CIFAR10-Keras

February 6, 2021

0.0.1 CIFAR10 Classification Using Keras

[1]:
```
# modules required
import numpy as np
import pandas as pd

# visualisations
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('seaborn-white')
```

[2]:
```
# Dataset is preprocessed and can be accessed using keras directly
```

[3]:
```
from tensorflow.keras.datasets import cifar10
```

[4]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 3s 0us/step
```

0.0.2 Dimensionality Check

[5]:
```
x_train.shape
```

[5]:
```
(50000, 32, 32, 3)
```

[6]:
```
np.unique(y_train)
```

[6]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

[6]:
```

```

[7]:
```
train_total, h, w, channels = x_train.shape
class_names = ['plane', 'car', ' bird', 'cat', 'deer', 'dog', ' frog', 'horse',␣
↪'ship', 'truck']
num_classes = len(np.unique(y_train))
print(f'Total Training Samples : {train_total} ')
print(f'Image Dimensions HxW : {h}x{w}')
```

```python
print(f'Number of Color Channels : {channels}')
print(f'Labels - {class_names}')
print(f'Number of Classes {num_classes}') # in cifar10, 10 resembles the classes
```

```
Total Training Samples : 50000
Image Dimensions HxW : 32x32
Number of Color Channels : 3
Labels - ['plane', 'car', ' bird', 'cat', 'deer', 'dog', ' frog', 'horse',
'ship', 'truck']
Number of Classes 10
```

0.0.3 Sample Images

[8]:
```python
'''
This 5/5 Image Grid contain randomly selected samples of data

'''
def make_grid(x,y):
    plt.figure(figsize=(15,10)) # specifying the overall grid size

    for i in range(25):
        plt.subplot(5,5,i+1)    # the number of images in the grid is 5*5 (25)
        plt.imshow(x[i])
        plt.title(f'label -  {class_names[y[i][0]]} ,target - {y[i][0]}')

    plt.tight_layout()
    plt.show()

make_grid(x_train,y_train)
```

0.0.4 Data Normalisation and Label Encoding

[9]:
```
x_train = x_train/255.
x_test = x_test/255.
```

[10]:
```
# image-grid after normalisation
make_grid(x_train,y_train)
```

[11]:
```
# One hot encoding Y for softmax
```

[12]:
```
from tensorflow.keras.utils import to_categorical
```

[13]:
```
y_cat_train = to_categorical(y_train,10)
y_cat_test = to_categorical(y_test,10)
```

[14]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Input, Add, Dense,Dropout, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D, MaxPool2D
```

[15]:
```
# Network Architecture
model = Sequential()

model.add(Conv2D(filters=32,kernel_size=(3,3), padding='same',input_shape=x_train.shape[1:],activation='relu'))
model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.4))
```

```
model.add(Conv2D(filters=64, kernel_size=(3,3),␣
 ↪padding='same',activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3,3),activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(filters=64, kernel_size=(3,3),␣
 ↪padding='same',activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3,3),activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(512,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

[16]:
```
opt = Adam(learning_rate=0.001)
model.compile(loss='categorical_crossentropy',␣
 ↪optimizer=opt,metrics=['accuracy'])
```

[17]:
```
from tensorflow.keras.callbacks import EarlyStopping
```

[18]:
```
import time
start = time.perf_counter()
model.
 ↪fit(x_train,y_cat_train,epochs=50,batch_size=256,validation_data=(x_test,y_cat_test))
elapsed = time.perf_counter() - start
```

```
Epoch 1/50
196/196 [==============================] - 12s 21ms/step - loss: 2.0824 -
accuracy: 0.2001 - val_loss: 1.6407 - val_accuracy: 0.3938
Epoch 2/50
196/196 [==============================] - 3s 18ms/step - loss: 1.5978 -
accuracy: 0.4061 - val_loss: 1.3422 - val_accuracy: 0.5075
Epoch 3/50
196/196 [==============================] - 3s 18ms/step - loss: 1.3854 -
accuracy: 0.4922 - val_loss: 1.1918 - val_accuracy: 0.5633
Epoch 4/50
196/196 [==============================] - 3s 18ms/step - loss: 1.2404 -
accuracy: 0.5520 - val_loss: 1.0903 - val_accuracy: 0.6080
Epoch 5/50
196/196 [==============================] - 4s 18ms/step - loss: 1.1579 -
accuracy: 0.5869 - val_loss: 1.0174 - val_accuracy: 0.6370
Epoch 6/50
196/196 [==============================] - 4s 18ms/step - loss: 1.0857 -
accuracy: 0.6120 - val_loss: 0.9614 - val_accuracy: 0.6536
```

```
Epoch 7/50
196/196 [==============================] - 3s 18ms/step - loss: 1.0386 -
accuracy: 0.6295 - val_loss: 0.9113 - val_accuracy: 0.6779
Epoch 8/50
196/196 [==============================] - 4s 18ms/step - loss: 0.9963 -
accuracy: 0.6439 - val_loss: 0.8808 - val_accuracy: 0.6937
Epoch 9/50
196/196 [==============================] - 4s 18ms/step - loss: 0.9408 -
accuracy: 0.6690 - val_loss: 0.8384 - val_accuracy: 0.7074
Epoch 10/50
196/196 [==============================] - 4s 18ms/step - loss: 0.9029 -
accuracy: 0.6809 - val_loss: 0.7841 - val_accuracy: 0.7235
Epoch 11/50
196/196 [==============================] - 4s 18ms/step - loss: 0.8645 -
accuracy: 0.6966 - val_loss: 0.8716 - val_accuracy: 0.6987
Epoch 12/50
196/196 [==============================] - 4s 18ms/step - loss: 0.8362 -
accuracy: 0.7086 - val_loss: 0.7862 - val_accuracy: 0.7299
Epoch 13/50
196/196 [==============================] - 4s 18ms/step - loss: 0.8117 -
accuracy: 0.7140 - val_loss: 0.7135 - val_accuracy: 0.7523
Epoch 14/50
196/196 [==============================] - 4s 18ms/step - loss: 0.7890 -
accuracy: 0.7248 - val_loss: 0.7739 - val_accuracy: 0.7298
Epoch 15/50
196/196 [==============================] - 4s 18ms/step - loss: 0.7832 -
accuracy: 0.7273 - val_loss: 0.6993 - val_accuracy: 0.7630
Epoch 16/50
196/196 [==============================] - 4s 18ms/step - loss: 0.7604 -
accuracy: 0.7357 - val_loss: 0.7176 - val_accuracy: 0.7505
Epoch 17/50
196/196 [==============================] - 4s 18ms/step - loss: 0.7461 -
accuracy: 0.7391 - val_loss: 0.6881 - val_accuracy: 0.7596
Epoch 18/50
196/196 [==============================] - 4s 18ms/step - loss: 0.7296 -
accuracy: 0.7427 - val_loss: 0.6611 - val_accuracy: 0.7673
Epoch 19/50
196/196 [==============================] - 4s 18ms/step - loss: 0.7188 -
accuracy: 0.7499 - val_loss: 0.6385 - val_accuracy: 0.7806
Epoch 20/50
196/196 [==============================] - 4s 18ms/step - loss: 0.7026 -
accuracy: 0.7572 - val_loss: 0.6575 - val_accuracy: 0.7714
Epoch 21/50
196/196 [==============================] - 4s 18ms/step - loss: 0.7085 -
accuracy: 0.7530 - val_loss: 0.6483 - val_accuracy: 0.7730
Epoch 22/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6890 -
accuracy: 0.7597 - val_loss: 0.6580 - val_accuracy: 0.7689
```

```
Epoch 23/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6775 -
accuracy: 0.7639 - val_loss: 0.6167 - val_accuracy: 0.7864
Epoch 24/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6879 -
accuracy: 0.7622 - val_loss: 0.6099 - val_accuracy: 0.7879
Epoch 25/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6612 -
accuracy: 0.7704 - val_loss: 0.6363 - val_accuracy: 0.7820
Epoch 26/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6554 -
accuracy: 0.7696 - val_loss: 0.6385 - val_accuracy: 0.7805
Epoch 27/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6492 -
accuracy: 0.7757 - val_loss: 0.6292 - val_accuracy: 0.7829
Epoch 28/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6418 -
accuracy: 0.7773 - val_loss: 0.6199 - val_accuracy: 0.7856
Epoch 29/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6312 -
accuracy: 0.7794 - val_loss: 0.5898 - val_accuracy: 0.7987
Epoch 30/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6272 -
accuracy: 0.7801 - val_loss: 0.6275 - val_accuracy: 0.7845
Epoch 31/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6168 -
accuracy: 0.7830 - val_loss: 0.5900 - val_accuracy: 0.7960
Epoch 32/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6375 -
accuracy: 0.7804 - val_loss: 0.6067 - val_accuracy: 0.7891
Epoch 33/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6303 -
accuracy: 0.7815 - val_loss: 0.6481 - val_accuracy: 0.7787
Epoch 34/50
196/196 [==============================] - 4s 18ms/step - loss: 0.6245 -
accuracy: 0.7827 - val_loss: 0.5854 - val_accuracy: 0.8020
Epoch 35/50
196/196 [==============================] - 4s 19ms/step - loss: 0.6042 -
accuracy: 0.7880 - val_loss: 0.5672 - val_accuracy: 0.8067
Epoch 36/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5907 -
accuracy: 0.7921 - val_loss: 0.5772 - val_accuracy: 0.8025
Epoch 37/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5990 -
accuracy: 0.7917 - val_loss: 0.5758 - val_accuracy: 0.8066
Epoch 38/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5881 -
accuracy: 0.7966 - val_loss: 0.5807 - val_accuracy: 0.8033
```

```
Epoch 39/50
196/196 [==============================] - 4s 18ms/step - loss: 0.5916 -
accuracy: 0.7894 - val_loss: 0.5870 - val_accuracy: 0.8000
Epoch 40/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5939 -
accuracy: 0.7944 - val_loss: 0.5791 - val_accuracy: 0.8053
Epoch 41/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5914 -
accuracy: 0.7973 - val_loss: 0.5839 - val_accuracy: 0.7993
Epoch 42/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5813 -
accuracy: 0.7969 - val_loss: 0.5672 - val_accuracy: 0.8034
Epoch 43/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5714 -
accuracy: 0.8013 - val_loss: 0.5518 - val_accuracy: 0.8097
Epoch 44/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5673 -
accuracy: 0.8040 - val_loss: 0.5679 - val_accuracy: 0.8049
Epoch 45/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5662 -
accuracy: 0.7998 - val_loss: 0.5604 - val_accuracy: 0.8097
Epoch 46/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5645 -
accuracy: 0.8050 - val_loss: 0.5595 - val_accuracy: 0.8095
Epoch 47/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5609 -
accuracy: 0.8047 - val_loss: 0.5675 - val_accuracy: 0.8057
Epoch 48/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5646 -
accuracy: 0.8020 - val_loss: 0.5857 - val_accuracy: 0.7981
Epoch 49/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5618 -
accuracy: 0.8056 - val_loss: 0.5771 - val_accuracy: 0.8035
Epoch 50/50
196/196 [==============================] - 4s 19ms/step - loss: 0.5533 -
accuracy: 0.8068 - val_loss: 0.5487 - val_accuracy: 0.8095
```

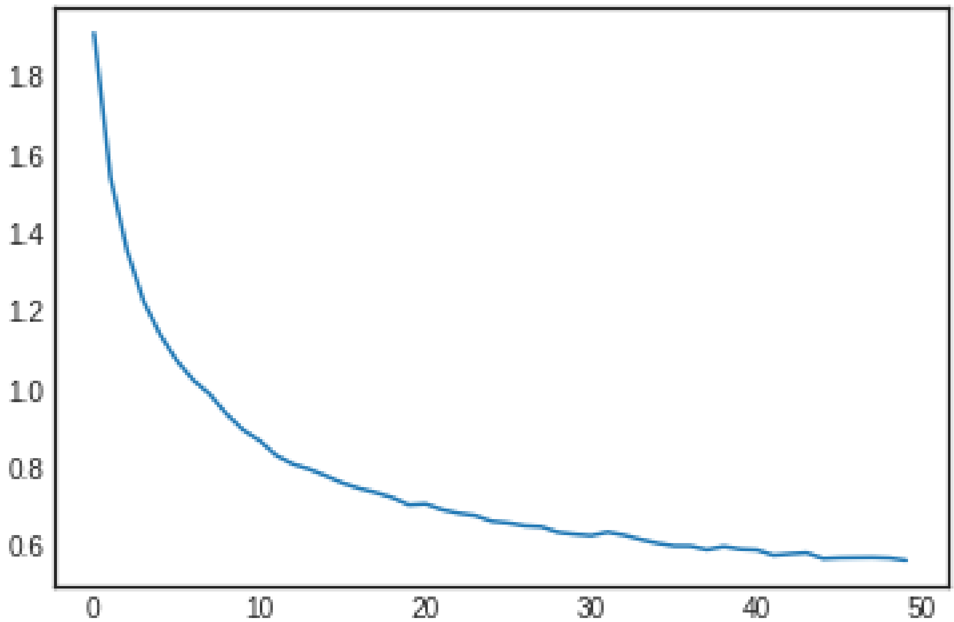
[19]:
```python
print(f'Time Taken : {elapsed/60:.2f}')
```

```
Time Taken : 3.14
```

[20]:
```python
history = pd.DataFrame(model.history.history)
```

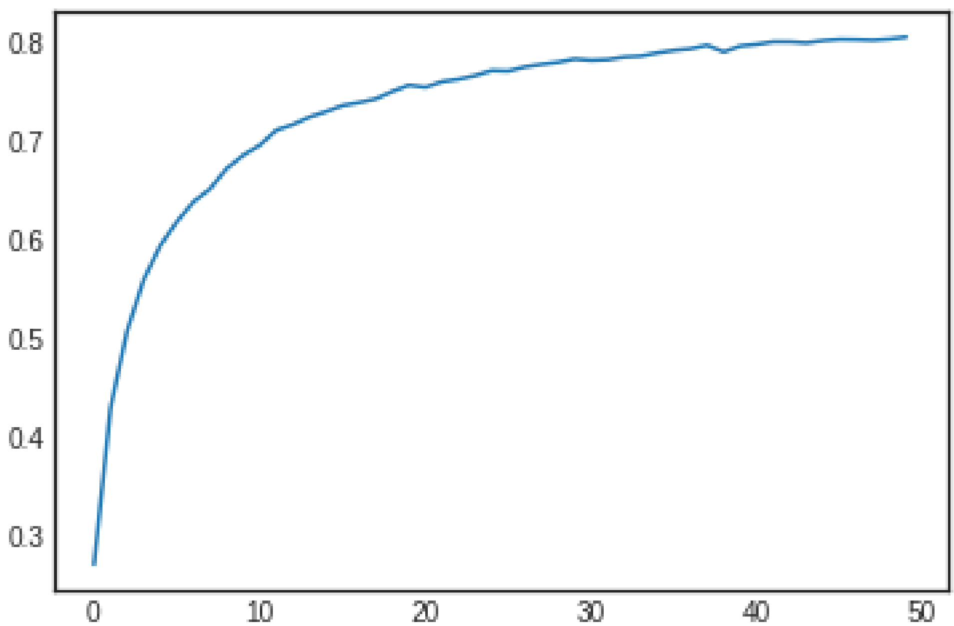
[21]:
```python
history.loss.plot()
```

[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9e9d276d68>
```



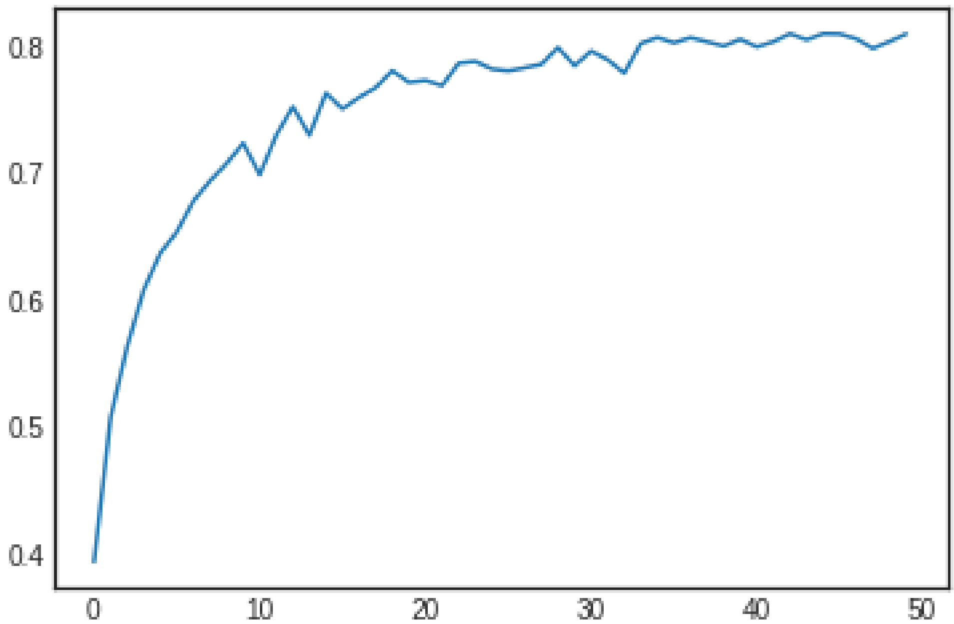
[22]:
```
history.accuracy.plot()
```

[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9e9d156a20>



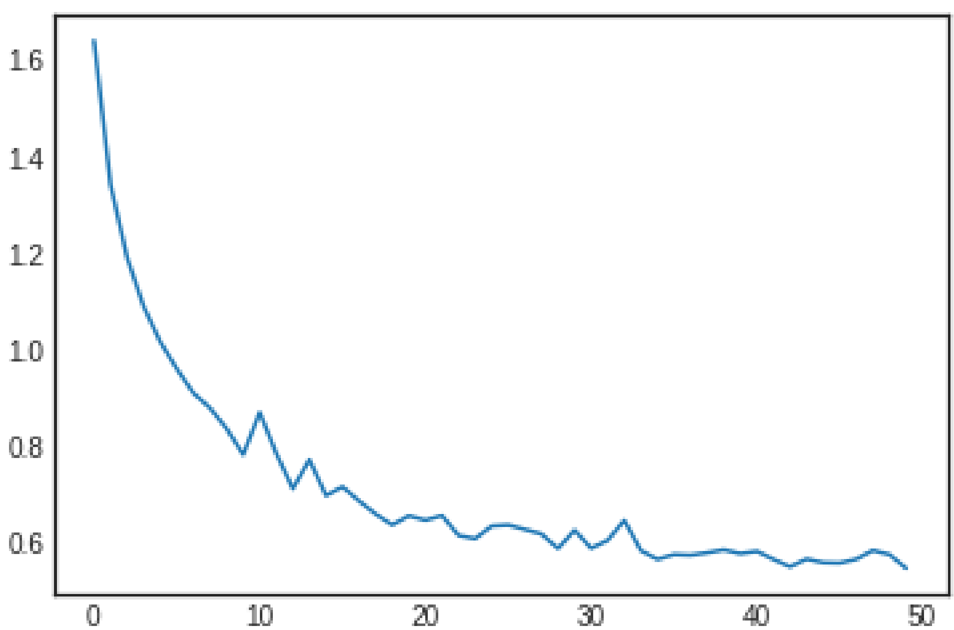
[23]:
```
history.val_accuracy.plot()
```

[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9e904b24e0>

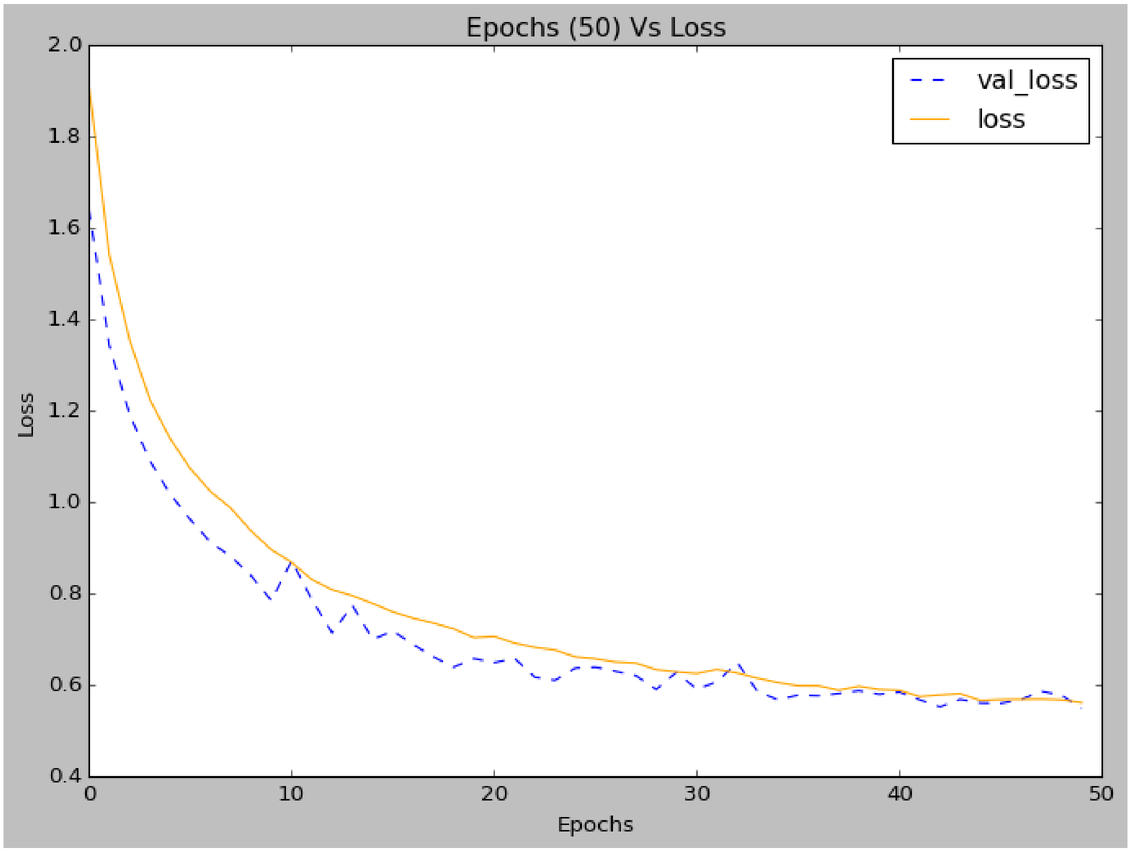


[24]:
```
history.val_loss.plot()
```

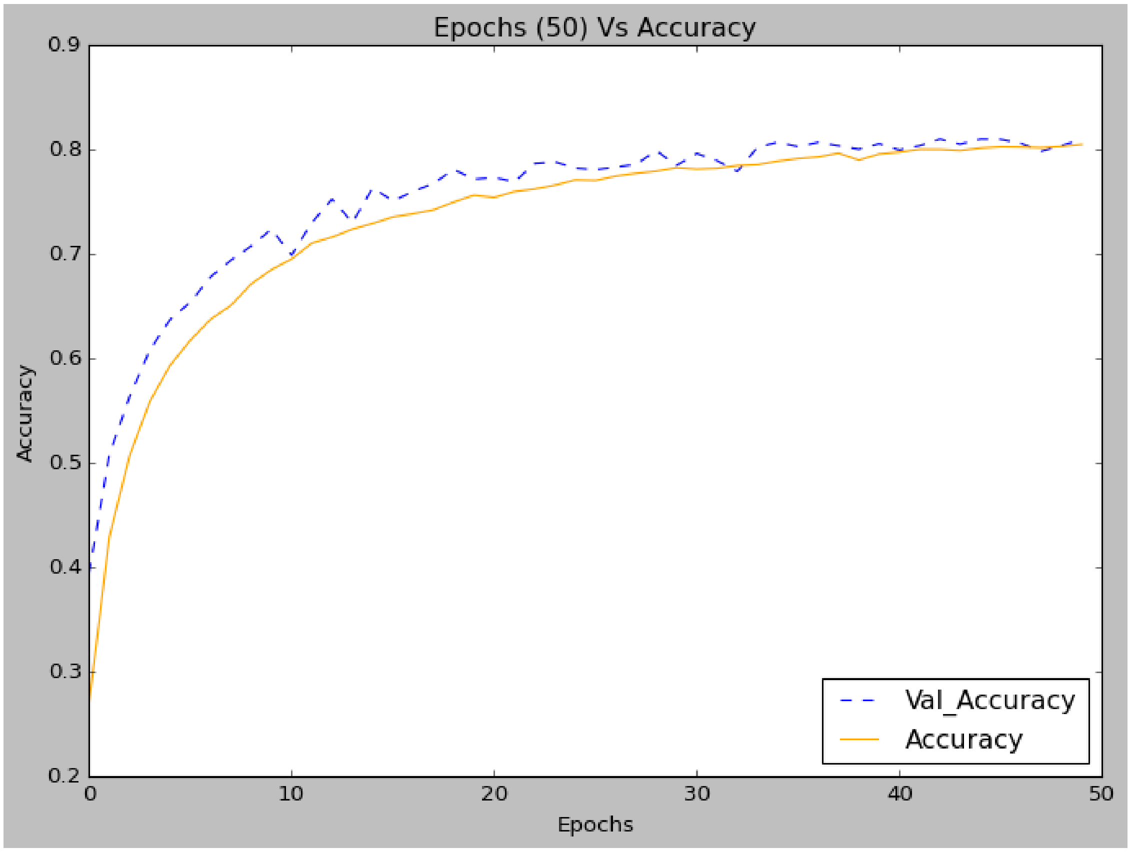
[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9e9d571898>



[25]:
```
plt.style.use('classic')
plt.figure(figsize=(10,7))
plt.plot(history['val_loss'], '--',label='val_loss')
plt.plot(history['loss'],color='orange',label='loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Epochs (50) Vs Loss')
plt.legend()
plt.show()
```



[26]:
```
# Accuracy
plt.style.use('classic')
plt.figure(figsize=(10,7))
plt.plot(history['val_accuracy'], '--',label='Val_Accuracy')
plt.plot(history['accuracy'],color='orange',label='Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Epochs (50) Vs Accuracy')
plt.legend(loc='lower right')
plt.show()
```



[27]:

```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896

_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 32)        9248

_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0

_________________________________________________________________
dropout (Dropout)            (None, 15, 15, 32)        0

_________________________________________________________________
conv2d_2 (Conv2D)            (None, 15, 15, 64)        18496

_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 64)        36928

_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0

_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0
```

```
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 6, 6, 64)          36928
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 512)               131584
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 276,138
Trainable params: 276,138
Non-trainable params: 0
_________________________________________________________________
```

[28]:
```
# Evaluation
predictions = model.predict_classes(x_test)
predictions
```

```
/usr/local/lib/python3.6/dist-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
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).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

[28]:
```
array([3, 8, 8, …, 5, 1, 7])
```

[29]:
```
counts = 0
for pred,y in zip(predictions,y_test):
  if pred == y:
    counts+=1
  else:
    pass
print('Precise Number of Elements Correctly predicted {}'.format(counts))
```

```
Precise Number of Elements Correctly predicted 8095
```

[30]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

[31]:
```
print(f'classes - {class_names}')
print()
print(classification_report(predictions,y_test))
```

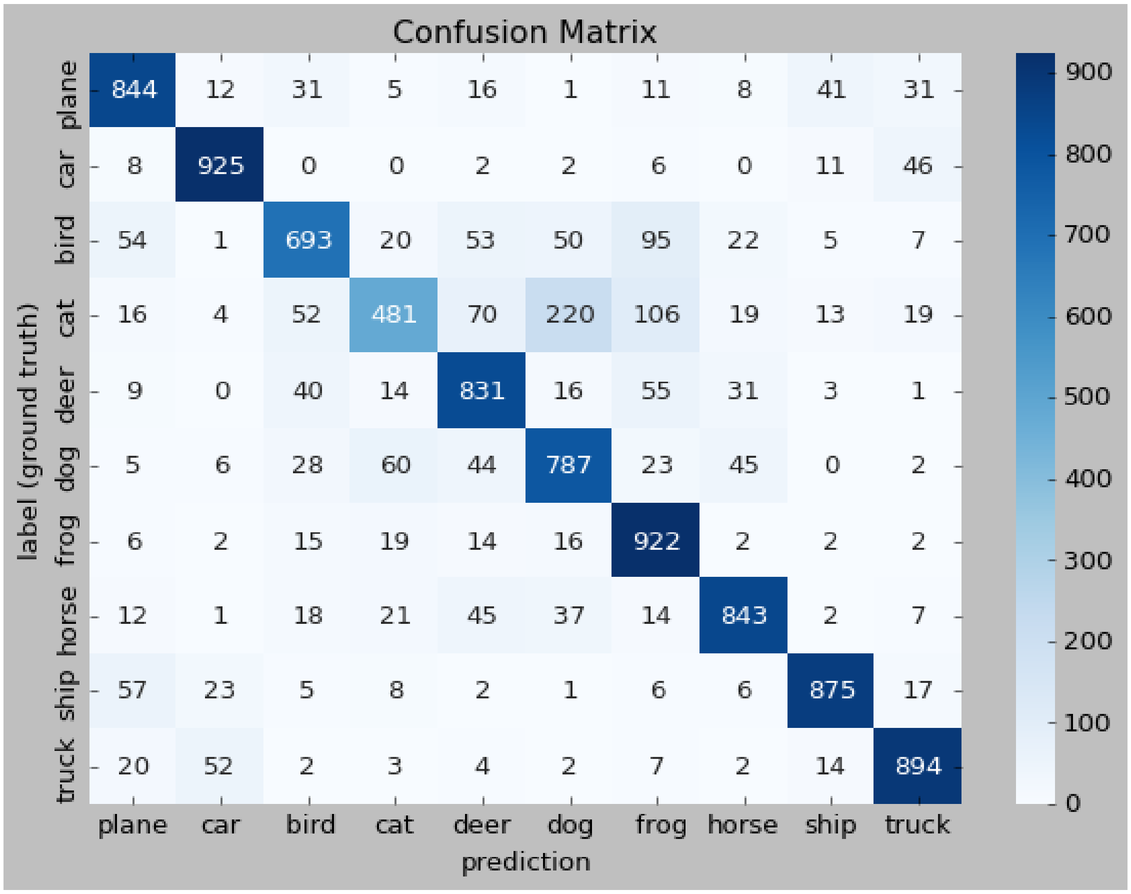
```
classes - ['plane', 'car', ' bird', 'cat', 'deer', 'dog', ' frog', 'horse',
'ship', 'truck']

              precision    recall  f1-score   support

           0       0.84      0.82      0.83      1031
           1       0.93      0.90      0.91      1026
           2       0.69      0.78      0.74       884
           3       0.48      0.76      0.59       631
           4       0.83      0.77      0.80      1081
           5       0.79      0.70      0.74      1132
           6       0.92      0.74      0.82      1245
           7       0.84      0.86      0.85       978
           8       0.88      0.91      0.89       966
           9       0.89      0.87      0.88      1026

    accuracy                           0.81     10000
   macro avg       0.81      0.81      0.81     10000
weighted avg       0.83      0.81      0.81     10000
```

[32]:
```
arr = confusion_matrix(y_test, predictions)
df_cm = pd.DataFrame(arr, class_names, class_names)
plt.figure(figsize = (9,6))
sns.heatmap(df_cm, annot=True, fmt="d", cmap='Blues')
plt.xlabel("prediction")
plt.ylabel("label (ground truth)")
plt.title('Confusion Matrix')
plt.show();
```



[33]:
```
# save model
model.save('model.h5')
```

0.1 Miscellaneous

[34]:
```
import os
path = './samples/'
overview_path = './samples/overview.txt'
eval_path  = './samples/evaluate.txt'

if os.path.exists(path):

    print('samples dir, exists..checking for dictionaries existence..')

    if os.path.exists(overview_path) and os.path.exists(eval_path):
        print('Data exists. no need of overwritting.')
    else:
        print("overview and eval doesn't exist, proceed to step-2")
```

```
else:
    print("samples/ dir is non-existent, Establishing one..")
    os.mkdir(path) # samples directory
```

```
samples/ dir is non-existent, Establishing one..
```

[35]:
```
x_train[[1110,8696,170]]
y_train[[1110,8696,170]]
```

[35]:
```
array([[2],
       [1],
       [8]], dtype=uint8)
```

[36]:
```
names = list()
names.append(class_names[2].strip())
names.append(class_names[1])
names.append(class_names[8])
names
```

[36]:
```
['bird', 'car', 'ship']
```

[37]:
```
x_train.shape
```

[37]:
```
(50000, 32, 32, 3)
```

[38]:
```
foreval = []
for x in y_test[0:50]:
  foreval.append(class_names[x[0]].strip())
foreval
```

[38]:
```
['cat',
 'ship',
 'ship',
 'plane',
 'frog',
 'frog',
 'car',
 'frog',
 'cat',
 'car',
 'plane',
 'truck',
 'dog',
 'horse',
 'truck',
 'ship',
 'dog',
```

```
 'horse',
 'ship',
 'frog',
 'horse',
 'plane',
 'deer',
 'truck',
 'dog',
 'bird',
 'deer',
 'plane',
 'truck',
 'frog',
 'frog',
 'dog',
 'deer',
 'dog',
 'truck',
 'bird',
 'deer',
 'car',
 'truck',
 'dog',
 'deer',
 'frog',
 'dog',
 'frog',
 'plane',
 'truck',
 'cat',
 'truck',
 'horse',
 'frog']
```

[39]:

```
import pickle
# dictionary init
overview_dict = {}
eval_dict = {}

# fill the following -
# for overview
#string
kind = 'Image Data'
#tuple
dimensions = x_train.shape
#labels : str(list of unique target values)
targets = class_names
```

```
#nd.array
data = x_train[[1110,8696,170]]
#nd.array
labels = names

vars0 = ['kind','dimensions', 'targets', 'data', 'labels']

# filling overview_dict
for x in vars0:
    try:
        overview_dict[x] = eval(x)
    except:
        overview_dict[x] = x

# evaluate_dict

eval_dict = {'test_cases' : x_test[0:50], 'true': y_cat_test[0:50],␣
↪'class_names':class_names,'model':'/model.h5'}


# dump 1
with open(overview_path,'wb') as f:
    pickle.dump(overview_dict,f)

# dump 2
with open(eval_path,'wb') as f:
    pickle.dump(eval_dict,f)
```

[40]:

```
```

[40]:

```
```

[ ]:

```
# dictionary init
overview_dict = {}
eval_dict = {}

# fill the following -
# for overview
#string
kind = 'Image Data'
#tuple
dimensions = x_train.shape
#labels : str(list of unique target values)
targets = list(np.unique(y_test))
#nd.array
data = x_train[0:3]
#nd.array
```

```python
labels = class_names

vars0 = ['kind','dimensions', 'targets', 'data', 'labels']

# filling overview_dict
for x in vars0:
    try:
        overview_dict[x] = eval(x)
    except:
        overview_dict[x] = x
```

[ ]:

```python
s = '''
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 6, 6, 64)          36928
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 512)               131584
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
```

```
=================================================================
Total params: 276,138
Trainable params: 276,138
Non-trainable params: 0
_________________________________________________________________

'''
```

[ ]:
```
# desc-----string
# project_name-----string
# framework-----string
# prediction_type-----string
# network_type-----string
# architecture-----model()
# layers-----int
# hidden_units-----int
# activations-----string(list)
# epochs-----int
# metrics-----string(list)
# loss-----string
# optimiser-----string
# learning_rate-----float
# batch_size-----int/string
# train_performance-----float
# test_performance-----float
# classification_report-----string
# elapsed-----float
# summary-----string
# ipynb-----path
# plots-----path
```

[44]:
```

```

```
anscombe.json                  mnist_test.csv
california_housing_test.csv    mnist_train_small.csv
california_housing_train.csv   README.md
```

[ ]:
```
report = '''
classes - ['plane', 'car', ' bird', 'cat', 'deer', 'dog', ' frog', 'horse',␣
↪'ship', 'truck']

              precision    recall  f1-score   support

           0       0.80      0.85      0.83       945
           1       0.89      0.93      0.91       962
           2       0.62      0.85      0.72       736
           3       0.66      0.65      0.66      1012
```

```
           4       0.76      0.78      0.77       972
           5       0.72      0.77      0.74       936
           6       0.94      0.65      0.77      1456
           7       0.79      0.90      0.84       878
           8       0.93      0.85      0.89      1091
           9       0.90      0.89      0.89      1012

    accuracy                           0.80     10000
   macro avg       0.80      0.81      0.80     10000
weighted avg       0.81      0.80      0.80     10000

'''
```

[ ]:
```
# desc = '''The CIFAR-10 dataset consists of 60000 32x32 colour images in 10␣
↪classes, with 6000 images per class. There are 50000 training images and␣
↪10000 test images. The classes include various cars, ships, deers, dogs and␣
↪cats, trucks etc.'''
# project_name = 'CIFAR-10'
# framework = 'Keras'
# prediction_type = 'Multi-Class Classification of 10 Classes'
# network_type = 'Convolutional Neural Network'
# architecture = s
# layers = 12
# hidden_units = 'None'
# activations = ['relu','softmax']
# epochs = 50
# metrics = 'Accuracy'
# loss = 'Categorical Cross-Entropy'
# optimiser = 'Adam'
# learning_rate = 0.001
# batch_size = 256
# train_performance = '80.60%'
# test_performance = '80.15%'
# classification_report = report
# elapsed = '3.3 Mins'
# summary = '''This Dataset being so discrete with the images is difficult to␣
↪train to get the optimum accuracy. 80% accuracy on this Image dataset is␣
↪great for a novice learner. Images of a particular class vary so much that␣
↪it almost forces the network to learn all the changes again and one more␣
↪analogy is that the images are blurry to be able to identify proper␣
↪characteristics.'''
# ipynb = './Projects/CIFAR10/Keras/CIFAR10-Keras.pdf'
# plots = './Projects/CIFAR10/Keras/Plots'
```

[ ]:
```
# var = ['desc','project_name', 'framework','prediction_type','network_type',
#      'architecture','layers','hidden_units','activations','epochs',
```

```
#     ␣
↪'metrics','loss','optimiser','learning_rate','batch_size','train_performace','test_performa
#     ,'ipynb','plots']
# param = {}
# for val in var:

#     try:
#         param[val] = eval(val)

#     except:
#         param[val] = val
```

[ ]:
```
# param['train_performace'] = '80.60%'
```

[ ]:
```
# import pickle
# file = open("artefacts.txt", "wb")
# dictionary = param
# pickle.dump(dictionary, file)
# file.close()
```