jupyter IIIT_MIDAS-2.3 Last Checkpoint: 8 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Code

Importing requred librabries

these are useful as they are needed to run all the fuctions in our notebook

In [74]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.layers import Conv2D, Flatten, Dense, Dropout, Input
from tensorflow.keras.models import Model
import tensorflow as tf
import matplotlib.pyplot as plt
from skimage import io
from tqdm import tqdm
import numpy as np
import os
import shutil
import random
import pandas as pd
from keras.datasets import mnist
import cv2 as cv
# from PIL import Image
```

Generating xtrain and ytrain

Here we are generating xtrain, for that we have to go through the following steps -

1)locate to the Dataset

2)convert the images to grayscale, we ae converting to grayscale as other colors aren't present in our dataset, black and white are the colors available

3)resize the image to 28,28. We are doing this as the standard MNIST dataset is also in 28,28. also cause 28x28 pixels are enough to train ou model

4)appending the image pixels value and label in an array to finally get our xtrain

Printing a counter in the loop so we know how much images have been processed yet

In [75]:
```
train_dir = os.listdir(r'E:\Internship\IIITD MIDAS\Task 3\Dataset')
root_dir = r'E:\Internship\IIITD MIDAS\Task 3\Dataset'
X = []
f=0
for cls in train_dir:
  src = root_dir +'\\'+ cls
  allFileNames = os.listdir(src)
  a = cls
  a = a[-2:]
  p = int(a)
  p
  # print(cls)
  # # print(src + '/'+ allFileNames)

  for img in allFileNames:
    # ytrain.append(p)
    IMG_LOC=src+'\\'+img
    # image = io.imread(IMG_LOC)
    image=cv.imread(IMG_LOC, cv.IMREAD_GRAYSCALE)
    image=cv.resize(image,(28,28))
    print(f)
    f=f+1
    # testing_data.append([np.array(img),img_num])
    X.append([np.array(image),np.array(p)])
  # print(X)
  # print(ytrain)
    # print(image)
```

```
31613
31614
31615
31616
31617
31618
31619
31620
31621
31622
31623
31624
31625
31626
31627
31628
31629
31630
31631
31632
```

Reshaping the array X and dividing it in Xtrain and Ytrain. We need xtrain and y train to train our model

In [76]:
```
x_train = np.array([i[0] for i in X]).reshape(-1,28,28,1)
y_train = np.array([i[1] for i in X])
```

Printing the shape of Xtrain and Ytrain, to verify if its same as what we need in our model

In [77]:
```
print(x_train.shape)
print(y_train.shape)
```

```
(60000, 28, 28, 1)
(60000,)
```

Loading MNIST dataset for Xtest and Ytest

In [78]:
```
#Loading
(mnist_x_train, mnist_y_train), (x_test, y_test) = mnist.load_data()
```

In [79]:
```
x_test = x_test.reshape(-1,28,28,1)
```

In [80]:
```
print(x_test.shape)
print(y_test.shape)
print(x_train.shape)
print(y_train.shape)
```

```
(10000, 28, 28, 1)
(10000,)
(60000, 28, 28, 1)
(60000,)
```

Data Normalization

As the CNN algorithm converge faster on the [0..1] data than on [0..255]

In [81]:
```
x_train, x_test=x_train/255.0, x_test/255.0
y_train, y_test=y_train.flatten(), y_test.flatten()
```

In [82]:
```
y_train
```

Out[82]:
```
array([0, 0, 0, ..., 9, 9, 9])
```

Label encoding

We need to give the y test and train in one hot encoded form

used this article for Label encoding - https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

it basically converts all the labels from 0 to the end. so that it becomes easier in one hot encoding.

In [83]:
```

# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
y_train= label_encoder.fit_transform(y_train)
y_train
# y_train.unique()
```

Out[83]:
```
array([0, 0, 0, ..., 9, 9, 9], dtype=int64)
```

Used the to_categorical function to convert the data to one hot encoded form

In [84]:
```
from numpy import array
from numpy import argmax
from keras.utils import to_categorical
# define example
data = array(y_train)
print(data)
# one hot encode
encoded = to_categorical(data)
print(encoded)
y_train = encoded
```

```
[0 0 0 ... 9 9 9]
[[1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]]
```

In [85]:
```
np.unique(y_train)
```

Out[85]:
```
array([0., 1.], dtype=float32)
```

Same for Y test

In [86]:
```
label_encoder = preprocessing.LabelEncoder()
y_test= label_encoder.fit_transform(y_test)
```

In [87]:
```
y_test
```

Out[87]:
```
array([7, 2, 1, ..., 4, 5, 6], dtype=int64)
```

In [88]:
```
data = array(y_test)
print(data)
# one hot encode
encoded = to_categorical(data)
print(encoded)
y_test = encoded
```

```
[7 2 1 ... 4 5 6]
[[0. 0. 0. ... 1. 0. 0.]
 [0. 0. 1. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

In [89]:
```
print(x_test.shape)
print(y_test.shape)
print(x_train.shape)
print(y_train.shape)
```

```
(10000, 28, 28, 1)
(10000, 10)
(60000, 28, 28, 1)
```

(60000, 10)

SAVING CHECKPOINTS

We need to save the checkpoints of our model, in case any error occurs in between our training

i refered this article to save checkpoints - https://towardsdatascience.com/checkpointing-deep-learning-models-in-keras-a652570b8de6

In [90]:
```
checkpoint_path = "E:\Internship\IIITD MIDAS\Task 3checkpoints\cp-{epoch:04d}.ckpt"
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    save_weights_only=True,
    save_best_only=True,
    verbose=1)
```

Model

Importing required libraries

In [91]:
```
from sklearn.metrics import confusion_matrix
import itertools

from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
```

Here we have created a function to create a CNN model by adding the following Layers

I have added 2 convolution layers, with filer size as 32. Padding as Same, and activation function as relu

one Max Pooling layer with pool size 2,2 and 1 Droupout to prevent overfitting

I have repeated the above archetecture with a filter size of 64, as with more layers, the model might perform better

Finally i have added a flatten layer to flatten the input, so that i can feed it to the dense layer.

I have then added the dense layer with relu as activation function.

At last i have added another dense layer with softmax as activation function because my final output has 60 classes.

After this i have added the required optimizers and i have finally compiled the model

In [92]:
```
def create_model():
    model = Sequential()

    model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                     activation ='relu', input_shape = (28,28,1)))
    model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                     activation ='relu'))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(Dropout(0.25))


    model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                     activation ='relu'))
    model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                     activation ='relu'))
    model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
    model.add(Dropout(0.25))


    model.add(Flatten())
    model.add(Dense(256, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(10, activation = "softmax"))

    optimizer = RMSprop(lr=0.001,
                        rho=0.9,
                        epsilon=1e-08,
                        decay=0.0)

    model.compile(optimizer = optimizer ,
                  loss = "categorical_crossentropy",
                  metrics=["accuracy"])
    return model
```

Creating a Model

In [93]:
```
model_ckpt= create_model()
```

Setting the number of epochs and batch size

In [94]:
```
epochs = 10 # Turn epochs to 30 to get 0.9967 accuracy
batch_size = 86
```

Data Augmentation

In data augmentation, we add a few image to the training dataset, with a little variation such as rotation or whitening and fliping of images. We do so, so that our model do not over fit.

Getting over fit means our model will get so much used to our training dataset, that its prediction on new images will be very faulty

In [95]:
```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

```
datagen.fit(x_train)
```

We are finally running the model we have built, by giving all the attributes to the model.fit_generator function

In [96]:
```
history = model_ckpt.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                              epochs = epochs,
                              validation_data = (x_test,y_test),
                              verbose = 2,
                              steps_per_epoch=x_train.shape[0] // batch_size
                              ,callbacks=[cp_callback])
```

```
Epoch 6/10

Epoch 00006: val_loss did not improve from 2.85897
697/697 - 199s - loss: 2.2636 - accuracy: 0.1106 - val_loss: 3.8315 - val_accuracy: 3.0000e-04
Epoch 7/10

Epoch 00007: val_loss did not improve from 2.85897
697/697 - 203s - loss: 2.2633 - accuracy: 0.1115 - val_loss: 3.6516 - val_accuracy: 4.0000e-04
Epoch 8/10

Epoch 00008: val_loss did not improve from 2.85897
697/697 - 208s - loss: 2.2635 - accuracy: 0.1125 - val_loss: 3.3721 - val_accuracy: 5.0000e-04
Epoch 9/10

Epoch 00009: val_loss did not improve from 2.85897
697/697 - 208s - loss: 2.2640 - accuracy: 0.1124 - val_loss: 3.4454 - val_accuracy: 8.0000e-04
Epoch 10/10

Epoch 00010: val_loss did not improve from 2.85897
697/697 - 210s - loss: 2.2654 - accuracy: 0.1130 - val_loss: 3.8605 - val_accuracy: 4.0000e-04
```

Calculating the final accuracy and loss

In [97]:
```
loss,acc = model_ckpt.evaluate(x_test,  y_test, verbose=2)
```

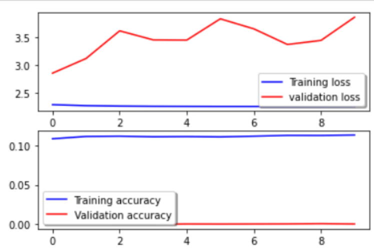
```
313/313 - 6s - loss: 3.8605 - accuracy: 4.0000e-04
```

Evaluation

In [98]:
```
model = model_ckpt
```

Plotting the loss and accuracy curves for training and validation

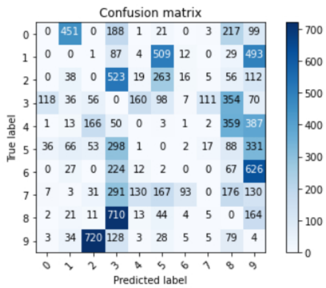
In [99]:
```
# Plot the loss and accuracy curves for training and validation
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training loss")
ax[0].plot(history.history['val_loss'], color='r', label="validation loss",axes =ax[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")
ax[1].plot(history.history['val_accuracy'], color='r',label="Validation accuracy")
legend = ax[1].legend(loc='best', shadow=True)
```



Plotting the Confusion matrix

In [100]:
```
# Look at confusion matrix

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Predict the values from the validation dataset
Y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
```

```
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))
```



loading pretrained weights 2.2 from our previous custom dataset

In [64]:
```
checkpoint_path = "E:\Internship\IIITD MIDAS\Task 2.2\checkpoints\cp-{epoch:04d}.ckpt"
```

In [65]:
```
checkpoint_dir = os.path.dirname(checkpoint_path)
latest = tf.train.latest_checkpoint(checkpoint_dir)

model_ckpt_1 = create_model()
model_ckpt_1.load_weights(latest)
```

Out[65]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x238e415bdf0>
```

In [66]:
```
loss,acc = model_ckpt_1.evaluate(x_test,  y_test, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```

```
313/313 - 5s - loss: 18.9440 - accuracy: 0.1059
Restored model, accuracy: 10.59%
```

Train our Pre trained weights on the faulty Dataset

In [67]:
```
# # model_ckpt2.fit(train_images,
#           train_labels,
#           batch_size=64,
#           epochs=10,
#           validation_data=(test_images,test_labels),
#           callbacks=[cp_callback])


history = model_ckpt_1.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                              epochs = 10,
                              validation_data = (x_test,y_test),
                              verbose = 2,
                              steps_per_epoch=x_train.shape[0] // batch_size
                              , callbacks=[cp_callback])
```

```
Epoch 1/10

Epoch 00001: val_loss improved from 3.00182 to 2.45490, saving model to E:\Internship\IIITD MIDAS\Task 3checkpoints\cp-0001.ckp
t
697/697 - 202s - loss: 2.3515 - accuracy: 0.1026 - val_loss: 2.4549 - val_accuracy: 0.0513
Epoch 2/10

Epoch 00002: val_loss did not improve from 2.45490
697/697 - 190s - loss: 2.2944 - accuracy: 0.1087 - val_loss: 2.8594 - val_accuracy: 0.0046
Epoch 3/10

Epoch 00003: val_loss did not improve from 2.45490
697/697 - 207s - loss: 2.2877 - accuracy: 0.1111 - val_loss: 3.0567 - val_accuracy: 0.0024
Epoch 4/10

Epoch 00004: val_loss did not improve from 2.45490
697/697 - 200s - loss: 2.2810 - accuracy: 0.1113 - val_loss: 3.2721 - val_accuracy: 5.0000e-04
Epoch 5/10

Epoch 00005: val_loss did not improve from 2.45490
697/697 - 203s - loss: 2.2770 - accuracy: 0.1132 - val_loss: 3.4099 - val_accuracy: 3.0000e-04
Epoch 6/10

Epoch 00006: val_loss did not improve from 2.45490
697/697 - 217s - loss: 2.2749 - accuracy: 0.1122 - val_loss: 3.3292 - val_accuracy: 3.0000e-04
Epoch 7/10

Epoch 00007: val_loss did not improve from 2.45490
697/697 - 207s - loss: 2.2741 - accuracy: 0.1118 - val_loss: 3.1295 - val_accuracy: 0.0000e+00
Epoch 8/10

Epoch 00008: val_loss did not improve from 2.45490
697/697 - 237s - loss: 2.2729 - accuracy: 0.1113 - val_loss: 3.0721 - val_accuracy: 5.0000e-04
Epoch 9/10

Epoch 00009: val_loss did not improve from 2.45490
697/697 - 215s - loss: 2.2722 - accuracy: 0.1115 - val_loss: 3.3107 - val_accuracy: 5.0000e-04
Epoch 10/10

Epoch 00010: val_loss did not improve from 2.45490
697/697 - 307s - loss: 2.2721 - accuracy: 0.1109 - val_loss: 3.2401 - val_accuracy: 6.0000e-04
```

In [69]:
```
loss,acc = model_ckpt2.evaluate(x_test,  y_test, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```

```
313/313 - 12s - loss: 2.3043 - accuracy: 0.1029
Restored model, accuracy: 10.29%
```

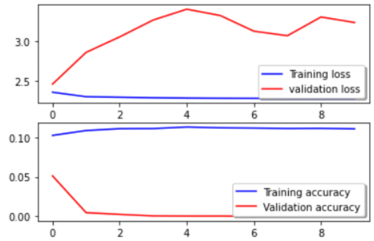
Evaluation

In [70]:
```
model = model_ckpt_1
```

Plotting the loss and accuracy curves for training and validation

Plotting the loss and accuracy curves for training and validation

In [71]:
```python
# Plot the loss and accuracy curves for training and validation
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training loss")
ax[0].plot(history.history['val_loss'], color='r', label="validation loss",axes =ax[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")
ax[1].plot(history.history['val_accuracy'], color='r',label="Validation accuracy")
legend = ax[1].legend(loc='best', shadow=True)
```



Plotting the Confusion matrix

In [72]:
```python
# Look at confusion matrix

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Predict the values from the validation dataset
Y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))
```