, 0, 0, 0, 5, 0, 0],
        [0, 2, 0, 6, 0, 5, 0, 0, 4],
        [0, 0, 0, 4, 6, 0, 3, 7, 8],
        [0, 1, 3, 0, 0, 7, 9, 0, 6],
        [6, 0, 0, 0, 0, 2, 0, 5, 1],
        [0, 7, 2, 1, 0, 0, 0, 0, 3],
        [8, 0, 0, 0, 0, 6, 0, 0, 7],
        [9, 0, 0, 0, 2, 0, 1, 8, 0]]

game = solve_sudoku(game)
print(game)
```

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

In [51]:
```
#check if the array is correct
np.sum(game, axis=1)
```

Out[51]:
```
array([45, 45, 45, 45, 45, 45, 45, 45, 45], dtype=int64)
```

In [ ]: