prabhudayala@gmail.com_13

July 22, 2019

[1]:
```
# Credits: https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.
↪py


from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.layers.normalization import BatchNormalization

batch_size = 512
num_classes = 10
epochs = 5

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

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)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

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

```
Using TensorFlow backend.

x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

[2]:
```
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

[3]:
```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
```

```
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
WARNING: Logging before flag parsing goes to stderr.
W0717 21:39:39.668090  8044 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:74: The name tf.get_default_graph
is deprecated. Please use tf.compat.v1.get_default_graph instead.

W0717 21:39:39.680058  8044 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:517: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

W0717 21:39:39.682081  8044 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:4138: The name tf.random_uniform is
deprecated. Please use tf.random.uniform instead.

W0717 21:39:39.701999  8044 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:3976: The name tf.nn.max_pool is
deprecated. Please use tf.nn.max_pool2d instead.

W0717 21:39:39.703994  8044 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:133: The name
tf.placeholder_with_default is deprecated. Please use
tf.compat.v1.placeholder_with_default instead.

W0717 21:39:39.711008  8044 deprecation.py:506] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
W0717 21:39:39.811706  8044 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\optimizers.py:790: The name tf.train.Optimizer is deprecated.
Please use tf.compat.v1.train.Optimizer instead.

W0717 21:39:39.816720  8044 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
```

```
packages\keras\backend\tensorflow_backend.py:3295: The name tf.log is
deprecated. Please use tf.math.log instead.

W0717 21:39:39.882516  8044 deprecation.py:323] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\tensorflow\python\ops\math_grad.py:1250:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where

Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 64s 1ms/step - loss: 0.5720 -
acc: 0.8175 - val_loss: 0.1035 - val_acc: 0.9649
Epoch 2/5
60000/60000 [==============================] - 67s 1ms/step - loss: 0.1262 -
acc: 0.9628 - val_loss: 0.0631 - val_acc: 0.9772
Epoch 3/5
60000/60000 [==============================] - 70s 1ms/step - loss: 0.0880 -
acc: 0.9740 - val_loss: 0.0415 - val_acc: 0.9858
Epoch 4/5
60000/60000 [==============================] - 69s 1ms/step - loss: 0.0691 -
acc: 0.9799 - val_loss: 0.0384 - val_acc: 0.9870
Epoch 5/5
60000/60000 [==============================] - 69s 1ms/step - loss: 0.0569 -
acc: 0.9835 - val_loss: 0.0472 - val_acc: 0.9836
```

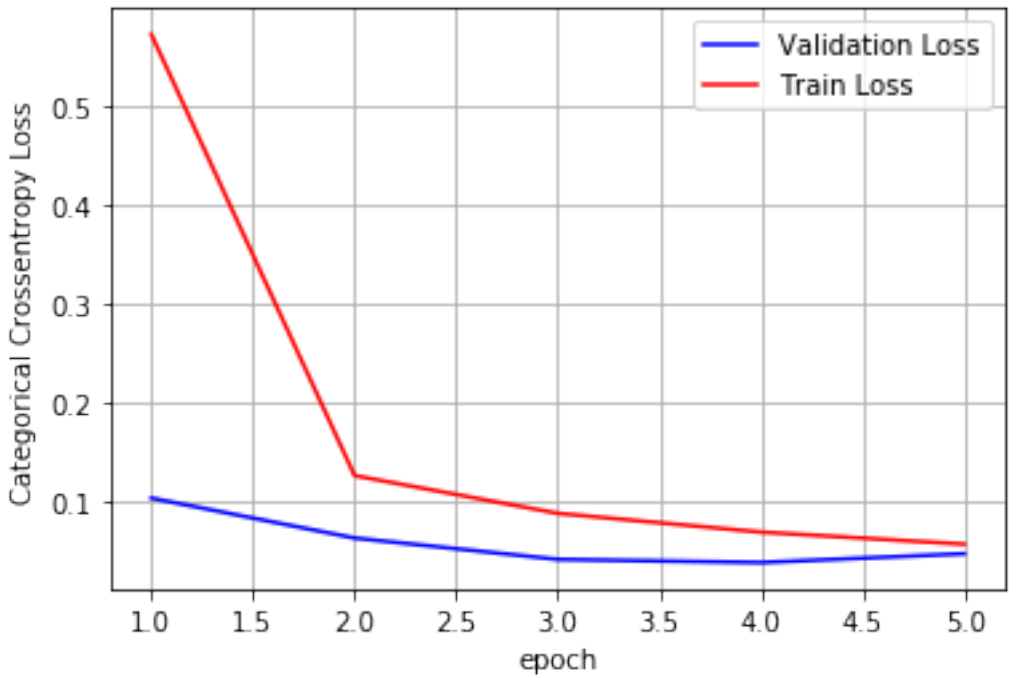
[4]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

x = list(range(1,epochs+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.04723946803898434
Test accuracy: 0.9836
```



[5]:
```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 58s 960us/step - loss: 0.5140 -
```

```
acc: 0.8404 - val_loss: 0.1440 - val_acc: 0.9570
Epoch 2/5
60000/60000 [==============================] - 61s 1ms/step - loss: 0.1436 -
acc: 0.9574 - val_loss: 0.0626 - val_acc: 0.9802
Epoch 3/5
60000/60000 [==============================] - 58s 965us/step - loss: 0.1025 -
acc: 0.9697 - val_loss: 0.0500 - val_acc: 0.9838
Epoch 4/5
60000/60000 [==============================] - 57s 957us/step - loss: 0.0816 -
acc: 0.9759 - val_loss: 0.0534 - val_acc: 0.9816
Epoch 5/5
60000/60000 [==============================] - 54s 905us/step - loss: 0.0705 -
acc: 0.9788 - val_loss: 0.0367 - val_acc: 0.9877
```

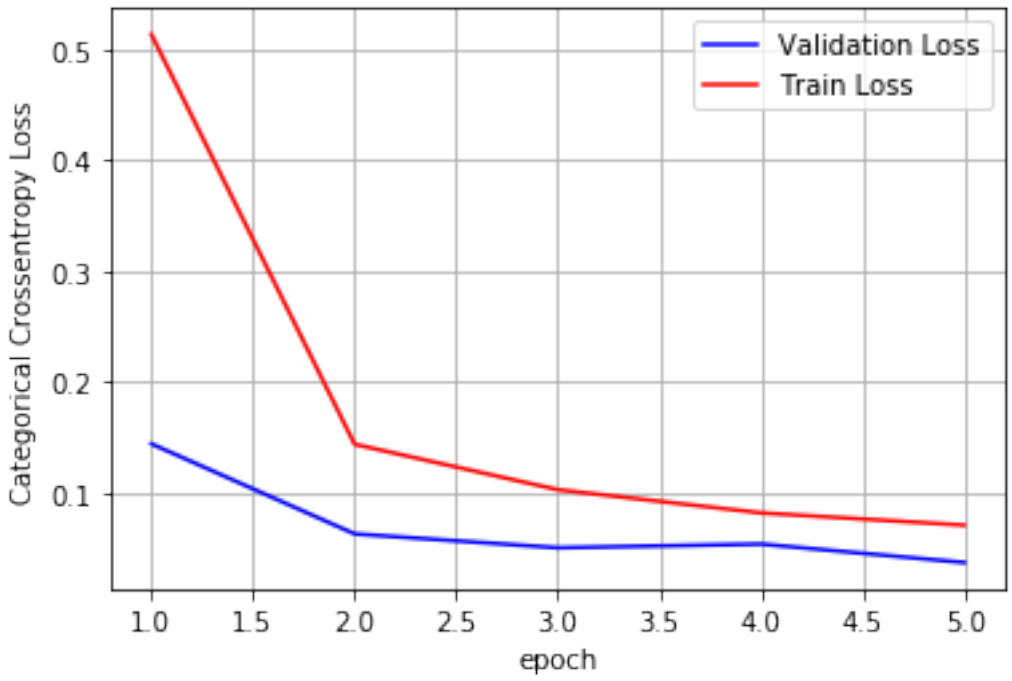
[6]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

x = list(range(1,epochs+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.03671102816282073
Test accuracy: 0.9877
```



[7]:
```
model = Sequential()
model.add(Conv2D(128, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
```

```
          validation_data=(x_test, y_test))
```

```
W0717 21:50:12.915995  8044 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:1834: The name
tf.nn.fused_batch_norm is deprecated. Please use
tf.compat.v1.nn.fused_batch_norm instead.


Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 245s 4ms/step - loss: 0.2719 -
acc: 0.9372 - val_loss: 0.0902 - val_acc: 0.9773
Epoch 2/5
60000/60000 [==============================] - 244s 4ms/step - loss: 0.0584 -
acc: 0.9872 - val_loss: 0.0449 - val_acc: 0.9893
Epoch 3/5
60000/60000 [==============================] - 244s 4ms/step - loss: 0.0352 -
acc: 0.9919 - val_loss: 0.0397 - val_acc: 0.9886
Epoch 4/5
60000/60000 [==============================] - 540s 9ms/step - loss: 0.0249 -
acc: 0.9939 - val_loss: 0.0343 - val_acc: 0.9903
Epoch 5/5
60000/60000 [==============================] - 618s 10ms/step - loss: 0.0184 -
acc: 0.9954 - val_loss: 0.0327 - val_acc: 0.9895
```

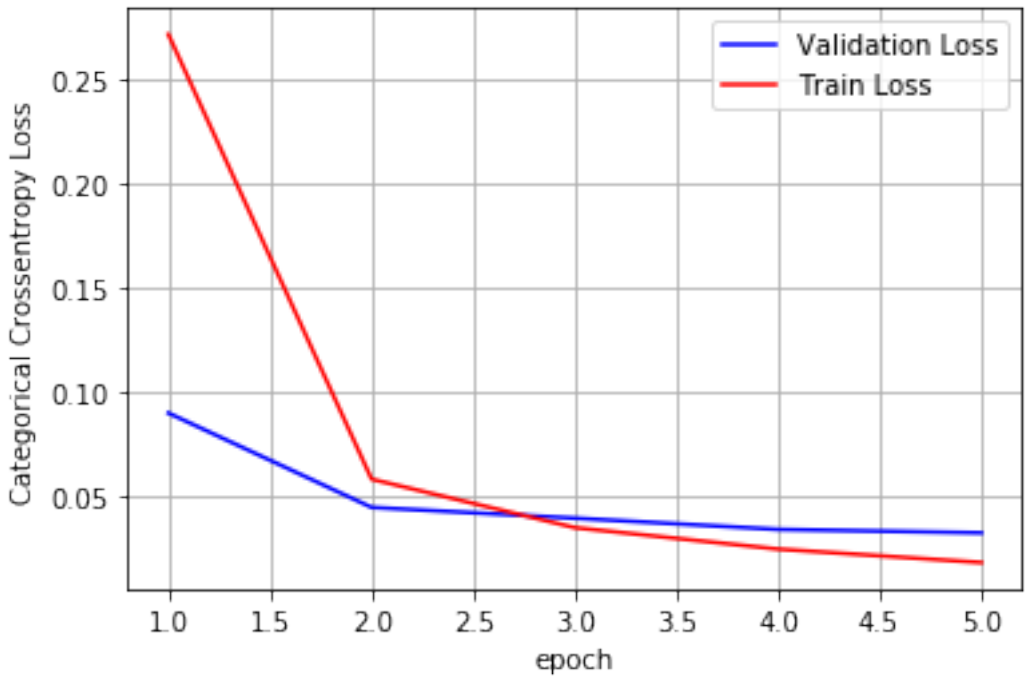
[8]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

x = list(range(1,epochs+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.03268356256503612
Test accuracy: 0.9895
```



This model converged very fast than other model.

[9]:
```
model = Sequential()
model.add(Conv2D(64, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (5, 5), activation='relu'))
model.add(Dropout(0.25))
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
```

```
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 84s 1ms/step - loss: 0.6515 -
acc: 0.8001 - val_loss: 0.1776 - val_acc: 0.9513
Epoch 2/5
60000/60000 [==============================] - 83s 1ms/step - loss: 0.1761 -
acc: 0.9476 - val_loss: 0.0963 - val_acc: 0.9704
Epoch 3/5
60000/60000 [==============================] - 81s 1ms/step - loss: 0.1195 -
acc: 0.9637 - val_loss: 0.1092 - val_acc: 0.9649
Epoch 4/5
60000/60000 [==============================] - 81s 1ms/step - loss: 0.0958 -
acc: 0.9713 - val_loss: 0.0569 - val_acc: 0.9803
Epoch 5/5
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0821 -
acc: 0.9740 - val_loss: 0.1095 - val_acc: 0.9625
```

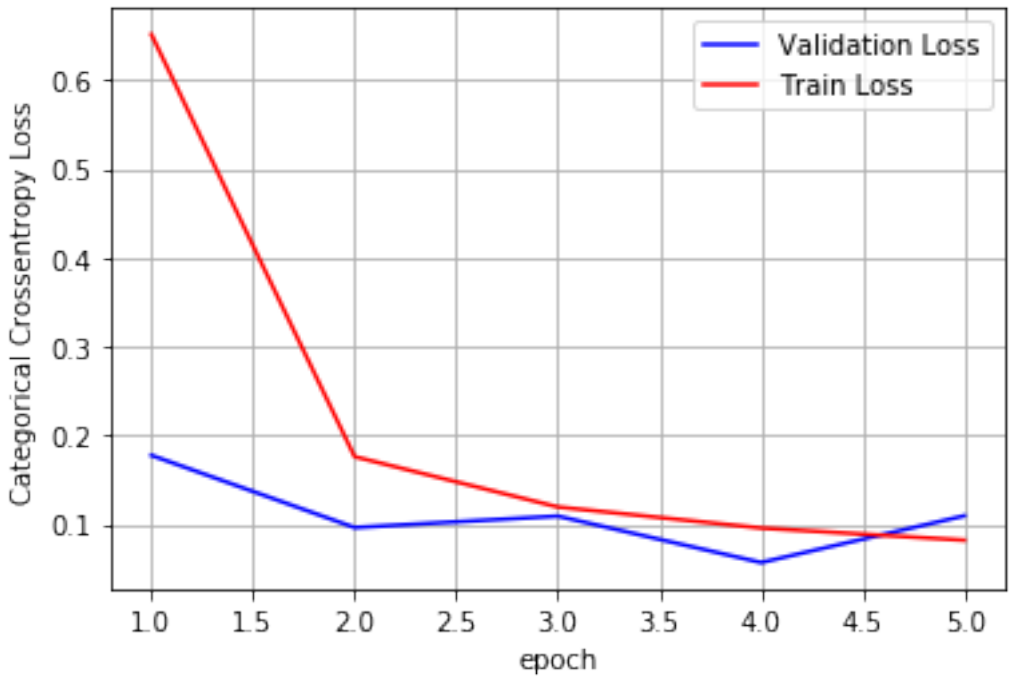
[10]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

x = list(range(1,epochs+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.1095163794439286
Test accuracy: 0.9625
```



The abnormality in test score might be caused by dropout just after two layers of convolution. May be using Using 3x3 kernel after 5x5 kernel hurts the model.

[11]:
```
model = Sequential()
model.add(Conv2D(64, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
          batch_size=batch_size,
```

```
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 78s 1ms/step - loss: 0.3386 -
acc: 0.8995 - val_loss: 0.1138 - val_acc: 0.9630
Epoch 2/5
60000/60000 [==============================] - 73s 1ms/step - loss: 0.0945 -
acc: 0.9720 - val_loss: 0.0652 - val_acc: 0.9798
Epoch 3/5
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0698 -
acc: 0.9789 - val_loss: 0.0426 - val_acc: 0.9875
Epoch 4/5
60000/60000 [==============================] - 71s 1ms/step - loss: 0.0559 -
acc: 0.9829 - val_loss: 0.0366 - val_acc: 0.9878
Epoch 5/5
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0479 -
acc: 0.9851 - val_loss: 0.0389 - val_acc: 0.9880
```

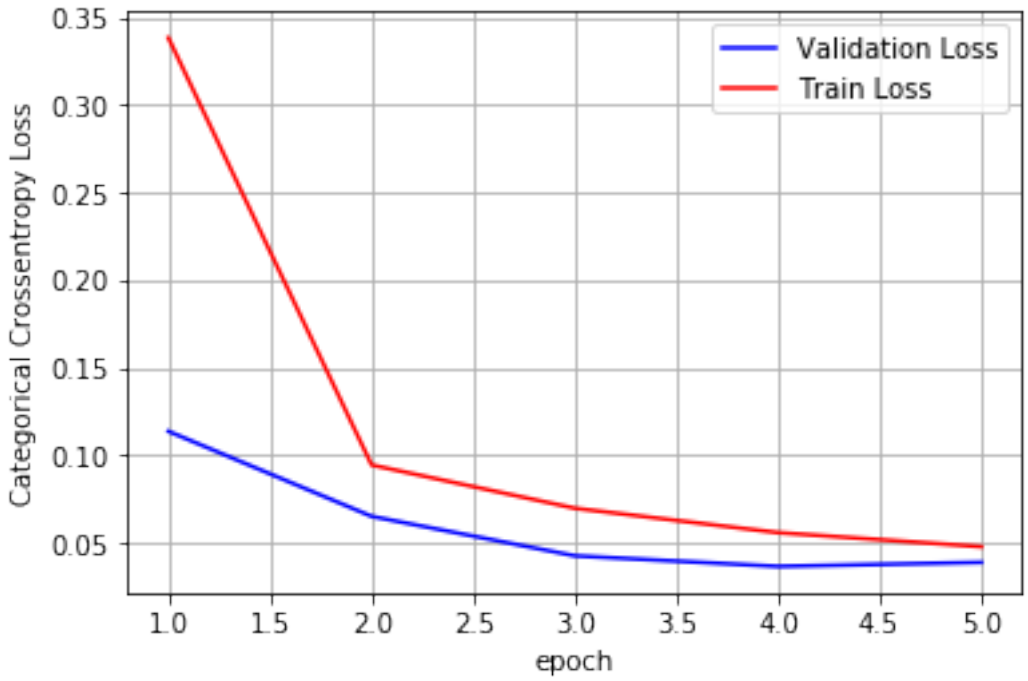
[12]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

x = list(range(1,epochs+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.03889234391124919
Test accuracy: 0.988
```



After removing the dropout from the first layer the model performs better.

[13]:
```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Model", "Convolution", "MaxPooling", "Dropout", "Batch␣
 ↪Normalization", "optimizer", "Train Accuracy", "Test Accuracy"]
x.add_row(["Model 1", "(2x2)", "(2x2)", "Yes", "Yes", "Adadelta",0.983,0.983])
x.add_row(["Model 2", "(3x3)", "(2x2)",  "Yes", "No", "Adadelta",0.978,0.987])
x.add_row(["Model 3", "(3x3) (5x5) (3x3)", "(2x2)", "Yes", "Yes", "Adadelta",0.
 ↪995,0.989])
x.add_row(["Model 4", "(3x3) (5x5) (3x3)", "(2x2)", "Yes", "Yes", "Adadelta",0.
 ↪974,0.962])
x.add_row(["Model 5", "(3x3) (5x5)", "(2x2)",  "Yes", "Yes", "Adadelta",0.985,0.
 ↪988])
x.border=True
print(x)
```

```
+---------+-------------------+------------+---------+---------------------+----
-------+---------------+---------------+
|  Model  |    Convolution    | MaxPooling | Dropout | Batch Normalization |
optimizer | Train Accuracy | Test Accuracy |
+---------+-------------------+------------+---------+---------------------+----
-------+---------------+---------------+
| Model 1 |       (2x2)       |   (2x2)    |   Yes   |         Yes         |
Adadelta |     0.983      |     0.983     |
```

Model 2	(3x3)	(2x2)	Yes	No	Adadelta	0.978	0.987
Model 3	(3x3) (5x5) (3x3)	(2x2)	Yes	Yes	Adadelta	0.995	0.989
Model 4	(3x3) (5x5) (3x3)	(2x2)	Yes	Yes	Adadelta	0.974	0.962
Model 5	(3x3) (5x5)	(2x2)	Yes	Yes	Adadelta	0.985	0.988