MLP on MNIST Dataset

In [1]:

```
import keras
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
import matplotlib.pyplot as plt
import numpy as np
import time
```

```
Using TensorFlow backend.
```

[1] Dataset Loading and pre-processing

In [2]:

```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [3]:

```
print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d, %d)"%(X_
train.shape[1], X_train.shape[2]))
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d,
%d)"%(X_test.shape[1], X_test.shape[2]))
```

```
Number of training examples : 60000 and each image is of shape (28, 28)
Number of training examples : 10000 and each image is of shape (28, 28)
```

In [4]:

```
from keras import backend as K

# input image dimensions
img_rows, img_cols = 28, 28

if K.image_data_format() == 'channels_first':
    X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
    X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
    X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

In [5]:

```
print("Number of training examples :", X_train.shape[0], "and each image is of shape :",X_train.sh
ape)
print("Number of training examples :", X_test.shape[0], "and each image is of shape
:",X_test.shape)
```

```
Number of training examples : 60000 and each image is of shape : (60000, 28, 28, 1)
Number of training examples : 10000 and each image is of shape : (10000, 28, 28, 1)
```

In [6]:

```
# Normalization
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

```
X_train /= 255
X_test /= 255
```

In [7]:

```
# convert class vectors to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

[2] CNN Models

In [8]:

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout,Flatten
from keras.layers.normalization import BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
```

In [9]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 30
```

In [10]:

```
def plt_epoch_vs_loss(x, vy, ty):
    fig = plt.figure(figsize=(9,7))
    sns.set_style("whitegrid",{'axes.grid' : True})
    plt.plot(x, vy, 'b', label="Validation Loss")
    plt.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.xlabel("epoch")
    plt.ylabel("Categorical Crossentropy Loss")
    plt.title("Loss")
    plt.show()
```

[2.1] Model 1

Input(28,28) - CONV - ReLu - CONV - ReLu - Pool - Dropout - CONV - ReLu - Pool - Dropout - Flatten - ReLu - Dropout - Softmax(Output(10)) - Adam Optimizer

In [11]:

```
# Model 1 parameters :
#       Conv layers : layer1 = 32,layer2 = 64,layer3 = 128
#       kernal : (2,2)
#       Pooling : (2,2)
#       Dropout : 0.5

model1 = Sequential()
model1.add(Conv2D(32, kernel_size=(2, 2),padding='same',activation='relu',input_shape=input_shape))
model1.add(Conv2D(64, kernel_size=(2, 2),padding='same',activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.5))
model1.add(Conv2D(128, kernel_size=(2, 2),padding='same',activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.5))
model1.add(Flatten())
model1.add(Dense(128, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(output_dim, activation='softmax'))
```

```
model1.summary()
```

```
WARNING:tensorflow:From C:\Users\sanjeev\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:4070: The name tf.nn.max_pool is deprecated. Please u
se tf.nn.max_pool2d instead.

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        160
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 64)        8256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 128)       32896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               802944
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 845,546
Trainable params: 845,546
Non-trainable params: 0
_________________________________________________________________
```

In [12]:

```
model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model1.fit(X_train, y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validatio
n_data=(X_test, y_test))
```

```
WARNING:tensorflow:From C:\Users\sanjeev\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Plea
se use tf.compat.v1.global_variables instead.

Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 159s 3ms/step - loss: 0.3913 - accuracy: 0.8763 - v
al_loss: 0.0818 - val_accuracy: 0.9743
Epoch 2/30
60000/60000 [==============================] - 163s 3ms/step - loss: 0.1400 - accuracy: 0.9571 - v
al_loss: 0.0459 - val_accuracy: 0.9854
Epoch 3/30
60000/60000 [==============================] - 154s 3ms/step - loss: 0.1083 - accuracy: 0.9676 - v
al_loss: 0.0368 - val_accuracy: 0.9883
Epoch 4/30
60000/60000 [==============================] - 164s 3ms/step - loss: 0.0942 - accuracy: 0.9712 - v
al_loss: 0.0353 - val_accuracy: 0.9877
Epoch 5/30
60000/60000 [==============================] - 157s 3ms/step - loss: 0.0835 - accuracy: 0.9746 - v
al_loss: 0.0316 - val_accuracy: 0.9902
Epoch 6/30
60000/60000 [==============================] - 152s 3ms/step - loss: 0.0779 - accuracy: 0.9762 - v
al_loss: 0.0291 - val_accuracy: 0.9903
Epoch 7/30
60000/60000 [==============================] - 152s 3ms/step - loss: 0.0732 - accuracy: 0.9776 - v
al_loss: 0.0284 - val_accuracy: 0.9907
Epoch 8/30
60000/60000 [==============================] - 153s 3ms/step - loss: 0.0674 - accuracy: 0.9795 - v
al_loss: 0.0286 - val_accuracy: 0.9897
Epoch 9/30
```

```
60000/60000 [==============================] - 151s 3ms/step - loss: 0.0626 - accuracy: 0.9816 - v
al_loss: 0.0273 - val_accuracy: 0.9918
Epoch 10/30
60000/60000 [==============================] - 155s 3ms/step - loss: 0.0603 - accuracy: 0.9819 - v
al_loss: 0.0278 - val_accuracy: 0.9907
Epoch 11/30
60000/60000 [==============================] - 166s 3ms/step - loss: 0.0564 - accuracy: 0.9822 - v
al_loss: 0.0255 - val_accuracy: 0.9911
Epoch 12/30
60000/60000 [==============================] - 157s 3ms/step - loss: 0.0577 - accuracy: 0.9825 - v
al_loss: 0.0253 - val_accuracy: 0.9917
Epoch 13/30
60000/60000 [==============================] - 158s 3ms/step - loss: 0.0515 - accuracy: 0.9836 - v
al_loss: 0.0235 - val_accuracy: 0.9929
Epoch 14/30
60000/60000 [==============================] - 158s 3ms/step - loss: 0.0520 - accuracy: 0.9845 - v
al_loss: 0.0221 - val_accuracy: 0.9926
Epoch 15/30
60000/60000 [==============================] - 158s 3ms/step - loss: 0.0492 - accuracy: 0.9846 - v
al_loss: 0.0223 - val_accuracy: 0.9929
Epoch 16/30
60000/60000 [==============================] - 150s 3ms/step - loss: 0.0481 - accuracy: 0.9851 - v
al_loss: 0.0221 - val_accuracy: 0.9929
Epoch 17/30
60000/60000 [==============================] - 153s 3ms/step - loss: 0.0475 - accuracy: 0.9845 - v
al_loss: 0.0230 - val_accuracy: 0.9923
Epoch 18/30
60000/60000 [==============================] - 146s 2ms/step - loss: 0.0453 - accuracy: 0.9861 - v
al_loss: 0.0211 - val_accuracy: 0.9937
Epoch 19/30
60000/60000 [==============================] - 146s 2ms/step - loss: 0.0459 - accuracy: 0.9860 - v
al_loss: 0.0225 - val_accuracy: 0.9926
Epoch 20/30
60000/60000 [==============================] - 146s 2ms/step - loss: 0.0427 - accuracy: 0.9870 - v
al_loss: 0.0218 - val_accuracy: 0.9932
Epoch 21/30
60000/60000 [==============================] - 149s 2ms/step - loss: 0.0413 - accuracy: 0.9869 - v
al_loss: 0.0200 - val_accuracy: 0.9929
Epoch 22/30
60000/60000 [==============================] - 147s 2ms/step - loss: 0.0402 - accuracy: 0.9873 - v
al_loss: 0.0209 - val_accuracy: 0.9933
Epoch 23/30
60000/60000 [==============================] - 147s 2ms/step - loss: 0.0401 - accuracy: 0.9872 - v
al_loss: 0.0215 - val_accuracy: 0.9929
Epoch 24/30
60000/60000 [==============================] - 146s 2ms/step - loss: 0.0402 - accuracy: 0.9873 - v
al_loss: 0.0198 - val_accuracy: 0.9938
Epoch 25/30
60000/60000 [==============================] - 151s 3ms/step - loss: 0.0391 - accuracy: 0.9876 - v
al_loss: 0.0203 - val_accuracy: 0.9934
Epoch 26/30
60000/60000 [==============================] - 146s 2ms/step - loss: 0.0396 - accuracy: 0.9878 - v
al_loss: 0.0184 - val_accuracy: 0.9944
Epoch 27/30
60000/60000 [==============================] - 146s 2ms/step - loss: 0.0370 - accuracy: 0.9878 - v
al_loss: 0.0213 - val_accuracy: 0.9925
Epoch 28/30
60000/60000 [==============================] - 146s 2ms/step - loss: 0.0371 - accuracy: 0.9883 - v
al_loss: 0.0189 - val_accuracy: 0.9942
Epoch 29/30
60000/60000 [==============================] - 147s 2ms/step - loss: 0.0348 - accuracy: 0.9891 - v
al_loss: 0.0190 - val_accuracy: 0.9931
Epoch 30/30
60000/60000 [==============================] - 146s 2ms/step - loss: 0.0349 - accuracy: 0.9888 - v
al_loss: 0.0187 - val_accuracy: 0.9944
```

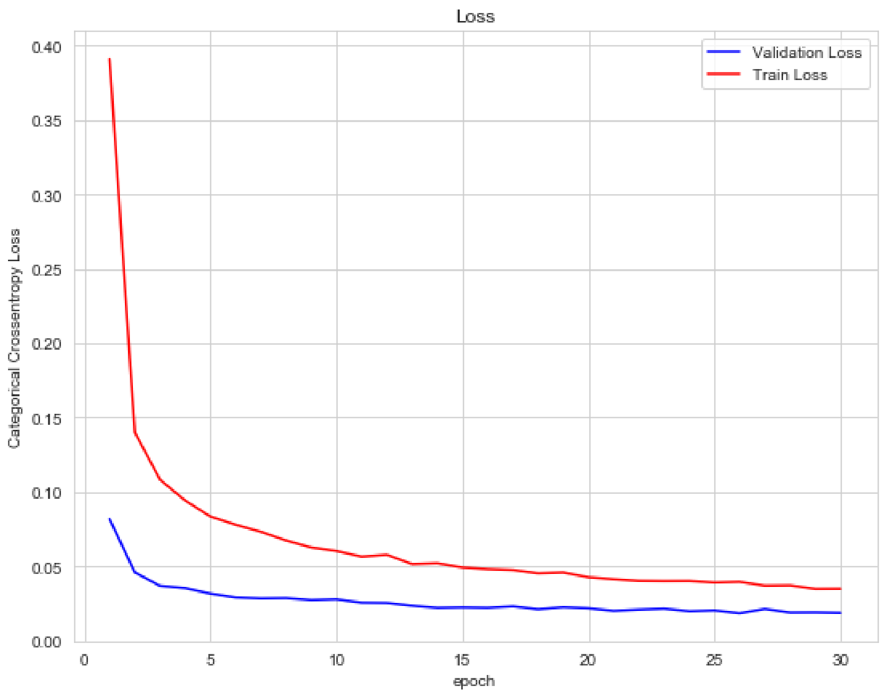
In [14]:

```
score1 = model1.evaluate(X_test, y_test, verbose=0)
print('Test score:', score1[0])
print('Test accuracy:', score1[1])

# list of epoch numbers
epochs = list(range(1,nb_epoch+1))
val_loss = history.history['val_loss']
train_loss = history.history['loss']
```

```
plt_epoch_vs_loss(epochs, val_loss, train_loss)
```

```
Test score: 0.018688983194074717
Test accuracy: 0.9944000244140625
```



[2.2] Model 2

Input(28,28) - CONV - ReLu - CONV - ReLu - Pool - Dropout - CONV - ReLu - CONV - ReLu - Pool - Dropout - CONV - ReLu - Pool - Dropout - Flatten - ReLu - Dropout - Softmax(Output(10)) - Adam Optimizer

In [15]:

```
# Model 2 parameters :
#       Conv layers : 5
#       layer1 = 32,layer2 = 64,layer3 = 128,layer4 = 256,layer5 = 512
#       kernal  : (3,3)
#       Pooling : (3,3)
#       Dropout : 0.5

from keras.initializers import glorot_normal

model2 = Sequential()
model2.add(Conv2D(32, kernel_size=(3, 3),padding='same',activation='relu',input_shape=input_shape))
model2.add(Conv2D(64, kernel_size=(3, 3),padding='same',activation='relu'))
model2.add(MaxPooling2D(pool_size=(3, 3)))
model2.add(Dropout(0.5))
model2.add(Conv2D(128, kernel_size=(3, 3),padding='same',activation='relu'))
model2.add(Conv2D(256, kernel_size=(3, 3),padding='same',activation='relu'))
model2.add(MaxPooling2D(pool_size=(3, 3)))
model2.add(Dropout(0.5))
model2.add(Conv2D(512, kernel_size=(3, 3),padding='same',activation='relu'))
model2.add(MaxPooling2D(pool_size=(3, 3)))
model2.add(Dropout(0.5))
model2.add(Flatten())
model2.add(Dense(256, activation='relu',kernel_initializer = glorot_normal(seed=None)))
model2.add(Dropout(0.5))
model2.add(Dense(output_dim, activation='softmax'))

model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 32)        320
```

```
conv2d_4 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 64)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 9, 9, 64)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 9, 9, 128)         73856
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 9, 9, 256)         295168
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 3, 3, 256)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 3, 3, 512)         1180160
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 1, 1, 512)         0
_________________________________________________________________
dropout_6 (Dropout)          (None, 1, 1, 512)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                2570
=================================================================
Total params: 1,701,898
Trainable params: 1,701,898
Non-trainable params: 0
_________________________________________________________________
```

In [16]:

```
model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model2.fit(X_train, y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validatio
n_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 516s 9ms/step - loss: 0.5370 - accuracy: 0.8160 - v
al_loss: 0.0441 - val_accuracy: 0.9861
Epoch 2/30
60000/60000 [==============================] - 513s 9ms/step - loss: 0.0954 - accuracy: 0.9723 - v
al_loss: 0.0296 - val_accuracy: 0.9912
Epoch 3/30
60000/60000 [==============================] - 500s 8ms/step - loss: 0.0692 - accuracy: 0.9801 - v
al_loss: 0.0215 - val_accuracy: 0.9932
Epoch 4/30
60000/60000 [==============================] - 498s 8ms/step - loss: 0.0563 - accuracy: 0.9843 - v
al_loss: 0.0176 - val_accuracy: 0.9947
Epoch 5/30
60000/60000 [==============================] - 499s 8ms/step - loss: 0.0550 - accuracy: 0.9844 - v
al_loss: 0.0205 - val_accuracy: 0.9936
Epoch 6/30
60000/60000 [==============================] - 502s 8ms/step - loss: 0.0464 - accuracy: 0.9866 - v
al_loss: 0.0183 - val_accuracy: 0.9946
Epoch 7/30
60000/60000 [==============================] - 498s 8ms/step - loss: 0.0449 - accuracy: 0.9874 - v
al_loss: 0.0198 - val_accuracy: 0.9943
Epoch 8/30
60000/60000 [==============================] - 499s 8ms/step - loss: 0.0405 - accuracy: 0.9884 - v
al_loss: 0.0204 - val_accuracy: 0.9935
Epoch 9/30
60000/60000 [==============================] - 563s 9ms/step - loss: 0.0382 - accuracy: 0.9895 - v
al_loss: 0.0223 - val_accuracy: 0.9938
Epoch 10/30
60000/60000 [==============================] - 549s 9ms/step - loss: 0.0378 - accuracy: 0.9894 - v
al_loss: 0.0167 - val_accuracy: 0.9953
Epoch 11/30
60000/60000 [==============================] - 528s 9ms/step - loss: 0.0357 - accuracy: 0.9897 - v
al_loss: 0.0249 - val_accuracy: 0.9932
```

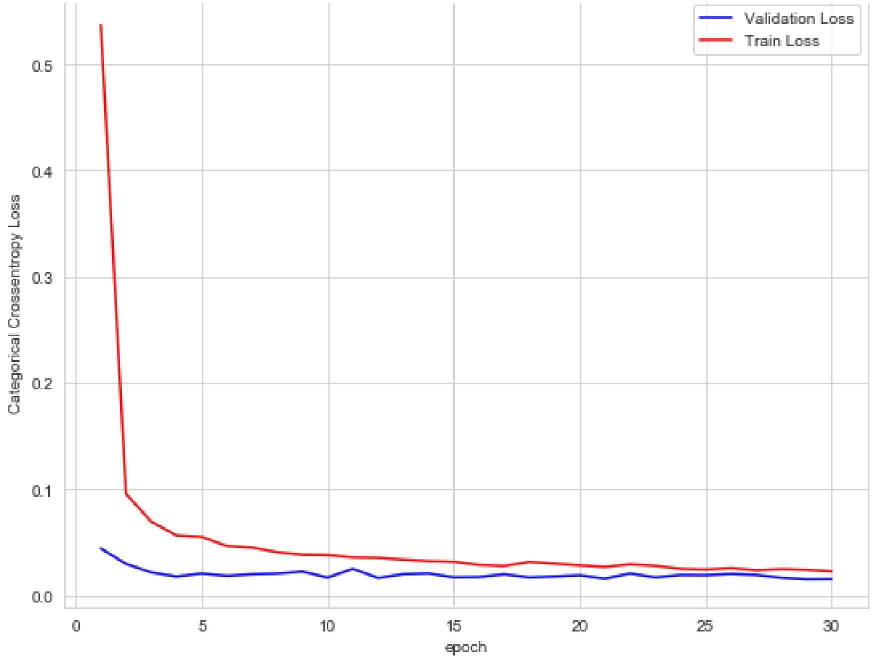
```
Epoch 12/30
60000/60000 [==============================] - 523s 9ms/step - loss: 0.0353 - accuracy: 0.9901 - v
al_loss: 0.0163 - val_accuracy: 0.9952
Epoch 13/30
60000/60000 [==============================] - 518s 9ms/step - loss: 0.0335 - accuracy: 0.9903 - v
al_loss: 0.0199 - val_accuracy: 0.9942
Epoch 14/30
60000/60000 [==============================] - 519s 9ms/step - loss: 0.0321 - accuracy: 0.9907 - v
al_loss: 0.0205 - val_accuracy: 0.9951
Epoch 15/30
60000/60000 [==============================] - 517s 9ms/step - loss: 0.0315 - accuracy: 0.9909 - v
al_loss: 0.0169 - val_accuracy: 0.9946
Epoch 16/30
60000/60000 [==============================] - 527s 9ms/step - loss: 0.0286 - accuracy: 0.9912 - v
al_loss: 0.0172 - val_accuracy: 0.9954
Epoch 17/30
60000/60000 [==============================] - 522s 9ms/step - loss: 0.0276 - accuracy: 0.9917 - v
al_loss: 0.0198 - val_accuracy: 0.9945
Epoch 18/30
60000/60000 [==============================] - 520s 9ms/step - loss: 0.0313 - accuracy: 0.9911 - v
al_loss: 0.0168 - val_accuracy: 0.9956
Epoch 19/30
60000/60000 [==============================] - 513s 9ms/step - loss: 0.0298 - accuracy: 0.9915 - v
al_loss: 0.0177 - val_accuracy: 0.9951
Epoch 20/30
60000/60000 [==============================] - 515s 9ms/step - loss: 0.0281 - accuracy: 0.9919 - v
al_loss: 0.0188 - val_accuracy: 0.9947
Epoch 21/30
60000/60000 [==============================] - 516s 9ms/step - loss: 0.0267 - accuracy: 0.9926 - v
al_loss: 0.0157 - val_accuracy: 0.9954
Epoch 22/30
60000/60000 [==============================] - 512s 9ms/step - loss: 0.0291 - accuracy: 0.9918 - v
al_loss: 0.0205 - val_accuracy: 0.9949
Epoch 23/30
60000/60000 [==============================] - 516s 9ms/step - loss: 0.0277 - accuracy: 0.9918 - v
al_loss: 0.0168 - val_accuracy: 0.9949
Epoch 24/30
60000/60000 [==============================] - 512s 9ms/step - loss: 0.0249 - accuracy: 0.9927 - v
al_loss: 0.0190 - val_accuracy: 0.9951
Epoch 25/30
60000/60000 [==============================] - 512s 9ms/step - loss: 0.0242 - accuracy: 0.9930 - v
al_loss: 0.0189 - val_accuracy: 0.9953
Epoch 26/30
60000/60000 [==============================] - 512s 9ms/step - loss: 0.0256 - accuracy: 0.9930 - v
al_loss: 0.0201 - val_accuracy: 0.9948
Epoch 27/30
60000/60000 [==============================] - 512s 9ms/step - loss: 0.0236 - accuracy: 0.9932 - v
al_loss: 0.0191 - val_accuracy: 0.9955
Epoch 28/30
60000/60000 [==============================] - 512s 9ms/step - loss: 0.0247 - accuracy: 0.9933 - v
al_loss: 0.0165 - val_accuracy: 0.9952
Epoch 29/30
60000/60000 [==============================] - 511s 9ms/step - loss: 0.0239 - accuracy: 0.9934 - v
al_loss: 0.0151 - val_accuracy: 0.9957
Epoch 30/30
60000/60000 [==============================] - 518s 9ms/step - loss: 0.0226 - accuracy: 0.9936 - v
al_loss: 0.0152 - val_accuracy: 0.9955
```

In [17]:

```python
score2 = model2.evaluate(X_test, y_test, verbose=0)
print('Test score:', score2[0])
print('Test accuracy:', score2[1])

# list of epoch numbers
epochs = list(range(1,nb_epoch+1))
val_loss = history.history['val_loss']
train_loss = history.history['loss']
plt_epoch_vs_loss(epochs, val_loss, train_loss)
```

```
Test score: 0.015234262454181245
Test accuracy: 0.9955000281333923
```



[2.3] Model3

Input(28,28) - CONV - ReLu - CONV - ReLu - Pool - Dropout - CONV - ReLu - CONV - ReLu - Pool - Dropout - CONV - ReLu - CONV - ReLu - Pool - Dropout - CONV - ReLu - Pool - Dropout - Flatten - ReLu(BatchNormalization()) - Dropout - Softmax(Output(10)) - Adam Optimizer

In [18]:

```
# Model 7 parameters :
#       Conv layers : 7
#       layer1 = 100,layer2 = 150,layer3 = 200,layer4 = 250,layer5 = 350,layer = 400,layer = 512
#       kernal : (5,5)
#       Pooling : (5,5)
#       Dropout : 0.25

from keras.initializers import he_normal

model3 = Sequential()
model3.add(Conv2D(100, kernel_size=(5, 5),padding='same',activation='relu',input_shape=input_shape)
)
model3.add(Conv2D(150, kernel_size=(5, 5),padding='same',activation='relu'))
model3.add(MaxPooling2D(pool_size=(5, 5),padding = 'same'))
model3.add(Dropout(0.25))
model3.add(Conv2D(200, kernel_size=(5, 5),padding='same',activation='relu'))
model3.add(Conv2D(250, kernel_size=(5, 5),padding='same',activation='relu'))
model3.add(MaxPooling2D(pool_size=(5, 5),padding = 'same'))
model3.add(Dropout(0.25))
model3.add(Conv2D(350, kernel_size=(5, 5),padding='same',activation='relu'))
model3.add(Conv2D(400, kernel_size=(5, 5),padding='same',activation='relu'))
model3.add(MaxPooling2D(pool_size=(5, 5),padding = 'same'))
model3.add(Dropout(0.25))
model3.add(Conv2D(512, kernel_size=(5, 5),padding='same',activation='relu'))
model3.add(MaxPooling2D(pool_size=(5, 5),padding = 'same'))
model3.add(Dropout(0.25))
model3.add(Flatten())
model3.add(Dense(256, activation='relu',kernel_initializer = he_normal(seed=None)))
model3.add(BatchNormalization())
model3.add(Dropout(0.25))
model3.add(Dense(output_dim, activation='softmax'))

model3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_9 (Conv2D)            (None, 28, 28, 100)       2600      
```

```
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 28, 28, 150)       375150    
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 6, 6, 150)         0         
_________________________________________________________________
dropout_8 (Dropout)          (None, 6, 6, 150)         0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 6, 6, 200)         750200    
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 6, 6, 250)         1250250   
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 2, 2, 250)         0         
_________________________________________________________________
dropout_9 (Dropout)          (None, 2, 2, 250)         0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 2, 2, 350)         2187850   
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 2, 2, 400)         3500400   
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 1, 1, 400)         0         
_________________________________________________________________
dropout_10 (Dropout)         (None, 1, 1, 400)         0         
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 1, 1, 512)         5120512   
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 1, 1, 512)         0         
_________________________________________________________________
dropout_11 (Dropout)         (None, 1, 1, 512)         0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 512)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328    
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_12 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570      
=================================================================
Total params: 13,321,884
Trainable params: 13,321,372
Non-trainable params: 512
_________________________________________________________________
```

In [19]:

```
model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model3.fit(X_train, y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validatio
n_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 3974s 66ms/step - loss: 0.3190 - accuracy: 0.8878 -
val_loss: 0.0387 - val_accuracy: 0.9889
Epoch 2/30
60000/60000 [==============================] - 3763s 63ms/step - loss: 0.0574 - accuracy: 0.9847 -
val_loss: 0.0377 - val_accuracy: 0.9900
Epoch 3/30
60000/60000 [==============================] - 3763s 63ms/step - loss: 0.0429 - accuracy: 0.9887 -
val_loss: 0.0325 - val_accuracy: 0.9912
Epoch 4/30
60000/60000 [==============================] - 3730s 62ms/step - loss: 0.0309 - accuracy: 0.9918 -
val_loss: 0.0272 - val_accuracy: 0.9927
Epoch 5/30
60000/60000 [==============================] - 3719s 62ms/step - loss: 0.0294 - accuracy: 0.9924 -
val_loss: 0.0301 - val_accuracy: 0.9918
Epoch 6/30
60000/60000 [==============================] - 3719s 62ms/step - loss: 0.0257 - accuracy: 0.9934 -
val_loss: 0.0352 - val_accuracy: 0.9912
Epoch 7/30
60000/60000 [==============================] - 3710s 62ms/step - loss: 0.0238 - accuracy: 0.9938 -
val_loss: 0.0255 - val_accuracy: 0.9932
Epoch 8/30
60000/60000 [==============================] - 3720s 62ms/step - loss: 0.0188 - accuracy: 0.9951 -
val_loss: 0.0591 - val_accuracy: 0.9854
```

```
val_loss: 0.0391 - val_accuracy: 0.9854
Epoch 9/30
60000/60000 [==============================] - 3715s 62ms/step - loss: 0.0185 - accuracy: 0.9951 -
val_loss: 0.0326 - val_accuracy: 0.9925
Epoch 10/30
60000/60000 [==============================] - 3786s 63ms/step - loss: 0.0177 - accuracy: 0.9952 -
val_loss: 0.0198 - val_accuracy: 0.9945
Epoch 11/30
60000/60000 [==============================] - 3980s 66ms/step - loss: 0.0145 - accuracy: 0.9960 -
val_loss: 0.0353 - val_accuracy: 0.9919
Epoch 12/30
60000/60000 [==============================] - 4411s 74ms/step - loss: 0.0152 - accuracy: 0.9961 -
val_loss: 0.0295 - val_accuracy: 0.9935
Epoch 13/30
60000/60000 [==============================] - 3882s 65ms/step - loss: 0.0129 - accuracy: 0.9966 -
val_loss: 0.0226 - val_accuracy: 0.9947
Epoch 14/30
60000/60000 [==============================] - 3630s 61ms/step - loss: 0.0090 - accuracy: 0.9975 -
val_loss: 0.0258 - val_accuracy: 0.9941
Epoch 15/30
60000/60000 [==============================] - 3636s 61ms/step - loss: 0.0121 - accuracy: 0.9969 -
val_loss: 0.0246 - val_accuracy: 0.9944
Epoch 16/30
60000/60000 [==============================] - 3630s 61ms/step - loss: 0.0122 - accuracy: 0.9970 -
val_loss: 0.0216 - val_accuracy: 0.9937
Epoch 17/30
60000/60000 [==============================] - 3630s 60ms/step - loss: 0.0098 - accuracy: 0.9973 -
val_loss: 0.0258 - val_accuracy: 0.9940
Epoch 18/30
60000/60000 [==============================] - 3627s 60ms/step - loss: 0.0083 - accuracy: 0.9980 -
val_loss: 0.0236 - val_accuracy: 0.9947
Epoch 19/30
60000/60000 [==============================] - 3643s 61ms/step - loss: 0.0114 - accuracy: 0.9972 -
val_loss: 0.0240 - val_accuracy: 0.9949
Epoch 20/30
60000/60000 [==============================] - 3632s 61ms/step - loss: 0.0075 - accuracy: 0.9979 -
val_loss: 0.0286 - val_accuracy: 0.9926
Epoch 21/30
60000/60000 [==============================] - 3805s 63ms/step - loss: 0.0077 - accuracy: 0.9980 -
val_loss: 0.0290 - val_accuracy: 0.9936
Epoch 22/30
60000/60000 [==============================] - 3782s 63ms/step - loss: 0.0098 - accuracy: 0.9973 -
val_loss: 0.0260 - val_accuracy: 0.9936
Epoch 23/30
60000/60000 [==============================] - 3700s 62ms/step - loss: 0.0065 - accuracy: 0.9982 -
val_loss: 0.0221 - val_accuracy: 0.9950
Epoch 24/30
60000/60000 [==============================] - 3705s 62ms/step - loss: 0.0068 - accuracy: 0.9983 -
val_loss: 0.0208 - val_accuracy: 0.9953
Epoch 25/30
60000/60000 [==============================] - 3712s 62ms/step - loss: 0.0063 - accuracy: 0.9984 -
val_loss: 0.0286 - val_accuracy: 0.9932
Epoch 26/30
60000/60000 [==============================] - 3698s 62ms/step - loss: 0.0064 - accuracy: 0.9984 -
val_loss: 0.0320 - val_accuracy: 0.9936
Epoch 27/30
60000/60000 [==============================] - 3644s 61ms/step - loss: 0.0059 - accuracy: 0.9985 -
val_loss: 0.0201 - val_accuracy: 0.9958
Epoch 28/30
60000/60000 [==============================] - 3644s 61ms/step - loss: 0.0051 - accuracy: 0.9986 -
val_loss: 0.0232 - val_accuracy: 0.9955
Epoch 29/30
60000/60000 [==============================] - 3664s 61ms/step - loss: 0.0050 - accuracy: 0.9987 -
val_loss: 0.0239 - val_accuracy: 0.9952
Epoch 30/30
60000/60000 [==============================] - 3644s 61ms/step - loss: 0.0054 - accuracy: 0.9987 -
val_loss: 0.0254 - val_accuracy: 0.9951
```

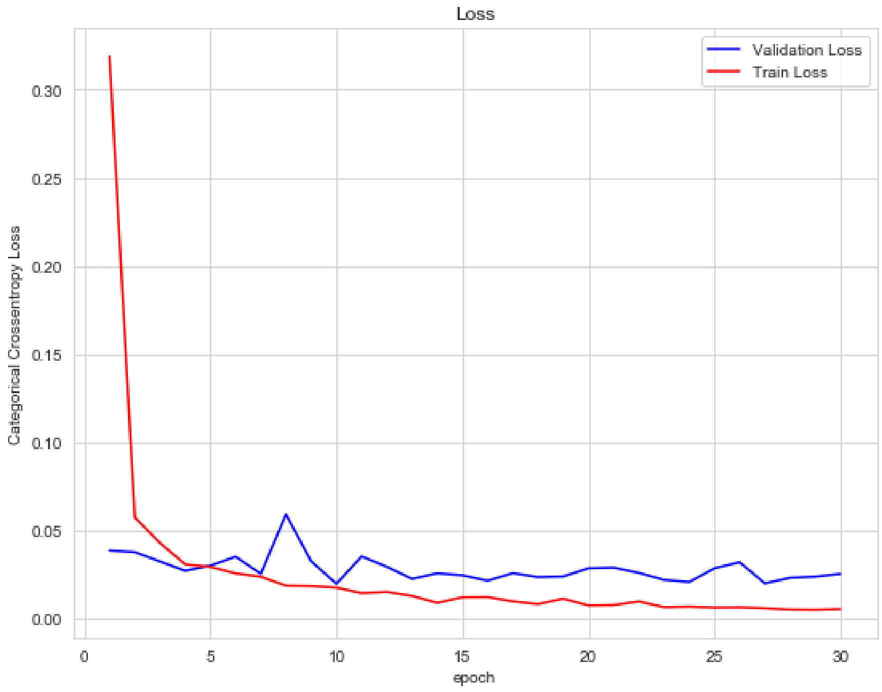
In [20]:

```python
score3 = model3.evaluate(X_test, y_test, verbose=0)
print('Test score:', score3[0])
print('Test accuracy:', score3[1])

# list of epoch numbers
epoch = list(range(1,nb_epoch+1))
val_loss = history.history['val_loss']
```

```
val_loss = history.history['val_loss']
train_loss = history.history['loss']
plt_epoch_vs_loss(epoch, val_loss, train_loss)
```

```
Test score: 0.02541344242626779
Test accuracy: 0.9951000213623047
```



[3] Results

In [22]:

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Model","Conv Layers","Score","Accuracy"]
table.add_row([1,3 ,round(score1[0],3),round(score1[1],3)])
table.add_row([2,5,round(score2[0],3),round(score2[1],3)])
table.add_row([3,7,round(score3[0],3),round(score3[1],3)])

print(table.get_string(title="Results"))
```

```
+------------------------------------------+
|                 Results                  |
+-------+-------------+-------+----------+
| Model | Conv Layers | Score | Accuracy |
+-------+-------------+-------+----------+
|   1   |      3      | 0.019 |  0.994   |
|   2   |      5      | 0.015 |  0.996   |
|   3   |      7      | 0.025 |  0.995   |
+-------+-------------+-------+----------+
```

[4] Conclusion

There is no much difference in the accuracy of all models.