ASSIGNMENT-4

APRIL 16, 2021

RISHABH SHARMA
20MAI0082

Github Link :- https://github.com/rishabh5197/Deep-Learning-Assignments/tree/main/Assignment-4

Task-1

In [1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
from sklearn.model_selection import train_test_split
```

In [2]:
```
(xtrain,ytrain),(xtest,ytest) = keras.datasets.cifar10.load_data()
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

In [3]:
```
plt.imshow(xtrain[0])
plt.title(classes[ytrain[0][0]],)
plt.axis(False)
plt.show()
```

In [4]:
```
#  selecting 50% less data from xtrain
(xtrain_new_50,xtest_new_50,ytrain_new_50,ytest_new_50) =train_test_split(xtrain,ytrain,test_size=0.5,random_state=
xtrain_1,xtest_1,ytrain_1,ytest_1 = train_test_split(xtrain_new_50,ytrain_new_50,test_size=0.3,random_state=0)
print("Xtrain orignal :- ",xtrain.shape)
print("Xtrain 50% selected from xtrain:-",xtrain_new_50.shape)
print("70% selected from that 50% training :- ",xtrain_1.shape)
```

```
Xtrain orignal :-  (50000, 32, 32, 3)
Xtrain 50% selected from xtrain:- (25000, 32, 32, 3)
70% selected from that 50% training :-  (17500, 32, 32, 3)
```

In [5]:
```
from keras.models import Sequential
from keras.layers import Conv2D,Activation,BatchNormalization,MaxPooling2D,Dense,Flatten,Dropout
```

In [6]:
```
from keras.utils import to_categorical

ytrain = to_categorical(ytrain,10)
ytrain_new_50 = to_categorical(ytrain_new_50,10)
ytrain_1 = to_categorical(ytrain_1,10)

ytest = to_categorical(ytest,10)
ytest_new_50 = to_categorical(ytest_new_50,10)
ytest_1 = to_categorical(ytest_1,10)
```

In [7]:
```
xtrain_1[0].shape
```

Out[7]: (32, 32, 3)

In [8]:

```
AlexNet = Sequential()

#1st Convolutional Layer
AlexNet.add(Conv2D(filters=96, input_shape=xtrain_1[0].shape, kernel_size=(11,11), strides=(4,4), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

# 1st Maxpooling Layer
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#2nd Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

# 2nd Maxpooling Layer
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#3rd Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#4th Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#5th Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

# 3rd Maxpooling Layer
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#Passing it to a Fully Connected layer
AlexNet.add(Flatten())

# 1st Fully Connected Layer
AlexNet.add(Dense(4096, input_shape=(32,32,3,)))
AlexNet.add(BatchNormalization())
```

```
AlexNet.add(Activation('relu'))

# Add Dropout to prevent overfitting
AlexNet.add(Dropout(0.4))

#2nd Fully Connected Layer
AlexNet.add(Dense(4096))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
#Add Dropout
AlexNet.add(Dropout(0.4))

#3rd Fully Connected Layer
# AlexNet.add(Dense(1000))                    # this is being removed to perform ConvNet as fixed feature extra
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
#Add Dropout
AlexNet.add(Dropout(0.4))

#Output Layer
AlexNet.add(Dense(10))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('softmax'))

#Model Summary
AlexNet.summary()
# https://cs231n.github.io/transfer-learning/
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 8, 8, 96)          34944
_________________________________________________________________
batch_normalization (BatchNo (None, 8, 8, 96)          384
_________________________________________________________________
activation (Activation)      (None, 8, 8, 96)          0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 4, 4, 96)          0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 4, 4, 256)         614656
_________________________________________________________________
batch_normalization_1 (Batch (None, 4, 4, 256)         1024
```

activation_1 (Activation)	(None, 4, 4, 256)	0
max_pooling2d_1 (MaxPooling2	(None, 2, 2, 256)	0
conv2d_2 (Conv2D)	(None, 2, 2, 384)	885120
batch_normalization_2 (Batch	(None, 2, 2, 384)	1536
activation_2 (Activation)	(None, 2, 2, 384)	0
conv2d_3 (Conv2D)	(None, 2, 2, 384)	1327488
batch_normalization_3 (Batch	(None, 2, 2, 384)	1536
activation_3 (Activation)	(None, 2, 2, 384)	0
conv2d_4 (Conv2D)	(None, 2, 2, 256)	884992
batch_normalization_4 (Batch	(None, 2, 2, 256)	1024
activation_4 (Activation)	(None, 2, 2, 256)	0
max_pooling2d_2 (MaxPooling2	(None, 1, 1, 256)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 4096)	1052672
batch_normalization_5 (Batch	(None, 4096)	16384
activation_5 (Activation)	(None, 4096)	0
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
batch_normalization_6 (Batch	(None, 4096)	16384
activation_6 (Activation)	(None, 4096)	0
dropout_1 (Dropout)	(None, 4096)	0

```
batch_normalization_7 (Batch (None, 4096)              16384
_________________________________________________________________
activation_7 (Activation)    (None, 4096)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                40970
_________________________________________________________________
batch_normalization_8 (Batch (None, 10)                40
_________________________________________________________________
activation_8 (Activation)    (None, 10)                0
=================================================================
Total params: 21,676,850
Trainable params: 21,649,502
Non-trainable params: 27,348
_________________________________________________________________
```

In [9]:
```
AlexNet.compile(loss = 'categorical_crossentropy', optimizer= 'adam', metrics=['accuracy'])
```

In [10]:
```
# checkpoint = ModelCheckpoint('AlexNet.h5',save_best_only=True, monitor='val_accuracy',mode='max')
history =AlexNet.fit(xtrain_1,ytrain_1,epochs=100,batch_size=32)
```

```
547/547 [==============================] - 7s 13ms/step - loss: 0.1324 - accuracy: 0.9609
Epoch 32/100
547/547 [==============================] - 7s 13ms/step - loss: 0.1261 - accuracy: 0.9617
Epoch 33/100
547/547 [==============================] - 7s 13ms/step - loss: 0.1211 - accuracy: 0.9665
Epoch 34/100
547/547 [==============================] - 7s 13ms/step - loss: 0.1212 - accuracy: 0.9645
Epoch 35/100
547/547 [==============================] - 7s 13ms/step - loss: 0.1023 - accuracy: 0.9682
Epoch 36/100
547/547 [==============================] - 7s 13ms/step - loss: 0.1170 - accuracy: 0.9663
Epoch 37/100
547/547 [==============================] - 7s 13ms/step - loss: 0.0979 - accuracy: 0.9706
Epoch 38/100
547/547 [==============================] - 7s 13ms/step - loss: 0.1012 - accuracy: 0.9693
Epoch 39/100
547/547 [==============================] - 7s 13ms/step - loss: 0.0876 - accuracy: 0.9740
Epoch 40/100
547/547 [==============================] - 7s 13ms/step - loss: 0.1026 - accuracy: 0.9692
Epoch 41/100
```

In [11]:
```
history = pd.DataFrame(history.history)
```

In [12]:
```
history
```

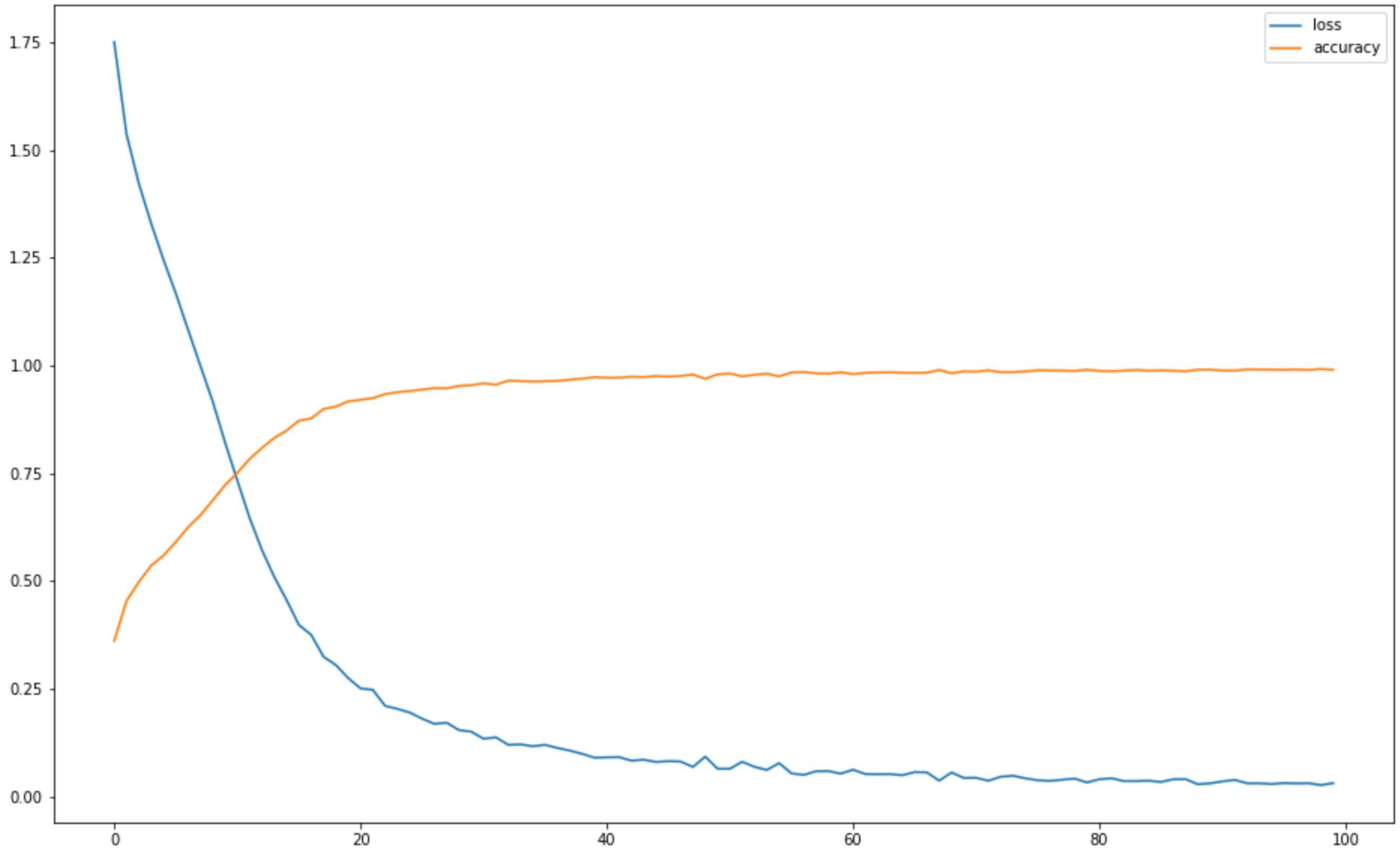
Out[12]:

	loss	accuracy
0	1.749619	0.361543
1	1.536217	0.454286
2	1.422057	0.498286
3	1.329333	0.536057
4	1.245421	0.559543
...	...	...
95	0.032239	0.990114
96	0.031350	0.990800
97	0.031872	0.989829
98	0.027423	0.992000
99	0.032049	0.990057

100 rows × 2 columns

In [13]:
```
1 history.plot.line(figsize=(16,10),)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f539a459250>



In [14]:
```
ypred = np.argmax(AlexNet.predict(xtest_1),axis=1)

ytrain = np.argmax(ytrain,axis=1)
ytrain_new_50 = np.argmax(ytrain_new_50,axis=1)
ytrain_1 = np.argmax(ytrain_1,axis=1)

ytest = np.argmax(ytest,axis=1)
ytest_new_50 = np.argmax(ytest_new_50,axis=1)
ytest_1 = np.argmax(ytest_1,axis=1)
```

In [15]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
accuracy_score(ypred,ytest_1)
```

Out[15]: 0.5650666666666667

In [16]:
```
ytrain = to_categorical(ytrain,10)
ytrain_new_50 = to_categorical(ytrain_new_50,10)
ytrain_1 = to_categorical(ytrain_1,10)

ytest = to_categorical(ytest,10)
ytest_new_50 = to_categorical(ytest_new_50,10)
ytest_1 = to_categorical(ytest_1,10)
```

In [17]:
```
history =AlexNet.fit(xtrain,ytrain,epochs=100,batch_size=32)
```

```
Epoch 1/100
1563/1563 [==============================] - 20s 13ms/step - loss: 0.8553 - accuracy: 0.7369
Epoch 2/100
1563/1563 [==============================] - 20s 13ms/step - loss: 0.6615 - accuracy: 0.7933
Epoch 3/100
1563/1563 [==============================] - 20s 13ms/step - loss: 0.5335 - accuracy: 0.8344
Epoch 4/100
1563/1563 [==============================] - 20s 13ms/step - loss: 0.4304 - accuracy: 0.8664
Epoch 5/100
1563/1563 [==============================] - 20s 13ms/step - loss: 0.3458 - accuracy: 0.8939
Epoch 6/100
1563/1563 [==============================] - 20s 13ms/step - loss: 0.2831 - accuracy: 0.9126
Epoch 7/100
1563/1563 [==============================] - 20s 13ms/step - loss: 0.2330 - accuracy: 0.9264
Epoch 8/100
1563/1563 [==============================] - 20s 13ms/step - loss: 0.1898 - accuracy: 0.9396
Epoch 9/100
1563/1563 [==============================] - 20s 13ms/step - loss: 0.1634 - accuracy: 0.9495
Epoch 10/100
```

In [18]:
```
history = pd.DataFrame(history.history)
```

In [19]:
```
1 history
```

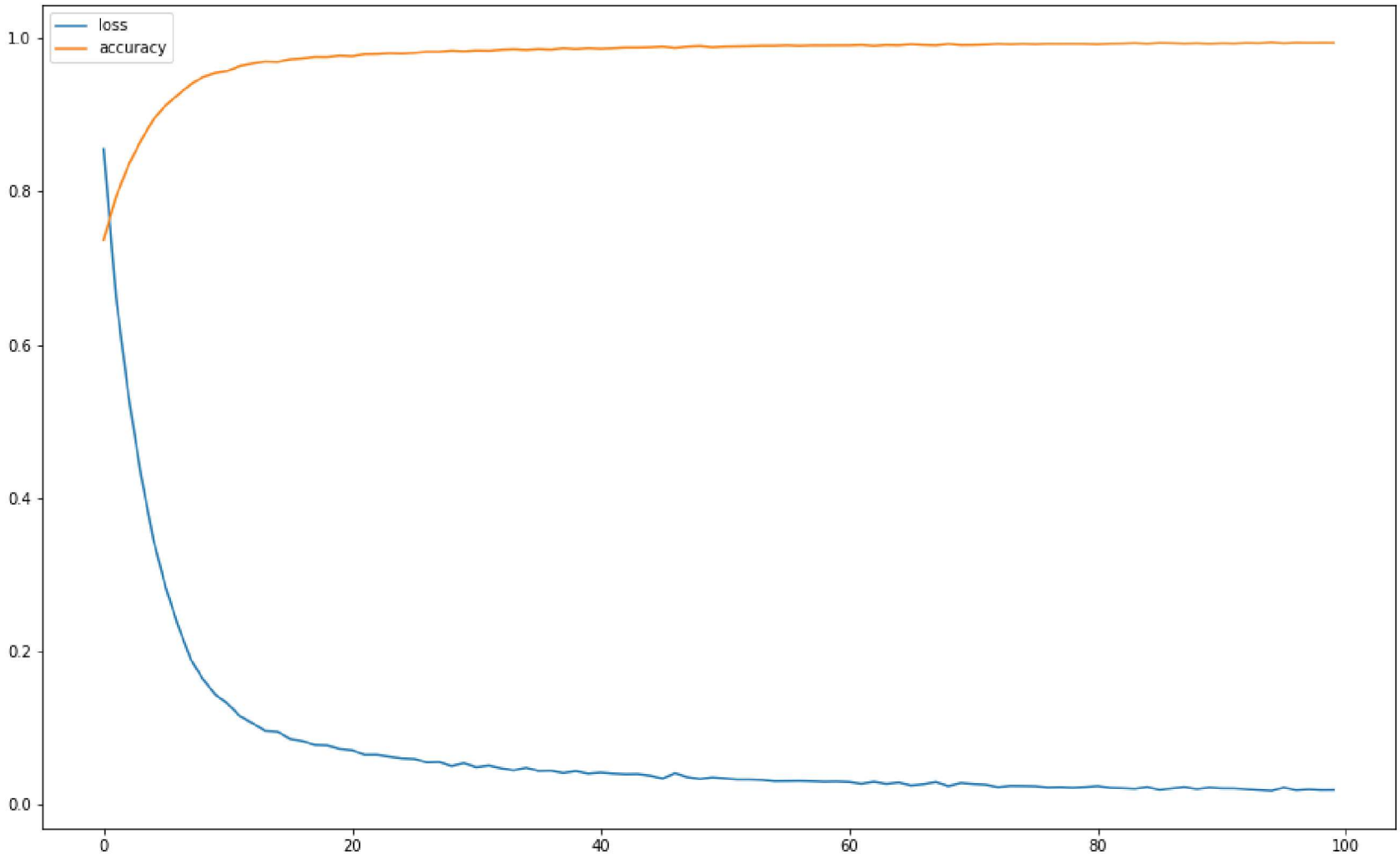
Out[19]:

	loss	accuracy
0	0.855332	0.73690
1	0.661502	0.79328
2	0.533495	0.83442
3	0.430364	0.86640
4	0.345819	0.89390
...	...	...
95	0.022198	0.99338
96	0.018639	0.99414
97	0.019822	0.99392
98	0.018850	0.99404
99	0.018977	0.99400

100 rows × 2 columns

In [20]:
```
1 history.plot.line(figsize=(16,10))
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f53d01a0a90>



In [21]:
```
ypred = np.argmax(AlexNet.predict(xtest),axis=1)

ytrain = np.argmax(ytrain,axis=1)
ytrain_new_50 = np.argmax(ytrain_new_50,axis=1)
ytrain_1 = np.argmax(ytrain_1,axis=1)

ytest = np.argmax(ytest,axis=1)
ytest_new_50 = np.argmax(ytest_new_50,axis=1)
ytest_1 = np.argmax(ytest_1,axis=1)
```

In [22]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
accuracy_score(ypred,ytest)
```

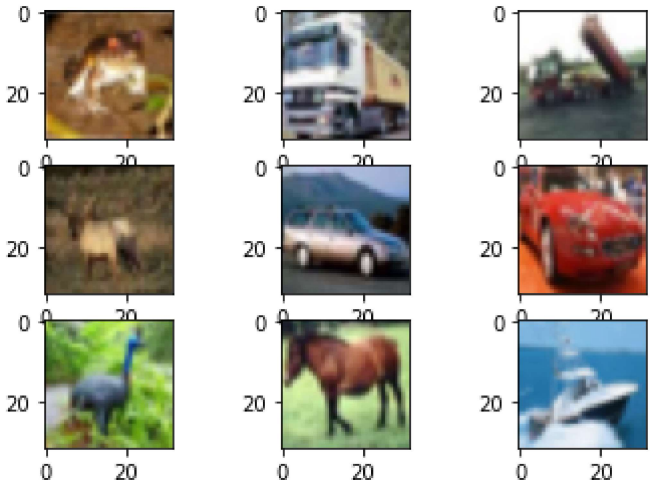
Out[22]: 0.6501

Task-2

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import keras as k
import tensorflow as tf
from keras.datasets import cifar10
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.model_selection import train_test_split
```

In [2]:
```
(trainX, trainy), (testX, testy) = cifar10.load_data()
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))
for i in range(9):
    plt.subplot(330 + 1 + i)
    plt.imshow(trainX[i])
plt.show()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170500096/170498071 [==============================] - 2s 0us/step
Train: X=(50000, 32, 32, 3), y=(50000, 1)
Test: X=(10000, 32, 32, 3), y=(10000, 1)
```



In [3]:
```
from keras.utils import to_categorical
trainy = to_categorical(trainy)
testy = to_categorical(testy)
```

In [4]:
```
from keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,BatchNormalization
from keras.models import Sequential
from keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator
```

In [5]:
```
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
```

In [6]:

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization()) # Adding Batch Normalization
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(10, activation='softmax'))
# compile model
opt = SGD(lr=0.001, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 16, 16, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
```

```
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 128)               262272
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 550,698
Trainable params: 550,634
Non-trainable params: 64
_________________________________________________________________
```

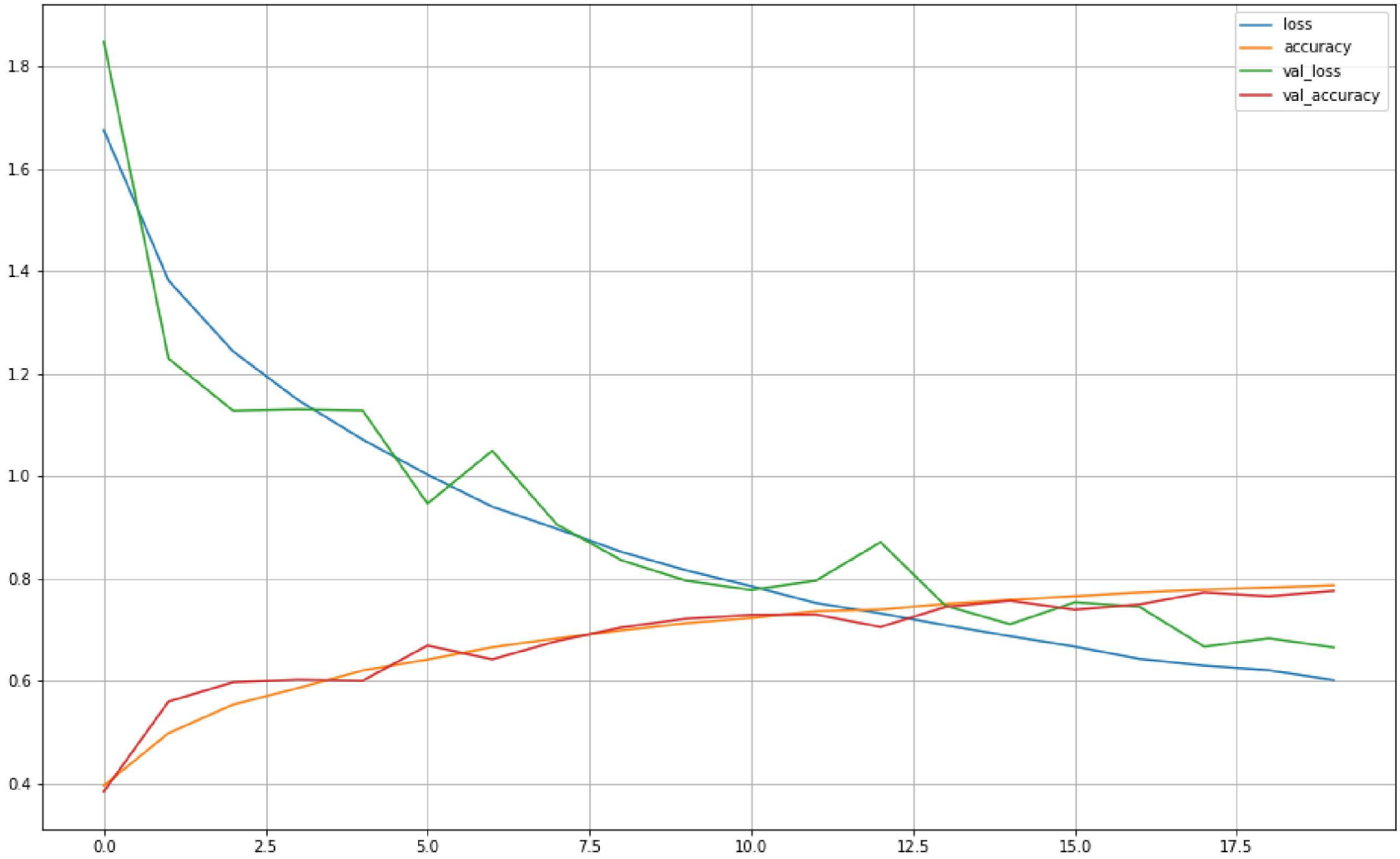
In [7]:
```
1 it_train = datagen.flow(trainX, trainy, batch_size=64)
```

In [8]:
```
1 steps = int(trainX.shape[0] / 64)
2 history = model.fit_generator(it_train, steps_per_epoch=steps, epochs=20, validation_data=(testX, testy), verbose=0)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
```

In [9]:
```
1 history = pd.DataFrame(history.history)
```

In [16]:
```
1 history.plot.line(figsize=(16,10)).grid("whitegrid")
```



In [11]:
```
1 ypred = np.argmax(model.predict(testX),axis=1)
```

In [12]:
```
1 ypred
```

Out[12]:
```
array([3, 8, 8, ..., 5, 1, 7])
```

In [13]:
```
1 testty = np.argmax(testy,axis=1)
```

In [14]:
```
1 testty
```

Out[14]:
```
array([3, 8, 8, ..., 5, 1, 7])
```

In [15]:
```
1 accuracy_score(ypred,testty)
```

Out[15]:
```
0.7764
```