Simple_image_classifier_using_cnn

January 30, 2021

[ ]:
```
#Isaac Mwendwa
#SCT211-0087/2016
#BSc. Commputer Science 4.2
#ICS 2407 Digital Image Processing
```

1 Objective

To create a CNN model and use the model to classify handwritten digits.

2 workflow

1. Load the dataset MNIST and examine the structure

- use any library to load the dataset (include both tensorflow and pytorch modules)
- take a look at data, inspecting its size,shape and quantity.
- view random samples using either openCV or MATPLOTLIB of the handwritten digits and observe the complexiy of the image

2. using Numpy to prepare the dataset for the training

- Ensure the format or shape of the data is appropriate for input into the model. (One-hot-encoding) https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/
- Ensure the data types are correct and data is normalized

3. Create a CNN with the following specifications

- Input dimensions 28 * 28 * 1
- Two Convolution layers (Kernel size 3*3) first with 64 filters, second 32. Use ReLU (Rectified Linear Unit) - activation layer
- Max Pooling size 2*2
- Dropout Layer 0.25
- Dense layer with 128 outputs
- Add another dropout layer with rate setting of 0.5
- Add final dropout layer that indicates the class probabilities.

4. Train the CNN on the MNIST dataset prepared in step 2

- Train for atleast 10 epochs using batch size of 32

5. Plot gragh showing how your training and validation loss and accuracy chached with respect to Epochs completed.
6. Save the model, will be used in part two.
7. Test the models on random samples on the test data.

3 deliverable

Jupyter Notebook that documents the workflow as we take the MNIST dataset, view samples, convert into right shape/format as required for the deep learning library

4 Loading the Handwritten Digit Dataset (MNIST)

""

[ ]:
```
from tensorflow.keras.datasets import mnist
```

load the dataset(divide into train and test data)

[ ]:
```
(x_train, y_train),(x_test, y_test) = mnist.load_data()
```

5 Display the number of samples in x_train, x_test, y_train, y_test

[ ]:
```
print("initial shape or dimensions of x_train",str(x_train.shape)+'\n')
print('Number of samples in training data: '+str(len(x_train)))
print('Number of labels in training data: '+ str(len(y_train)))
print('Number of samples in test data: '+str(len(x_test)))
print('Number of labels in test data: '+str(len(y_test))+ '\n')
print('Dimensions of x_train: '+str(x_train[0].shape))
print('Labels in x_train: '+str(y_train.shape)+'\n')
print('Dimensions of x_test: '+str(x_test[0].shape))
print('Labels in x_test: '+str(y_test.shape)+'\n')
```

```
initial shape or dimensions of x_train (60000, 28, 28)

Number of samples in training data: 60000
Number of labels in training data: 60000
Number of samples in test data: 10000
Number of labels in test data: 10000

Dimensions of x_train: (28, 28)
Labels in x_train: (60000,)

Dimensions of x_test: (28, 28)
Labels in x_test: (10000,)
```

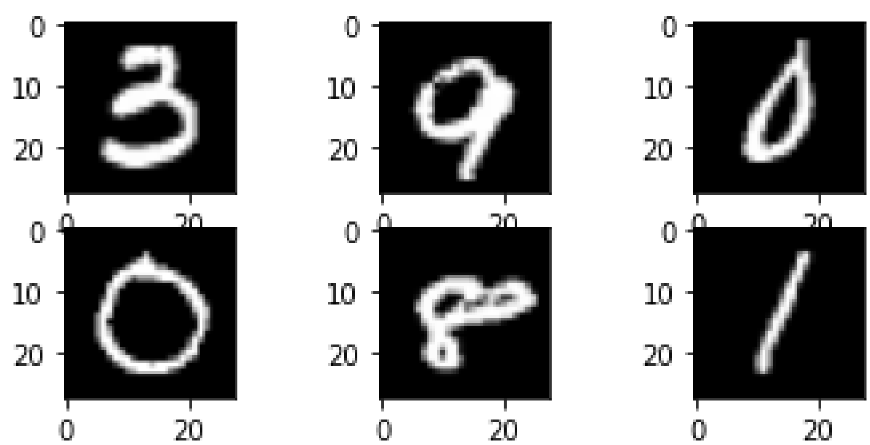
6 Take a look at the images in the Dataset

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
```

7 Plot 6 images in subplots

8 set the colormap to grey since our image data is in greyscale

[ ]:
```
plt.subplot(331)
random_num = np.random.randint(0, len(x_train))
_=plt.imshow(x_train[random_num],cmap=plt.get_cmap('gray'))
plt.subplot(332)
random_num = np.random.randint(0, len(x_train))
_=plt.imshow(x_train[random_num],cmap=plt.get_cmap('gray'))

plt.subplot(333)
random_num = np.random.randint(0, len(x_train))
_=plt.imshow(x_train[random_num],cmap=plt.get_cmap('gray'))

plt.subplot(334)
random_num = np.random.randint(0, len(x_train))
_=plt.imshow(x_train[random_num],cmap=plt.get_cmap('gray'))

plt.subplot(335)
random_num = np.random.randint(0, len(x_train))
_=plt.imshow(x_train[random_num],cmap=plt.get_cmap('gray'))

plt.subplot(336)
random_num = np.random.randint(0, len(x_train))
_=plt.imshow(x_train[random_num],cmap=plt.get_cmap('gray'))
```



9 Preparing Dataset for Keras

Keras Requires input data as a 4-d shape of (60000,28,28,1). When we initially loaded our data, x_train was (60000, 28,28). We need out label to be one-hot-encoded (). """

10 Store rows and columns

[ ]:
```
img_rows = x_train[0].shape[0]
img_cols = x_train[0].shape[1]
```

11 get data in right shape for keras.

12 add a forth dimensio to our data (60000,28,28) to (60000,28,28,1)

[ ]:
```
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)
```

13 Store shape of single image for future use as a variable storing our input shape

[ ]:
```
input_shape = (img_rows,img_cols,1)
```

14 Change image type to float

[ ]:
```
x_train=x_train.astype('float32')
x_test = x_test.astype('float32')
```

15 Normalize data by changing the range from 0-255 to 0-1

[ ]:
```
x_train /=255.0
x_test /=255.0
print('x_train shape: ',x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

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

16 Perform One-hot-encoding of data labels

```
[ ]: from tensorflow.keras.utils import to_categorical
```

17 one hot encode for output

```
[ ]: y_train = to_categorical(y_train)
     y_test = to_categorical(y_test)
```

18 count cols in our hot encoded matrix

```
[ ]: print('Number of classes: '+str(y_test.shape[1]))
     num_classes = y_test.shape[1]
```

```
Number of classes: 10
```

19 Create the CNN Model

```
[ ]: from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, Dropout, Flatten
     from tensorflow.keras.layers import Conv2D, MaxPooling2D
     from tensorflow.keras import backend as k
     from tensorflow.keras.optimizers import SGD
```

20 create the model

```
[ ]: model = Sequential()
```

21 First Convolutional Layer, Filter size 32 which reduces layer size to 262632

22 We use ReLU activation and our input shape 28281

```
[ ]: model.add(Conv2D(32, kernel_size=(3,3), activation='relu',␣
      ↪input_shape=input_shape))
```

23 Second Convolutional layer, Filter size of 64 which reduces our layer size to 242464

[ ]:
```
model.add(Conv2D(64, (3,3), activation='relu'))
```

24 Use maxpooling with kernel size of 22 *reducing size to 1212*64

[ ]:
```
model.add(MaxPooling2D(pool_size=(2,2)))
```

25 Dropout P setting as 0.25 ro reduce overfitting

[ ]:
```
model.add(Dropout(0.25))
#Flatten our Tensor object befor input into our dense layer
#A flatten op on a tensor reshapes the tensor to have the shape that is
#equal to the number of elements in the tensor
#Our CNN goes from 12*12*64 to 9*16*1
model.add(Flatten())
#We use another Dropout layer
model.add(Dropout(0.5))
#Create a fully connected/Dense layer with an output of each class (10)
model.add(Dense(num_classes, activation='softmax'))
#Compileour model, creates an object that stores the model. We set the␣
↪optimizer
#to use stochastic Gradient Descent (Learning rate of 0.01)
#We set loss function to be categorical_crossentropy as it's suitable for␣
↪multiclass
#problems. And finally the metrics ( to judge the performance of the model)
#We use accuracy
model.compile(loss='categorical_crossentropy', optimizer=SGD(0.
↪01),metrics=['accuracy'])
#The summary function can be used to display the model layers and parameters
print(model.summary())
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 12, 12, 64)        0
_________________________________________________________________
dropout_16 (Dropout)         (None, 12, 12, 64)        0
```

```
_________________________________________________________________
flatten_4 (Flatten)          (None, 9216)              0
_________________________________________________________________
dropout_17 (Dropout)         (None, 9216)              0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                92170
=================================================================
Total params: 110,986
Trainable params: 110,986
Non-trainable params: 0
_________________________________________________________________
None
```

26 Train the CNN

[ ]:
```
batch_size = 32
epochs = 10
#Store the results for plotting later
# in our fit function we specify our dataset (X_train, y_train)
#batch_size (typically 16 to 128 --Depending on RAM). NUmber of epochs (10 to␣
↪100)
#Validation dataset (X_test, y_test)
#Verbose = 1, setting training to output performance metric every epoch
history = model.fit(x_train, y_train, batch_size=batch_size,epochs=epochs,
                    verbose=1,
                    validation_data=(x_test,y_test))
#We obtain accuracy score using the evaluative fn
score=model.evaluate(x_test,y_test,verbose=0)
print('Test loss: ', score[0])
print('Test Accuracy: ', score[1])
```

```
Epoch 1/10
 317/1875 [====>...] - ETA: 2:02 - loss: 1.2664 -


      ␣
↪---------------------------------------------------------------------------

       KeyboardInterrupt                         Traceback (most recent call␣
↪last)

       <ipython-input-93-44368ac66c30> in <module>()
         8 history = model.fit(x_train, y_train,␣
↪batch_size=batch_size,epochs=epochs,
         9                     verbose=1,
   ---> 10                     validation_data=(x_test,y_test))
        11 #We obtain accuracy score using the evaluative fn
```

```
     12 score=model.evaluate(x_test,y_test,verbose=0)


    /usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/
↪training.py in _method_wrapper(self, *args, **kwargs)
        106   def _method_wrapper(self, *args, **kwargs):
        107     if not self._in_multi_worker_mode():  # pylint:␣
↪disable=protected-access
    --> 108       return method(self, *args, **kwargs)
        109
        110     # Running inside `run_distribute_coordinator` already.


    /usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/
↪training.py in fit(self, x, y, batch_size, epochs, verbose, callbacks,␣
↪validation_split, validation_data, shuffle, class_weight, sample_weight,␣
↪initial_epoch, steps_per_epoch, validation_steps, validation_batch_size,␣
↪validation_freq, max_queue_size, workers, use_multiprocessing)
       1096                 batch_size=batch_size):
       1097               callbacks.on_train_batch_begin(step)
    -> 1098               tmp_logs = train_function(iterator)
       1099               if data_handler.should_sync:
       1100                 context.async_wait()


    /usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/
↪def_function.py in __call__(self, *args, **kwds)
        778       else:
        779         compiler = "nonXla"
    --> 780         result = self._call(*args, **kwds)
        781
        782       new_tracing_count = self._get_tracing_count()


    /usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/
↪def_function.py in _call(self, *args, **kwds)
        805       # In this case we have created variables on the first call, so␣
↪we run the
        806       # defunned version which is guaranteed to never create␣
↪variables.
    --> 807       return self._stateless_fn(*args, **kwds)  # pylint:␣
↪disable=not-callable
        808     elif self._stateful_fn is not None:
        809       # Release the lock early so that multiple threads can perform␣
↪the call
```

```
    /usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/function.
↪py in __call__(self, *args, **kwargs)
   2827     with self._lock:
   2828       graph_function, args, kwargs = self.
↪_maybe_define_function(args, kwargs)
-> 2829     return graph_function._filtered_call(args, kwargs)  # pylint:␣
↪disable=protected-access
   2830
   2831   @property


    /usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/function.
↪py in _filtered_call(self, args, kwargs, cancellation_manager)
   1846                            resource_variable_ops.
↪BaseResourceVariable))],
   1847         captured_inputs=self.captured_inputs,
-> 1848         cancellation_manager=cancellation_manager)
   1849
   1850   def _call_flat(self, args, captured_inputs,␣
↪cancellation_manager=None):


    /usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/function.
↪py in _call_flat(self, args, captured_inputs, cancellation_manager)
   1922       # No tape is watching; skip to running the function.
   1923       return self._build_call_outputs(self._inference_function.call(
-> 1924           ctx, args, cancellation_manager=cancellation_manager))
   1925     forward_backward = self._select_forward_and_backward_functions(
   1926         args,


    /usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/function.
↪py in call(self, ctx, args, cancellation_manager)
    548               inputs=args,
    549               attrs=attrs,
--> 550               ctx=ctx)
    551         else:
    552           outputs = execute.execute_with_cancellation(


    /usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/execute.
↪py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,␣
↪op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
```

```
     62     if name is not None:

    KeyboardInterrupt:
```

27 plot loss charts

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
import matplotlib.pyplot as plt
#use the history object to get our saved performance results
history_dict = history.history

#extract the loss and the validation losses
loss_values=history_dict['loss']
val_loss_values=history_dict['val_loss']

#get the number of epochs and create an array up to that number using range()
epochs=range(1, len(loss_values) +1)

#Plot line charts for both validation and loss
line1 = plt.plot(epochs, val_loss_values, label='Validation/Test loss')
line2 = plt.plot(epochs, loss_values, label='Training loss')
plt.setp(line1, linewidth=2.0, marker='+', markersize=10.0)
plt.setp(line2, linewidth=2.0, marker='4', markersize=10.0)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()
plt.show()
```

28 Plot of Accuracy

[ ]:
```
#Plotting the accuracy chart
import matplotlib.pyplot as plt
#Use the history object to get our svaed performace results
from keras.callbacks import History
history_dict=history.history
```

29 extract the loss and the validation losses

[ ]:
```
acc_values=history_dict['accuracy']
val_acc_values=history_dict['val_accuracy']
#get the number of epochs and create an array up to that number using range()
epochs=range(1, len(acc_values) +1)
```

30 Plot line charts for both validation and loss

[ ]:
```
line1 = plt.plot(epochs, val_acc_values, label='Validation/Test Accuracy')
line2 = plt.plot(epochs, acc_values, label='Training')
plt.setp(line1, linewidth=2.0, marker='+', markersize=10.0)
plt.setp(line2, linewidth=2.0, marker='4', markersize=10.0)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()
plt.show()
```

31 saving the model

[ ]:
```
model.save('/content/drive/My Drive/Colab Notebooks/mnist_simple_cnn_10_Epochs.
↪h5')
print('model is saved')
import numpy as np
figure=plt.figure(figsize=(20,20))
for i in range(5):
  figure.add_subplot(1,5,i+1)
  random_idx=np.random.randint(0,len(x_test))
  plt.imshow(x_test[random_idx,:,:,0],cmap='gray')
  plt.axis('off')
  print(np.squeeze(np.argmax(model.predict(x_test[random_idx].
↪reshape(1,28,28,1)), axis=1),axis=0))
```

32 Part Two

[ ]:
```
#reload the data and model
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist

#load the dataset
(x_train,y_train),(x_test,y_test) = mnist.load_data()
```

```
model = load_model('/content/drive/My Drive/Colab Notebooks/
↪mnist_simple_cnn_10_Epochs.h5')
```

[ ]:
```
#get data into right shape for keras
#have a variable for number of rows and columns
img_rows = x_train[0].shape[0]
img_columns = x_train[0].shape[1]
x_test = x_test.reshape(x_test.shape[0], img_rows, img_columns, 1) #similar to␣
↪one-hot encoding

#store the shape of a single image
input_shape = (img_rows, img_columns, 1)

#change image type to float32
x_test = x_test.astype('float32')

#normalize the data by changing the range
x_test /= 255.0
y_test = to_categorical(y_test)
print(x_test.shape[0], 'test samples')
```

```
10000 test samples
```

[ ]:
```
#displaying the classification report
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

y_pred = model.predict_classes(x_test)

print(classification_report(np.argmax(y_test,axis=1),y_pred))
print(confusion_matrix(np.argmax(y_test,axis=1),y_pred))
```

```
WARNING:tensorflow:From <ipython-input-96-7005bb2820f5>:5:
Sequential.predict_classes (from tensorflow.python.keras.engine.sequential) is
deprecated and will be removed after 2021-01-01.
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.99      0.99      0.99      1010
           4       0.99      0.98      0.99       982
```

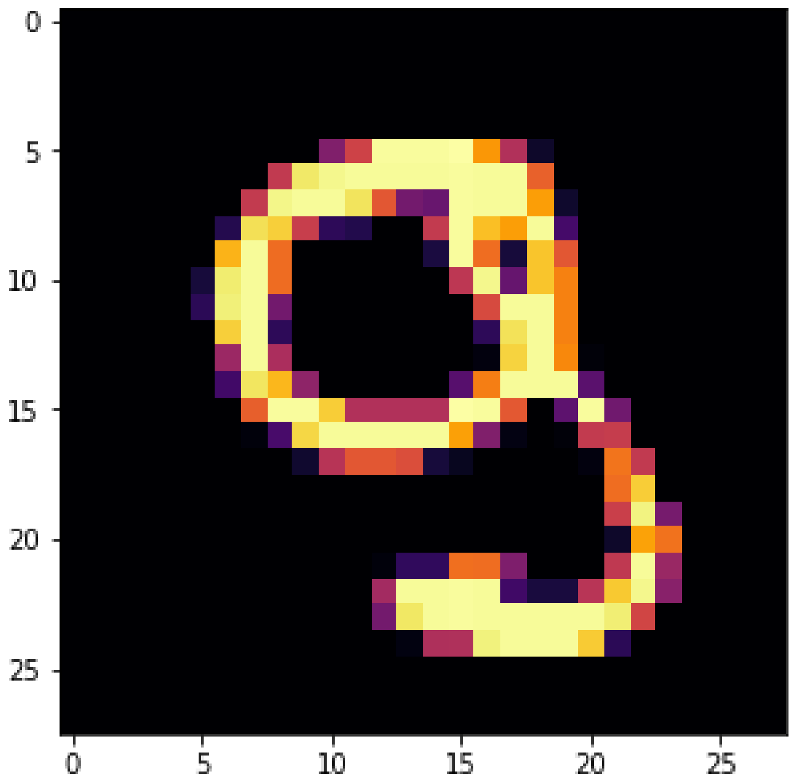
```
           5       0.99      0.99      0.99       892
           6       0.99      0.98      0.99       958
           7       0.97      0.98      0.98      1028
           8       0.98      0.98      0.98       974
           9       0.99      0.97      0.98      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000

[[ 974    0    1    0    0    1    1    1    2    0]
 [   0 1128    2    1    0    1    2    0    1    0]
 [   1    3 1009    4    1    0    1    8    5    0]
 [   0    0    2 1001    0    2    0    3    2    0]
 [   1    0    1    0  965    0    3    0    3    9]
 [   2    0    0    4    0  882    2    1    1    0]
 [   6    3    0    0    1    2  943    0    3    0]
 [   1    3    8    3    0    0    0 1009    3    1]
 [   5    0    2    1    1    2    2    5  953    3]
 [   5    5    0    2    3    2    0    9    3  980]]
```

[ ]:
```python
import cv2
import numpy as np
from tensorflow.keras.datasets import mnist

#load the data
(x_train,y_train),(x_test,y_test)=mnist.load_data()
#use numeric python to craete an array that store values of 1 when␣
↪misclassification ocurs
result=np.absolute(y_test,y_pred)
result_indices=np.nonzero(result>0)

#display the indices of the misclassification
print('indices misclassification data are:\n\n'+ str(result_indices))
```

```
indices misclassification data are:

(array([   0,    1,    2, ..., 9997, 9998, 9999]),)
```

[ ]:
```python
from tensorflow.keras.models import Model

#extract the output of the top 7 layers-layer outputs
layer_outputs = [layer.output for layer in model.layers[:7]]

#create a model to retun those outputs given the model's input
activation_model =Model(inputs=model.input,outputs=layer_outputs)
```

```python
#display the test image of the activation model
img_tensor = x_test[151].reshape(1,28,28,1)
figure1 = plt.figure(figsize=(5,5))
plt.imshow(img_tensor[0,:,:,0],cmap='inferno')
plt.axis='off'
```



[ ]:
```python
#run the model in predict mode to get the activation layers
#when an image is read, it returns the values of the activation

activations = activation_model.predict(img_tensor)
print("number of activation layers: "+ str(len(activations)))
```

```
number of activation layers: 7
```

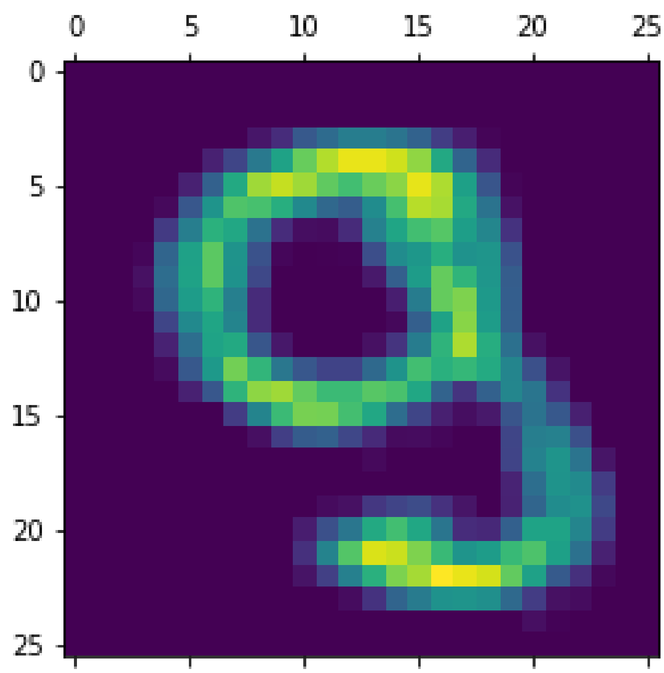
[ ]:
```python
#activation of the first conv layer for the image input
first_layer_activation = activations[0]
print(first_layer_activation.shape)
print(model.summary)
```

```
(1, 26, 26, 32)
<bound method Model.summary of
<tensorflow.python.keras.engine.sequential.Sequential object at 0x7fb3996a1da0>>
```

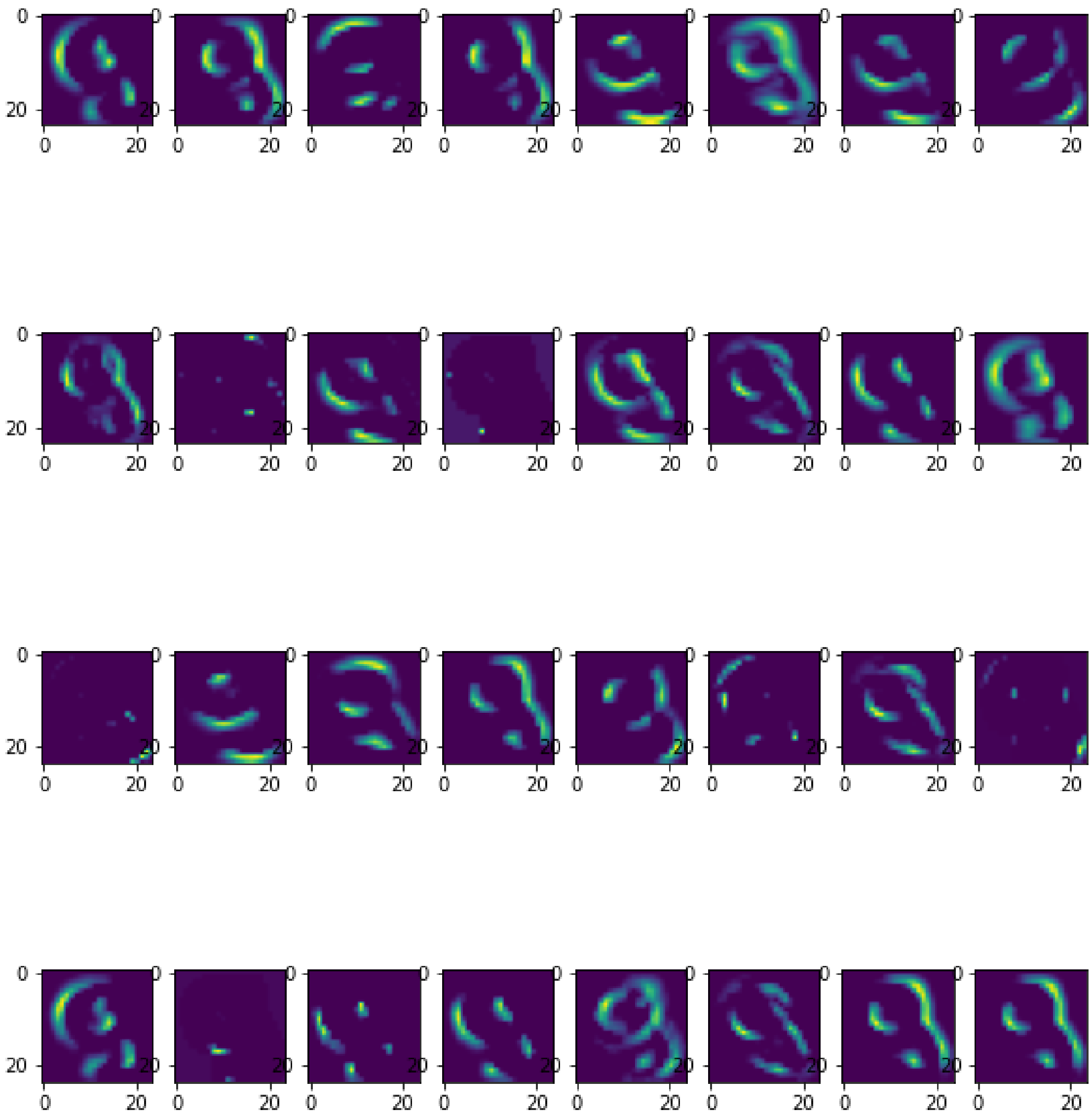
32.0.1 Plot output of 4th conv filter in the 1st conv layer

[ ]:
```
plt.matshow(first_layer_activation[0,:,:,3],cmap = 'viridis')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7fb38cda7978>
```



[ ]:
```
#create a function to display the actiavrtion of a specific layer the behaviour␣
↪of how the model makes a decision

def display_activations(activations,col_size,row_size,act_index):
  activation = activations[act_index]
  activation_index = 0
  fig,ax = plt.subplots(row_size, col_size, figsize=(row_size*2.5,col_size*1.5))
  for row in range(0,row_size):
    for col in range(0,col_size):
      ax[row][col].imshow(activation[0,:,:,activation_index],cmap = 'viridis')
      activation_index += 1


display_activations(activations,8,4,1)
```



32.1 Part Two: Assignment

[ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

[ ]:
```python
import os
import imageio
import numpy as np
from matplotlib import pyplot as plt

# load the model
from keras.models import load_model
```

```
model = load_model("/content/drive/My Drive/Colab Notebooks/
 ↪mnist_simple_cnn_10_Epochs.h5")

path = "/content/drive/My Drive/Colab Notebooks/Handwritten_Digits_Dataset/"



files = os.listdir(path)

im_files = []


for file in files:
  filename = os.path.basename(file)
  full_name = path + filename
  im_files.append(full_name)




#filenames.sort() # now you have the filenames and can do something with them
print("Number of images in dataset:  " + str(len(im_files)))

for im in im_files:
  im = imageio.imread(im)
  gray = np.dot(im[...,:3], [0.299, 0.587, 0.114])
  # reshape the image
  gray = gray.reshape(1, img_rows, img_cols, 1)

  # normalize image
  gray /= 255


  # predict digit
  prediction = model.predict(gray)
  print(prediction.argmax())
```

```
Number of images in dataset:  30
8
9
8
9
3
9
9
9
9
```

```
8
9
9
9
8
9
9
9
9
9
9
8
9
9
9
9
9
9
9
8
8
```

[ ]:

[ ]:

[ ]:

[ ]:

33 Part 3: Transfer of Learning

[10:10 AM] Lawrence B Nderu
Workflow

1. Load the CIFAR10 dataset (keras datasets) and train a Deeper CNN with various configulations.

2. Train this CNN for 10 Epochs or more using a Batch Size of 32 (batch size does not matter signigicantly as this deepend on the size of the RAM)

- Examine the performance metrics of the trained CNN. The accuracy after 10 Epochs should be between 60 and 65% on the test data. How can we perform better on this. 3.See what our CNN is capable of by testing the model on some of the Test images used in part 1. How would you compare this it classification performance to a human (assume)

4. The model created could be disappoiting, we then use the Transfer learning to significantly improve it. (Load the weights of a pre-trained CNN such as VGG16) - to import that model: from tensorflow.keras.applications import vgg16 as vgg.

5. Do not include the top layer when loading,, we are using this model to apply the concept of Transfer Learning - the function *vgg.VGG16(weights='imagenet', include_top=False, input_shape=(48,48,3))*

6. Extract the last layer from the third block of the VGG16 model. We wil be using the VGG model upto *block3_pool*

7. Add the classification layers for the CIFAR10 classed on top of it.

8. Freeze all layers in the pretrained VGG16 model since will be reusing them and compile merged model. Iterate through our base_model.layers and set the trainable parameter to be false by using layer.trainable=False

9. Keras data generator loading the image data.

10. Train the model for at least 5 (10 is the best) Epochs and note the improvement

11. Visualize the filters of the pre-trained VGG16 model - Reload the VGG16 model, Extract the conv layers since we want the filters and the biased values of these. Will inspect the bias and weights using third Conv layer using get_weights(). == a good idea of what filters and biases are.

12. Plot the first 6 conv filters- First normalixe the filter values (0-1) --get filters using f=filters[:,:,:,i]. Plot and visualixe it using plt.imshow(f[f:,:,:,j], cmap='gray')

13. Visualixe the features map of the VGG16 by running an input image (Create your own) through the model. -- redefine the model to output right after the first hidden layer using model=Model(input=model.input,outputs=model.layers[1].output. using the inbuilt keras preprocessing functions load_img and img_to_img along the numpy's expand_dims and keras VGG16 funtion from tensorflow.keras.applictions.vgg16 import preprocess_input

14. plot these features map for the output of the 5 Convolution Blocks indexed as [2,5,9,13,17]

34 Training a model for CIFAR10 Using Deeper CNN

[ ]:
```
from  __future__ import print_function
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout, Activation,Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
import os
```

[ ]:
```
batch_size = 32
num_classes = 10
epochs = 10
```

[ ]:
```
#load the CIFAR10
(x_train,y_train),(x_test,y_test)=cifar10.load_data()
```

[ ]:
```
#display image data shape/dim
print('x_train shape',x_train.shape)
print(x_train.shape[0],'train samples')
print(x_test.shape[0],'test samples')
```

```
x_train shape (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

[ ]:
```
#format the training data by normalizing and changing datya type
x_train=x_train.astype('float32')
x_test=x_test.astype('float32')
x_train /=255.0
x_test /=255.0
```

[ ]:
```
#hot encioding
y_train=to_categorical(y_train,num_classes)
y_test=to_categorical(y_test,num_classes)
```

[ ]:
```
model=Sequential()
#Padding = 'same' results in padding the input such that
#the output has the same length as the original input
model.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(64,(3,3),padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

#model build
model.compile(loss='categorical_crossentropy',optimizer=SGD(0.
 ↪01),metrics=['accuracy'])
```

```python
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
activation_1 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 13, 13, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1180160
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
_________________________________________________________________
None
```