In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installe
d
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/doc
ker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list
 all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that gets
 preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved o
utside of the current session
```

```
/kaggle/input/digit-recognizer/test.csv
/kaggle/input/digit-recognizer/sample_submission.csv
/kaggle/input/digit-recognizer/train.csv
```

In [2]:

```
# LOAD THE DATA
train = pd.read_csv("../input/digit-recognizer/train.csv")
test = pd.read_csv("../input/digit-recognizer/test.csv")
```

Importing libraries

In [3]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormal
ization
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import LearningRateScheduler
```

Preprocessing Data

In [4]:

```
# PREPARE DATA FOR NEURAL NETWORK
X_train = train.drop(labels = ["label"],axis = 1)
Y_train = train["label"]
# Normalize pixels
X_train = X_train / 255.0
X_test = test / 255.0
```

In [5]:

```
X_train.shape
```

Out[5]:

```
(42000, 784)
```

In [6]:

```
X_train = X_train.values.reshape(-1,28,28,1)
X_test = X_test.values.reshape(-1,28,28,1)
Y_train = to_categorical(Y_train, num_classes = 10)
# This will convert digits into flag array of size 10
```

In [7]:

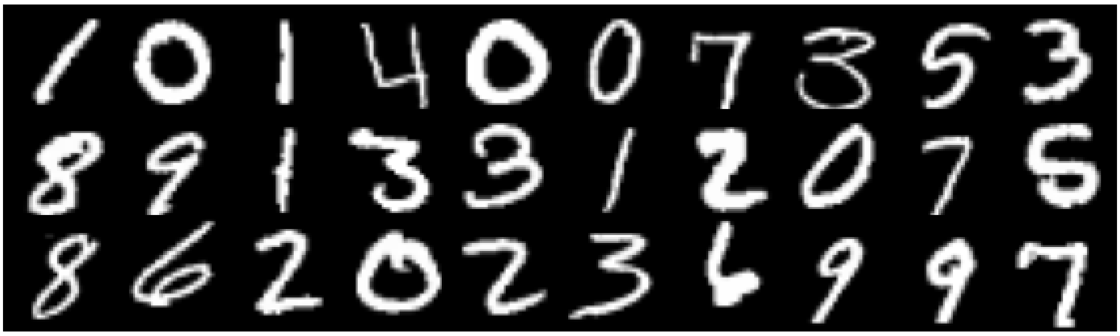
```
X_train.shape
```

Out[7]:

```
(42000, 28, 28, 1)
```

In [8]:

```
import matplotlib.pyplot as plt
# Plotting mnist images
plt.figure(figsize=(10,3))
for i in range(30):
    plt.subplot(3, 10, i+1)
    plt.imshow(X_train[i].reshape((28,28)),cmap=plt.cm.gray)
    plt.axis('off')
plt.subplots_adjust(wspace=-0.1, hspace=-0.1)
plt.show()
```



Let's Create some more data

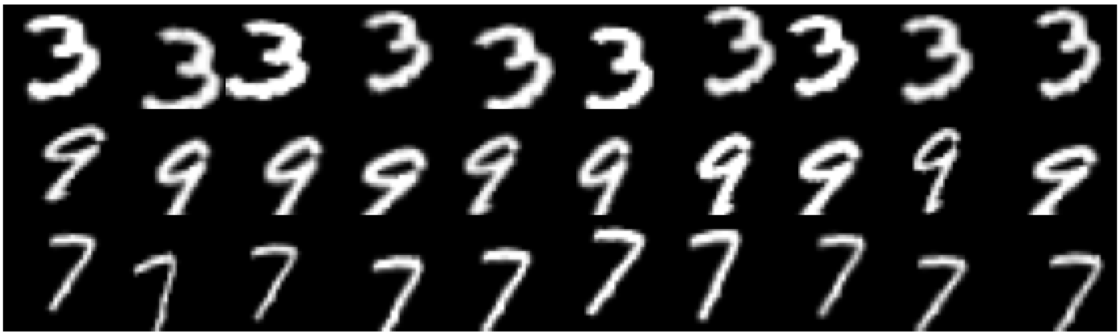
DATA AUGMENTATION

In [9]:

```
datagenerated = ImageDataGenerator(
        rotation_range=14,
        zoom_range = 0.1,
        channel_shift_range=0.4,
        width_shift_range=0.13,
        height_shift_range=0.13)
```

In [10]:

```
# Taking 1 element from train dataset
X_train3 = X_train[9,].reshape((1,28,28,1))
Y_train3 = Y_train[9,].reshape((1,10))
plt.figure(figsize=(10,3))
# Applying the filter
for i in range(30):
    plt.subplot(3, 10, i+1)
    X_train2, Y_train2 = datagenerated.flow(X_train3,Y_train3).next()
    plt.imshow(X_train2[0].reshape((28,28)),cmap=plt.cm.gray)
    plt.axis('off')
    if i==9: X_train3 = X_train[11,].reshape((1,28,28,1))
    if i==19: X_train3 = X_train[18,].reshape((1,28,28,1))
plt.subplots_adjust(wspace=-0.1, hspace=-0.1)
plt.show()
```



In [11]:

```
# BUILD CONVOLUTIONAL NEURAL NETWORKS
layers = 15
#Taking default [32, 64, 128] conv2d filter
model = Sequential()

model.add(Conv2D(32, kernel_size = 3, activation='relu', input_shape = (28, 28, 1)))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size = 5, strides=2, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(64, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size = 5, strides=2, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(128, kernel_size = 4, activation='relu'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))

# COMPILE WITH ADAM OPTIMIZER AND CROSS ENTROPY COST
model.compile(optimizer="RMSprop", loss="categorical_crossentropy", metrics=["accuracy"])
```

Our model looks like this now:

In [12]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320       
_________________________________________________________________
batch_normalization (BatchNo (None, 26, 26, 32)        128       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 32)        9248      
_________________________________________________________________
batch_normalization_1 (Batch (None, 24, 24, 32)        128       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 32)        25632     
_________________________________________________________________
batch_normalization_2 (Batch (None, 12, 12, 32)        128       
_________________________________________________________________
dropout (Dropout)            (None, 12, 12, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        18496     
_________________________________________________________________
batch_normalization_3 (Batch (None, 10, 10, 64)        256       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 64)          36928     
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 64)          256       
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          102464    
_________________________________________________________________
batch_normalization_5 (Batch (None, 4, 4, 64)          256       
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 64)          0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 1, 1, 128)         131200    
_________________________________________________________________
batch_normalization_6 (Batch (None, 1, 1, 128)         512       
_________________________________________________________________
flatten (Flatten)            (None, 128)               0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense (Dense)                (None, 10)                1290      
=================================================================
Total params: 327,242
Trainable params: 326,410
Non-trainable params: 832
_________________________________________________________________
```

import tensorflow as tf

if tf.test.gpu_device_name(): print('Default GPU Device:{}'.format(tf.test.gpu_device_name())) else: print("Please install GPU version of TF")

Check cuda avaialable or not:

In [13]:

```
import time
import glob
import torch
import os

from IPython.display import Image, clear_output
print('PyTorch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0)
if torch.cuda.is_available() else 'CPU'))
```

```
PyTorch 1.5.1 _CudaDeviceProperties(name='Tesla P100-PCIE-16GB', maj
or=6, minor=0, total_memory=16280MB, multi_processor_count=56)
```

In [14]:

```
# DECREASE LEARNING RATE EACH EPOCH
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.951 ** x)
# TRAIN NETWORKS
History = None
epochs = 40
X_train2, X_val2, Y_train2, Y_val2 = train_test_split(X_train, Y_train, test_siz
e = 0.1)
History = model.fit(datagenerated.flow(X_train2,Y_train2, batch_size=64),
                                    epochs = epochs, steps_per_epoch = X_train2.
shape[0]//128,
                                    validation_data = (X_val2,Y_val2), callbacks
=[annealer], verbose=1)
```

```
Epoch 1/40
295/295 [==============================] - 10s 35ms/step - loss: 0.9
598 - accuracy: 0.7047 - val_loss: 0.7892 - val_accuracy: 0.7781 - l
r: 0.0010
Epoch 2/40
295/295 [==============================] - 10s 35ms/step - loss: 0.2
426 - accuracy: 0.9271 - val_loss: 0.1698 - val_accuracy: 0.9464 - l
r: 9.5100e-04
Epoch 3/40
295/295 [==============================] - 10s 35ms/step - loss: 0.1
719 - accuracy: 0.9472 - val_loss: 0.0509 - val_accuracy: 0.9855 - l
r: 9.0440e-04
Epoch 4/40
295/295 [==============================] - 10s 34ms/step - loss: 0.1
366 - accuracy: 0.9590 - val_loss: 0.0534 - val_accuracy: 0.9850 - l
r: 8.6009e-04
Epoch 5/40
295/295 [==============================] - 11s 38ms/step - loss: 0.1
149 - accuracy: 0.9667 - val_loss: 0.0373 - val_accuracy: 0.9883 - l
r: 8.1794e-04
Epoch 6/40
295/295 [==============================] - 10s 36ms/step - loss: 0.1
015 - accuracy: 0.9702 - val_loss: 0.0389 - val_accuracy: 0.9883 - l
r: 7.7786e-04
Epoch 7/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
856 - accuracy: 0.9732 - val_loss: 0.0352 - val_accuracy: 0.9886 - l
r: 7.3975e-04
Epoch 8/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
795 - accuracy: 0.9763 - val_loss: 0.0339 - val_accuracy: 0.9910 - l
r: 7.0350e-04
Epoch 9/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
773 - accuracy: 0.9766 - val_loss: 0.0339 - val_accuracy: 0.9900 - l
r: 6.6903e-04
Epoch 10/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
777 - accuracy: 0.9778 - val_loss: 0.0312 - val_accuracy: 0.9917 - l
r: 6.3625e-04
Epoch 11/40
295/295 [==============================] - 11s 37ms/step - loss: 0.0
667 - accuracy: 0.9793 - val_loss: 0.0276 - val_accuracy: 0.9936 - l
r: 6.0507e-04
Epoch 12/40
295/295 [==============================] - 11s 36ms/step - loss: 0.0
598 - accuracy: 0.9820 - val_loss: 0.0287 - val_accuracy: 0.9931 - l
r: 5.7542e-04
Epoch 13/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
629 - accuracy: 0.9815 - val_loss: 0.0315 - val_accuracy: 0.9919 - l
r: 5.4723e-04
Epoch 14/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
619 - accuracy: 0.9822 - val_loss: 0.0279 - val_accuracy: 0.9924 - l
r: 5.2041e-04
Epoch 15/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
583 - accuracy: 0.9830 - val_loss: 0.0292 - val_accuracy: 0.9926 - l
r: 4.9491e-04
Epoch 16/40
```

```
295/295 [==============================] - 11s 37ms/step - loss: 0.0
476 - accuracy: 0.9859 - val_loss: 0.0309 - val_accuracy: 0.9931 - l
r: 4.7066e-04
Epoch 17/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
535 - accuracy: 0.9846 - val_loss: 0.0218 - val_accuracy: 0.9940 - l
r: 4.4760e-04
Epoch 18/40
295/295 [==============================] - 11s 36ms/step - loss: 0.0
527 - accuracy: 0.9853 - val_loss: 0.0271 - val_accuracy: 0.9924 - l
r: 4.2567e-04
Epoch 19/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
511 - accuracy: 0.9842 - val_loss: 0.0243 - val_accuracy: 0.9933 - l
r: 4.0481e-04
Epoch 20/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
452 - accuracy: 0.9867 - val_loss: 0.0288 - val_accuracy: 0.9933 - l
r: 3.8497e-04
Epoch 21/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
507 - accuracy: 0.9847 - val_loss: 0.0258 - val_accuracy: 0.9924 - l
r: 3.6611e-04
Epoch 22/40
295/295 [==============================] - 11s 37ms/step - loss: 0.0
459 - accuracy: 0.9868 - val_loss: 0.0240 - val_accuracy: 0.9938 - l
r: 3.4817e-04
Epoch 23/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
415 - accuracy: 0.9873 - val_loss: 0.0330 - val_accuracy: 0.9929 - l
r: 3.3111e-04
Epoch 24/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
411 - accuracy: 0.9881 - val_loss: 0.0288 - val_accuracy: 0.9926 - l
r: 3.1488e-04
Epoch 25/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
430 - accuracy: 0.9875 - val_loss: 0.0241 - val_accuracy: 0.9926 - l
r: 2.9946e-04
Epoch 26/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
393 - accuracy: 0.9887 - val_loss: 0.0253 - val_accuracy: 0.9926 - l
r: 2.8478e-04
Epoch 27/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
370 - accuracy: 0.9886 - val_loss: 0.0289 - val_accuracy: 0.9921 - l
r: 2.7083e-04
Epoch 28/40
295/295 [==============================] - 11s 36ms/step - loss: 0.0
381 - accuracy: 0.9892 - val_loss: 0.0254 - val_accuracy: 0.9938 - l
r: 2.5756e-04
Epoch 29/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
399 - accuracy: 0.9886 - val_loss: 0.0250 - val_accuracy: 0.9936 - l
r: 2.4494e-04
Epoch 30/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
368 - accuracy: 0.9892 - val_loss: 0.0257 - val_accuracy: 0.9933 - l
r: 2.3294e-04
Epoch 31/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
```

```
327 - accuracy: 0.9901 - val_loss: 0.0246 - val_accuracy: 0.9936 - l
r: 2.2152e-04
Epoch 32/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
342 - accuracy: 0.9897 - val_loss: 0.0251 - val_accuracy: 0.9936 - l
r: 2.1067e-04
Epoch 33/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
341 - accuracy: 0.9901 - val_loss: 0.0271 - val_accuracy: 0.9936 - l
r: 2.0034e-04
Epoch 34/40
295/295 [==============================] - 11s 37ms/step - loss: 0.0
293 - accuracy: 0.9911 - val_loss: 0.0255 - val_accuracy: 0.9945 - l
r: 1.9053e-04
Epoch 35/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
336 - accuracy: 0.9907 - val_loss: 0.0275 - val_accuracy: 0.9924 - l
r: 1.8119e-04
Epoch 36/40
295/295 [==============================] - 10s 35ms/step - loss: 0.0
336 - accuracy: 0.9907 - val_loss: 0.0264 - val_accuracy: 0.9926 - l
r: 1.7231e-04
Epoch 37/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
300 - accuracy: 0.9911 - val_loss: 0.0273 - val_accuracy: 0.9931 - l
r: 1.6387e-04
Epoch 38/40
295/295 [==============================] - 10s 34ms/step - loss: 0.0
309 - accuracy: 0.9906 - val_loss: 0.0280 - val_accuracy: 0.9931 - l
r: 1.5584e-04
Epoch 39/40
295/295 [==============================] - 11s 36ms/step - loss: 0.0
282 - accuracy: 0.9918 - val_loss: 0.0234 - val_accuracy: 0.9940 - l
r: 1.4820e-04
Epoch 40/40
295/295 [==============================] - 11s 36ms/step - loss: 0.0
293 - accuracy: 0.9917 - val_loss: 0.0272 - val_accuracy: 0.9936 - l
r: 1.4094e-04
```

In [15]:

```
print("Final train_data accuracy:",History.history['accuracy'][-1])
print("Final val_data accuracy:",History.history['val_accuracy'][-1])
```

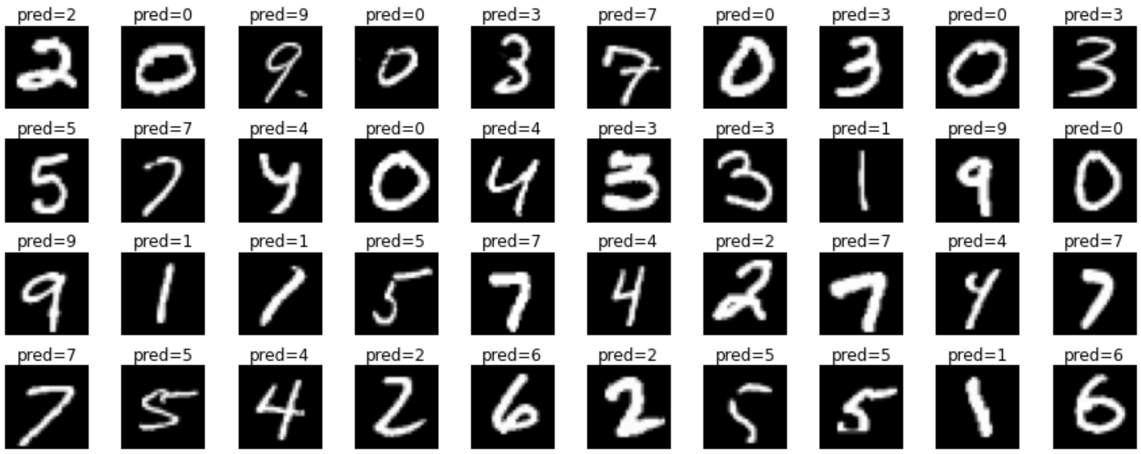
```
Final train_data accuracy: 0.9916843175888062
Final val_data accuracy: 0.993571400642395
```

In [16]:

```
# ENSEMBLE PREDICTIONS AND SUBMIT
results = np.zeros( (X_test.shape[0],10) )
results = results + model.predict(X_test)
results = np.argmax(results,axis = 1)
results = pd.Series(results,name="Label")
submission = pd.concat([pd.Series(range(1,28001),name = "ImageId"),results],axis
= 1)
submission.to_csv("MNIST-LeNet.csv",index=False)
```

In [17]:

```
# PREVIEW PREDICTIONS
plt.figure(figsize=(15,6))
for i in range(40):
    plt.subplot(4, 10, i+1)
    plt.imshow(X_test[i].reshape((28,28)),cmap=plt.cm.gray)
    plt.title("pred=%d" % results[i],y=0.9 ,pad=10)
    plt.axis('off')
plt.subplots_adjust(wspace=0.4, hspace=0.1)
plt.show()
```



Let's save our model in a zip file (I want to use it later)

In [18]:

```
model.save('model_data')
```

In [19]:

```
import os
import zipfile

zf = zipfile.ZipFile("model_data.zip", "w")
for dirname, subdirs, files in os.walk("model_data"):
    zf.write(dirname)
    for filename in files:
        zf.write(os.path.join(dirname, filename))
zf.close()
```