cnn-cifar10-dataset-1

March 3, 2024

Small Image Classification Using Convolutional Neural Network (CNN)

In this notebook, we will classify small images cifar10 dataset from tensorflow keras datasets. There are total 10 classes as shown below. We will use CNN for classification

[1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

```
WARNING:tensorflow:From C:\Users\Pratik
Nagare\AppData\Roaming\Python\Python311\site-packages\keras\src\losses.py:2976:
The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Load the dataset

[4]:
```
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 267s 2us/step
```

[4]:
```
(50000, 32, 32, 3)
```

[5]:
```
X_test.shape
```

[5]:
```
(10000, 32, 32, 3)
```

Here we see there are 50000 training images and 10000 test images

[6]:
```
y_train.shape
```

[6]:
```
(50000, 1)
```

[ ]:
```
y_train[:5]
```

y_train is a 2D array, for our classification having 1D array is good enough. so we will convert this to now 1D array

[7]:
```
y_train = y_train.reshape(-1,)
y_train[:5]
```

[7]:
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

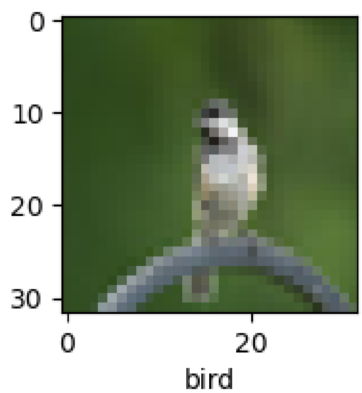
[8]:
```
y_test = y_test.reshape(-1,)
```

[9]:
```
classes =␣
 ↪["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

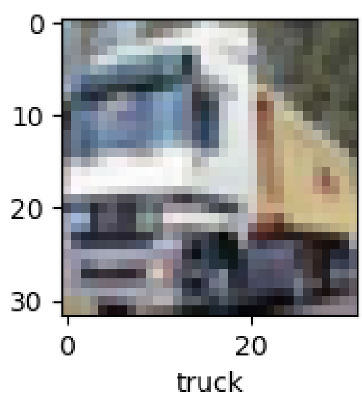
Let's plot some images to see what they are

[10]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

[12]:
```
plot_sample(X_train, y_train, 54)
```



[13]:
```
plot_sample(X_train, y_train, 1)
```



Normalize the images to a number from 0 to 1. Image has 3 channels (R,G,B) and each value in the channel can range from 0 to 255. Hence to normalize in 0–>1 range, we need to divide it by 255

Normalizing the training data

[ ]:
```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Build simple artificial neural network for image classification

[14]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])

ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
WARNING:tensorflow:From C:\Users\aaive\anaconda3\Lib\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\aaive\anaconda3\Lib\site-
packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/5
WARNING:tensorflow:From C:\Users\aaive\anaconda3\Lib\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\aaive\anaconda3\Lib\site-
packages\keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

 986/1563 [=================>…] - ETA: 43s - loss: nan - accuracy:
0.0994
```

```
---------------------------------------------------------------------------
```

```
KeyboardInterrupt                          Traceback (most recent call last)
Cell In[14], line 12
      1 ann = models.Sequential([
      2         layers.Flatten(input_shape=(32,32,3)),
      3         layers.Dense(3000, activation='relu'),
      4         layers.Dense(1000, activation='relu'),
      5         layers.Dense(10, activation='sigmoid')
      6     ])
      8 ann.compile(optimizer='SGD',
      9               loss='sparse_categorical_crossentropy',
     10               metrics=['accuracy'])
---> 12 ann.fit(X_train, y_train, epochs=5)

File ~\anaconda3\Lib\site-packages\keras\src\utils\traceback_utils.py:65, in␣
 ↪filter_traceback.<locals>.error_handler(*args, **kwargs)
     63 filtered_tb = None
     64 try:
---> 65     return fn(*args, **kwargs)
     66 except Exception as e:
     67     filtered_tb = _process_traceback_frames(e.__traceback__)

File ~\anaconda3\Lib\site-packages\keras\src\engine\training.py:1807, in Model.
 ↪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)
   1799 with tf.profiler.experimental.Trace(
   1800     "train",
   1801     epoch_num=epoch,
   (…)
   1804     _r=1,
   1805 ):
   1806     callbacks.on_train_batch_begin(step)
-> 1807     tmp_logs = self.train_function(iterator)
   1808     if data_handler.should_sync:
   1809         context.async_wait()

File ~\anaconda3\Lib\site-packages\tensorflow\python\util\traceback_utils.py:
 ↪150, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File␣
 ↪~\anaconda3\Lib\site-packages\tensorflow\python\eager\polymorphic_function\polymorphic_func
 ↪py:832, in Function.__call__(self, *args, **kwds)
    829 compiler = "xla" if self._jit_compile else "nonXla"
```

```
    831 with OptionalXlaContext(self._jit_compile):
--> 832   result = self._call(*args, **kwds)
    834 new_tracing_count = self.experimental_get_tracing_count()
    835 without_tracing = (tracing_count == new_tracing_count)

File␣
 ↪~\anaconda3\Lib\site-packages\tensorflow\python\eager\polymorphic_function\po ymorphic_func
 ↪py:868, in Function._call(self, *args, **kwds)
    865   self._lock.release()
    866   # In this case we have created variables on the first call, so we run␣
 ↪the
    867   # defunned version which is guaranteed to never create variables.
--> 868   return tracing_compilation.call_function(
    869       args, kwds, self._no_variable_creation_config
    870   )
    871 elif self._variable_creation_config is not None:
    872   # Release the lock early so that multiple threads can perform the cal
    873   # in parallel.
    874   self._lock.release()

File␣
 ↪~\anaconda3\Lib\site-packages\tensorflow\python\eager\polymorphic_function\tr cing_compilat
 ↪py:139, in call_function(args, kwargs, tracing_options)
    137 bound_args = function.function_type.bind(*args, **kwargs)
    138 flat_inputs = function.function_type.unpack_inputs(bound_args)
--> 139 return function._call_flat(  # pylint: disable=protected-access
    140     flat_inputs, captured_inputs=function.captured_inputs
    141 )

File␣
 ↪~\anaconda3\Lib\site-packages\tensorflow\python\eager\polymorphic_function\co crete_functio
 ↪py:1323, in ConcreteFunction._call_flat(self, tensor_inputs, captured_inputs)
   1319 possible_gradient_type = gradients_util.PossibleTapeGradientTypes(args)
   1320 if (possible_gradient_type == gradients_util.POSSIBLE_GRADIENT_TYPES_NO E
   1321     and executing_eagerly):
   1322   # No tape is watching; skip to running the function.
-> 1323   return self._inference_function.call_preflattened(args)
   1324 forward_backward = self._select_forward_and_backward_functions(
   1325     args,
   1326     possible_gradient_type,
   1327     executing_eagerly)
   1328 forward_function, args_with_tangents = forward_backward.forward()

File␣
 ↪~\anaconda3\Lib\site-packages\tensorflow\python\eager\polymorphic_function\at mic_function
 ↪py:216, in AtomicFunction.call_preflattened(self, args)
    214 def call_preflattened(self, args: Sequence[core.Tensor]) -> Any:
    215   """Calls with flattened tensor inputs and returns the structured␣
 ↪output."""
```

```
--> 216   flat_outputs = self.call_flat(*args)
    217   return self.function_type.pack_output(flat_outputs)

File␣
 ↪~\anaconda3\Lib\site-packages\tensorflow\python\eager\polymorphic_function\atomic_function
 ↪py:251, in AtomicFunction.call_flat(self, *args)
    249 with record.stop_recording():
    250   if self._bound_context.executing_eagerly():
--> 251     outputs = self._bound_context.call_function(
    252         self.name,
    253         list(args),
    254         len(self.function_type.flat_outputs),
    255     )
    256   else:
    257     outputs = make_call_op_in_graph(
    258         self,
    259         list(args),
    260         self._bound_context.function_call_options.as_attrs(),
    261     )

File ~\anaconda3\Lib\site-packages\tensorflow\python\eager\context.py:1486, in␣
 ↪Context.call_function(self, name, tensor_inputs, num_outputs)
   1484 cancellation_context = cancellation.context()
   1485 if cancellation_context is None:
-> 1486   outputs = execute.execute(
   1487       name.decode("utf-8"),
   1488       num_outputs=num_outputs,
   1489       inputs=tensor_inputs,
   1490       attrs=attrs,
   1491       ctx=self,
   1492   )
   1493 else:
   1494   outputs = execute.execute_with_cancellation(
   1495       name.decode("utf-8"),
   1496       num_outputs=num_outputs,
   (…)
   1500       cancellation_manager=cancellation_context,
   1501   )

File ~\anaconda3\Lib\site-packages\tensorflow\python\eager\execute.py:53, in␣
 ↪quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51 try:
     52   ctx.ensure_initialized()
---> 53   tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name
     54                                       inputs, attrs, num_outputs)
     55 except core._NotOkStatusException as e:
     56   if name is not None:
```

```
KeyboardInterrupt:
```

You can see that at the end of 5 epochs, accuracy is at around 48.48%

[ ]:
```python
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test,␣
 ↪y_pred_classes))
```

Now let us build a convolutional neural network to train our images

[ ]:
```python
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu',␣
 ↪input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

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

[ ]:
```python
cnn.fit(X_train, y_train, epochs=10)
```

With CNN, at the end 5 epochs, accuracy was at around 70.28% which is a significant improvement over ANN. CNN's are best for image classification and gives superb accuracy. Also computation is much less compared to simple ANN as maxpooling reduces the image dimensions while still preserving the features

[ ]:
```python
cnn.evaluate(X_test,y_test)
```

[ ]:
```python
y_pred = cnn.predict(X_test)
y_pred[:5]
```

[ ]:
```python
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

[ ]:
```python
y_test[:5]
```

[ ]:
```
plot_sample(X_test, y_test,3)
```

[ ]:
```
classes[y_classes[3]]
```

[ ]:
```
classes[y_classes[3]]
```

Exercise

Use CNN to do handwritten digits classification using MNIST dataset. You can use this notebook as a reference: https://github.com/codebasics/py/blob/master/DeepLearningML/1_digits_recognition/digits_reco

Above we used ANN for digits classification. You need to modify this code to use CNN instead. Check how accuracy improves fast with CNN and figure out how CNN can be a better choice for doing image classification compared to ANN. Once you have worked on this problem on your own, you can check my solution by clicking on this link: Solution