experiment-8

September 22, 2025

Name:	Tufan Kundu
Registration no:	24MDT0184
Course Name:	Deep Learning Lab
Course Code:	PMDS603P
Experiment:	8
Date:	18 September,2025

0.1 Question 1: First try to fit a basic CNN model to accomplish the same task.Import necessary modules and classes first. Load the CIFAR 10 dataset and normalize the data and reshape. Now you can check the shape of xtrain and xtest etc which you are loading. we need to provide $32 \times 32 \times 3$ images to the CNN. Next compile the model and fit the model and check the performance. Also include early stopping in your model. Since we have a classification problem we have to use the softmax activation in the final layer.

0.1.1 Importing the necessary libraries

[1]:
```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout, Conv2D, MaxPool2D
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
import numpy as np

import warnings
warnings.filterwarnings('ignore')
```

```
2025-09-22 07:01:41.466626: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
```

```
E0000 00:00:1758524501.799542      19 cuda_dnn.cc:8310] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1758524501.892278      19 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
```

0.1.2 Loading and preprocessing the dataset

[2]:
```
(x_train,y_train),(x_test,y_test) = cifar10.load_data()
x_train = x_train/255.0
x_test = x_test/255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071            2s
0us/step
```

0.1.3 Displaying random images from the dataset

[3]:
```
class_names = ["Airplane","Automobile","Bird","Cat","Deer",
"Dog","Frog","Horse","Ship","Truck"]

indices = np.random.choice(len(x_train), size = 5, replace = False)
plt.figure(figsize = (2*5,3))
for i,idx in enumerate(indices):
    ax = plt.subplot(1,5,i+1)
    img = x_train[idx]
    plt.imshow(img)
    label = class_names[y_train[idx][0]]
    plt.title(label)
    plt.axis('off')
plt.show()
```

0.1.4 Building the CNN

[4]:
```
model = Sequential()
model.add(Conv2D(32,5, strides = (1,1), activation = 'relu', padding='same',␣
 ↪input_shape = x_train.shape[1:]))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2), padding='valid'))
model.add(Dropout(0.5))
model.add(Conv2D(32,3, strides = (1,1), activation = 'relu', padding='same'))
model.add(MaxPool2D(pool_size=(2,2), strides=(1,1), padding='same'))
model.add(Dropout(0.3))
model.add(Conv2D(64,3, strides = (1,1), activation = 'relu', padding='same'))
model.add(MaxPool2D(pool_size=(2,2), strides=(1,1), padding='valid'))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(256, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
I0000 00:00:1758524526.043863      19 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB memory:  -> device:
0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5
I0000 00:00:1758524526.044545      19 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB memory:  -> device:
1, name: Tesla T4, pci bus id: 0000:00:05.0, compute capability: 7.5
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	2,432
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 16, 16, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_1 (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 64)	0

dropout_2 (Dropout)	(None, 15, 15, 64)	0
flatten (Flatten)	(None, 14400)	0
dense (Dense)	(None, 256)	3,686,656
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1,290

Total params: 3,751,018 (14.31 MB)

Trainable params: 3,751,018 (14.31 MB)

Non-trainable params: 0 (0.00 B)

[5]:

```
estop = EarlyStopping(monitor = 'val_loss', min_delta= 1e-4, patience= 5,␣
↪verbose = 1, restore_best_weights=True)
model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam',␣
↪metrics = ['accuracy'])
```

[6]:

```
history = model.fit(x_train,y_train,batch_size=128, epochs = 200, verbose = 1,␣
↪validation_data=(x_test,y_test), callbacks=[estop])
```

```
Epoch 1/200
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1758524532.675782      60 service.cc:148] XLA service 0x79ade000e340
initialized for platform CUDA (this does not guarantee that XLA will be used).
Devices:
I0000 00:00:1758524532.677284      60 service.cc:156]   StreamExecutor device
(0): Tesla T4, Compute Capability 7.5
I0000 00:00:1758524532.677305      60 service.cc:156]   StreamExecutor device
(1): Tesla T4, Compute Capability 7.5
I0000 00:00:1758524533.089177      60 cuda_dnn.cc:529] Loaded cuDNN version
90300
```

```
 17/391            3s 11ms/step - accuracy:
0.0992 - loss: 2.5474
```

```
I0000 00:00:1758524538.780024      60 device_compiler.h:188] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.
391/391              18s 24ms/step -
accuracy: 0.1974 - loss: 2.1412 - val_accuracy: 0.4352 - val_loss: 1.5567
Epoch 2/200
391/391              3s 8ms/step -
accuracy: 0.4373 - loss: 1.5330 - val_accuracy: 0.5071 - val_loss: 1.3733
Epoch 3/200
391/391              3s 8ms/step -
accuracy: 0.5047 - loss: 1.3702 - val_accuracy: 0.5463 - val_loss: 1.2609
Epoch 4/200
391/391              3s 8ms/step -
accuracy: 0.5592 - loss: 1.2411 - val_accuracy: 0.5457 - val_loss: 1.3063
Epoch 5/200
391/391              3s 8ms/step -
accuracy: 0.5835 - loss: 1.1716 - val_accuracy: 0.5649 - val_loss: 1.2374
Epoch 6/200
391/391              3s 8ms/step -
accuracy: 0.6104 - loss: 1.1073 - val_accuracy: 0.5987 - val_loss: 1.1134
Epoch 7/200
391/391              3s 8ms/step -
accuracy: 0.6280 - loss: 1.0520 - val_accuracy: 0.6404 - val_loss: 1.0358
Epoch 8/200
391/391              3s 8ms/step -
accuracy: 0.6495 - loss: 0.9930 - val_accuracy: 0.6511 - val_loss: 0.9789
Epoch 9/200
391/391              3s 8ms/step -
accuracy: 0.6595 - loss: 0.9636 - val_accuracy: 0.6597 - val_loss: 0.9773
Epoch 10/200
391/391              3s 8ms/step -
accuracy: 0.6672 - loss: 0.9446 - val_accuracy: 0.6658 - val_loss: 0.9494
Epoch 11/200
391/391              3s 8ms/step -
accuracy: 0.6867 - loss: 0.8915 - val_accuracy: 0.6867 - val_loss: 0.8908
Epoch 12/200
391/391              3s 8ms/step -
accuracy: 0.6922 - loss: 0.8735 - val_accuracy: 0.6938 - val_loss: 0.8763
Epoch 13/200
391/391              3s 8ms/step -
accuracy: 0.6981 - loss: 0.8566 - val_accuracy: 0.6957 - val_loss: 0.8615
Epoch 14/200
391/391              3s 8ms/step -
accuracy: 0.7149 - loss: 0.8138 - val_accuracy: 0.6922 - val_loss: 0.8784
Epoch 15/200
391/391              3s 8ms/step -
accuracy: 0.7176 - loss: 0.8000 - val_accuracy: 0.7113 - val_loss: 0.8355
Epoch 16/200
```

```
391/391                3s 8ms/step -
accuracy: 0.7230 - loss: 0.7870 - val_accuracy: 0.6967 - val_loss: 0.8705
Epoch 17/200
391/391                3s 8ms/step -
accuracy: 0.7265 - loss: 0.7724 - val_accuracy: 0.7134 - val_loss: 0.8180
Epoch 18/200
391/391                3s 8ms/step -
accuracy: 0.7335 - loss: 0.7496 - val_accuracy: 0.7195 - val_loss: 0.7988
Epoch 19/200
391/391                3s 8ms/step -
accuracy: 0.7381 - loss: 0.7398 - val_accuracy: 0.7240 - val_loss: 0.7961
Epoch 20/200
391/391                3s 8ms/step -
accuracy: 0.7428 - loss: 0.7283 - val_accuracy: 0.7221 - val_loss: 0.7998
Epoch 21/200
391/391                3s 8ms/step -
accuracy: 0.7511 - loss: 0.7032 - val_accuracy: 0.7244 - val_loss: 0.7977
Epoch 22/200
391/391                3s 8ms/step -
accuracy: 0.7545 - loss: 0.6965 - val_accuracy: 0.7075 - val_loss: 0.8427
Epoch 23/200
391/391                3s 8ms/step -
accuracy: 0.7563 - loss: 0.6949 - val_accuracy: 0.7232 - val_loss: 0.7966
Epoch 24/200
391/391                3s 8ms/step -
accuracy: 0.7643 - loss: 0.6695 - val_accuracy: 0.7100 - val_loss: 0.8499
Epoch 24: early stopping
Restoring model weights from the end of the best epoch: 19.
```

[7]:
```
loss, val_accuracy = model.evaluate(x_test,y_test)
```

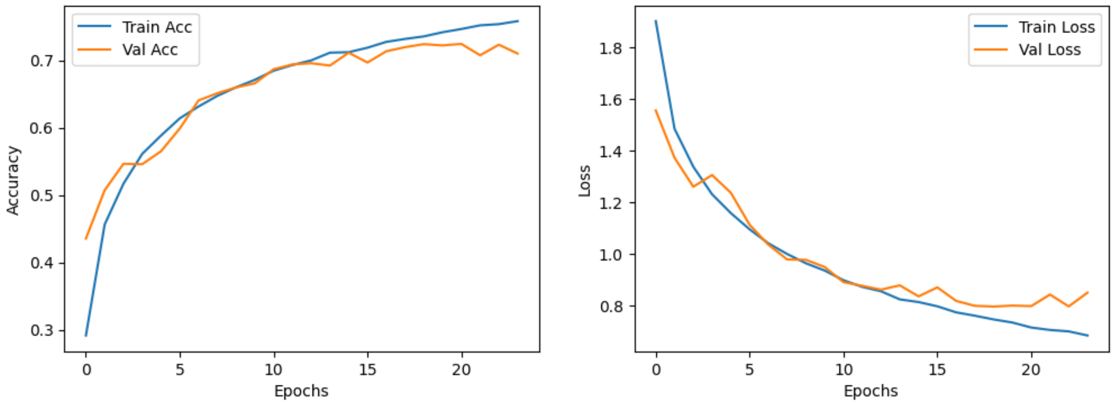
```
313/313                1s 2ms/step -
accuracy: 0.7251 - loss: 0.7901
```

[8]:
```
print(f"Validation Accuracy of Cifar10 dataset with CNN model:␣
↪{val_accuracy*100:.4f}%")
```

```
Validation Accuracy of Cifar10 dataset with CNN model: 72.4000%
```

[9]:
```
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
```

```
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel("Epochs")
plt.ylabel('Loss')
plt.legend()
plt.show()
```



0.2 Question 2: Next we will see how you can define a new model incorporating the pre-trained model VGG16 in your model as its part. For that first you have to import the respective model details along with the necessary modules and classes.

0.2.1 Importing the necessary libraries

[10]:
```
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense, Resizing, Dropout
from keras.applications import VGG16
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from keras.layers import GlobalAveragePooling2D
```

0.2.2 Loading and preprocessing of the dataset

[11]:
```
(x_train,y_train),(x_test,y_test) = cifar10.load_data()
x_train = x_train/255.0
x_test = x_test/255.0
```

[12]:
```
base_model = VGG16(include_top = False, weights = 'imagenet', input_shape =␣
↪(224,224,3))
for layer in base_model.layers:
    layer.trainable = False
```

```python
## Building the model
model = Sequential()
model.add(Resizing(224,224, input_shape = (32,32,3)))
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(100, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(50, activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(10, activation = 'softmax'))

model.summary()
model.compile(
    loss = 'sparse_categorical_crossentropy',
    optimizer = 'adam',
    metrics = ['accuracy']
)

estop = EarlyStopping(monitor = 'val_loss', min_delta= 1e-4, patience= 5,
 ↪verbose = 1, restore_best_weights=True)
history = model.fit(x_train,y_train,batch_size=128, epochs = 200, verbose = 1,
 ↪validation_data=(x_test,y_test), callbacks=[estop])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256              0s
0us/step
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 224, 224, 3)	0
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense_3 (Dense)	(None, 100)	51,300
dropout_5 (Dropout)	(None, 100)	0
dense_4 (Dense)	(None, 50)	5,050
dropout_6 (Dropout)	(None, 50)	0

```
 dense_5 (Dense)                  (None, 10)                         510


 Total params: 14,771,548 (56.35 MB)

 Trainable params: 56,860 (222.11 KB)

 Non-trainable params: 14,714,688 (56.13 MB)

Epoch 1/200
2025-09-22 07:03:59.900674: E
external/local_xla/xla/service/slow_operation_alarm.cc:65] Trying algorithm
eng11{k2=1,k3=0} for conv (f32[128,64,224,224]{3,2,1,0}, u8[0]{0}) custom-
call(f32[128,64,224,224]{3,2,1,0}, f32[64,64,3,3]{3,2,1,0}, f32[64]{0}),
window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
onv_backend_config":{"activation_mode":"kRelu","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_
id":"0","wait_on_operation_queues":[]} is taking a while…
2025-09-22 07:04:00.167501: E
external/local_xla/xla/service/slow_operation_alarm.cc:133] The operation took
1.267004147s
Trying algorithm eng11{k2=1,k3=0} for conv (f32[128,64,224,224]{3,2,1,0},
u8[0]{0}) custom-call(f32[128,64,224,224]{3,2,1,0}, f32[64,64,3,3]{3,2,1,0},
f32[64]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
onv_backend_config":{"activation_mode":"kRelu","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_
id":"0","wait_on_operation_queues":[]} is taking a while…
2025-09-22 07:04:10.300850: E
external/local_xla/xla/service/slow_operation_alarm.cc:65] Trying algorithm
eng36{k2=3,k3=0} for conv (f32[128,128,112,112]{3,2,1,0}, u8[0]{0}) custom-
call(f32[128,128,112,112]{3,2,1,0}, f32[128,128,3,3]{3,2,1,0}, f32[128]{0}),
window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
onv_backend_config":{"activation_mode":"kRelu","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_
id":"0","wait_on_operation_queues":[]} is taking a while…
2025-09-22 07:04:10.988668: E
external/local_xla/xla/service/slow_operation_alarm.cc:133] The operation took
1.687989021s
Trying algorithm eng36{k2=3,k3=0} for conv (f32[128,128,112,112]{3,2,1,0},
u8[0]{0}) custom-call(f32[128,128,112,112]{3,2,1,0}, f32[128,128,3,3]{3,2,1,0},
f32[128]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
```

```
onv_backend_config":{"activation_mode":"kRelu","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_
id":"0","wait_on_operation_queues":[]} is taking a while…
```

```
391/391              359s 822ms/step -
accuracy: 0.1941 - loss: 2.1650 - val_accuracy: 0.4281 - val_loss: 1.6345
Epoch 2/200
391/391              286s 731ms/step -
accuracy: 0.3755 - loss: 1.6891 - val_accuracy: 0.4951 - val_loss: 1.4140
Epoch 3/200
391/391              284s 727ms/step -
accuracy: 0.4337 - loss: 1.5387 - val_accuracy: 0.5288 - val_loss: 1.3201
Epoch 4/200
391/391              284s 726ms/step -
accuracy: 0.4705 - loss: 1.4653 - val_accuracy: 0.5440 - val_loss: 1.2820
Epoch 5/200
391/391              286s 731ms/step -
accuracy: 0.4826 - loss: 1.4295 - val_accuracy: 0.5642 - val_loss: 1.2317
Epoch 6/200
391/391              285s 728ms/step -
accuracy: 0.5031 - loss: 1.3861 - val_accuracy: 0.5725 - val_loss: 1.2089
Epoch 7/200
391/391              284s 726ms/step -
accuracy: 0.5128 - loss: 1.3609 - val_accuracy: 0.5762 - val_loss: 1.1942
Epoch 8/200
391/391              284s 727ms/step -
accuracy: 0.5156 - loss: 1.3495 - val_accuracy: 0.5866 - val_loss: 1.1713
Epoch 9/200
391/391              285s 730ms/step -
accuracy: 0.5256 - loss: 1.3329 - val_accuracy: 0.5889 - val_loss: 1.1565
Epoch 10/200
391/391              284s 727ms/step -
accuracy: 0.5339 - loss: 1.3103 - val_accuracy: 0.5987 - val_loss: 1.1308
Epoch 11/200
391/391              284s 727ms/step -
accuracy: 0.5375 - loss: 1.3088 - val_accuracy: 0.6039 - val_loss: 1.1226
Epoch 12/200
391/391              284s 727ms/step -
accuracy: 0.5435 - loss: 1.2854 - val_accuracy: 0.6087 - val_loss: 1.1319
Epoch 13/200
391/391              284s 727ms/step -
accuracy: 0.5457 - loss: 1.2880 - val_accuracy: 0.6098 - val_loss: 1.1193
Epoch 14/200
391/391              284s 727ms/step -
accuracy: 0.5492 - loss: 1.2716 - val_accuracy: 0.6082 - val_loss: 1.1163
Epoch 15/200
391/391              284s 727ms/step -
accuracy: 0.5489 - loss: 1.2674 - val_accuracy: 0.6166 - val_loss: 1.0942
```

```
Epoch 16/200
391/391            285s 729ms/step -
accuracy: 0.5596 - loss: 1.2488 - val_accuracy: 0.6235 - val_loss: 1.0898
Epoch 17/200
391/391            285s 729ms/step -
accuracy: 0.5587 - loss: 1.2463 - val_accuracy: 0.6172 - val_loss: 1.1146
Epoch 18/200
391/391            285s 730ms/step -
accuracy: 0.5645 - loss: 1.2449 - val_accuracy: 0.6239 - val_loss: 1.0686
Epoch 19/200
391/391            285s 729ms/step -
accuracy: 0.5654 - loss: 1.2334 - val_accuracy: 0.6188 - val_loss: 1.0937
Epoch 20/200
391/391            285s 729ms/step -
accuracy: 0.5656 - loss: 1.2420 - val_accuracy: 0.6291 - val_loss: 1.0641
Epoch 21/200
391/391            285s 729ms/step -
accuracy: 0.5717 - loss: 1.2233 - val_accuracy: 0.6211 - val_loss: 1.0839
Epoch 22/200
391/391            287s 735ms/step -
accuracy: 0.5653 - loss: 1.2345 - val_accuracy: 0.6279 - val_loss: 1.0642
Epoch 23/200
391/391            285s 730ms/step -
accuracy: 0.5699 - loss: 1.2209 - val_accuracy: 0.6235 - val_loss: 1.0740
Epoch 24/200
391/391            284s 727ms/step -
accuracy: 0.5702 - loss: 1.2209 - val_accuracy: 0.6323 - val_loss: 1.0532
Epoch 25/200
391/391            284s 727ms/step -
accuracy: 0.5755 - loss: 1.2150 - val_accuracy: 0.6375 - val_loss: 1.0556
Epoch 26/200
391/391            284s 727ms/step -
accuracy: 0.5780 - loss: 1.2083 - val_accuracy: 0.6209 - val_loss: 1.0729
Epoch 27/200
391/391            284s 727ms/step -
accuracy: 0.5727 - loss: 1.2148 - val_accuracy: 0.6356 - val_loss: 1.0585
Epoch 28/200
391/391            284s 727ms/step -
accuracy: 0.5758 - loss: 1.2060 - val_accuracy: 0.6363 - val_loss: 1.0438
Epoch 29/200
391/391            284s 727ms/step -
accuracy: 0.5808 - loss: 1.1903 - val_accuracy: 0.6377 - val_loss: 1.0486
Epoch 30/200
391/391            285s 728ms/step -
accuracy: 0.5827 - loss: 1.1939 - val_accuracy: 0.6358 - val_loss: 1.0545
Epoch 31/200
391/391            285s 729ms/step -
accuracy: 0.5841 - loss: 1.1966 - val_accuracy: 0.6379 - val_loss: 1.0461
```

```
Epoch 32/200
391/391            285s 729ms/step -
accuracy: 0.5817 - loss: 1.2005 - val_accuracy: 0.6398 - val_loss: 1.0341
Epoch 33/200
391/391            285s 728ms/step -
accuracy: 0.5861 - loss: 1.1897 - val_accuracy: 0.6439 - val_loss: 1.0354
Epoch 34/200
391/391            285s 729ms/step -
accuracy: 0.5830 - loss: 1.1858 - val_accuracy: 0.6321 - val_loss: 1.0636
Epoch 35/200
391/391            285s 730ms/step -
accuracy: 0.5806 - loss: 1.2009 - val_accuracy: 0.6395 - val_loss: 1.0410
Epoch 36/200
391/391            287s 733ms/step -
accuracy: 0.5901 - loss: 1.1758 - val_accuracy: 0.6483 - val_loss: 1.0253
Epoch 37/200
391/391            285s 729ms/step -
accuracy: 0.5869 - loss: 1.1792 - val_accuracy: 0.6478 - val_loss: 1.0335
Epoch 38/200
391/391            284s 727ms/step -
accuracy: 0.5905 - loss: 1.1744 - val_accuracy: 0.6396 - val_loss: 1.0356
Epoch 39/200
391/391            284s 727ms/step -
accuracy: 0.5869 - loss: 1.1850 - val_accuracy: 0.6454 - val_loss: 1.0205
Epoch 40/200
391/391            285s 729ms/step -
accuracy: 0.5890 - loss: 1.1850 - val_accuracy: 0.6466 - val_loss: 1.0345
Epoch 41/200
391/391            285s 730ms/step -
accuracy: 0.5962 - loss: 1.1602 - val_accuracy: 0.6416 - val_loss: 1.0266
Epoch 42/200
391/391            285s 730ms/step -
accuracy: 0.5937 - loss: 1.1743 - val_accuracy: 0.6493 - val_loss: 1.0205
Epoch 43/200
391/391            285s 730ms/step -
accuracy: 0.5897 - loss: 1.1756 - val_accuracy: 0.6413 - val_loss: 1.0251
Epoch 44/200
391/391            284s 727ms/step -
accuracy: 0.5935 - loss: 1.1702 - val_accuracy: 0.6468 - val_loss: 1.0196
Epoch 45/200
391/391            284s 727ms/step -
accuracy: 0.5928 - loss: 1.1625 - val_accuracy: 0.6456 - val_loss: 1.0331
Epoch 46/200
391/391            285s 729ms/step -
accuracy: 0.5928 - loss: 1.1719 - val_accuracy: 0.6543 - val_loss: 1.0122
Epoch 47/200
391/391            286s 731ms/step -
accuracy: 0.5940 - loss: 1.1685 - val_accuracy: 0.6461 - val_loss: 1.0156
```

```
Epoch 48/200
391/391            284s 727ms/step -
accuracy: 0.5970 - loss: 1.1505 - val_accuracy: 0.6520 - val_loss: 1.0174
Epoch 49/200
391/391            284s 727ms/step -
accuracy: 0.6017 - loss: 1.1496 - val_accuracy: 0.6428 - val_loss: 1.0254
Epoch 50/200
391/391            284s 727ms/step -
accuracy: 0.5991 - loss: 1.1509 - val_accuracy: 0.6548 - val_loss: 0.9992
Epoch 51/200
391/391            285s 730ms/step -
accuracy: 0.5973 - loss: 1.1534 - val_accuracy: 0.6526 - val_loss: 1.0083
Epoch 52/200
391/391            285s 729ms/step -
accuracy: 0.6000 - loss: 1.1559 - val_accuracy: 0.6431 - val_loss: 1.0277
Epoch 53/200
391/391            285s 729ms/step -
accuracy: 0.5993 - loss: 1.1635 - val_accuracy: 0.6437 - val_loss: 1.0237
Epoch 54/200
391/391            287s 733ms/step -
accuracy: 0.6047 - loss: 1.1439 - val_accuracy: 0.6591 - val_loss: 0.9959
Epoch 55/200
391/391            285s 729ms/step -
accuracy: 0.6036 - loss: 1.1447 - val_accuracy: 0.6589 - val_loss: 0.9961
Epoch 56/200
391/391            285s 729ms/step -
accuracy: 0.6051 - loss: 1.1465 - val_accuracy: 0.6444 - val_loss: 1.0154
Epoch 57/200
391/391            285s 729ms/step -
accuracy: 0.5945 - loss: 1.1630 - val_accuracy: 0.6589 - val_loss: 0.9936
Epoch 58/200
391/391            285s 730ms/step -
accuracy: 0.5931 - loss: 1.1541 - val_accuracy: 0.6588 - val_loss: 1.0005
Epoch 59/200
391/391            286s 732ms/step -
accuracy: 0.6005 - loss: 1.1477 - val_accuracy: 0.6532 - val_loss: 1.0074
Epoch 60/200
391/391            284s 727ms/step -
accuracy: 0.6020 - loss: 1.1494 - val_accuracy: 0.6514 - val_loss: 1.0351
Epoch 61/200
391/391            284s 726ms/step -
accuracy: 0.6037 - loss: 1.1454 - val_accuracy: 0.6525 - val_loss: 0.9974
Epoch 62/200
391/391            284s 727ms/step -
accuracy: 0.6030 - loss: 1.1469 - val_accuracy: 0.6548 - val_loss: 1.0089
Epoch 62: early stopping
Restoring model weights from the end of the best epoch: 57.
```

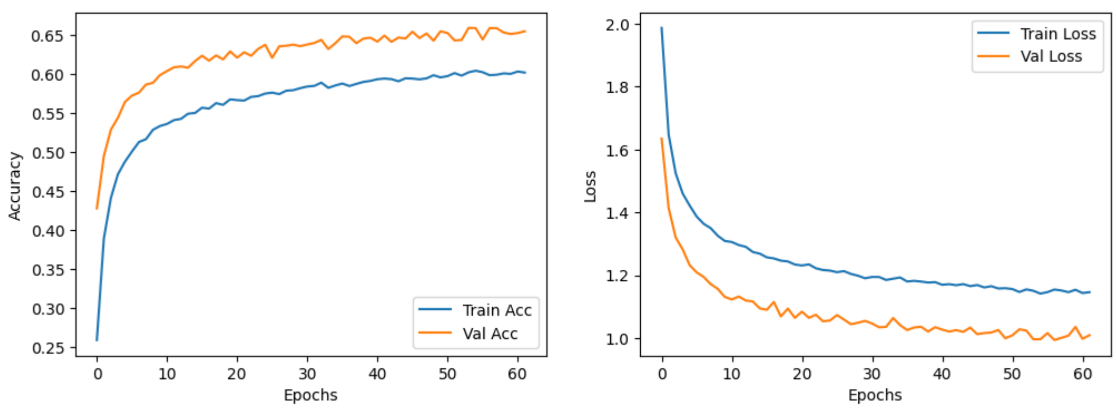
[13]:
```
loss, val_accuracy = model.evaluate(x_test,y_test)
```

```
313/313              76s 200ms/step -
accuracy: 0.6574 - loss: 0.9963
```

[14]:
```
print(f"Validation Accuracy of Cifar10 dataset with CNN model:␣
↪{val_accuracy*100:.4f}%")
```

```
Validation Accuracy of Cifar10 dataset with CNN model: 65.8900%
```

[15]:
```
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel("Epochs")
plt.ylabel('Loss')
plt.legend()
plt.show()
```



0.3 Question 3: Try to use the pre-trained model ResNet50 and design a new model in similar way and report the results obtained.

0.3.1 Importing the necessary libraries

[16]:
```
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense, Resizing, Dropout, GlobalAveragePooling2D
```

```python
from keras.applications import ResNet50
from keras.callbacks import EarlyStopping
from keras.datasets import cifar10
import matplotlib.pyplot as plt
```

0.3.2 Loading and preprocessing the dataset

[17]:
```python
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train / 255.0
x_test  = x_test / 255.0
```

[18]:
```python
base_model = ResNet50(
    include_top=False,
    weights='imagenet',
    input_shape=(224, 224, 3)
)

for layer in base_model.layers:
    layer.trainable = False
## Building the model
model = Sequential()
model.add(Resizing(224, 224, input_shape=(32, 32, 3)))
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(10, activation='softmax'))

model.summary()

model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

estop = EarlyStopping(
    monitor='val_loss',
    min_delta=1e-4,
    patience=5,
    verbose=1,
    restore_best_weights=True
)

history = model.fit(
```

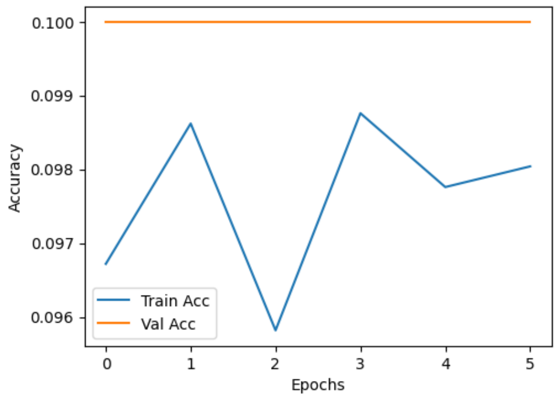
```
    x_train, y_train,
    batch_size=128,
    epochs=200,
    verbose=1,
    validation_data=(x_test, y_test),
    callbacks=[estop]
)

loss, val_accuracy = model.evaluate(x_test, y_test)
print(f"Validation Accuracy of CIFAR-10 with ResNet50 model: {val_accuracy*100:.
 ↪4f}%")

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel("Epochs")
plt.ylabel('Loss')
plt.legend()

plt.show()
```

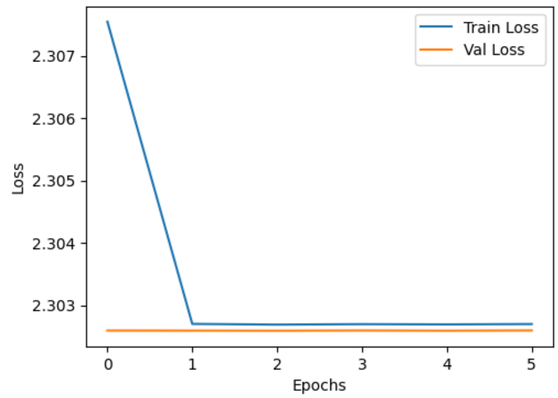
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736             0s
0us/step
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
resizing_1 (Resizing)	(None, 224, 224, 3)	0
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 2048)	0
dense_6 (Dense)	(None, 100)	204,900

```
dropout_7 (Dropout)              (None, 100)                    0
dense_7 (Dense)                  (None, 50)                 5,050
dropout_8 (Dropout)              (None, 50)                     0
dense_8 (Dense)                  (None, 10)                   510

Total params: 23,798,172 (90.78 MB)

Trainable params: 210,460 (822.11 KB)

Non-trainable params: 23,587,712 (89.98 MB)

Epoch 1/200
391/391             205s 460ms/step -
accuracy: 0.0983 - loss: 2.3250 - val_accuracy: 0.1000 - val_loss: 2.3026
Epoch 2/200
391/391             161s 411ms/step -
accuracy: 0.0999 - loss: 2.3027 - val_accuracy: 0.1000 - val_loss: 2.3026
Epoch 3/200
391/391             161s 411ms/step -
accuracy: 0.0963 - loss: 2.3027 - val_accuracy: 0.1000 - val_loss: 2.3026
Epoch 4/200
391/391             161s 412ms/step -
accuracy: 0.0992 - loss: 2.3027 - val_accuracy: 0.1000 - val_loss: 2.3026
Epoch 5/200
391/391             161s 412ms/step -
accuracy: 0.0975 - loss: 2.3027 - val_accuracy: 0.1000 - val_loss: 2.3026
Epoch 6/200
391/391             161s 412ms/step -
accuracy: 0.0972 - loss: 2.3027 - val_accuracy: 0.1000 - val_loss: 2.3026
Epoch 6: early stopping
Restoring model weights from the end of the best epoch: 1.
313/313             32s 87ms/step -
accuracy: 0.0995 - loss: 2.3026
Validation Accuracy of CIFAR-10 with ResNet50 model: 10.0000%
```


0.100
0.099
0.098
0.097
0.096
Accuracy
0
1
2
3
4
5
Epochs
Train Acc
Val Acc


2.307
2.306
2.305
2.304
2.303
Loss
0
1
2
3
4
5
Epochs
Train Loss
Val Loss