

Angel Reddy Nakkala

700748217

ICP7

```
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
#from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.utils import to_categorical
#from keras import backend as K
#K.set_image_dim_ordering('th')
# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# convert from int to float and normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# one hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
num_classes = y_test.shape[1]

# transpose the dimensions of the input data
X_train = np.transpose(X_train, (0, 3, 1, 2))
X_test = np.transpose(X_test, (0, 3, 1, 2))

# Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(3, 32, 32), padding='same',
activation='relu'))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

```

# Compile model
epochs = 5
lr = 0.01
decay = lr/epochs
sgd = SGD(lr=lr)
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))

```

## Output:

### In class programming:

1. Follow the instruction below and then report how the performance changed.(apply all at once)

- Convolutional input layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
- Dropout layer at 20%.
- Convolutional layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
- Max Pool layer with size 2×2.
- Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
- Dropout layer at 20%.
- Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
- Max Pool layer with size 2×2.
- Convolutional layer, 128 feature maps with a size of 3×3 and a rectifier activation function.
- Dropout layer at 20%.
- Convolutional layer, 128 feature maps with a size of 3×3 and a rectifier activation function.
- Max Pool layer with size 2×2.
- Flatten layer.
- Dropout layer at 20%.
- Fully connected layer with 1024 units and a rectifier activation function.
- Dropout layer at 20%.
- Fully connected layer with 512 units and a rectifier activation function.
- Dropout layer at 20%.
- Fully connected output layer with 10 units and a Softmax activation function

Did the performance change?

```

import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential

```

```

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
#from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.utils import to_categorical
# from keras import backend as K
# K.tensorflow_backend.set_image_dim_ordering('th')
# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# convert from int to float and normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# one hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
num_classes = y_test.shape[1]

# transpose the dimensions of the input data
X_train = np.transpose(X_train, (0, 3, 1, 2))
X_test = np.transpose(X_test, (0, 3, 1, 2))

# Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(3, 32, 32), padding='same',
activation='relu'))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(1, 1)))
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(1, 1)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))

```

```

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
epochs = 5
lr = 0.01
decay = lr/epochs
sgd = SGD(lr=lr)
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))

```

Output:

The screenshot shows a Google Colab notebook interface. The notebook is titled "ImageClassification.ipynb". The output of the code execution is displayed in a text area. It shows the model summary, total parameters, and training progress over 5 epochs.

```

Model: "model"
Layer (type)                 Output Shape         Param #   Connected to
=================================================================
flatten_3 (Flatten)          (None, 2048)         0         input[0][0]
dropout_13 (Dropout)          (None, 2048)         0         flatten_3
dense_7 (Dense)               (None, 1024)         2098176    dropout_13
dropout_14 (Dropout)          (None, 1024)         0         dense_7
dense_8 (Dense)               (None, 512)          524800     dropout_14
dropout_15 (Dropout)          (None, 512)          0         dense_8
dense_9 (Dense)               (None, 10)           5130       dropout_15
Total params: 2923466 (11.15 MB)
Trainable params: 2923466 (11.15 MB)
Non-trainable params: 0 (0.00 Byte)

None
Epoch 1/5
1563/1563 [=====] - 13s 8ms/step - loss: 2.2965 - accuracy: 0.1222 - val_loss: 2.2593 - val_accuracy: 0.1747
Epoch 2/5
1563/1563 [=====] - 11s 7ms/step - loss: 2.1235 - accuracy: 0.2153 - val_loss: 2.0188 - val_accuracy: 0.2731
Epoch 3/5
1563/1563 [=====] - 12s 8ms/step - loss: 1.9569 - accuracy: 0.2815 - val_loss: 1.8338 - val_accuracy: 0.3313
Epoch 4/5
1563/1563 [=====] - 11s 7ms/step - loss: 1.8362 - accuracy: 0.3258 - val_loss: 1.7425 - val_accuracy: 0.3629
Epoch 5/5
1563/1563 [=====] - 11s 7ms/step - loss: 1.7523 - accuracy: 0.3585 - val_loss: 1.6617 - val_accuracy: 0.3957
Accuracy: 39.57%

```

2. Predict the first 4 images of the test data using the above model. Then, compare with the actual label for those 4 images to check whether or not the model has predicted correctly

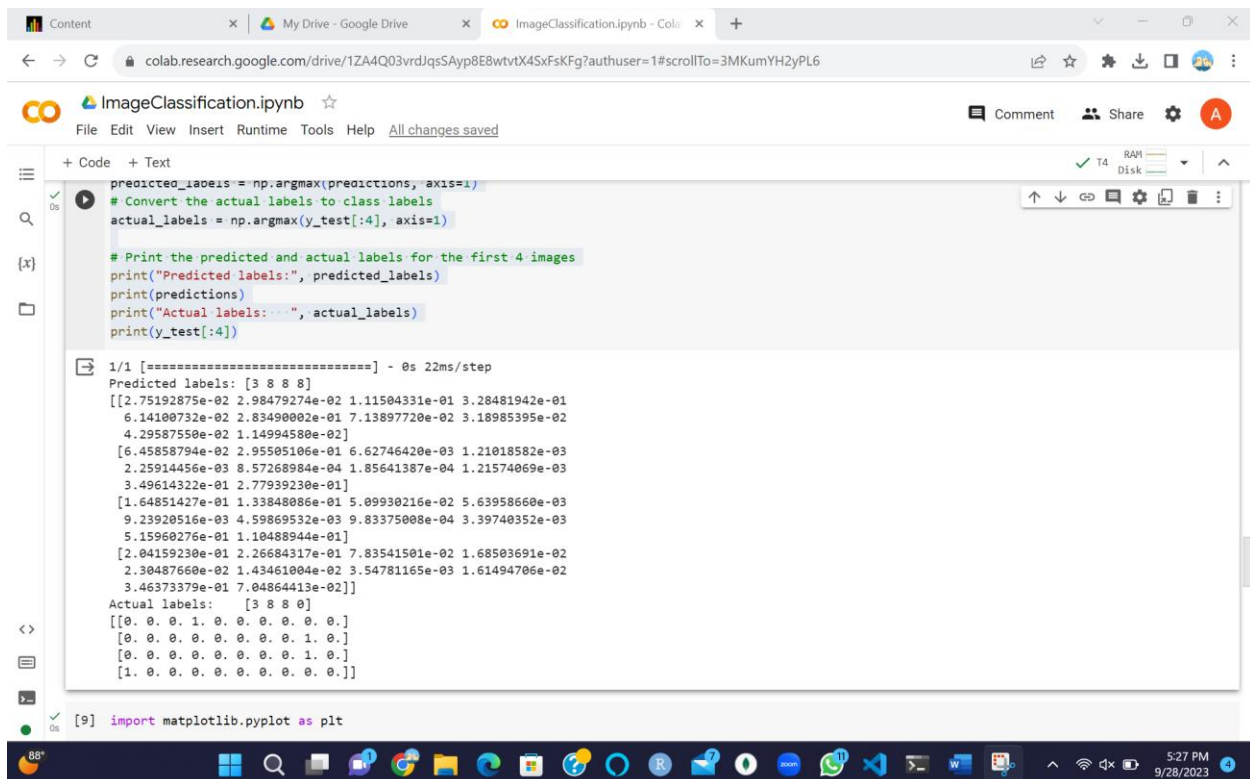
```

# Predict the first 4 images of the test data
predictions = model.predict(X_test[:4])
# Convert the predictions to class labels
predicted_labels = np.argmax(predictions, axis=1)
# Convert the actual labels to class labels
actual_labels = np.argmax(y_test[:4], axis=1)

# Print the predicted and actual labels for the first 4 images
print("Predicted labels:", predicted_labels)
print(predictions)
print("Actual labels:  ", actual_labels)
print(y_test[:4])

```

Output:



```

1/1 [=====] - 0s 22ms/step
Predicted labels: [3 8 8 8]
[[2.75192875e-02 2.98479274e-02 1.11504331e-01 3.28481942e-01
  6.14100732e-02 2.83490002e-01 7.13897720e-02 3.18985395e-02
  4.29587550e-02 1.14994580e-02]
 [6.45858794e-02 2.95505106e-01 6.62746420e-03 1.21018582e-03
  2.25914456e-03 8.57268984e-04 1.85641387e-04 1.21574069e-03
  3.49614322e-01 2.77939230e-01]
 [1.64851427e-01 1.33848086e-01 5.09930216e-02 5.63958660e-03
  9.23920516e-03 4.59869532e-03 9.83375008e-04 3.39740352e-03
  5.15960276e-01 1.10488944e-01]
 [2.04159230e-01 2.26684317e-01 7.83541501e-02 1.68503691e-02
  2.30487660e-02 1.43461004e-02 3.54781165e-03 1.61494706e-02
  3.46373379e-01 7.04864413e-02]]
Actual labels: [3 8 8 8]
[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

```

### 3. Visualize Loss and Accuracy using the history object

```

import matplotlib.pyplot as plt

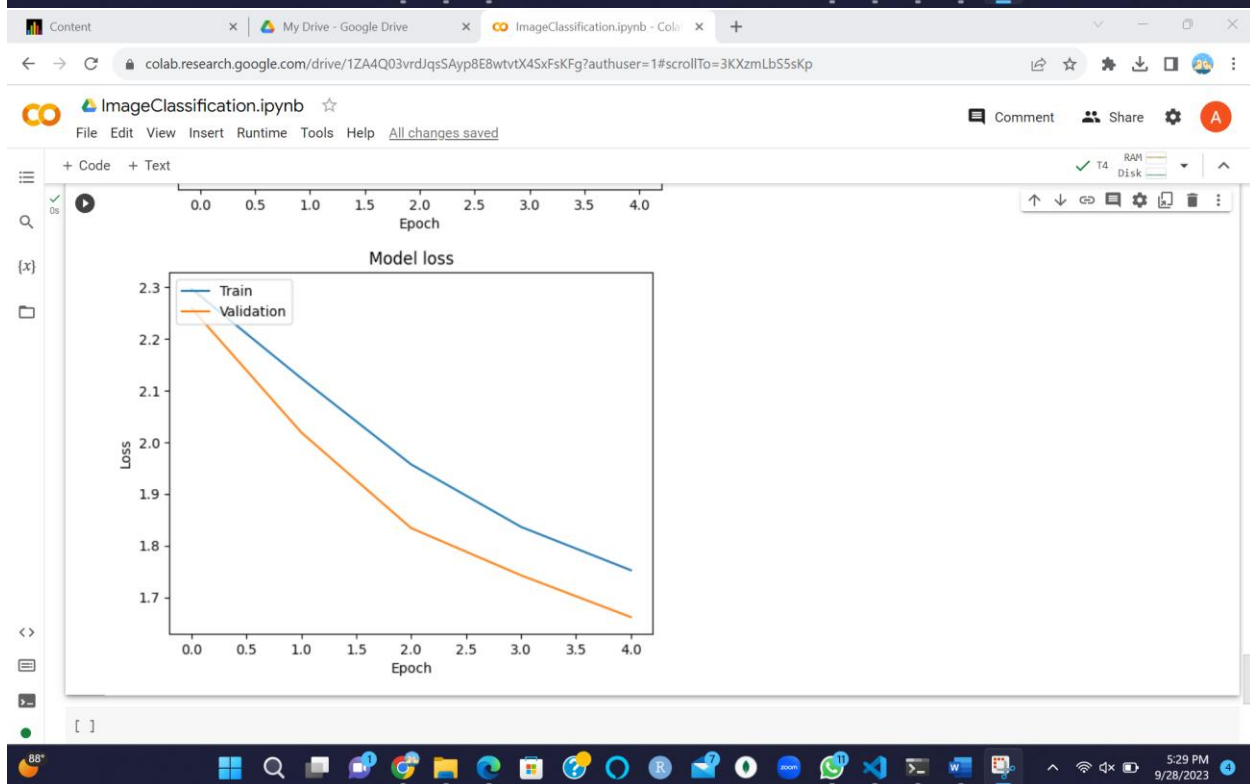
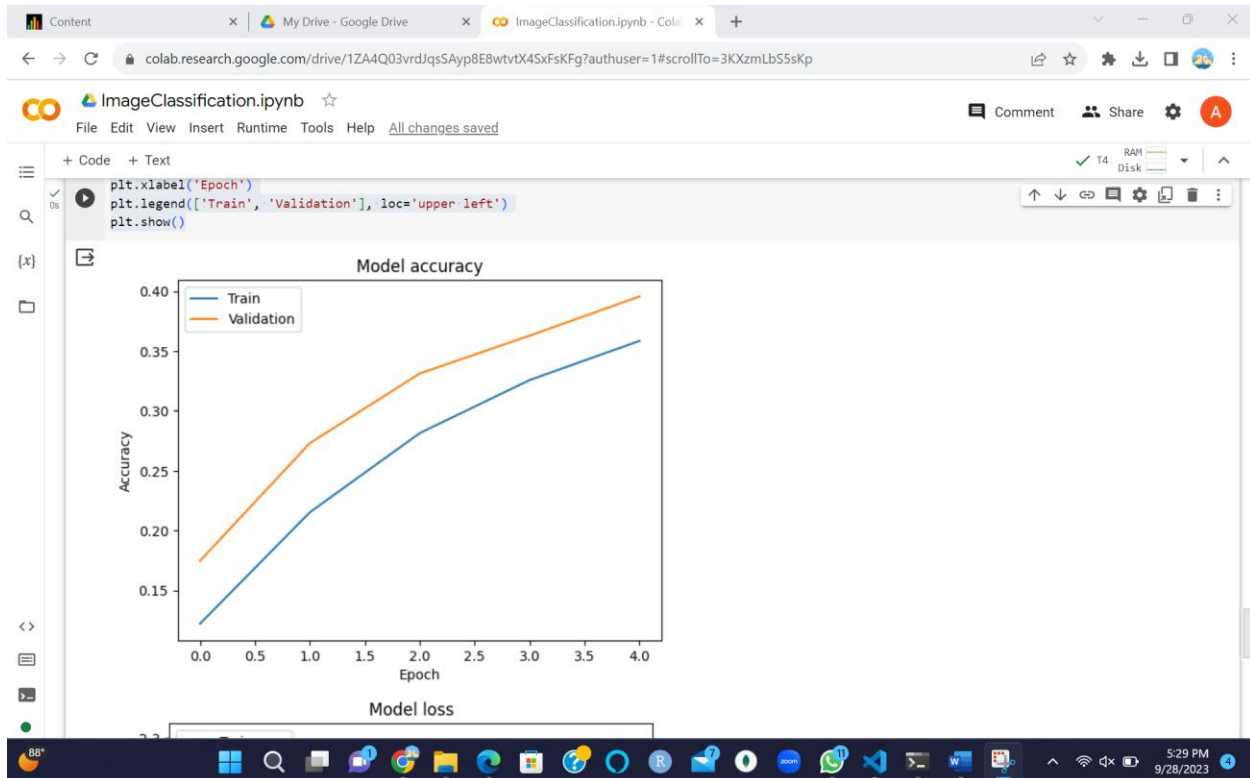
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')

```

```
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
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()
```

Output:



GitHub :

<https://github.com/angelreddy09/DNN/tree/main/ICP7>

