ICP -7

VVS Murthy KOLLA
700729142

Github: https://github.com/murthykolla/ICP-7.git
Video:https://docs.google.com/document/d/1swTmUqeImsZy85msyjTUdI4mLNsYSD0oIgXOI96snKs/edit?usp=share_link

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

```python
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.constraints import maxnorm
from keras.utils import np_utils
from keras.optimizers import SGD

# Fix random seed for reproducibility
np.random.seed(7)

# 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') / 255.0
X_test = X_test.astype('float32') / 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()
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'))
```

```python
# Compile model
epochs = 50
learning_rate = 0.001
decay_rate = learning_rate / epochs
sgd = SGD(lr=learning_rate, momentum=0.9, decay=decay_rate, 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)

# Evaluate the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1] * 100))
```

```
_________________________________________________________________
 conv2d_12 (Conv2D)          (None, 32, 32, 32)        896

 dropout_12 (Dropout)        (None, 32, 32, 32)        0

 conv2d_13 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_6 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_14 (Conv2D)          (None, 16, 16, 64)        18496

 dropout_13 (Dropout)        (None, 16, 16, 64)        0

 conv2d_15 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_7 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_16 (Conv2D)          (None, 8, 8, 128)         73856

 dropout_14 (Dropout)        (None, 8, 8, 128)         0

 conv2d_17 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_8 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten_2 (Flatten)         (None, 2048)              0

 dropout_15 (Dropout)        (None, 2048)              0

 dense_6 (Dense)             (None, 1024)              2098176

 dropout_16 (Dropout)        (None, 1024)              0

 dense_7 (Dense)             (None, 512)               524800

 dropout_17 (Dropout)        (None, 512)               0

 dense_8 (Dense)             (None, 10)                5130

=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
_________________________________________________________________
--
```

```
1563/1563 [==============================] - 13s 9ms/step - loss: 0.5153 - accuracy: 0.8176 - val_loss: 0.6257 - val_accuracy: 0.7866
Epoch 37/50
1563/1563 [==============================] - 13s 8ms/step - loss: 0.5085 - accuracy: 0.8180 - val_loss: 0.6315 - val_accuracy: 0.7793
Epoch 38/50
1563/1563 [==============================] - 13s 8ms/step - loss: 0.4917 - accuracy: 0.8255 - val_loss: 0.6208 - val_accuracy: 0.7872
Epoch 39/50
1563/1563 [==============================] - 14s 9ms/step - loss: 0.4858 - accuracy: 0.8287 - val_loss: 0.6193 - val_accuracy: 0.7866
Epoch 40/50
1563/1563 [==============================] - 14s 9ms/step - loss: 0.4656 - accuracy: 0.8353 - val_loss: 0.6159 - val_accuracy: 0.7906
Epoch 41/50
1563/1563 [==============================] - 13s 9ms/step - loss: 0.4618 - accuracy: 0.8342 - val_loss: 0.6168 - val_accuracy: 0.7894
Epoch 42/50
1563/1563 [==============================] - 14s 9ms/step - loss: 0.4521 - accuracy: 0.8392 - val_loss: 0.6036 - val_accuracy: 0.7929
Epoch 43/50
1563/1563 [==============================] - 13s 8ms/step - loss: 0.4453 - accuracy: 0.8414 - val_loss: 0.6071 - val_accuracy: 0.7921
Epoch 44/50
1563/1563 [==============================] - 13s 9ms/step - loss: 0.4304 - accuracy: 0.8472 - val_loss: 0.6263 - val_accuracy: 0.7884
Epoch 45/50
1563/1563 [==============================] - 14s 9ms/step - loss: 0.4245 - accuracy: 0.8485 - val_loss: 0.6078 - val_accuracy: 0.7926
Epoch 46/50
1563/1563 [==============================] - 13s 9ms/step - loss: 0.4134 - accuracy: 0.8515 - val_loss: 0.6093 - val_accuracy: 0.7938
Epoch 47/50
1563/1563 [==============================] - 14s 9ms/step - loss: 0.4090 - accuracy: 0.8543 - val_loss: 0.5997 - val_accuracy: 0.7963
Epoch 48/50
1563/1563 [==============================] - 14s 9ms/step - loss: 0.3977 - accuracy: 0.8575 - val_loss: 0.6009 - val_accuracy: 0.7970
Epoch 49/50
1563/1563 [==============================] - 14s 9ms/step - loss: 0.3869 - accuracy: 0.8612 - val_loss: 0.6138 - val_accuracy: 0.7931
Epoch 50/50
1563/1563 [==============================] - 14s 9ms/step - loss: 0.3774 - accuracy: 0.8649 - val_loss: 0.6042 - val_accuracy: 0.7975
Accuracy: 79.75%
```

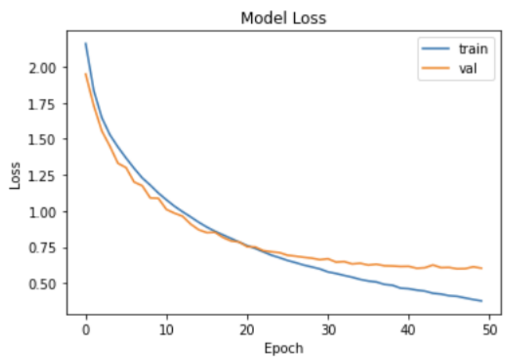
```
[ ] #2. Predict the first 4 images of the test data using the above model. Then, compare with the actual label for those 4
    #images to check whether or not the model has predicted correctly.
```

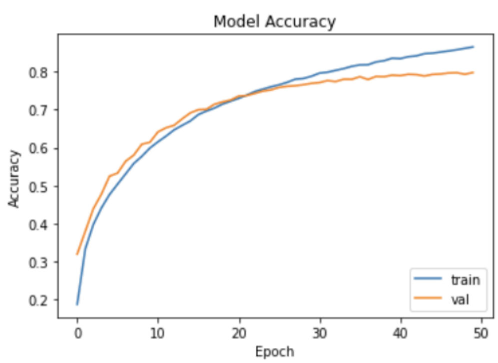
```python
# Predicting the first 4 images of the test data
predictions = model.predict(X_test[:4])
# Converting the predictions to class labels
predicted_labels = np.argmax(predictions, axis=1)
# Converting the actual labels to class labels
actual_labels = np.argmax(y_test[:4], axis=1)

# Printing the predicted and actual labels for the first 4 images
print("Actual labels:    ", actual_labels)
print("Predicted labels:", predicted_labels)
```

```
1/1 [==============================] - 0s 28ms/step
Actual labels:    [3 8 8 0]
Predicted labels: [3 8 8 0]
```

```
[ ] #3. Visualize Loss and Accuracy using the history object
```

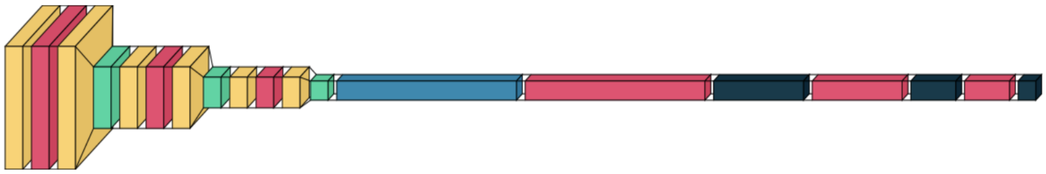
```
import matplotlib.pyplot as plt

# plot history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

# plot history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='lower right')
plt.show()
```






```
[ ]
```

```
[ ] #visualize the neural network model,
    import visualkeras
    visualkeras.layered_view(model)
```



#We can display the generated image using the following command.

```
#We can display the generated image using the following command.
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```