Neural Networks & Deep Learning - ICP-7

CS 5720 (CRN 23216)

Student ID: 700745451

Student Name: Kamala Ramesh

Original Code Execution after correcting the Error:

Corrected the Input Shape in order of (height, width, channel)

input_shape(32, 32, 3)

```
# Simple CNN model for CIFAR-10
import numpy
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
#from keras import backend as K
#K.set_image_dim_ordering('th')

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

```
# Create the model
model = Sequential()

#Encountered Error in the below line
#model.add(Conv2D(32, (3, 3), input_shape=(3, 32, 32), padding='same', activation='relu', kernel_constraint=maxnorm(3)))

#Corrected line
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
# Compile model
epochs = 25
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
_________________________________________________________________
/usr/local/lib/python3.9/dist-packages/keras/optimizers/optimizer_v2/gradient_descent.py:114: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
None
Epoch 1/25
1563/1563 [==============================] - 19s 7ms/step - loss: 1.6976 - accuracy: 0.3872 - val_loss: 1.3758 - val_accuracy: 0.5101
Epoch 2/25
1563/1563 [==============================] - 10s 6ms/step - loss: 1.3129 - accuracy: 0.5296 - val_loss: 1.1629 - val_accuracy: 0.5809
Epoch 3/25
1563/1563 [==============================] - 10s 6ms/step - loss: 1.1592 - accuracy: 0.5901 - val_loss: 1.1421 - val_accuracy: 0.5981
Epoch 4/25
1563/1563 [==============================] - 10s 6ms/step - loss: 1.0428 - accuracy: 0.6313 - val_loss: 1.0464 - val_accuracy: 0.6291
Epoch 5/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.9412 - accuracy: 0.6664 - val_loss: 0.9875 - val_accuracy: 0.6478
Epoch 6/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.8576 - accuracy: 0.6952 - val_loss: 0.9486 - val_accuracy: 0.6657
Epoch 7/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.7841 - accuracy: 0.7221 - val_loss: 0.9302 - val_accuracy: 0.6768
Epoch 8/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.7161 - accuracy: 0.7474 - val_loss: 0.9138 - val_accuracy: 0.6848
Epoch 9/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.6605 - accuracy: 0.7660 - val_loss: 0.8941 - val_accuracy: 0.6921
Epoch 10/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.6089 - accuracy: 0.7848 - val_loss: 0.9057 - val_accuracy: 0.6961
Epoch 11/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5622 - accuracy: 0.8011 - val_loss: 0.9224 - val_accuracy: 0.6919
Epoch 12/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5264 - accuracy: 0.8154 - val_loss: 0.9424 - val_accuracy: 0.6945
Epoch 13/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4855 - accuracy: 0.8276 - val_loss: 0.9148 - val_accuracy: 0.7031
Epoch 14/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.4473 - accuracy: 0.8416 - val_loss: 0.9482 - val_accuracy: 0.6950
Epoch 15/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4180 - accuracy: 0.8529 - val_loss: 0.9709 - val_accuracy: 0.7030
Epoch 16/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3925 - accuracy: 0.8621 - val_loss: 0.9744 - val_accuracy: 0.7004
Epoch 17/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3689 - accuracy: 0.8686 - val_loss: 0.9872 - val_accuracy: 0.7031
Epoch 18/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3429 - accuracy: 0.8790 - val_loss: 0.9949 - val_accuracy: 0.7017
Epoch 19/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3223 - accuracy: 0.8862 - val_loss: 0.9913 - val_accuracy: 0.7061
Epoch 20/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3080 - accuracy: 0.8914 - val_loss: 1.0083 - val_accuracy: 0.7081
Epoch 21/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2885 - accuracy: 0.8986 - val_loss: 1.0165 - val_accuracy: 0.7095
Epoch 22/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2725 - accuracy: 0.9043 - val_loss: 1.0589 - val_accuracy: 0.7032
Epoch 23/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2582 - accuracy: 0.9106 - val_loss: 1.0628 - val_accuracy: 0.7108
Epoch 24/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2455 - accuracy: 0.9138 - val_loss: 1.0660 - val_accuracy: 0.7084
Epoch 25/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2382 - accuracy: 0.9169 - val_loss: 1.0816 - val_accuracy: 0.7077
Accuracy: 70.77%
```

1. Applied the instructions given in the assignment and executed them all at once

```python
# Simple CNN model for CIFAR-10
import numpy
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
#from keras import backend as K
#K.set_image_dim_ordering('th')

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
# one hot encode outputs
y_test_out = y_test
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

```
# Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))

model.add(Dense(num_classes, activation='softmax'))
```

```
# Compile model
epochs = 25
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())
# Fit the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 18s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

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

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

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 dropout_1 (Dropout)         (None, 16, 16, 64)        0

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 dropout_2 (Dropout)         (None, 8, 8, 128)         0

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dropout_3 (Dropout)         (None, 2048)              0

 dense (Dense)               (None, 1024)              2098176

 dropout_4 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 512)               524800

 dropout_5 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 10)                5130

=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
_________________________________________________________________
/usr/local/lib/python3.9/dist-packages/keras/optimizers/optimizer_v2/gradient_descent.py:114: UserWarning: The `lr` argument is deprecated
  super().__init__(name, **kwargs)
None
Epoch 1/25
1563/1563 [==============================] - 26s 8ms/step - loss: 1.8107 - accuracy: 0.3268 - val_loss: 1.5910 - val_accuracy: 0.4371
Epoch 2/25
1563/1563 [==============================] - 13s 8ms/step - loss: 1.3706 - accuracy: 0.4994 - val_loss: 1.1744 - val_accuracy: 0.5776
Epoch 3/25
1563/1563 [==============================] - 14s 9ms/step - loss: 1.1574 - accuracy: 0.5842 - val_loss: 1.0379 - val_accuracy: 0.6333
Epoch 4/25
1563/1563 [==============================] - 13s 8ms/step - loss: 1.0009 - accuracy: 0.6428 - val loss: 0.9136 - val accuracy: 0.6849
```

```
Epoch 5/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.8910 - accuracy: 0.6836 - val_loss: 0.8346 - val_accuracy: 0.7102
Epoch 6/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.8053 - accuracy: 0.7154 - val_loss: 0.8000 - val_accuracy: 0.7239
Epoch 7/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.7414 - accuracy: 0.7396 - val_loss: 0.7395 - val_accuracy: 0.7437
Epoch 8/25
1563/1563 [==============================] - 14s 9ms/step - loss: 0.6840 - accuracy: 0.7593 - val_loss: 0.7252 - val_accuracy: 0.7461
Epoch 9/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.6408 - accuracy: 0.7736 - val_loss: 0.6901 - val_accuracy: 0.7612
Epoch 10/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.6007 - accuracy: 0.7882 - val_loss: 0.6499 - val_accuracy: 0.7794
Epoch 11/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.5696 - accuracy: 0.7992 - val_loss: 0.6425 - val_accuracy: 0.7763
Epoch 12/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.5415 - accuracy: 0.8100 - val_loss: 0.6271 - val_accuracy: 0.7891
Epoch 13/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.5102 - accuracy: 0.8199 - val_loss: 0.6397 - val_accuracy: 0.7812
Epoch 14/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.4898 - accuracy: 0.8243 - val_loss: 0.6151 - val_accuracy: 0.7935
Epoch 15/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.4678 - accuracy: 0.8348 - val_loss: 0.6074 - val_accuracy: 0.7957
Epoch 16/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.4474 - accuracy: 0.8399 - val_loss: 0.6122 - val_accuracy: 0.7937
Epoch 17/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.4293 - accuracy: 0.8487 - val_loss: 0.6018 - val_accuracy: 0.7976
Epoch 18/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.4113 - accuracy: 0.8536 - val_loss: 0.6046 - val_accuracy: 0.7933
Epoch 19/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.4012 - accuracy: 0.8562 - val_loss: 0.5902 - val_accuracy: 0.8057
Epoch 20/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.3825 - accuracy: 0.8642 - val_loss: 0.5905 - val_accuracy: 0.8019
Epoch 21/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.3719 - accuracy: 0.8677 - val_loss: 0.6025 - val_accuracy: 0.7982
Epoch 22/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.3583 - accuracy: 0.8721 - val_loss: 0.5900 - val_accuracy: 0.8034
Epoch 23/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.3479 - accuracy: 0.8750 - val_loss: 0.5924 - val_accuracy: 0.8036
Epoch 24/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.3355 - accuracy: 0.8811 - val_loss: 0.5744 - val_accuracy: 0.8104
Epoch 25/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.3239 - accuracy: 0.8836 - val_loss: 0.5865 - val_accuracy: 0.8091
Accuracy: 80.91%
```

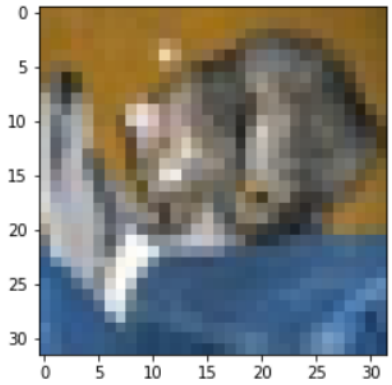
Did the performance change?

Yes, the performance has changed. The accuracy increased after implementing the model with the updated changes. In this execution, the updated model has more than the original model given.

2. Prediction of first 4 images of the test data using the above model.

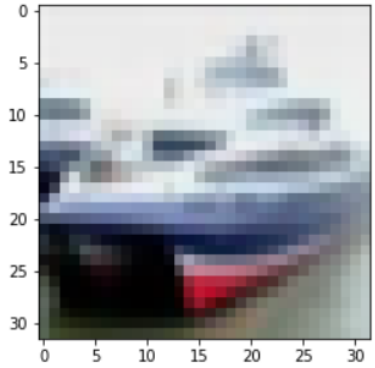
```
import numpy as np
import matplotlib.pyplot as plt
def predict_test_img(i):
  img = X_test[i]
  img_class = y_test_out[i][0]
  prediction = model.predict(np.array([img]),verbose=0)

  print("Prediction of Image",i)
  # Plot the selected image
  plt.imshow(img)
  plt.show()
  print("Predicted class: ", np.argmax(prediction[0]))
  print("Actual class: ", img_class,"\n")

predict_test_img(0)
predict_test_img(1)
predict_test_img(2)
predict_test_img(3)
```

Prediction of Image 0



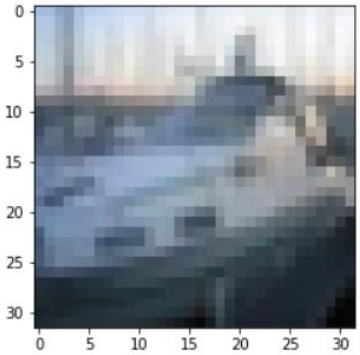
Predicted class: 3
Actual class: 3

Prediction of Image 1



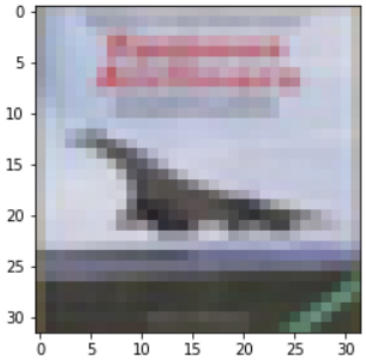
Predicted class: 8
Actual class: 8

Prediction of Image 2



Predicted class: 8
Actual class: 8

Prediction of Image 3

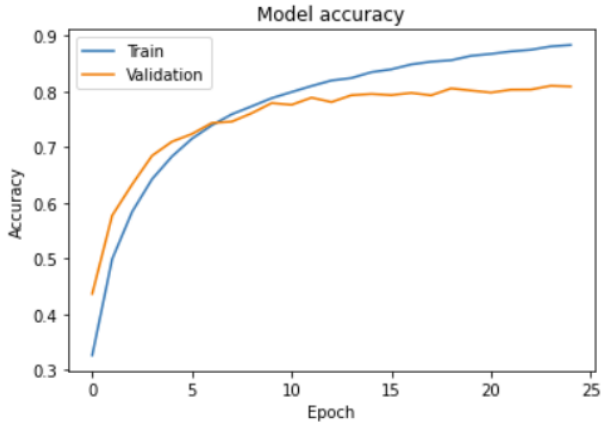


Predicted class: 0
Actual class: 0

For the first 4 images of the test data, the model has predicted correctly.

3. Visualization of Loss and Accuracy using the history object

```
[8]  # Plot the training and validation accuracy over epochs
     plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('Model accuracy')
     plt.ylabel('Accuracy')
     plt.xlabel('Epoch')
     plt.legend(['Train', 'Validation'], loc='upper left')
     plt.show()
```




```
     # Plot the training and validation loss over epochs
     plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.title('Model loss')
     plt.ylabel('Loss')
     plt.xlabel('Epoch')
     plt.legend(['Train', 'Validation'], loc='upper left')
     plt.show()
```