CS5542 Big Data Apps and Analytics

In Class Programming –5 Report
(Jongkook Son)

Project Overview:

Use the same data (that we used in ICP4 `from keras.datasets import cifar10)`**) and use the model provided in ICP5 to perform image classification. You must change 4 hyper parameters in the source code. Report your findings in detail.**
Note: please indicate in your reports which 4 hyperparameters you changed in the source code and why in your opinion these changes are logical.

Requirements/Task(s):

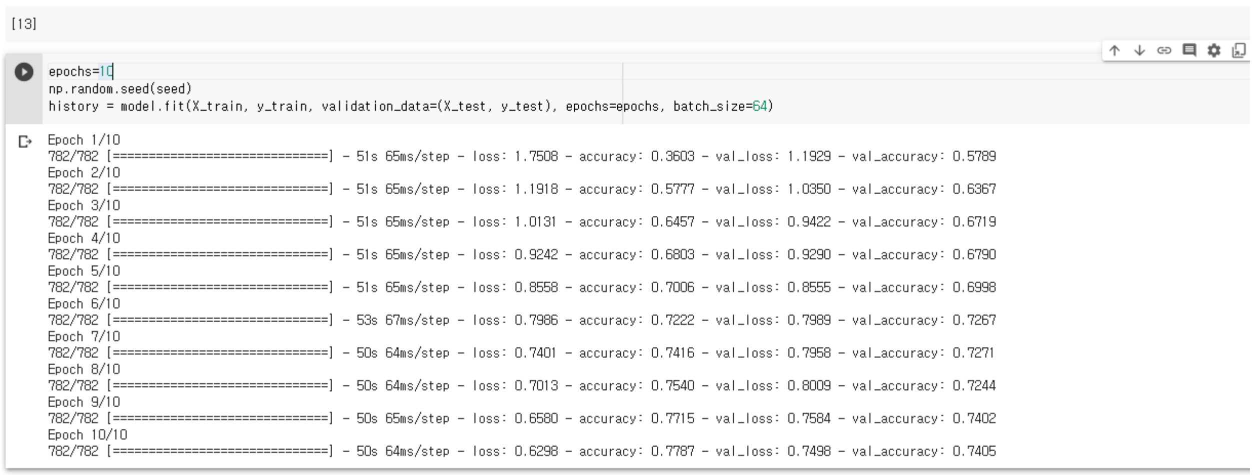
1) Successfully executing the code and changing 4 hyperparameters in the model (75 points)
2) Validating the model on 5 new images (that are not present in the data set and are not used in training or testing but are taken form internet) (5 points)
3) Providing the logical explanation of the changes that you made to hyper parameters and over all code quality (10 points)
4) Pdf Report quality, video explanation (10 points)

What I learned in ICP:

I could have learned how to classifying images using deep learning model. Also Thanks to this ICP5 I could get some insight how to improve deeplearning model by adjusting some hyperparameters. In this ICP, I adjusted class number, activation function, filter numbers, batch normalization. So that the overall accuracy of the model could be improved. I keep thinking about which layer and hyperparameter should I add and which activation fuction should I use to maximize accuracy for model. Still I am confused and do not have intuitive about this concept. But I am sure I can have more insight and knowledge on this by keep exploring and learn in upcoming class.

ICP description what was the task you were performing and Screen shots that shows the successful execution of each required step of your code

The model’s accuracy before change in hyperparameter

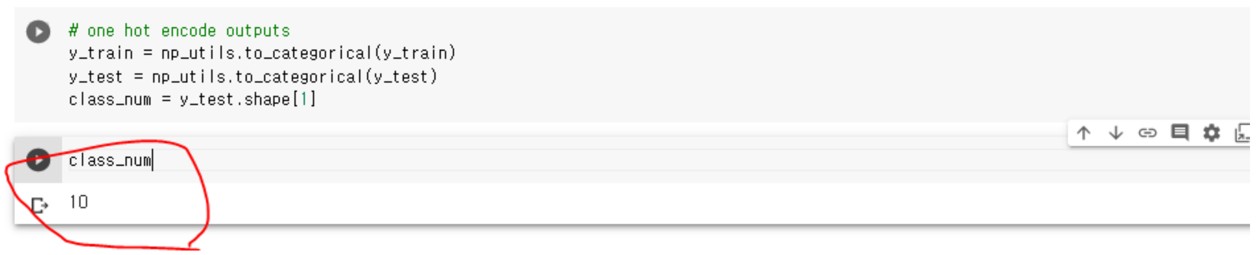


```
[13]

epochs=10
np.random.seed(seed)
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=64)

Epoch 1/10
782/782 [==============================] - 51s 65ms/step - loss: 1.7508 - accuracy: 0.3603 - val_loss: 1.1929 - val_accuracy: 0.5789
Epoch 2/10
782/782 [==============================] - 51s 65ms/step - loss: 1.1918 - accuracy: 0.5777 - val_loss: 1.0350 - val_accuracy: 0.6367
Epoch 3/10
782/782 [==============================] - 51s 65ms/step - loss: 1.0131 - accuracy: 0.6457 - val_loss: 0.9422 - val_accuracy: 0.6719
Epoch 4/10
782/782 [==============================] - 51s 65ms/step - loss: 0.9242 - accuracy: 0.6803 - val_loss: 0.9290 - val_accuracy: 0.6790
Epoch 5/10
782/782 [==============================] - 51s 65ms/step - loss: 0.8558 - accuracy: 0.7006 - val_loss: 0.8555 - val_accuracy: 0.6998
Epoch 6/10
782/782 [==============================] - 53s 67ms/step - loss: 0.7986 - accuracy: 0.7222 - val_loss: 0.7989 - val_accuracy: 0.7267
Epoch 7/10
782/782 [==============================] - 50s 64ms/step - loss: 0.7401 - accuracy: 0.7416 - val_loss: 0.7958 - val_accuracy: 0.7271
Epoch 8/10
782/782 [==============================] - 50s 64ms/step - loss: 0.7013 - accuracy: 0.7540 - val_loss: 0.8009 - val_accuracy: 0.7244
Epoch 9/10
782/782 [==============================] - 50s 65ms/step - loss: 0.6580 - accuracy: 0.7715 - val_loss: 0.7584 - val_accuracy: 0.7402
Epoch 10/10
782/782 [==============================] - 50s 64ms/step - loss: 0.6298 - accuracy: 0.7787 - val_loss: 0.7498 - val_accuracy: 0.7405
```

➪ The accuracy is about 0.7789 before the change

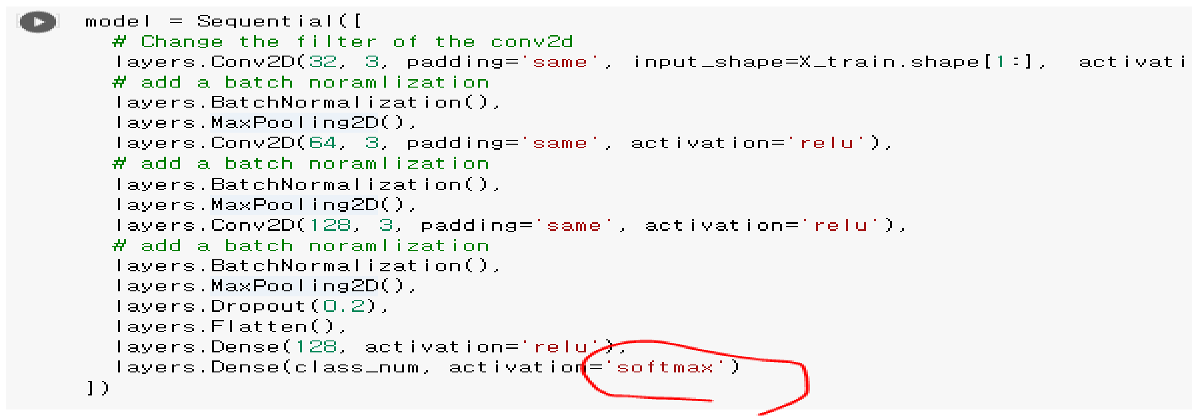
Changing 4 hyper parameters

1. Change the class number and activation function



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

```
class_num

10
```

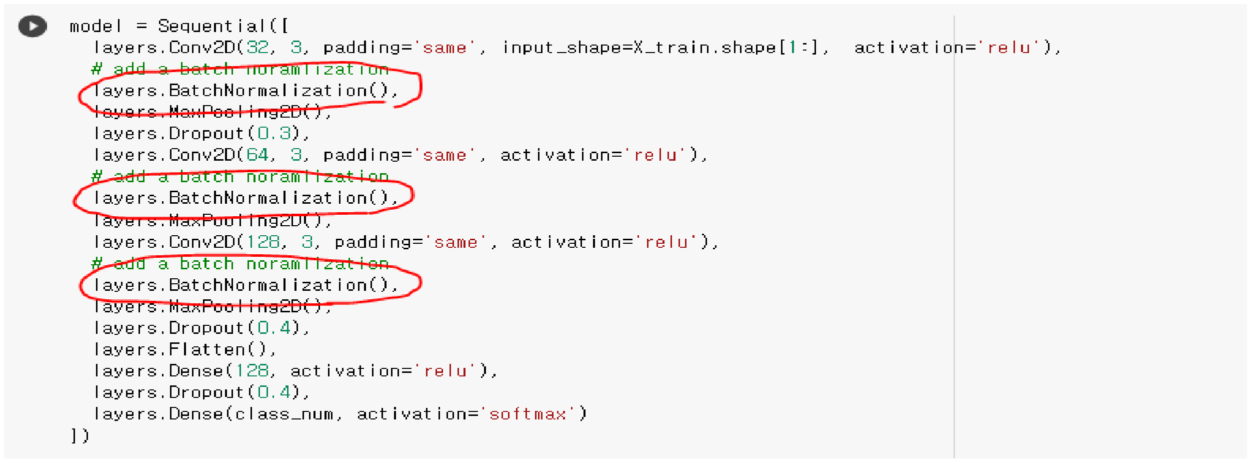


```
model = Sequential([
    # Change the filter of the conv2d
    layers.Conv2D(32, 3, padding='same', input_shape=X_train.shape[1:],  activati
    # add a batch noramlization
    layers.BatchNormalization(),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    # add a batch noramlization
    layers.BatchNormalization(),
    layers.MaxPooling2D(),
    layers.Conv2D(128, 3, padding='same', activation='relu'),
    # add a batch noramlization
    layers.BatchNormalization(),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(class_num, activation='softmax')
])
```

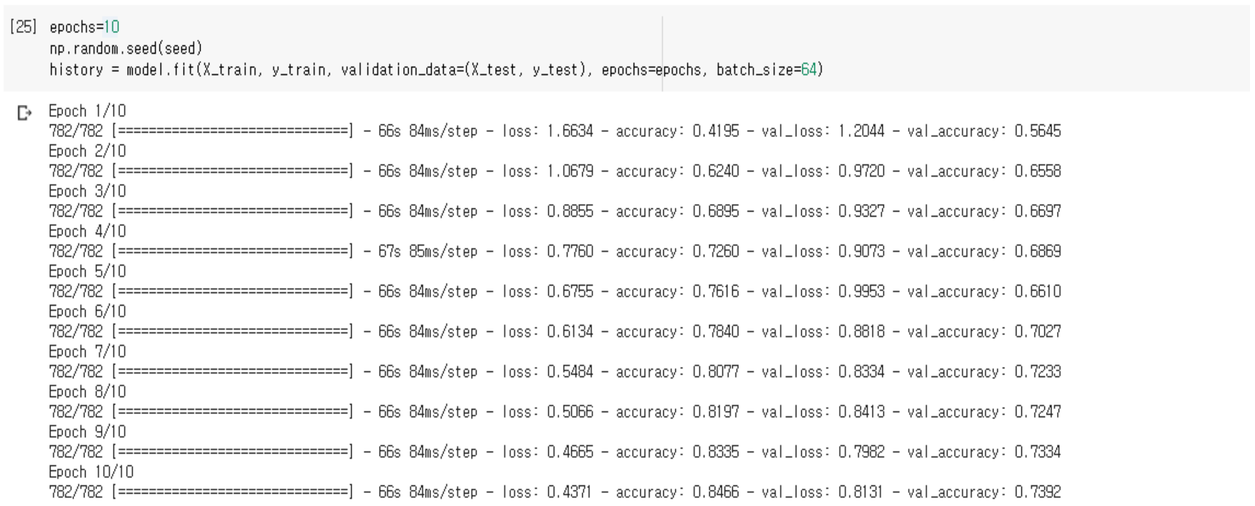
➪ **First of all, We need change the number of classes from 5 to 10. Because in this data set the number of classes are 10. Also we should change the last layer’s**

activation function that is because The reason why softmax is needed here is because it converts the output of the last layer in your neural network into what is essentially a probability distribution. So it will be better than relu when it comes to classifying multi classes.

2. Add a batch normalization to the model



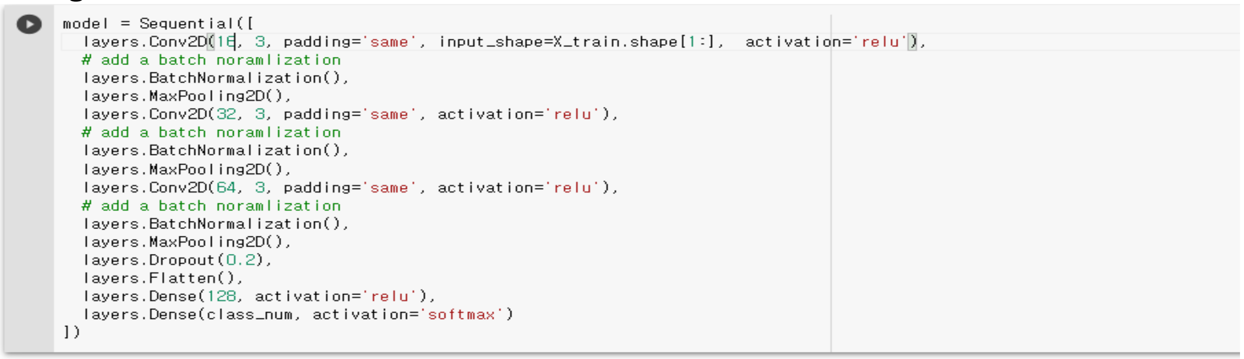
```
model = Sequential([
  layers.Conv2D(32, 3, padding='same', input_shape=X_train.shape[1:],  activation='relu'),
  # add a batch noramlization
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.3),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  # add a batch noramlization
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  # add a batch noramlization
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.4),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dropout(0.4),
  layers.Dense(class_num, activation='softmax')
])
```



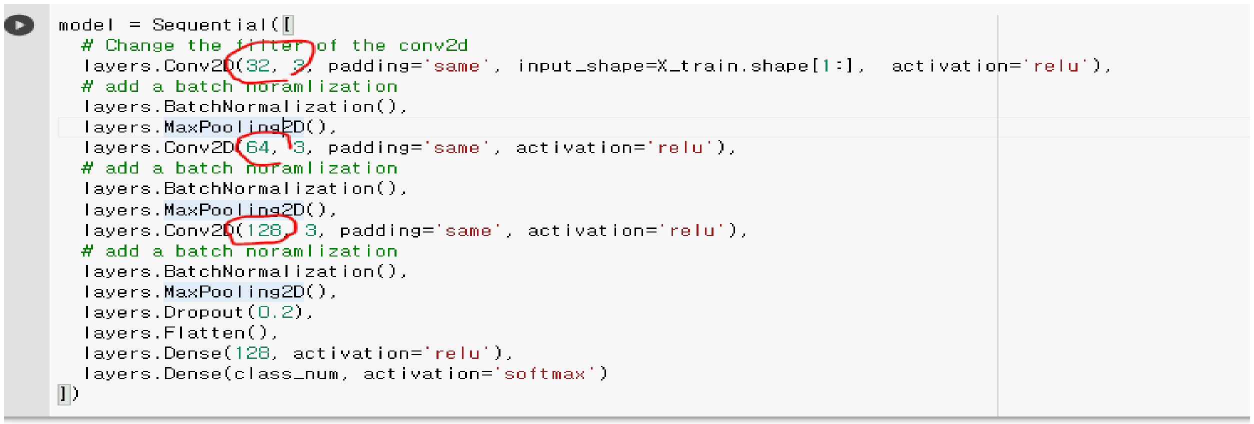
```
[25] epochs=10
     np.random.seed(seed)
     history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=64)

Epoch 1/10
782/782 [==============================] - 66s 84ms/step - loss: 1.6634 - accuracy: 0.4195 - val_loss: 1.2044 - val_accuracy: 0.5645
Epoch 2/10
782/782 [==============================] - 66s 84ms/step - loss: 1.0679 - accuracy: 0.6240 - val_loss: 0.9720 - val_accuracy: 0.6558
Epoch 3/10
782/782 [==============================] - 66s 84ms/step - loss: 0.8855 - accuracy: 0.6895 - val_loss: 0.9327 - val_accuracy: 0.6697
Epoch 4/10
782/782 [==============================] - 67s 85ms/step - loss: 0.7760 - accuracy: 0.7260 - val_loss: 0.9073 - val_accuracy: 0.6869
Epoch 5/10
782/782 [==============================] - 66s 84ms/step - loss: 0.6755 - accuracy: 0.7616 - val_loss: 0.9953 - val_accuracy: 0.6610
Epoch 6/10
782/782 [==============================] - 66s 84ms/step - loss: 0.6134 - accuracy: 0.7840 - val_loss: 0.8818 - val_accuracy: 0.7027
Epoch 7/10
782/782 [==============================] - 66s 84ms/step - loss: 0.5484 - accuracy: 0.8077 - val_loss: 0.8334 - val_accuracy: 0.7233
Epoch 8/10
782/782 [==============================] - 66s 84ms/step - loss: 0.5066 - accuracy: 0.8197 - val_loss: 0.8413 - val_accuracy: 0.7247
Epoch 9/10
782/782 [==============================] - 66s 84ms/step - loss: 0.4665 - accuracy: 0.8335 - val_loss: 0.7982 - val_accuracy: 0.7334
Epoch 10/10
782/782 [==============================] - 66s 84ms/step - loss: 0.4371 - accuracy: 0.8466 - val_loss: 0.8131 - val_accuracy: 0.7392
```

⇨ **After adding batchnormalization to the model, accuracy is increased to 0.8466. This is because Batch normalization works just the same way as we normalize the input data where we divided the x_train/255. What we are trying to do here is we are arranging all the features in same scale so that model converges easily and we can reduce the distrotions. We passs the CNN throuh a batch normalization layer we are normalizing the weights so that our model will be stable and we can train model longer and also use larger learning rate.**

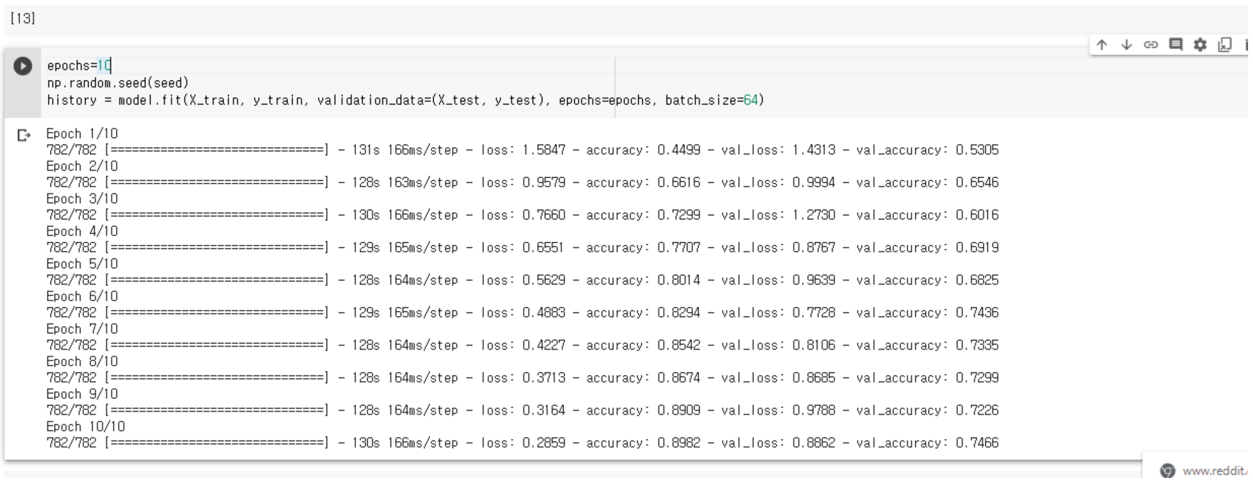
3. Change the filter number of Conv2d



```
model = Sequential([
  layers.Conv2D(16, 3, padding='same', input_shape=X_train.shape[1:],  activation='relu'),
  # add a batch noramlization
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  # add a batch noramlization
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  # add a batch noramlization
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(class_num, activation='softmax')
])
```



```
model = Sequential([
  # Change the filter of the conv2d
  layers.Conv2D(32, 3, padding='same', input_shape=X_train.shape[1:],  activation='relu'),
  # add a batch noramlization
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  # add a batch noramlization
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  # add a batch noramlization
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(class_num, activation='softmax')
])
```

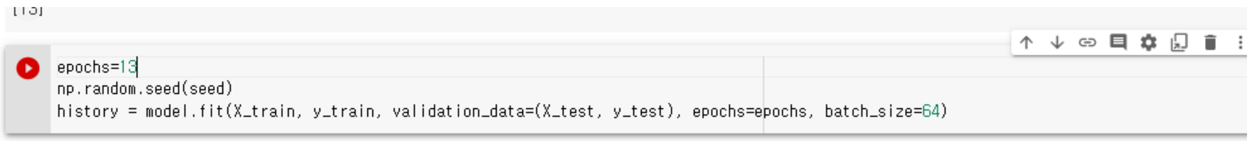


```
[13]

epochs=10
np.random.seed(seed)
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=64)

Epoch 1/10
782/782 [==============================] - 131s 166ms/step - loss: 1.5847 - accuracy: 0.4499 - val_loss: 1.4313 - val_accuracy: 0.5305
Epoch 2/10
782/782 [==============================] - 128s 163ms/step - loss: 0.9579 - accuracy: 0.6616 - val_loss: 0.9994 - val_accuracy: 0.6546
Epoch 3/10
782/782 [==============================] - 130s 166ms/step - loss: 0.7660 - accuracy: 0.7299 - val_loss: 1.2730 - val_accuracy: 0.6016
Epoch 4/10
782/782 [==============================] - 129s 165ms/step - loss: 0.6551 - accuracy: 0.7707 - val_loss: 0.8767 - val_accuracy: 0.6919
Epoch 5/10
782/782 [==============================] - 128s 164ms/step - loss: 0.5629 - accuracy: 0.8014 - val_loss: 0.9639 - val_accuracy: 0.6825
Epoch 6/10
782/782 [==============================] - 129s 165ms/step - loss: 0.4883 - accuracy: 0.8294 - val_loss: 0.7728 - val_accuracy: 0.7436
Epoch 7/10
782/782 [==============================] - 128s 164ms/step - loss: 0.4227 - accuracy: 0.8542 - val_loss: 0.8106 - val_accuracy: 0.7335
Epoch 8/10
782/782 [==============================] - 128s 164ms/step - loss: 0.3713 - accuracy: 0.8674 - val_loss: 0.8685 - val_accuracy: 0.7299
Epoch 9/10
782/782 [==============================] - 128s 164ms/step - loss: 0.3164 - accuracy: 0.8909 - val_loss: 0.9788 - val_accuracy: 0.7226
Epoch 10/10
782/782 [==============================] - 130s 166ms/step - loss: 0.2859 - accuracy: 0.8982 - val_loss: 0.8862 - val_accuracy: 0.7466
```

⇨ **The higher the number of filters, the higher the number of *abstractions* that your Network is able to extract from image data. The reason why the number of filters is generally ascending is that at the input layer the Network receives raw pixel data. So we can upgrade our model by adjusting filter number. By increasing the filter number of Conv2d the overall accuracy has been increased to 0.8982.**

⇨

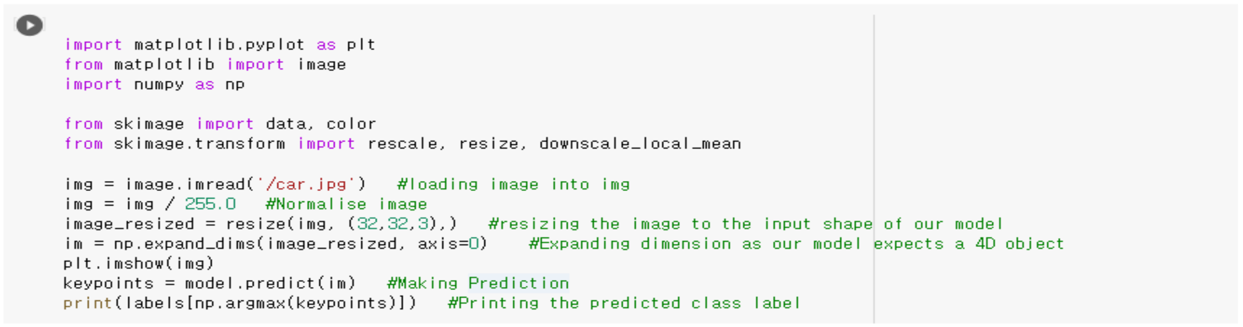
4. Increase the epoch number



```
[13]
epochs=13
np.random.seed(seed)
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=64)
```

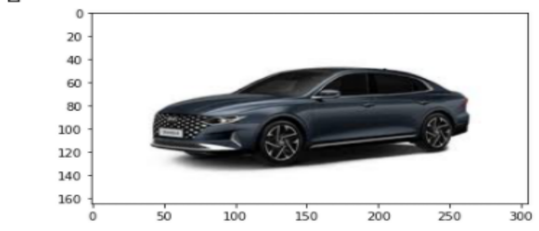
⇨ **Increase the epoch 10 to 13. I want to improve more accuracy by increasing accuracy But I found that If I set epoch more than 13 the validation accuracy tend to decrease So I limit the epoch number to 13**

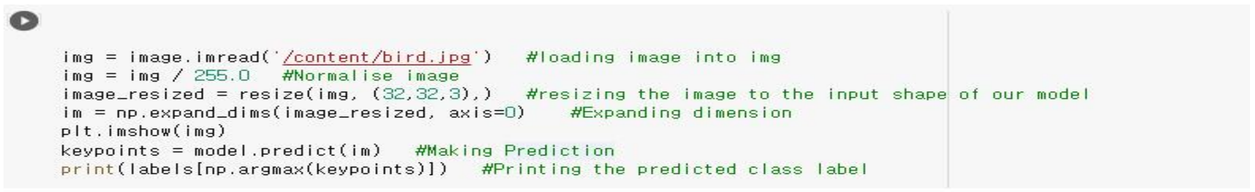
Validating the model on 5 new images



```
import matplotlib.pyplot as plt
from matplotlib import image
import numpy as np

from skimage import data, color
from skimage.transform import rescale, resize, downscale_local_mean

img = image.imread('/car.jpg')   #loading image into img
img = img / 255.0   #Normalise image
image_resized = resize(img, (32,32,3),)   #resizing the image to the input shape of our model
im = np.expand_dims(image_resized, axis=0)   #Expanding dimension as our model expects a 4D object
plt.imshow(img)
keypoints = model.predict(im)   #Making Prediction
print(labels[np.argmax(keypoints)])   #Printing the predicted class label
```

```
automobile
```





```
img = image.imread('/content/bird.jpg')   #loading image into img
img = img / 255.0   #Normalise image
image_resized = resize(img, (32,32,3),)   #resizing the image to the input shape of our model
im = np.expand_dims(image_resized, axis=0)   #Expanding dimension
plt.imshow(img)
keypoints = model.predict(im)   #Making Prediction
print(labels[np.argmax(keypoints)])   #Printing the predicted class label
```

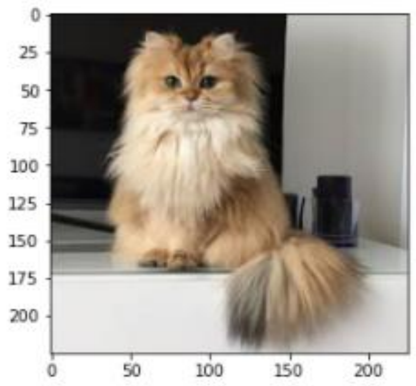
```
bird
```

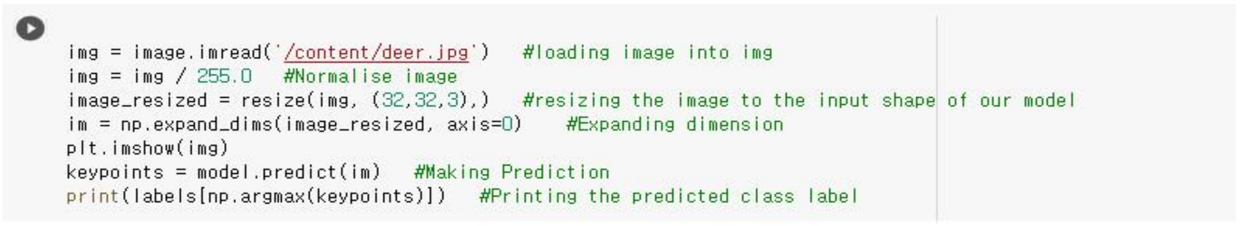
```
img = image.imread('/content/ship.jpg')   #loading image into img
img = img / 255.0   #Normalise image
image_resized = resize(img, (32,32,3),)   #resizing the image to the input shape of our model
im = np.expand_dims(image_resized, axis=0)   #Expanding dimension
plt.imshow(img)
keypoints = model.predict(im)   #Making Prediction
print(labels[np.argmax(keypoints)])   #Printing the predicted class label
```

```
ship
```

```
img = image.imread('/content/cat.jpg')   #loading image into img
img = img / 255.0   #Normalise image
image_resized = resize(img, (32,32,3),)   #resizing the image to the input shape of our mode
im = np.expand_dims(image_resized, axis=0) #Expanding dimension
plt.imshow(img)
keypoints = model.predict(im)   #Making Prediction
print(labels[np.argmax(keypoints)])   #Printing the predicted class label
```

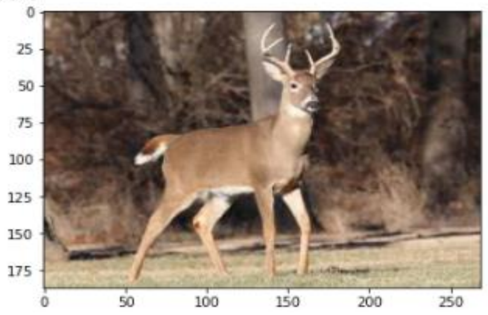
```
dog
```





```
img = image.imread('/content/deer.jpg')   #loading image into img
img = img / 255.0   #Normalise image
image_resized = resize(img, (32,32,3),)   #resizing the image to the input shape of our model
im = np.expand_dims(image_resized, axis=0)    #Expanding dimension
plt.imshow(img)
keypoints = model.predict(im)   #Making Prediction
print(labels[np.argmax(keypoints)])   #Printing the predicted class label
```

```
deer
```



- **I downloaded 5 random image from internet and predict the picture with the model that I created. In result 4 of 5 prediction is correct while one is wrong. So we can say it is pretty good model**

Challenges that I faced:

The most difficult challenge that I faced was that how to improve model and change the hyper parameter appropriately. I have no idea on this at first. But After exploring data and reviewing what We did on the class. I could finally get some insight about this.

Video link

https://www.youtube.com/watch?v=p_42cNk3zlM