Machine Learning Laboratory

(410302)

BE Sem I Honors in AI/ML

Academic Year: 2021-22

Lab Assignment No.9

Name: Aboli Marathe

Roll Number: 41301

Branch: Department of Computer Engineering

Problem Statement:

Creating & Visualizing Neural Network for the given data.

Note:

1. Download dataset using Kaggle or you can use any other dataset.
2. Keras, ANN visualizer, graph viz libraries are required.
3. learn to preprocess your data, model, evaluate and optimize neural networks

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
%matplotlib inline

np.random.seed(2)

from sklearn.model_selection import train_test_split
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 tensorflow.keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
```

```
sns.set(style='white', context='notebook', palette='deep')
```

2. Data preparation

2.1 Load data

```
# Load the data
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

```
Y_train = train["label"]

# Drop 'label' column
X_train = train.drop(labels = ["label"],axis = 1)

# free some space
del train

g = sns.countplot(Y_train)

Y_train.value_counts()
```

Show hidden output

2.2 Check for null and missing values

```
# Check the data
X_train.isnull().any().describe()
```

Show hidden output

```
test.isnull().any().describe()
```

Show hidden output

2.3 Normalization

```
# Normalize the data
X_train = X_train / 255.0
test = test / 255.0
```

2.3 Reshape

```
# Reshape image in 3 dimensions (height = 28px, width = 28px , canal = 1)
X_train = X_train.values.reshape(-1,28,28,1)
test = test.values.reshape(-1,28,28,1)
```

2.5 Label encoding

```
# Encode labels to one hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
Y_train = to_categorical(Y_train, num_classes = 10)
```

Labels are 10 digits numbers from 0 to 9. We need to encode these lables to one hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0]).

2.6 Split training and valdiation set

```
# Set the random seed
random_seed = 2
```

```
# Split the train and the validation set for the fitting
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.1, rando
```

```
# Some examples
g = plt.imshow(X_train[0][:,:,0])
```

Show hidden output

3. CNN

3.1 Define the model

```
# Set the CNN model
# my CNN architechture is In -> [[Conv2D->relu]*2 -> MaxPool2D -> Dropout]*2 -> Flatten ->

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"))
```

3.2 Set the optimizer and annealer

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

Show hidden output

```
# Compile the model
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accurac
```

```
# Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

```
epochs = 1 # Turn epochs to 30 to get 0.9967 accuracy
batch_size = 86
```

3.3 Data augmentation

```
# Without data augmentation i obtained an accuracy of 0.98114
#history = model.fit(X_train, Y_train, batch_size = batch_size, epochs = epochs,
#          validation_data = (X_val, Y_val), verbose = 2)
```

```
# With data augmentation to prevent overfitting (accuracy 0.99286)

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 wi
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total hei
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(X_train)


# Fit the model
history = model.fit_generator(datagen.flow(X_train,Y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (X_val,Y_val),
                              verbose = 2, steps_per_epoch=X_train.shape[0] // batch_size
                              , callbacks=[learning_rate_reduction])
```

Show hidden output

4. Evaluate the model

```
# 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_val)
# 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_val,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))
```

Show hidden output

```
# Display some error results

# Errors are difference between predicted labels and true labels
errors = (Y_pred_classes - Y_true != 0)

Y_pred_classes_errors = Y_pred_classes[errors]
Y_pred_errors = Y_pred[errors]
Y_true_errors = Y_true[errors]
X_val_errors = X_val[errors]

def display_errors(errors_index,img_errors,pred_errors, obs_errors):
    """ This function shows 6 images with their predicted and real labels"""
    n = 0
    nrows = 2
    ncols = 3
    fig, ax = plt.subplots(nrows,ncols,sharex=True,sharey=True)
    for row in range(nrows):
        for col in range(ncols):
            error = errors_index[n]
            ax[row,col].imshow((img_errors[error]).reshape((28,28)))
            ax[row,col].set_title("Predicted label :{}\nTrue label :{}".format(pred_errors
            n += 1

# Probabilities of the wrong predicted numbers
Y_pred_errors_prob = np.max(Y_pred_errors,axis = 1)

# Predicted probabilities of the true values in the error set
true_prob_errors = np.diagonal(np.take(Y_pred_errors, Y_true_errors, axis=1))

# Difference between the probability of the predicted label and the true label
delta_pred_true_errors = Y_pred_errors_prob - true_prob_errors

# Sorted list of the delta prob errors
sorted_dela_errors = np.argsort(delta_pred_true_errors)

# Top 6 errors
most_important_errors = sorted_dela_errors[-6:]

# Show the top 6 errors
display_errors(most_important_errors, X_val_errors, Y_pred_classes_errors, Y_true_errors)
```

Show hidden output

```
# predict results
```

```
results = model.predict(test)

# select the indix with the maximum probability
results = np.argmax(results,axis = 1)

results = pd.Series(results,name="Label")
```

Visualization of Neural Network

```
pip install ann_visualizer
```

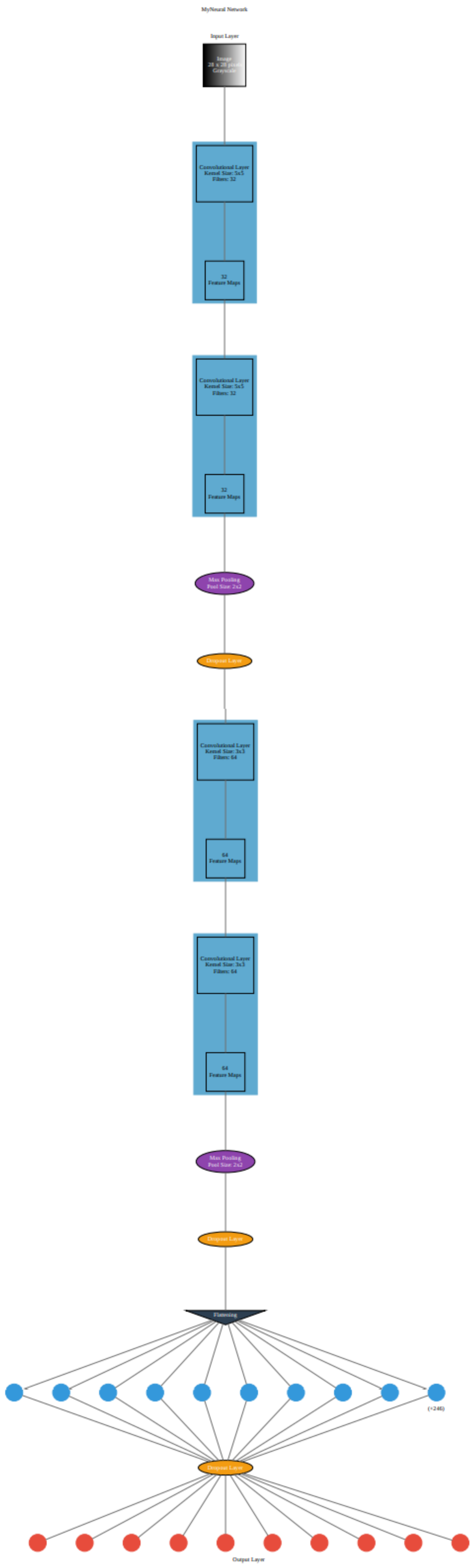
```
Collecting ann_visualizer
  Downloading ann_visualizer-2.5.tar.gz (4.7 kB)
Building wheels for collected packages: ann-visualizer
  Building wheel for ann-visualizer (setup.py) ... done
  Created wheel for ann-visualizer: filename=ann_visualizer-2.5-py3-none-any.whl size
  Stored in directory: /root/.cache/pip/wheels/1b/fc/58/2ab1c3b30350105929308becddda4
Successfully built ann-visualizer
Installing collected packages: ann-visualizer
Successfully installed ann-visualizer-2.5
```

```
!sudo apt-get install graphviz && pip3 install graphviz
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
graphviz is already the newest version (2.40.1-2).
The following package was automatically installed and is no longer required:
  libnvidia-common-460
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 37 not upgraded.
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (0
```

```
from ann_visualizer.visualize import ann_viz;
#Build your model here
ann_viz(model, view=True, filename="network.gv", title="MyNeural Network")
```

```
from IPython.display import Image
Image(filename='test.png')
```


MyNeural Network
Input Layer
Image
28 x 28 pixels
Grayscale
Convolutional Layer
Kernel Size: 5x5
Filters: 32
32
Feature Maps
Convolutional Layer
Kernel Size: 5x5
Filters: 32
32
Feature Maps
Max Pooling
Pool Size: 2x2
Dropout Layer
Convolutional Layer
Kernel Size: 3x3
Filters: 64
64
Feature Maps
Convolutional Layer
Kernel Size: 3x3
Filters: 64
64
Feature Maps
Max Pooling
Pool Size: 2x2
Dropout Layer
Flattening
(+246)
Dropout Layer
Output Layer

Conclusion

Thus in this assignment, we have learnt how to preprocess the digits data, create a neural network and recognize the digits. Then I used RMSprop optimizer which is an gradient based optimization technique to optimize the neural network. Finally I visualized the network using ANN Visualizer library.