Fractal-3 Assignment
MACHINE LEARNING

Problem 3: Chart Image Classification using CNN

Mayur Mehta
M22AI577
29 April 2023

https://github.com/M22AI577/ML_Assigmnet_FRACTAL3/blob/3d15f1ea808c2b5b6fde20f8c01ed92eb78eff75/Problem3_Chart_Image_Classification_using_CNN.ipynb

Chart Image Classification using CNN.

Task 1:

Images are given and corresponding label are give in CSV file : train_val.csv . In CSV file “**type**” column contains label value which slightly different from give lane value “Line”,“Dot Line”,“Horizontal Bar”,“Vertical Bar”, and “Pie”. Each Image file name is number.This number is corresponding index in CSV file using this we have to map Images and label name.

First, label names which are different require to map with given name we made mapping keys of “type” value along with array index of required label value array.

```
test_numberLabels = []


df = pd.read_csv("train_val.csv")
df.head()

# define some classes from the images we have observed

image_type_class = ["Line", "Dot Line", "Horizontal Bar", "Vertical Bar", 'Pie']

# map the categories to the labels array i.e y_train
image_type = {'line': 0, 'dot_line': 1, 'hbar_categorical': 2, 'vbar_categorical': 3, 'pie': 4}
```

First we map CSV in data frame using panda’s library so that while reading image we can directly map label name.

```python
def load_data(sourcePath = '',labelArray = [],imageArray = []) :

  # Loop over each folder from '0' to '9'

    number_folder = os.path.join(sourcePath)
    # Loop over each image in the folder
    for file in os.listdir(number_folder):
      file_path = os.path.join(number_folder, file)
      if file_path.endswith(('.png')):
        # read the image i graay scale and resize it to the imageSize size
        img = cv2.imread(file_path)
        img = cv2.resize(img, imageSize)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        # Append the image and label to the lists
        imageArray.append(img)
        name_type = df.at[int(os.path.splitext(file)[0]),'type']
        # print(os.path.splitext(file)[0]+ " " + name)
        # print(image_type_class[image_type[name_type]] + "   --->" + name_type)
        labelArray.append(image_type_class[image_type[name_type]])



# Loading Traiing dataset
df.head()
load_data(train_path,train_numberLabels,train_Images)

# Convert the lists to NumPy arrays

train_Images = np.array(train_Images)
# print(train_numberLabels)
train_numberLabels = np.array(train_numberLabels)

le = LabelEncoder()
train_numberLabels = le.fit_transform(train_numberLabels)

# Save the arrays in NumPy format
np.save('image_train.npy', train_Images)
np.save('label_train.npy', train_numberLabels)


image_train = np.load('image_train.npy')
label_train = np.load('label_train.npy')


# Loading Testing dataset
load_data(val_path,test_numberLabels,test_Images)


# Convert the lists to NumPy arrays
test_Images = np.array(test_Images)
```

load_data function is defined to load data, which take path as argument to read images.Whie reading images it used name of file to map label from dataframe . Name of file is digit so directly we are converting into int to use as index in data frame. From data frame using index from column name “type” we are getting label values which we again convert into give label values using or defined dictionary.

We are loading data into train_Images and train_numberLabels.Sameway we are loading test images.Then using sklearn mode selection we split data into 8:2 for training and testing data.

```python
from sklearn.model_selection import train_test_split
image_train, image_test, label_train, label_test = train_test_split(image_train, image_test, test_size=0.2, random_state=44)

model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

Task 2:

```
model = keras.Sequential([
tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (128, 128, 3)),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense (128, activation=tf.nn.relu),
tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])

from sklearn.model_selection import train_test_split
image_train, image_test, label_train, label_test = train_test_split(image_train, image_test, test_size=0.2, random_state=44)

model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])


history1 = model.fit(image_train, label_train,epochs= 10, validation_data=(image_test, label_test))
plot_accuracy_loss(history1)
```

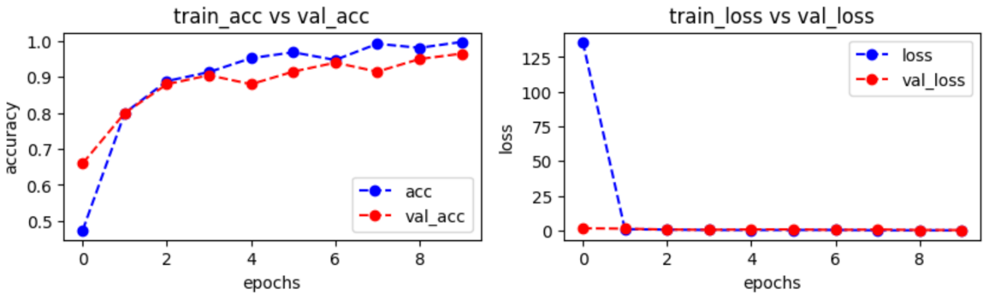
This is a neural network model defined using the Keras API in TensorFlow. The model consists of two convolutional layers with 32 filters each, each followed by a max pooling layer of size 2x2. The resulting feature maps are flattened and fed into a fully connected layer with 128 neurons and ReLU activation function. The final layer has 6 neurons and a softmax activation function, which is suitable for multi-class classification tasks. Overall, this model architecture can be used for image classification tasks on input images of size 128x128 with 3 color channels.

```
def plot_accuracy_loss(history):

  # Plot the accuracy and the loss during the training of the nn.
  fig = plt.figure(figsize=(10,5))
  # Plot accuracy
  plt.subplot(221)

  plt.plot(history.history['accuracy'],'bo--',label= "acc")

  plt.plot(history.history['val_accuracy'],'ro--',label= "val_acc")

  plt.title("train_acc vs val_acc")
  plt.ylabel("accuracy")
  plt.xlabel("epochs")
  plt.legend()


  # PLot Loss functton
  plt.subplot(222)
  plt.plot(history.history['loss'],'bo--',label= "loss")

  plt.plot(history.history['val_loss'],'ro--',label= "val_loss")

  plt.title("train_loss vs val_loss")
  plt.ylabel("loss")
  plt.xlabel("epochs")
  plt.legend()
  plt.show()
```

We have also defined plot_accuracy _loss , which shows the training and validation accuracy on one subplot and training and validation loss on another subplot. The blue line represents training metrics and the red line represents validation metrics. The function is a useful tool for visualizing the performance of a model during training.

So for given train data ad test data output is:

```
Epoch 1/10
25/25 [==============================] - 19s 729ms/step - loss: 135.8731 - accuracy: 0.4737 - val_loss: 1.4912 - val_accuracy: 0.6600
Epoch 2/10
25/25 [==============================] - 18s 727ms/step - loss: 0.8653 - accuracy: 0.8000 - val_loss: 1.2104 - val_accuracy: 0.8000
Epoch 3/10
25/25 [==============================] - 19s 763ms/step - loss: 0.5216 - accuracy: 0.8888 - val_loss: 0.5756 - val_accuracy: 0.8800
Epoch 4/10
25/25 [==============================] - 18s 709ms/step - loss: 0.2638 - accuracy: 0.9137 - val_loss: 0.3653 - val_accuracy: 0.9050
Epoch 5/10
25/25 [==============================] - 20s 808ms/step - loss: 0.2139 - accuracy: 0.9525 - val_loss: 0.6251 - val_accuracy: 0.8800
Epoch 6/10
25/25 [==============================] - 18s 718ms/step - loss: 0.2497 - accuracy: 0.9688 - val_loss: 0.6408 - val_accuracy: 0.9150
Epoch 7/10
25/25 [==============================] - 19s 769ms/step - loss: 0.2971 - accuracy: 0.9475 - val_loss: 0.3142 - val_accuracy: 0.9400
Epoch 8/10
25/25 [==============================] - 18s 710ms/step - loss: 0.0555 - accuracy: 0.9925 - val_loss: 0.5626 - val_accuracy: 0.9150
Epoch 9/10
25/25 [==============================] - 18s 711ms/step - loss: 0.0630 - accuracy: 0.9812 - val_loss: 0.2313 - val_accuracy: 0.9500
Epoch 10/10
25/25 [==============================] - 21s 834ms/step - loss: 0.0238 - accuracy: 0.9975 - val_loss: 0.1896 - val_accuracy: 0.9650
```



We can see accuracy around 0.96. From the plot, it can be observed that the training accuracy increases with each epoch, while the validation accuracy reaches a plateau after a few epochs. Similarly, the training loss decreases with each epoch while the validation loss plateaus after a few epochs. This indicates that the model is overfitting to the training data after a few epochs, and some regularization techniques such as dropout or early stopping may be necessary to improve generalization to new data.

Task 3: Finetune a pretrained network (e.g., AlexNet) for this task and report the results.

For finetuned we have used VGG16 which is a popular pre-trained convolutional neural network (CNN) model that was trained on the large scale ImageNet dataset.

```
1] from tensorflow.keras.applications.vgg16 import VGG16
   from tensorflow.keras.models import Sequential
   from tensorflow.keras.layers import Dense, Flatten, Dropout
   from tensorflow.keras.optimizers import Adam

   # Load the pre-trained VGG16 model
   vgg = VGG16(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

   # Freeze the layers of the pre-trained model
   for layer in vgg.layers:
       layer.trainable = False

   # Create a new model with VGG16 as the base
   model = Sequential()
   model.add(vgg)
   model.add(Flatten())
   model.add(Dense(256, activation='relu'))
   model.add(Dropout(0.5))
   model.add(Dense(10, activation='softmax'))

   from keras.utils import to_categorical

   label_train = to_categorical(label_train, num_classes=10)
   label_test = to_categorical(label_test, num_classes=10)

   # Compile the model
   model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

   # Train the model
   model.fit(image_train, label_train, epochs=10, batch_size=32, validation_data=(image_test, label_test))

   # Evaluate the model
   score = model.evaluate(image_test, label_test)
   print('Test loss:', score[0])
   print('Test accuracy:', score[1])
```

it creates a new model with VGG16 as the base, adds a Flatten layer, a Dense layer with 256 neurons, a Dropout layer with a rate of 0.5, and a final Dense layer with 10 neurons for classification. The labels are converted to categorical using the to_categorical function from Keras.

```
Epoch 1/10
25/25 [==============================] - 198s 8s/step - loss: 7.2222 - accuracy: 0.6513 - val_loss: 0.0270 - val_accuracy: 0.9850
Epoch 2/10
25/25 [==============================] - 199s 8s/step - loss: 0.2060 - accuracy: 0.9725 - val_loss: 0.1813 - val_accuracy: 0.9800
Epoch 3/10
25/25 [==============================] - 201s 8s/step - loss: 0.0613 - accuracy: 0.9925 - val_loss: 0.0988 - val_accuracy: 0.9900
Epoch 4/10
25/25 [==============================] - 197s 8s/step - loss: 0.0354 - accuracy: 0.9925 - val_loss: 0.0854 - val_accuracy: 0.9950
Epoch 5/10
25/25 [==============================] - 197s 8s/step - loss: 0.1407 - accuracy: 0.9850 - val_loss: 0.0793 - val_accuracy: 0.9950
Epoch 6/10
25/25 [==============================] - 192s 8s/step - loss: 0.0242 - accuracy: 0.9950 - val_loss: 0.1026 - val_accuracy: 0.9950
Epoch 7/10
25/25 [==============================] - 193s 8s/step - loss: 0.0047 - accuracy: 0.9987 - val_loss: 0.1018 - val_accuracy: 0.9950
Epoch 8/10
25/25 [==============================] - 188s 8s/step - loss: 0.0201 - accuracy: 0.9950 - val_loss: 0.0674 - val_accuracy: 0.9950
Epoch 9/10
25/25 [==============================] - 192s 8s/step - loss: 0.0233 - accuracy: 0.9962 - val_loss: 0.0709 - val_accuracy: 0.9950
Epoch 10/10
25/25 [==============================] - 191s 8s/step - loss: 0.0283 - accuracy: 0.9950 - val_loss: 0.0572 - val_accuracy: 0.9950
7/7 [==============================] - 37s 5s/step - loss: 0.0572 - accuracy: 0.9950
Test loss: 0.05722116306424141
Test accuracy: 0.9950000047683716
```

By seeing its output , we can conclude that VGG16 has hit accuracy. With CNN we have achieved accuracy near 0.96 but using VGG16 we have achieved accuracy around 0.99 near to 1.So VGG16 has slight edge over CNN.But we also have to note that time it took to give the output was very long. So it is computationally expensive.